

2010

An interoperable and secure architecture for internet-scale decentralized personal communication

David Alan Bryan

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Communication Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Bryan, David Alan, "An interoperable and secure architecture for internet-scale decentralized personal communication" (2010). *Dissertations, Theses, and Masters Projects*. Paper 1539623560.

<https://dx.doi.org/doi:10.21220/s2-5ae4-2v74>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

An Interoperable and Secure Architecture for
Internet-Scale Decentralized Personal
Communications

David Alan Bryan

Williamsburg, Virginia

M.S. Computer Science, The College of William and Mary, 2000
B.A. Physics, The Richard Stockton College of New Jersey, 1996
B.S. Computer Science, The Richard Stockton College of New Jersey, 1995

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
May, 2010

Copyright © 2010 David Alan Bryan
All rights reserved

APPROVAL PAGE


This dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

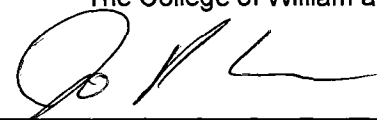


David Alan Bryan

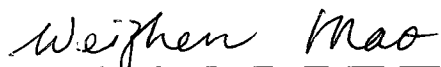
Approved by the Committee, April, 2010



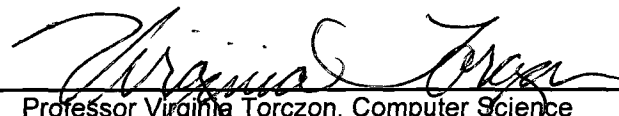
Committee Chair
Assistant Professor Bruce Lowekamp, Computer Science
The College of William and Mary



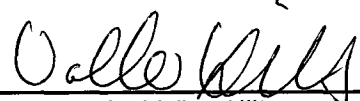
Chair and Associate Professor Phil Kearns, Computer Science
The College of William and Mary



Associate Professor Weizhen Mao, Computer Science
The College of William and Mary



Professor Virginia Torczon, Computer Science
The College of William and Mary



Dr. Volker Hilt
Bell Labs/Alcatel-Lucent

ABSTRACT PAGE

Interpersonal network communications, including Voice over IP (VoIP) and Instant Messaging (IM), are increasingly popular communications tools. However, systems to date have generally adopted a client-server model, requiring complex centralized infrastructure, or have not adhered to any VoIP or IM standard. Many deployment scenarios either require no central equipment, or due to unique properties of the deployment, are limited or rendered unattractive by central servers. To address these scenarios, we present a solution based on the Session Initiation Protocol (SIP) standard, utilizing a decentralized Peer-to-Peer (P2P) mechanism to distribute data. Our new approach, P2PSIP, enables users to communicate with minimal or no centralized servers, while providing secure, real-time, authenticated communications comparable in security and performance to centralized solutions.

We present two complete protocol descriptions and system designs. The first, the SOSIMPLE/dSIP protocol, is a P2P-over-SIP solution, utilizing SIP both for the transport of P2P messages and personal communications, yielding an interoperable, single-stack solution for P2P communications. The RELOAD protocol is a binary P2P protocol, designed for use in a SIP-using-P2P architecture where an existing SIP application is modified to use an additional, binary RELOAD stack to distribute user information without need for a central server.

To meet the unique security needs of a fully decentralized communications system, we propose an enrollment-time certificate authority model that provides asserted identity and strong P2P and user-level security. In this model, a centralized server is contacted only at enrollment time. No run-time connections to the servers are required.

Additionally, we show that traditional P2P message routing mechanisms are inappropriate for P2PSIP. The existing mechanisms are generally optimized for file sharing and neglect critical practical elements of the open Internet – namely link-level security and asymmetric connectivity caused by Network Address Translators (NATs). In response to these shortcomings, we introduce a new message routing paradigm, Adaptive Routing (AR), and using both analytical models and simulation show that AR significantly improves message routing performance for P2PSIP systems.

Our work has led to the creation of a new research topic within the P2P and interpersonal communications communities, P2PSIP. Our seminal publications have provided the impetus for subsequent P2PSIP publications, for the listing of P2PSIP as a topic in conference calls for papers, and for the formation of a new working group in the Internet Engineering Task Force (IETF), directed to develop an open Internet standard for P2PSIP.

To my wonderful wife Marcia and my son Alexander

Table of Contents

Acknowledgments	x
List of Figures	xi
1 Introduction	2
1.1 Research Problem	4
1.2 Requirements for a Solution	5
1.3 Why P2PSIP?	6
1.3.1 Why P2P and DHTs?	6
1.3.2 Why SIP?	7
1.4 Contributions	8
1.5 Outline	10
2 Background	12
2.1 Use cases	12
2.1.1 Groups of Users or Devices Best Served without Servers	13
2.1.2 Network or Equipment Failure Driven Scenarios	16
2.1.3 Social and Political Motivations	18
2.1.4 Large-scale or Global Communications Systems	20
2.2 NATs and NAT Traversal	21
2.3 Peer-to-Peer Systems	24

2.3.1	Defining P2P and Comparing with Client-Server	24
2.3.2	Applications of P2P Architectures	28
2.3.3	Unstructured P2P	29
2.3.4	Structured P2P	30
2.3.4.1	Distributed Hash Tables	31
2.3.4.2	Chord	34
2.3.4.3	Routing in DHTs	36
2.4	Network-based Personal Communications Systems	38
2.4.1	VoIP and IM Systems	38
2.4.2	SIP and SIMPLE	40
2.4.2.1	Registration	43
2.4.2.2	Establishing and Terminating Calls	43
2.4.2.3	SIMPLE: SIP for IM and Presence	45
2.5	Related Work	46
2.5.1	Using SIP for Other P2P Applications	47
2.5.2	Non-standards Based P2P Communications	47
2.5.3	P2PSIP Research	50
2.6	Terminology and Glossary	50
3	The Design of the Peer-to-Peer SIP (P2PSIP) Protocol	55
3.1	Choosing a Mechanism to Distribute Information	56
3.1.1	Requirements for Communications	56
3.2	Assignment of Peer-IDs	58
3.2.1	Why Users Cannot Control Peer-ID Assignment	58
3.2.2	Problems with IP-based Peer-IDs	60
3.3	Distributed Identity	61
3.3.1	Problems with Early Proposals	62
3.4	Securing the System with an Enrollment-time Certificate Server	64

3.4.1	Architecture of an Enrollment-time Certificate Server System	65
3.4.2	DHT Operations with Certificates	67
3.4.3	User Operations with Certificates	68
3.5	Providing Secure Redundancy	69
3.6	Recommendations for “Best Effort” Security in Ad-hoc Systems	70
3.7	Summary	71
4	The SOSIMPLE/dSIP and RELOAD Protocols	72
4.1	SOSIMPLE/dSIP: A SIP-Based Protocol for P2PSIP	73
4.1.1	Why P2P-over-SIP?	74
4.1.2	The SOSIMPLE/dSIP P2PSIP Architecture	74
4.1.3	Protocol Design of SOSIMPLE/dSIP	75
4.1.3.1	Use of SIP Messages	75
4.1.3.2	DHT Algorithms and IDs	76
4.1.3.3	Routing	78
4.1.3.4	Peer Operations	78
4.1.3.5	User Operations	81
4.1.3.6	NAT Traversal	84
4.1.3.7	Offline Message Storage, IM and Presence	85
4.2	RELOAD: A Binary Protocol for P2PSIP	86
4.2.1	Why SIP-using-P2P	86
4.2.2	The RELOAD P2PSIP Architecture	87
4.2.3	Protocol Design of RELOAD	88
4.2.3.1	Binary Messages	88
4.2.3.2	DHT Algorithms and IDs	89
4.2.3.3	Routing	89
4.2.3.4	P2PSIP Resources and Resource Registration	90
4.2.3.5	Peer Operations	90

4.2.3.6	Resource Operations	91
4.2.3.7	Large Messages and Fragmentation	92
4.2.3.8	NAT Traversal and Transport Operations	92
4.3	Summary	93
5	Message Routing for P2PSIP	94
5.1	Narrowing the Field	95
5.1.1	Eliminating Iterative Routing	97
5.1.2	Eliminating Forward-Only Routing	99
5.2	Return Routing and Asymmetric Connectivity	100
5.2.1	Deployment Environments	101
5.2.2	Defining An Analytical Model for Comparing Response Times	101
5.2.2.1	Modeling Symmetric Recursive Response Routing (SR) . .	103
5.2.2.2	Modeling Simple Direct Response Routing (SD)	104
5.2.2.3	Modeling Practical Direct Response Routing (DR)	105
5.2.3	Analytical Model Results	106
5.2.3.1	Comparing Message Count	106
5.2.3.2	Comparing Propagation Time	109
5.2.3.3	Comparing Processing by Intermediate Peers	110
5.2.3.4	Selecting a Technique	111
5.3	Adaptive Response (AR)	113
5.3.1	Evaluating Adaptive Response Analytically	114
5.3.1.1	Analytical Results for Adaptive Response	116
5.3.2	Evaluating Adaptive Response through Simulation	122
5.3.2.1	Simulating Hop Counts	124
5.3.2.2	Simulation Routing Times	126
5.3.3	Results of the Analysis	127
5.4	Other Concerns and Topics in P2P Routing	128

5.4.1	DoS Attack and Parallelization Risks	128
5.4.2	A (seemingly) Dead End: Social Network Routing	130
5.5	Summary	132
6	Conclusion	133
6.1	Results and Contributions	133
6.2	Future Work	135
6.3	Conclusion	137
A	dSIP Protocol Specification	138
A.1	Message Syntax	138
A.1.1	Option Tags	138
A.1.2	Hash Algorithms and Identifiers	139
A.1.2.1	Peer-IDs	139
A.1.2.2	Resource-IDs and the Replication	140
A.1.3	P2PSIP URIs	140
A.1.3.1	Peer URIs	141
A.1.3.2	Resource URIs and the resource-ID URI Parameter	142
A.1.4	The DHT-PeerID Header and Overlay Parameters	143
A.1.4.1	Hash Algorithms and the algorithm Parameter	144
A.1.4.2	Overlay Names and the overlay Parameter	145
A.1.4.3	DHT Algorithms and the dht Parameter	146
A.1.4.4	PeerID Expires header parameter	147
A.1.5	The DHT-Link Header	147
A.1.5.1	Expires Processing	148
A.2	Message Routing	148
A.2.1	Peer Registration	149
A.2.2	Resource Registration	150

A.2.3	Session Establishment	150
A.2.4	DHT Maintenance	151
A.3	Peer/DHT Operations	151
A.3.1	Peer Registration	152
A.3.1.1	Constructing a Peer Registration	152
A.3.1.2	Processing the Peer Registration	154
A.3.2	Peer Query	157
A.3.2.1	Constructing a Peer Query Message	157
A.3.2.2	Processing Peer Query Message	158
A.3.3	Populating the Joining Peer's Routing Table	159
A.3.4	Transferring User Registrations	160
A.3.5	Peers Leaving the Overlay Gracefully	160
A.3.6	NAT and Firewall Traversal	160
A.3.7	Handling Failed Requests	161
A.4	Resource Operations	161
A.4.1	Resource Registrations	161
A.4.2	Refreshing Resource Registrations	162
A.4.3	Removing Resource Registrations	163
A.4.4	Querying Resource Registrations	163
A.4.5	Session Establishment	164
A.4.6	Presence	164
A.4.7	Offline Storage	165
A.5	Pluggable DHT Algorithm Requirements	165
A.6	Security Considerations	166
A.6.1	Threat Model	167
A.6.2	Protecting the ID Namespace	167
A.6.2.1	Protection Using ID Hashing	168

A.6.2.2	Cryptographic Protection	169
A.6.3	Protecting the Resource Namespace	169
A.6.4	Protecting the Routing	170
A.6.5	Protecting the Signaling	170
A.6.6	Protecting the Media	171
B	RELOAD Protocol Specification	172
B.1	Message Syntax	172
B.1.1	Message Header	172
B.1.2	Message Attributes	177
B.1.2.1	RESPONSE-CODE	179
B.1.2.2	SOURCE-INFO and PEER-INFO	181
B.1.2.3	RESOURCE and RESOURCE-INFO	183
B.1.2.4	ROUTE-LOG	186
B.1.2.5	REDIRECT	188
B.1.2.6	SIGNATURE	188
B.1.2.7	SDP	188
B.1.2.8	APPLICATION	189
B.1.3	Adding New Attributes	190
B.2	Hash Algorithms and Identifiers	191
B.2.1	Hash Algorithms	191
B.2.1.1	Peer-IDs	191
B.3	Message Routing	192
B.3.1	SIP Session Establishment	193
B.4	Peer and DHT Operations	194
B.4.1	Peer Registration	194
B.4.1.1	Constructing a Peer Registration	194
B.4.1.2	Processing the Peer Registration	196

B.4.2	Peer Query	198
B.4.2.1	Constructing a Peer Query Message	198
B.4.2.2	Processing Peer Query Message	200
B.4.3	Populating the Joining Peer's Routing Table	200
B.4.4	Transferring User Registrations	201
B.4.5	Peers Leaving the Overlay Gracefully	201
B.4.6	NAT and Firewall Traversal	201
B.4.7	Handling Failed Requests	202
B.5	Resource Operations	202
B.5.1	Resource Registrations	202
B.5.2	Refreshing Resource Registrations	203
B.5.3	Removing Resource Registrations	204
B.5.4	Querying Resource Registrations	204
B.5.5	SIP Session Establishment	205
B.5.6	Offline Storage	206
B.6	Pluggable DHT Algorithm Requirements	206
B.7	Security Considerations	207
B.7.1	Protection Using ID Hashing	207
B.7.2	Protecting the Signaling	208
B.7.3	Replay Attacks	209
C	Tables of Simulation Results	210
	Bibliography	213
	Vita	222

ACKNOWLEDGMENTS

I would like most of all to acknowledge my best friend, constant companion, and wife Marcia, who has supported me in innumerable ways while I have worked on my thesis, and provided advice and much needed pep-talks on the many occasions it seemed I would never find time to finish. I'd also like to thank my family for their encouragement over the many, many years I've been working on my Doctorate, and for providing me with the education and opportunities that made me who I am today.

Special thanks to my team at SIPeerior Technologies (including Tiffany Broadbent, Jim Deverick, Mike Lehmann, Bruce Lowekamp, Tom O'Connor, Douglas Wadkins, Marcia Zangrilli, and many others) who helped me turn some of my early ideas in this thesis into a real-world product, and in so doing, provided inspiration and impetus for some of my later research.

As portions of publications and IETF drafts I have co-authored with others are included herein (including portions verbatim from [15–20, 22, 25, 50]), I would like thank my co-authors: Bruce Lowekamp, Marcia Zangrilli, Feng Cao, Spencer Dawkins, Cullen Jennings, Enrico Marocco, Philip Matthews, Eunsoo Shim, and Dean Willis.

I'd also like to thank my advisor (Bruce Lowekamp) and committee (Phil Kearns, Weizhen Mao, Virginia Torczon and Volker Hilt) for their valuable feedback. The IETF participants who reviewed my work and/or provided valuable insights, particularly Gonzalo Camarillo, Eric Cooper, Adrian Georgescu, Alan Johnston, Henning Schulzrinne and Henry Sinnreich, also deserve my gratitude.

Portions of this work were supported by grants from the Virginia Space Grant Consortium, The William and Mary Reves Center, and Cisco Systems.

List of Figures

2.1	An Example of a Network Deployment with a NAT	22
2.2	A Simplistic Conceptual Comparison of P2P and Client-Server Architectures	26
2.3	A Simplistic Conceptual Comparison of Structured and Unstructured P2P Overlays	29
2.4	An Example of Hashing Identifiers and Peer IDs into the Same Namespace	32
2.5	Example Ring-like DHT	33
2.6	An Iterative Routing Flow	37
2.7	A Symmetric Recursive Routing Flow	38
2.8	A Direct Response Routing Flow	38
2.9	A Strict Forwarding Routing Flow	38
2.10	Call Flow for a Basic, Successful, Proxy-brokered SIP Call	44
3.1	Examples of Placed ID and Sybil Attacks	59
3.2	Using Replica Keys for Distributed Replication	70

4.1	An Example of a New Peer With Peer-ID 503 Joining the Overlay	80
4.2	Alice Locates User Bob and Establishes a Communications Session (Iterative Routing)	82
4.3	Example Redundant Storage of Registrations Using Successor Method . .	84
5.1	MC vs. P by routing type	107
5.2	P_{EQ} vs. N for SR/DR routing	108
5.3	T vs. P by routing technique	109
5.4	P'_{EQ} vs. N for SR/DR routing	111
5.5	MC vs P ($N = 10K$)	116
5.6	MC vs. P ($N = 1M$)	117
5.7	MC vs C ($P = .5$)	118
5.8	MC vs C ($P = .9$)	119
5.9	T vs. P ($N = 10K$)	119
5.10	T vs. P ($N = 1M$)	120
5.11	T vs C ($P = .5$)	121
5.12	T vs C ($P = .9$)	121
5.13	Simulation Results, Hops	125
5.14	Simulation Results, Time	126
5.15	Simulation Results, Time (Detail)	127

5.16 Surrogate Attackers in a Recursive Approach	129
--	-----

**An Interoperable and Secure Architecture for Internet-Scale
Decentralized Personal Communications**

Chapter 1

Introduction

Voice over IP (VoIP) and Instant Messaging (IM) systems are increasingly popular communications systems for private, corporate, and academic purposes. Examples of these kind of systems include AOL's AIM, Microsoft's MSN Messenger, Yahoo's Y! Messenger, and Google's GTalk [4, 42, 64, 103]. Many centralized VoIP and IM systems are based on protocols specified as standards by the Internet Engineering Task Force (IETF) [49], such as the Session Initiation Protocol (SIP, for VoIP), SIP for IM and Presence (SIMPLE) and the Extensible Messaging and Presence Protocol (XMPP, for IM) [24, 79, 84, 87, 88]. The adoption of common standards has led to extensive development effort of, broad adoption of, and a focus on interoperability between applications using these protocols. These technologies can be extended to handle video/vision, resource sharing, and even remote control of physical hardware devices. Applications based on these protocols are now being used for more esoteric purposes, such as remote rehabilitation of injured persons out of reach of medical care, distributing information and communication for air traffic control, for wearable/on person computing environments, document sharing, communications within aircraft simulators, and many other purposes [2, 32, 102]. We discuss the Session Initiation Protocol (SIP), and communications systems in general, in more detail in section 2.4.

While these centralized systems are increasingly popular, the choice of a client-server architecture with a central server introduces a number of limitations. Despite their popularity, the need for central servers, whether maintained locally or by third parties, limits growth and places a burden on users. Many large companies concerned about intellectual property ban the use of centralized commercial IM systems. These systems consist of one or more client programs, run by end users, connected to one or a few centralized servers. There are a number of places where this approach is impractical. For small organizations, the costs, configuration and maintenance of these systems for communications can pose an insurmountable obstacle. For ad-hoc systems, small networks of nodes (such as sensor networks), or networks in locations with limited connectivity the need for a central server also poses a number of challenges.

To address these problems, we combine standards-based communications systems with Peer-to-Peer (P2P) mechanisms for locating resources, resulting in a fully distributed, and widely interoperable communications system we call P2PSIP. P2P computing has emerged as a mechanism for decentralized, server-free implementations of various applications. In a P2P architecture, a number of individual nodes or peers work together in concert to provide a service or store resources that would otherwise be stored by the central server. In particular, one form of P2P, the Distributed Hash Table (DHT) allows for rapid, deterministic location of resources and routing of information between devices. We present additional background on P2P and DHTs in Section 2.3. By removing the centralized server, communications systems can be quickly set up and torn down, scale without need for server management, be deployed in environments without Internet connectivity, and be inexpensively managed, even by very small organizations. In Section 2.1 we present detailed use cases motivating our work.

1.1 Research Problem

While a great deal of work has been done in both the areas of P2P and communications, very little work has addressed how to combine these technologies to enable interpersonal communications in new and novel ways. Our work focuses on solving the problems that arise when constructing a P2P personal communications system:

- What changes are required to existing approaches and protocols for interpersonal communications when centralized servers are removed?
- Existing P2P technology has often been motivated by file sharing or distributed file service considerations. In general, it is not optimized for the real-time constraints and traffic patterns required for locating and connecting users. Communications systems require guaranteed reachability and confirmation of remote identity, in contrast to the anonymity typically present in file sharing applications. In addition, real-time constraints pose additional security challenges that differ from file sharing. What changes are needed to enable P2P to be used for interpersonal communications?
- Because distributed communications systems must support users located in networks with non-optimal properties, particularly non-symmetric network connectivity caused by Network Address Translators (NATs), what modifications to existing P2P approaches are required to support these environments?
- In the absence of a run-time central authority, how can we properly secure a communications system and, using the primitives provided by a P2P solution, provide a reasonable level of asserted identity?

1.2 Requirements for a Solution

The environments we address with our P2P solution impose the following set of requirements:

- **No (or limited) Central Server/Connectivity:** In several scenarios, security concerns or connectivity restrictions prohibit contacting an outside server, therefore we require none be present. A central server for certificate issuance may be required in the global communications scenario. In such a case, this central server must not be needed except to issue the certificate to a new user.
- **Compatibility and Reuse:** The system must be compatible with as much existing infrastructure as possible, including VoIP and IM clients, and VoIP gateways. Code should be reused where possible. Users should be able to move from one client implementation to another and still participate in the same communications community.
- **Scalable Number of Users:** Additional users must be able to join the system, and the system should grow in response. No new resources, other than those the new user brings, should be required.
- **Scalable Security:** The system should allow for deployments with differing security levels. Ad-hoc and ephemeral systems may have very limited security needs. Conventional personal communications systems, either for use within an office or at a global scale require stronger security to prevent disruption. Some deployments, such as systems designed for anonymous communication or to circumvent censorship require strong but atypical security mechanisms.

- **Limit Information to System Users:** Users should be able to locate other users without having to go outside the system. Messages between users must not leave the system or be interceptable by other users within the system.
- **User Mobility Support:** Users should be able to move from device from device (from a mobile device in the field to a device in an office, for example) and maintain access to configuration information, friend lists, etc.
- **Multiple, Interconnected Communications Communities:** Users should be able to configure their own instance of a communication system and limit access. We refer to each such instance as a *Communications Community*. A security conscious group would typically utilize a private community for internal messages, and interact with external communities for outside communications. While traffic within a community should be restricted to that community, we must support interconnecting independent, separate communities, and allow some mechanism for users to communicate between these realms. In order to preserve privacy, this can not be done by implicit merging of the communities.

1.3 Why P2PSIP?

1.3.1 Why P2P and DHTs?

Our requirements of No Central Server, No Central Naming Authority, Scalable Number of Users, and User Mobility Support led to our decision to use P2P. Dynamic versions of the Domain Name System (DNS) [97, 100] have been suggested as one approach to create a serverless communication system. While a dynamic DNS approach addresses some of the issues we raised, there are several scenarios in which such an approach is lacking. Dynamic DNS requires a central DNS server be available. In situations where there is no

external connectivity, such as ad-hoc deployments or deployments in remote locations or in the developing world, this is not possible. Additionally, unless users maintain their own DNS servers, this is not a service a typical end user has access to. Finally, if device mobility is important, the latency in making changes to DNS may be undesirable.

Similarly, one possibility is to take a broadcast, replicate everything everywhere approach, in which all peers store all resources and broadcast updates to every peer. A very early commercial distributed telephone system, offered by NimCat Networks, took this very approach. This is very easy to implement, but unfortunately has numerous drawbacks. Broadcasts are not supported in version 6 of the Internet Protocol (IPv6) and have a very difficult time spanning networks boundaries, such as routers. While IP-multicast can address these limitations, a solution based on this approach still has obvious scalability problems at the Internet level. While this approach may work for a few hundred users, storing location information for the millions of users required by a global Internet deployment on every peer is clearly not possible.

Given the limitations of these approaches, the most obvious alternative, and the one we have chosen to pursue, is to take a P2P approach.

1.3.2 Why SIP?

SIP has emerged as the standard protocol for VoIP. Although some systems use older standards or proprietary mechanisms, the majority of new VoIP developments use SIP. The availability of the SIMPLE IM extensions means that a single design can be used for IM as well as voice and video.

Using the open SIP/SIMPLE standard helps us meet our requirement for compatibility and reuse, and allows the system to leverage open source stacks and applications. Additionally, the approach allows integration with existing SIP phones, SIMPLE IM clients,

and SIP gateways, used to enable SIP networks to connect to the PSTN (Public Switched Telephone Network – the current conventional phone network).

We also feel that this protocol was the most extensible of the existing VoIP/IM protocols, and was the most P2P in nature already, due to the fact that SIP nodes can communicate directly with one another with no intervening servers, so long as they can locate each other. In SIP, this location service is provided by a centralized registration server. As a DHT is essentially a distributed location mechanism, SIP is well suited to being combined with a DHT, having already been designed with location services largely decoupled from call control mechanisms, and ripe for replacement with a DHT-based location service.

1.4 Contributions

We present our work in creating a fully functional, serverless P2PSIP communications system, including the earliest published work in the field. Our work includes the creation of two protocols for P2PSIP, research into increasing the integrity of P2P systems via secure assignment of Peer-IDs, development of an enrollment-time distributed identity mechanism, and analysis of existing P2P routing techniques, leading to the development of a new P2P response routing mechanism. As a result of our work, a working group has been formed at the IETF, and P2PSIP has become an active topic area in both the personal communications and P2P spaces, frequently mentioned as a topic in P2P conference calls for papers. Specifically, our contributions include:

- Creating a fully-distributed, open P2P system for VoIP and IM by extending existing SIP standards. We have developed and present two complete protocols, one using SIP messages to convey P2P information (P2P-over-SIP) and one using a binary

encoding method (SIP-using-P2P). Both protocols are presented in Chapter 4 with additional details in Appendices A and B

- Creation of a new response routing protocol to address the inadequate NAT traversal and security properties of existing techniques. In Chapter 5 we present an analytical framework to analyze existing P2P routing techniques, taking the costs of NAT traversal and security into account. We present a novel response routing technique for P2P systems, Adaptive Response, developed in response to deficiencies in existing response routing techniques revealed by our analysis. We show through both application of our analytical model and simulation that Adaptive Response offers a better performance balance for both message count and response time than existing routing techniques in many common network environments.
- While emerging standards for NAT traversal make systems with more fully participating peers possible, securing a system using NATed peers remains challenging. The mechanisms proposed by most DHTs for generating Peer-IDs are vulnerable to attack. We demonstrate that, although hashing can be made more secure, providing provable security in light of an attacker's capabilities is unlikely to be completely successful. In particular, NATs and the adoption of IPv6 impose constraints on Peer-ID generation for deployments on the open Internet. Further, assuring unique identity in a distributed system requires some central arbiter, but it is highly undesirable to consult centralized servers at run-time. In Chapter 3 we present a certification authority (CA) based solution to both these problems. We show that a P2P system using this solution can achieve security properties comparable with SIP's server-based security, using no centralized operations after a one-time enrollment.
- In Chapter 3 we also present an improved resource replication scheme that can be used with certificate-based schemes, but can also be used in their absence to improve protection against a specific family of attacks against P2P systems, as well

as to provide limited shared-secret security for ad-hoc systems.

1.5 Outline

In Chapter 2, we provide background information, starting with a set of use cases motivating P2PSIP. We then present an introduction to the problems caused by NATs, to P2P systems, and to VoIP and IM systems. We close the chapter with a presentation of related work.

In Chapter 3, we explain the special constraints in developing a P2P system for distributed communications, and present some of the more challenging problems in creating our solution, including secure Peer-ID assignment and distributed identity. We present a solution using an enrollment-time certificate server, and illustrate how this approach addresses these concerns. We then discuss how to provide secure redundancy and close by offering guidance on how to achieve the best levels of security when a certificate server is unavailable. Chapter 4 presents the design of two complete P2PSIP protocols we developed using these solutions: the P2P-over-SIP SOSIMPLE/dSIP solution and the SIP-using-P2P RELOAD solution.

In Chapter 5 we discuss the selection of routing algorithms for P2PSIP, showing that some of the commonly proposed routing techniques in the DHT algorithm literature either do not function or are highly sub-optimal when security and NAT traversal are considered. We present an analytical model for comparing the costs of existing response routing in recursive routing systems in environments containing NATs and secured links, and show that each approach has drawbacks. Additionally, we present Adaptive Response, a novel technique for routing responses in P2P systems, and show that it provides a better balance of performance than existing techniques both in terms of simultaneously

optimizing message count and response time, as well as in terms of performing in diverse environments.

We close with Chapter 6, where we summarize our work and discuss future directions for this research. Appendices A and B present additional details of the SOSIMPLE/dSIP and RELOAD protocols, including the syntactic productions, packet diagrams, and operational instructions for message processing.

Chapter 2

Background

In this chapter, we provide the background information needed to understand P2PSIP properly. First, we provide a set of use cases, motivating the need for a distributed communications solution. Network address translators, or NATs are discussed, followed by a description of various peer-to-peer algorithms, and in particular the Chord algorithm used in our solution. We present a basic primer on Voice over IP and the SIP protocol, and discuss other work related to our P2PSIP research. Finally, we provide a glossary of terms used throughout this document.

2.1 Use cases

To help understand why there is a need for a serverless, standards-based interpersonal communications system, we begin by describing some scenarios in which traditional, client-server based communications systems, or distributed but closed systems, have failed or proven less than optimal. These approaches may have failed for technical, financial, security or social reasons. While we cannot present all such scenarios, the following provide a cross section of the problem.

Use cases are grouped according to either the characteristics of the environment in which the end users or devices participating in the P2P overlay are communicating, or by some motivating characteristics. Note there is some overlap and duplication between scenarios.

2.1.1 Groups of Users or Devices Best Served without Servers

Groups of users or devices may need to communicate in some unusual way, or in some location where a traditional solution is poorly suited. Several of the scenarios proposed in this section are among the most compelling motivations for serverless communications systems.

- **Small or Resource Constrained Organizations:** There are a number of reasons why small organizations may find existing solutions undesirable. These organizations may want private, internal communications, but may not be able to afford the equipment or expertise needed to install and maintain a centralized system.

As a result, these organizations may use a service such as AOL's AIM, Microsoft's Messenger, or the Vonage VoIP service [98], which use servers located at the provider. Because all communication, even transactions between two internal users, will pass through this external party, many organizations have banned the use of these services. While large organizations can install their own internal systems, the personnel and equipment requirements are often too great for smaller organizations. Small organizations which lack these resources may still have a need for internal communication.

Other organizations may simply want to operate their own system. Many small enterprises have a need for integrated communications systems. Examples of such

systems are IP-PBX (Internet Protocol Private Branch Exchange) systems. IP-PBXs support internal communications by handling phone calls, voicemail, and related functions. More elaborate systems are often referred to as “UC” or Unified Communications platforms, combining traditional telephone capabilities with advanced features such as instant messaging, integration with email, and *presence*. Presence is a mechanism that enables users to monitor properties of the system, for example another other user’s connection status. This can be used to enable notifications whenever a someone on the user’s friend or buddy list arrives or leaves.

Certain extremely security conscious small organizations may go a step further and have more demanding needs. These users require communications systems that allow users to communicate directly with one another regardless of their location, with strong encryption, and without any connectivity to or use of central equipment. This differ slightly from the above scenario in that these users are not looking for general purpose communications, but special purpose, point-to-point secure communications. Organizations with security needs in this category include military, national security, or intelligence organizations. These users may overlap with the anonymized communications case described later.

- **Ad-hoc and Ephemeral Groups:** Groups that collaborate may be in ephemeral settings, such as in a meeting or classroom, or in a field research, battlefield, or conference setting. These users should be able to communicate and collaborate using IM, voice, video, and collaboration tools without having to configure a server. Such systems may be used in scenarios where connectivity to the Internet or centralized servers is difficult or impossible, but may also simply be among groups that don’t share accounts on a centralized server. These systems should require minimal, ideally zero, configuration. Because users may join and leave such groups frequently, these systems must allow an arbitrary number of users to connect and

to come and go at their leisure. The underlying connectivity in this scenario may be an ad-hoc network, but such a network does not (by itself) provide a framework for locating and communicating with other users.

- **Multimedia Features for Consumer Electronics Devices:** As more and more multimedia consumer electronics devices, including cameras, camcorders and televisions, become network aware, sharing of multimedia content such as photos and video clips between family members and friends will be desirable. Presence is a useful and important feature for instant messaging and VoIP applications today, but in the future will be incorporated directly into these consumer devices, allowing users to comment on content and interact with one another. As an example, a digital camera may share photos on a digital picture frame with relatives thousands of kilometers away, who will be able to comment on and discuss the photos.

While our work primarily focuses on the use of SIP as a VoIP and IM protocol, SIP is a generic protocol for establishing multimedia sessions between devices. As a result, in addition to enabling users to communicate, it is well suited for connecting and enabling sharing of media between consumer devices. A P2P approach allows instant sharing of multimedia content while reducing or eliminating the need for the equipment vendor to provide infrastructure to support this capability.

While some proprietary solutions for this have been explored, most notably the Bluetooth standard [14], these closed standards generally cannot be used in open source projects or in an academic setting. Additionally, these solutions generally specify transport as well as application level functionality. We feel solutions in this space should be truly open, allowing devices to connect and communicate regardless of the underlying connectivity model used.

- **Emergency First Responder Networks:** Following a large-scale disaster such as a tsunami, earthquake, hurricane, or terrorist attack, access to traditional com-

munications devices of any kind — Internet, cellular, or traditional PSTN — may be compromised. The events following hurricane Katrina have shown that current first responder radio systems cannot be relied upon to interoperate effectively. A network of devices that can grow organically as responders arrive, requiring only wireless access, is required. As more personnel arrive, they should be able to join the network, locate other personnel, and communicate without any configuration of centralized devices.

Example: Following a disaster, the local fire department arrives. Each fire fighter has a wireless handset, and one or more trucks have wireless base stations. When a nearby locality sends additional rescuers, their wireless handsets should be able to instantly join the communications network and communicate. Note that this scenario overlaps somewhat with the limited or no Internet connectivity cases that follow.

2.1.2 Network or Equipment Failure Driven Scenarios

A number of scenarios revolve around failures, either of devices or the networks connecting these devices. In these scenarios, central servers often prove highly problematic, and P2P solutions offer a number of significant advantages.

- **Limited or No Internet Connectivity:** There are many reasons a user may not have external connectivity. If the user is located in a very remote location (Antarctica, a remote developing country, or even outer space), Internet connectivity could be intermittent, delayed, or non-existent. Additionally, there are many reasons connectivity may be interrupted, including service provider failure, infrastructure damage from a civil emergency, battlefield conditions, or simply being out of communications range. Security constraints, such as being located in a classified environ-

ment can also preclude outside connectivity, essentially creating a closed environment.

In some cases, connectivity to the global Internet may be available, but be very expensive, of limited capacity, or unstable, for example when using satellite connections. In such cases, it is preferable to localize communications as much as possible, reducing dependency on any infrastructure in the global Internet. In these scenarios devices should be able to communicate as long they can establish connectivity between themselves.

When there is no physical network available for stable deployment of a communications server or when an instant deployment of real-time communication systems is required, the P2P approach may be the only feasible solution. In such an environment, any type of manual configuration may be difficult to achieve, as technical support may not be available.

- **Extending the Reach of Mobile Devices:** A network of mobile devices can relay traffic between themselves to reach a base station, even if the base station is out of reach of that device. A P2PSIP approach would be particularly effective here, as many mobile telephone devices today already use the SIP protocol.

Example: A user has a handset for communication that cannot reach a base station. Some other user is within range of both that user and a base station. This intermediate user can serve as a relay for the caller who is out of range. A system might make this feature optional for standard communication and mandatory for emergency calls.

- **P2P for Redundancy in Centralized Communications Servers:** Service providers, large enterprises, or universities may wish to connect a farm of central communications servers together in a transparent way, passing resources (user registrations or other call information) between servers with as little configuration or traffic as

possible. Ideally, the redundancy and exchange of information should require a minimum of configuration between the devices. A P2P architecture between the servers allows server farms to be organized and operated in this way. With this approach, it is easy to add more servers with minimal service disruptions, and the overall robustness of the system is increased.

- **Fail-over for Centralized Systems** A traditional centralized communications server presents a single point of failure. Devices which fall back on a P2P paradigm when the server fails enable the service to function normally (or minimally in a degraded, but useful state) in the event of a server failure. If device that participates as a peer provides alternative connectivity to the traditional telephone network (for example via a mobile provider), the system can provide emergency functionality in the event connectivity to the ISP is lost.

Example: A small company has a central IP-PBX. When that device experiences a failure, the handsets transparently switch to P2P and continue operation until a replacement IP-PBX arrives.

2.1.3 Social and Political Motivations

Another set of motivating scenarios involve human factors, either social or political in nature.

- **Deployments in the Developing World:**

Certain locations in the developing world have limited, intermittent, or non-existent connectivity to the Internet. These locations typically also lack experienced people with the specialized skills needed to administer or maintain centralized communications servers. Even DNS servers may not exist or be reachable. A communications

system that is able to function reliably for internal communications, even in the presence of degraded or absent connectivity, is clearly needed. Such a system must also scale easily with little or no configuration and ideally should interface easily to existing communications systems when connectivity is available.

Example: A village in the developing world has connectivity that is limited by weather (microwave connection) or is solar powered. It would be desirable for intra-village communication to continue to function in the absence of Internet connectivity.

- **Censorship or Impeded Access:** Some users may not be allowed to access centralized, persistent servers because of government censorship or because a service provider prevents access to competing services. Such scenarios exist today. The current Iranian government severely restricts access to communications services, and similar (if less severe) restrictions have been imposed by the Chinese government in recent years. US ISPs and Universities have at times limited access to VoIP/IM services to ensure users are restricted to their service. Service providers of cellular devices providing data transmission may also block access to VoIP/IM systems to ensure more costly SMS (Short Message Service, used to send small text messages between mobile handsets) and per-minute services are used instead, a practice known as creating a “walled garden”.

As one mechanism to disable access is to block central servers, users should be able to communicate without the need to connect to any centralized servers. A fully decentralized system cannot be completely disconnected without removing connectivity at the basic Internet level. If connectivity to certain addresses is blocked, users can connect with different peers. If ports are blocked, the system can be run on different ports or use multiple ports. Examples:

- A user wishes to use an IP telephony service to communicate PC to PC with a friend, but the ports commonly used by these services, or the servers used

for authentication, are blocked by the ISP.

- A user with an Internet-enabled PDA devices wishes to connect with colleagues, but traditional services are blocked to ensure that SMS or voice minutes are used (at additional cost) instead.
 - A user in a country with an oppressive government is unable to connect to well-known hosts or ports to enable communications during a government crack-down.
- **Anonymous Communications:** Users occasionally have need to communicate in a completely anonymous fashion, whether due to political persecution, need for secrecy for commercial reasons, or threats of violence. In such a case, the need for a self-organizing, serverless system is imperative. Users of such a system could communicate with reduced risk of the system being monitored or their identities discovered. As with the impeded access scenario, the only way to disable such networks would be to completely disable Internet connectivity. If encryption and onion routing techniques [41] are combined with a decentralized system, intercept can be extremely difficult. Note that while enabling anonymous communications in “good” scenarios such as political persecution or risk of safety, lawful intercept for legitimate reasons such as preventing terror attacks is also very difficult in such architectures.

2.1.4 Large-scale or Global Communications Systems

A final motivating scenario is to create large-scale networks of devices, allowing communication between many disparate individuals from around the world without centralized servers. These cases assume many users across a very large domain, either without or with only a loosely coupled administrator. Communication paths between two devices

may span multiple administrative domains and should be assumed to be insecure. Most well-known P2P file sharing networks have operated in this type of environment.

While many users enjoy the functionality offered by services such as the Skype VoIP service [92], the reliance on a central authority for availability and access is inherently unscalable and failure prone. Recent research [7] has shown that Skype uses a central login server, responsible for management of registered usernames and contacted when a user connects to the system. A truly decentralized system is more reliable, easier to join, and ultimately less expensive for the users. Many of these systems also lock users into using non-standard clients. A system that replicates this functionality using standard-based clients while minimizing the dependence on a central authority is highly desirable. As a cost-saving measure, operators could also build and deploy solutions based on this technology, while retaining some level of centralized control.

A fully P2P VoIP network constructed in this way resembles the global Internet itself in that it has distributed management and growth, enables anyone to reach anyone else in the network, and allows any device supporting the standard protocols to be used. Anyone can join and leave the network freely and anyone can implement the software to participate. In such a system, the protocols used must be based on open standards. This approach does not have the single point of failure or scaling problems that a centralized system (or even a hybrid system with some centralized components) presents.

2.2 NATs and NAT Traversal

In many situations, end users have an inadequate number of IP addresses. This can be due to cost concerns or simply due to the scarcity of IPv4 addresses. Home users often obtain one IP address from their ISP, but it is very common today to have multiple computers or other devices requiring an IP address in the home. Similarly, corporations

may have several addresses, but not enough for their users. In many nations, particularly those outside of the US, there are far fewer IP addresses than there are users wishing to have one. This problem is fundamentally related to the inherent scarcity of IPv4 addresses. While a move to IPv6, with vastly more addresses, is intended to address this problem, this process is very slow, and at present the majority of Internet users do not have access to IPv6 connections and many resources on the Internet are not available using IPv6.

Network Address Translators, or NATs, are a widely deployed solution to this problem. While many types of NATs exist, with slightly different implications to protocol development, they all work in essentially the same way. The majority of home router/firewalls in use today are implemented as port address translating NATs, and we will use this type of NAT as an example for our discussion.

A NAT works by allowing a number of internal IP addresses to share a single external address. Devices on the inside are given private addresses out of a pool reserved for this use. Note that these private addresses are unique only behind a particular NAT—there are not enough reserved private addresses for them to be globally unique. As a result, while no two machines behind the same NAT will share an address, two users behind different NATs may be assigned the same private address.

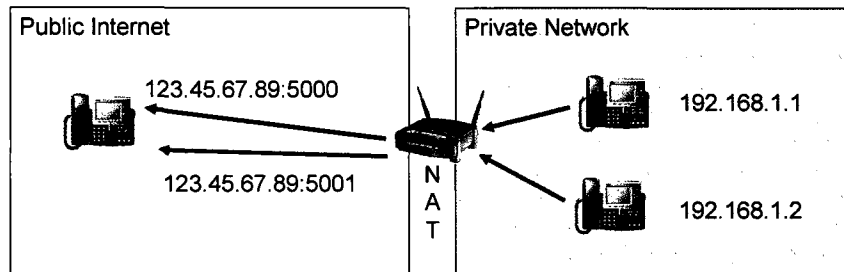


Figure 2.1: An Example of a Network Deployment with a NAT

As messages are sent from hosts behind the NAT to a public host, a particular mapping for this host is maintained by the NAT. We show an example of this in Figure 2.1. In the case of a port address translating NAT, packets from an internal host with a particular assigned private address of 192.168.1.1 will show as being from the public address of the NAT (in this case 123.45.67.89) and a particular port, here 5000. Later, another internal host with private address 192.168.1.2 sends packets, which may appear to be from 123.45.67.89:5001. The exact mechanism of how the port numbers are allocated is dependent on the type of NAT. When responses arrive for internal hosts, the NAT uses the internal mapping to determine which internal host the packets are directed to.

This presents a number of problems in protocol design. If a protocol includes embedded IP addresses in messages and these messages are intended for consumption by entities outside of the NAT, the host cannot place the actual assigned, private addresses into the messages, as these would be unreachable by external hosts and not globally unique. Instead, since there is no universally deployed mechanism to query the mapping from the NAT, a host must obtain the address by asking an external entity what the apparent address is. The SIP protocol is one such protocol that uses internal IP addresses

in messages.

Incoming packets are even more problematic. While Internet applications commonly used an assigned, well-known port to reach a particular application with incoming requests, if there are multiple hosts behind a NAT determining which host to send the message to is impossible absent explicit configuration. Without explicit configuration, this means that new incoming connections to a NAT are generally rejected. An internal host may need to send a message out to create and identify its mapping before a message can be received. In addition, for certain types of NATs, messages arriving on a particular port may even be directed to different internal hosts, depending on the source of the message. For these NATs, not only does a message have to be sent out before a connection can be established, but the reply must come from the same external host the request was sent to. For two devices each behind different NATs, this creates a sort of chicken and egg problem. In some cases, connections can only be established using a public relay intermediary. Both NATed devices open an outbound connection to the relay, which forwards packets between the devices on their behalf. Two SIP phones behind NATs often exhibit this problem.

The IETF has created a suite of protocols to deal with these problems: STUN, TURN and ICE. [60, 80–82] These tools provide mechanisms for NAT traversal, including defining how public servers can return the apparent public address of a query to a device behind a NAT, and specifying protocols for message relay. Additionally, they provide advice on determining which of these tools need to be used and under what circumstances to establish a connection through the NAT.

2.3 Peer-to-Peer Systems

2.3.1 Defining P2P and Comparing with Client-Server

There are a wide variety of definitions for Peer-to-Peer (P2P) systems, and systems can also be “P2P” to a greater or lesser degree. At the most basic level, a P2P system is one where multiple software applications interact directly with one another as peers or nodes to accomplish some task. The group of peers as a whole forming the logical cluster is often referred to as an *overlay* or *overlay network*. The overlay is so named because the peers form a sub-network at a higher logical level than the lower level physical/data link/network layer connections. From an architectural standpoint, a P2P architecture is nearly the opposite of a client/server architecture, in that resources are being moved from a central device to the edge. With that said, P2P application behavior can be indistinguishable from more traditional applications from the perspective of the end user. It is probably most accurate to view P2P as a philosophy of reducing or eliminating the need for central servers, rather than as a particular clearly defined architectural approach [37].

In a P2P network, every peer has equal importance in the network. Rather than a large number of clients contacting a few servers, peers interact directly with each other. In this way, the services or resources that would be provided by a centralized entity are instead available in a distributed fashion from the peers composing the system. While in some cases complete replication of all data may be used, in the vast majority of cases a P2P architecture doesn’t imply that every peer must provide every service or maintain all the available data. Collectively, all the peers in the overlay must be able to provide all services or provide a particular resource, but any one particular peer may only provide or store some small subset. For example, a collection of peers replicating a database might each store a small number of entries from the database. If a very large group of peers splits up the task, the odds of one particular entry being on a given peer are quite

low, but at least one peer in the overlay will store a given entry, guaranteeing that as a group the peers provide the full database service. The services or resources provided are often said to be provided by the overlay, because collectively the members of the overlay provide them. One can envision such a structure as one where a “pie” of services is distributed, slice by slice, among the peers, whereas in a client-server model, the entire pie is stored by the server. (Figure 2.2).

The P2P model can be contrasted with the more traditional client-server model, where one centralized piece of software (the server) processes requests from numerous clients. Choosing P2P or client-server is an architectural decision about where the processing of information takes place. For deployment scenarios such as disconnected networks of devices, a P2P solution may be the only option available. In other cases, the end user may be unaware of, and perhaps not care, where processing takes place. The choice may then be dictated by economic or configuration considerations.

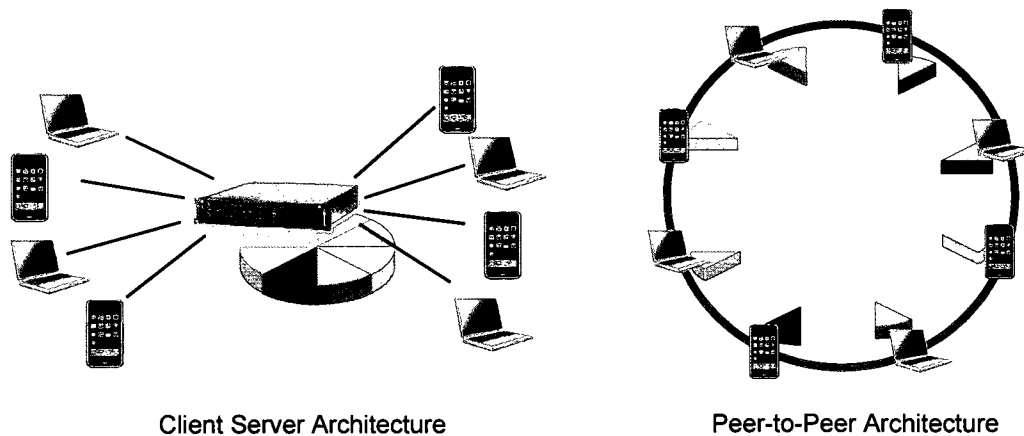


Figure 2.2: A Simplistic Conceptual Comparison of P2P and Client-Server Architectures

In contrast to centrally managed servers, in many P2P systems peers are assumed to be ephemeral in nature. Because the peer software may be running on unmanaged

end-user machines, they may only be available while the software is running and may disconnect at any time for a variety of reasons. The constant change in the makeup of the peers in the overlay is referred to as churn, and is an important consideration in developing P2P applications, particularly from the standpoint of replication of resources and for ensuring information can be properly routed by the overlay.

As P2P technology has evolved, the mechanisms to distribute and locate data have fallen into two broad categories, Unstructured P2P and Structured P2P. These two approaches will be discussed in detail later in this section.

In some P2P architectures, a subset of the peers provide more services than the others. These peers, often called “super-peers” or “super nodes”, may even be the only peers providing services. In such architectures, the super-peers collectively replace the servers, with the remaining “peers” behaving essentially as clients, communicating with super-peers for their services. This approach is often used in the presence of NATs, where peers behind NATs may be unable to receive requests and therefore can’t fully participate as peers.

Yet another common variation of P2P is the hybrid architecture. Hybrid P2P systems use a centralized server to locate a particular peer offering a service, but the service takes place directly between the peers. The best known use of P2P, online file sharing, often worked this way. In the original Napster file sharing system, for example, users connected to a centralized server to enumerate the files they stored and locate files others stored. Once a peer that stored a requested file was located, the transfer of the files took place directly between the participating peers. The central server in such a deployment is often called a tracker. An example of such a system is the popular BitTorrent file sharing system [13].

As we’ll point out later, VoIP and Instant-message systems based on SIP can be thought of as hybrid P2P systems, in that the user agents (UAs) or phones stream media

and commands directly between themselves and use a central server primarily to register and locate resources, in this case, other users to communicate with. This is in contrast to older VoIP systems, where all control and even relaying of media was centralized in the server.

All P2P systems must address bootstrapping, or locating an initial peer in the overlay. This problem can be solved by using some set of fixed peers, requiring that an initial peer be located using an offline mechanism, or using a broadcast/multicast mechanism.

P2P systems have several advantages over centralized architectures. P2P systems distribute resources, greatly reducing the potential for failure due to a single peer and providing a measure of protection against Denial-of-Service (DOS) attacks. P2P systems scale more easily as the number of peers increases, because each new peer brings with it some server-like functionality.

P2P systems have some drawbacks. While resource location can be made relatively efficient for a distributed system, typically it requires more than the single hop required to connect to a centralized server. Malicious peers can provide incorrect information or deny access to resource in the system by refusing to properly route messages or by falsely claiming resources do not exist. Additionally, users can sometimes create many peers in the system, possibly using this as a mechanism for hijacking the system. This family of attacks is referred to as Sybil attacks [34].

2.3.2 Applications of P2P Architectures

P2P systems can be used for many purposes, the most famous (or infamous) being file sharing. In file sharing, users share copies of media files such as movies or music. The distributed nature of P2P is used in these systems to help avoid detection. P2P has

also been used for gaming, distributed file storage, and as this paper describes, P2P communications.

The belief that P2P is a fundamentally new concept is a misconception. Many common network protocols, including Border Gateway Protocol (BGP) and even the Simple Mail Transfer Protocol (SMTP) [55, 72] are arguably P2P, but between instances of servers or routers, rather than end hosts. Increasingly we are seeing peer groups made up of end-user applications, in contrast to groups of managed servers. When someone refers to something as “P2P” today, they generally mean “P2P between end-user applications”.

Other P2P examples include Apple’s open-source enhanced version of TCP/IP, the Bonjour (formerly Rendezvous) protocol, which requires no Dynamic Host Configuration Protocol, or DHCP, server, and the IETF’s service location (SRVLoc) protocol for announcing and discovering network services [5, 35, 44]. Both eliminate the need for central servers and have been used constructively for years.

There have been many proposals to use P2P as an underlying protocol to store the information needed for a DNS system, the most popular being [29].

Like most technologies, P2P can be used for positive and negative applications, and the technology should not be ignored because of a few inappropriate uses. That said, negative connotations have a tendency to stick, and the term “P2P” is already at risk of joining disparaged terms like “hacker”, which used to mean a skilled software engineer adept at quickly adapting code. It will be interesting to see if P2P becomes a tainted term as well.

2.3.3 Unstructured P2P

The earliest P2P systems were unstructured (Left-hand side of Figure 2.3). In unstructured P2P, the peers organize in a haphazard way. Each new peer locates and connects to one or a few other peers in the overlay. There is no mechanism for selecting which peers a new peer connects to; any available peer will do. As a result, some peers may be connected to only a few other peers, while others are connected to many. The data stored or services provided by each peer are similarly randomly distributed. For data storage, this means any peer in the overlay could store a given piece of information, and that the information may not be well distributed among the peers. The connection between the peers is generally not related to the underlying physical network in any way.

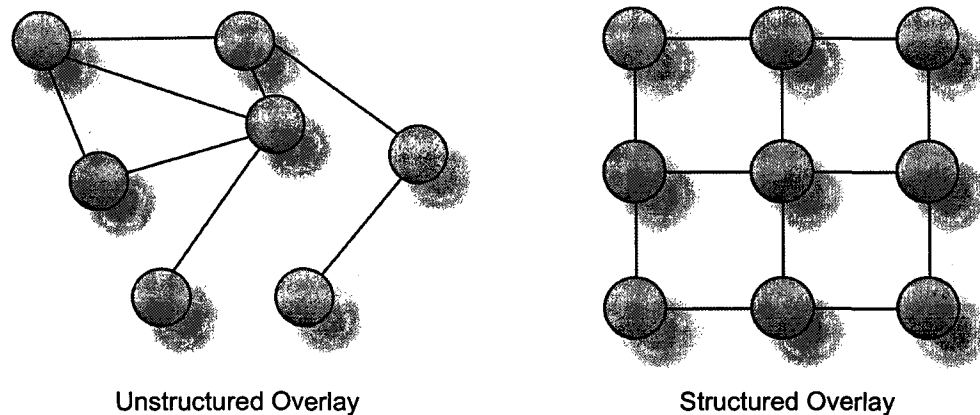


Figure 2.3: A Simplistic Conceptual Comparison of Structured and Unstructured P2P Overlays

Such an arrangement is very easy to form and requires little overhead to maintain, but searching can be very difficult as overlays grow larger. Because any peer could store the information, every peer must be queried to be certain the data isn't present in the overlay. As the structure is random, it is impossible to know how many other peers a peer

will be connected to, or how many “hops” away the furthest peer will be. Peers can be easily split from the overlay, since there is no structure ensuring redundant links between portions of the overlay.

In unstructured P2P, a peer searching for a resource asks their neighbors if they have a particular resource. If one of these peers has the resource, it will respond telling the requester they have the resource. If not, they pass the request on to their neighbors. The search can be limited to a particular depth using a Time to Live (TTL) mechanism, but since peers have no way of knowing if a response has already been transmitted by another peer, queries continue until the TTL is reached, even if some peers have already replied. Exhaustive searches can be very time consuming in large networks, and limiting the search by capping the depth of the searches using a TTL results in non-deterministic searches, since every peer is not consulted and a resource can be stored anywhere in an unstructured network. This approach, used by very early P2P systems such as Gnutella [40, 77] is often called the flood search approach, and proved inefficient, particularly in large networks.

For these reasons, unstructured P2P has fallen out of favor for use in large or Internet scale deployments. For smaller deployments, or deployments where the underlying network itself may have an unstructured arrangement (particularly sensor networks and other ad-hoc or wireless network arrangements) this approach has proven well suited and is still sometimes used.

2.3.4 Structured P2P

In contrast, in a structured P2P architecture (right-hand side of Figure 2.3) the peers are connected to one another in a defined, logical structure, for example a ring, tree, or grid. Peers are assigned a (hopefully) unique identifier when they join the overlay.

This peer identifier, or Peer-ID, could be assigned by some out of band mechanism, selected randomly, or determined by hashing a property of the peer such as the IP address. The Peer-ID determines which other peers the new peer makes connections with. For example, the new peer may connect to peers with identifiers that are “close” in some mathematical sense such as numerical value or number of matching binary digits, or are distributed among the peers in some pre-determined fashion. While such systems do exist, generally the location of peers within the structure is not tied to geographical or topological location.

Since the connections between peers are carefully controlled, a well-designed structured P2P algorithm can ensure that each peer is connected to several others, preventing partitioning when a single peer fails. The total distance between any two peers may also be mathematically limited by the structure of the overlay.

2.3.4.1 Distributed Hash Tables

One widely deployed form of structured P2P is the DHT, or Distributed Hash Table. Among the earliest DHTs proposed were Chord and Pastry [86, 94]. We discuss Chord in more detail in Section 2.3.4.2. In a DHT, not only is the structure of connectivity between the peers controlled in a mathematical way, but the placement of resources onto the peers is as well.

Each resource is assigned an identifier, or Resource-ID, in the same identifier space as the Peer-ID (Figure 2.4). That is, the range of values a Peer-ID or Resource-ID can take on are the same. The Resource-ID is the hash of a property of the resource such as a filename or keyword. A resource’s keyword is hashed to produce a Resource-ID, and the peer with the “closest” Peer-ID stores the resource. The peer with this Peer-ID is called the responsible peer. The hash space is divided up so that all of the hash space is

always the responsibility of some particular peer, although as peers enter and leave the system a particular peer's area of responsibility may change. Both the definition of "close" and provisions for redundancy are dependent on the particular DHT algorithm used.

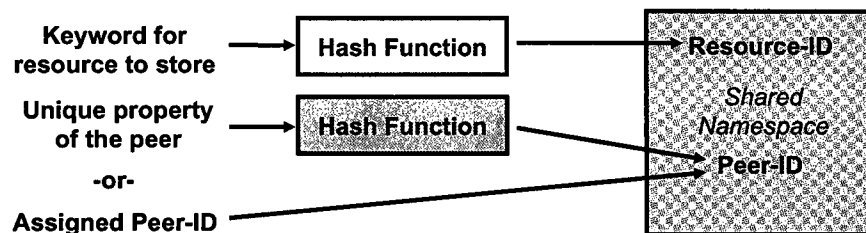


Figure 2.4: An Example of Hashing Identifiers and Peer IDs into the Same Namespace

Messages are exchanged between the peers in the DHT as the peers enter and leave to preserve the structure of the DHT and exchange stored entries. Peers keep information about the location of other peers in the hash space and typically know about many peers nearby in the hash space, and progressively fewer more distant peers. This table of other peers is referred to as a Routing Table or a Finger Table. When a peer wishes to search, it consults the list of peers it is aware of and contacts the peer with the Peer-ID nearest the desired Resource-ID. If that peer does not know how to find the resource, it either returns information about a closer peer it knows about, or forwards the request to a closer peer. In this fashion, the request eventually reaches the peer responsible for the resource, which then replies to the requester.

Various DHT implementations may visualize the hash space as a grid, circle, or line. Chord and Pastry use a ring-like structure for connecting the peers. In such an approach,

a resource with a Resource-ID of n might be stored on the peer with the Peer-ID closest to, but larger than n (see figure 3). In this case, we show 5 peers (circles), with Peer-IDs of 100, 200, 300, 400 and 500. The system stores 2 resources, with Resource-IDs of 345 and 444, shown by squares. The arrows indicate which peer is responsible for storing each resource in the system.

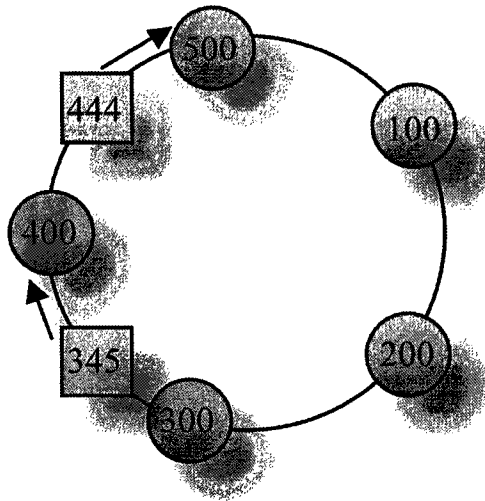


Figure 2.5: Example Ring-like DHT

When some other peer wants to locate a resource later, they hash the distinguished name of that resource, and use the overlay to contact the peer with the nearest Peer-ID, called the responsible peer. That peer provides the resource if present, or can report the resource doesn't exist if it is not stored by the overlay. This mechanism requires fewer messages to be sent to locate data, and provides deterministic search, ensuring that the unique responsible peer is queried for the resource.

Since each peer only connects directly to some subset of the entire overlay, the search will likely take more than one hop. A peer may ask a neighbor that is closer to the desired Peer-ID, which then asks another still closer neighbor, etc. Most of the DHTs used today are structured in such a way that they can guarantee that at most $\log(n)$ peers must be

consulted to locate a particular peer, where n is the total number of peers in the overlay. While this is obviously a higher search cost than the direct query and response of a client-server network, it results in relatively low search times even for overlays with a large number of peers, while eliminating central servers.

Some of the more popular DHTs include:

- **Pastry (Bamboo)** [75]: uses a ring-like structure, very similar to chord. 128 bit IDs are used. Bamboo is a modification on Pastry, optimized to support instances with high levels of churn.
- **Tapestry** [107]: features a mesh-like structure, and uses 160 bit IDs created by the SHA-1 hash algorithm. Tapestry attempts to provide a routing framework that takes locality into account in making routing decisions.
- **Kademlia** [63]: uses 160 bit IDs, and differs from the other algorithms in using an XOR metric in routing. Another unique property of the algorithm is the symmetry of the connections. In other words, the peers that a peer needs to connect to also will need to connect back to the original peer, allowing peers to learn about new, relevant neighbors from searches they process. This is in contrast to many other DHTs, where this property doesn't exist. The XOR metric also allows for easier proof of correctness and efficiency, but also increases the conceptual difficulty in describing (or implementing) Kademlia.

2.3.4.2 Chord

The Chord system [31, 94, 95] is an early and popular DHT algorithm, and is the one we have adapted for our work. Chord uses a ring-type structure for the peers in the overlay. In this structure, a peer with a hash of 0 would be located adjacent to a peer that

hashes to the highest possible hash value, essentially wrapping the namespace. Each peer keeps track of its immediate predecessor peer, as well as one (or more) successor peers in the ring. Each peer keeps information allowing it to directly contact some number of other peers in the overlay. In terms of the overlay network, these are the neighbors of the peer, since they are reachable in one hop.

The list of neighbors is referred to as a finger table in Chord. If the hash has 2^n bits in the output range, each peer will keep at most n finger table entries (pointers to other peers). These fingers are selected in a very specific way. The i^{th} entry in the finger table contains a pointer to a peer at least 2^i units away in the hash space. Essentially, the peer divides the overlay hash circle up into segments, the first being the segment from $[0 - 2^0)$ away from the peer, the second being from $[2^0 - 2^1)$, the third being from $[2^1 - 2^2)$, etc., all the way to the segment from $[2^{158} - 2^{159})$ away from peer. The highest finger table entry thus point $\frac{1}{2}$ of the way across the hash space, the next highest $\frac{1}{4}$, the next $\frac{1}{8}$, and the first entry points to a range only 1 away in the hash space. Each peer then stores an entry in the finger table for the first peer present in the system with a Peer-ID greater than or equal to the start of this interval. In this way, the peer has many entries pointing to nearby peers, and progressively fewer entries pointing to more remote peers. These tables are populated when the peer joins the overlay, and are kept current by periodically updating them.

Chord uses the SHA-1 hash, which produces 160 bit results. Therefore in chord, there are 160 finger table entries, pointing to peers in the space at least 2^i away from the peer, for $i \mid 0 \leq i \leq 159, i = i \bmod 2^{160}$.

Messages are routed by taking advantage of the key property of these finger tables, that a peer has more detailed, fine grained information about peers near it than further away, but it knows at least a few more distant peers. When locating a resource (or peer) with a particular key, the peer will send the request to the finger table entry with the Peer-

ID closest to (but preceding) the desired key. Since the peer receiving the request has many neighbors with similar Peer-IDs, it will presumably know of a peer with a Peer-ID closer to the key. In Chord, this request is then passed on to the closer peer, and the process repeats in a recursive fashion. When the peer that would finally be responsible for storing the particular key (the successor peer to that key) is reached, it can respond. In the case where the search is for a resource, it can indicate if that resource is present or not. In the case of storing a file, it can respond if the file exists, and provide a mechanism to retrieve it.

Several proposals have been made to extend Chord and improve efficiency. Most notable is Epichord [58], which seeks to improve the performance of Chord by including overlay maintenance requests in searches (reactive routing management), and by issuing parallel searches. By maintaining lists of predecessors as well as successors, and maintaining significantly more fingers, routing efficiency of only a few hops can be achieved in many cases. Note this is an example of the classic time-space tradeoff, as additional storage space is required.

2.3.4.3 Routing in DHTs

Messages can be routed in DHTs in a number of different ways. As we will see later, when deploying a system on the open Internet, the decision becomes very critical. There are two different types of forward routing, that is, ways to forward a message from the sender to the destination: iterative and recursive. Within recursive forward routing, there are three ways to route the response back to the sender : symmetric recursive (or symmetric response), direct response, and strict forwarding (or forward-only). As we will show in Section 5, the problem isn't quite as simple as presented here, but these descriptions will serve as a basic introduction to routing for the following sections.

Iterative Routing

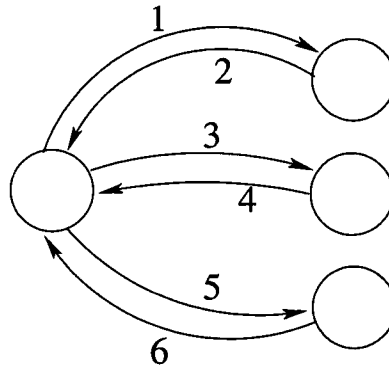


Figure 2.6: An Iterative Routing Flow

In Figure 2.6 we illustrate an iterative routing flow. The peer receiving the message replies to the sender suggesting a nearer peer, rather than forwarding the message. The initiator then sends a new request to the recommended (and nearer) peer, repeating until the target is reached. This is the technique used by many early DHTs, and is still frequently proposed in the DHT algorithm literature.

Recursive Routing

With recursive routing, the initiator issues a message to the peer it is aware of nearest the target. If the recipient peer is not the target, the message is forwarded to the nearest peer the recipient is aware of, and the process repeats until the target is reached.

Recursive algorithms can route the response in three ways. In a symmetric or true recursive algorithm (Figure 2.7) the response is returned simply by reversing the original path. Many recursive algorithms shortcut by sending the response directly back to the initiator, an approach termed direct response or semi-recursive routing (Figure 2.8). A response can also be routed as strict forwarding, with the response being routed back to the initiator in the same manner a new message from the responder to the initiator would be routed — in other words, the message will be routed forward through the links of the

overlay from the destination back to the source as if it was a message from destination to source (Figure 2.9). Here the shaded peers were not in the original forward path of the message.

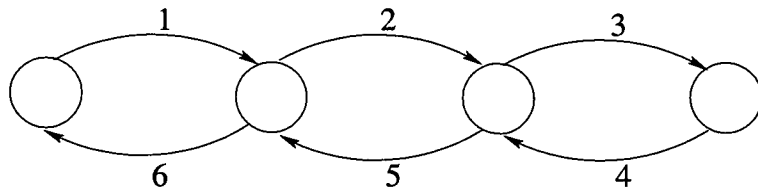


Figure 2.7: A Symmetric Recursive Routing Flow

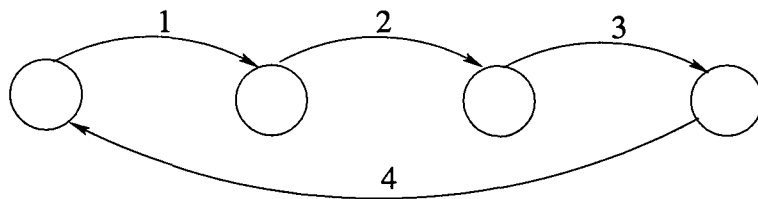


Figure 2.8: A Direct Response Routing Flow

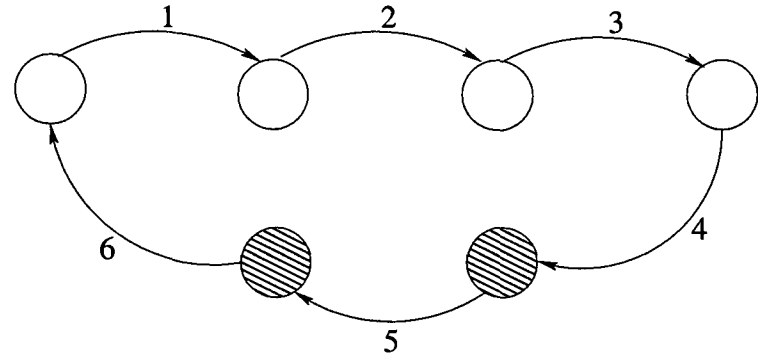


Figure 2.9: A Strict Forwarding Routing Flow

2.4 Network-based Personal Communications Systems

2.4.1 VoIP and IM Systems

Modern Voice Over IP (VoIP) and Instant Message (IM) systems are designed to allow users to communicate, either using voice/video conversations, or by the exchange of short text messages. These systems operate over the IP protocol, and are thus packet-switched, but in many cases have been designed to replicate the perceived end user behavior of earlier phone systems, which were circuit-switched. To implement these capabilities, these systems generally provide three basic functions:

- **Resource location** is responsible for identifying and locating other users so conversations can be established.
- **Session establishment and management** creates, controls, and terminates text or multimedia sessions between users.
- **Presence** is a mechanism that enables users to monitor properties of the system, for example another other user's connection status. This can be used to enable notifications whenever a someone on the user's friend or buddy list arrives or leaves,perhaps by highlighting that friend on a list displayed on the user's GUI. Presence can also be used to provide other notifications, for example waiting voicemail messages, parked calls available to be retrieved, etc.

The majority of today's VoIP and IM systems are centralized. Users connect using a User Agent (UA), which may be a software application such as an IM client program or a software-based phone, often called a soft client. UAs may also be hardware device such as an IP or mobile phone. In a VoIP architecture, many devices are actually implemented as UAs. For example a gateway which terminates VoIP calls and connects them to the

PSTN is often actually a special purpose UA from the point of view of the SIP system. Similarly, a voicemail system which receives and records messages may be implemented as a special purpose UA.

In a traditional client-server architecture, the UAs connect to a central sever. Depending on the particular VoIP protocol used and the architecture of the system, this server may only provide location and call routing capabilities, in which case it is usually called a proxy. The server may also provide additional services and features including voice-mail, interactive voice response (IVR) systems, etc. These more elaborate servers are typically called softswitches. In some more highly centralized systems, virtually all capability resides in the server. Historically, in these contexts the server has been called a gatekeeper.

Several VoIP protocols are in wide use. Older protocols such as the International Telecommunications Union (ITU) H.323 standard [48] place most intelligence in the central gatekeeper, while SIP pushes most of the intelligence to the UAs but relies on central servers for some functions, particularly locating users and helping to establish connections.

A large outlay of effort, capital and time has been invested in building networks and equipment based on these protocols. Most physical or “hard” UAs in use today implement either H.323 or SIP.

Commercial IM protocols used by AOL, Yahoo and MSN require users to connect to a server hosted at the provider's site. A few commercial products, as well as the IETF standards XMPP and SIMPLE allow a corporation or group to maintain their own server, but these are still centralized. XMPP is an XML based protocol that emerged from the Jabber project. SIMPLE is a set of extensions to the SIP protocol intended to support instant messaging.

2.4.2 SIP and SIMPLE

SIP and SIMPLE are text-based protocols derived from HTTP [36], therefore message types and traffic are similar to web traffic. SIP is a general protocol for establishing and controlling multimedia sessions, but is most widely used for VoIP. SIP embeds Session Description Protocol (SDP) [45] information to specify the media parameters. SIP generally runs on port 5060, using either TCP or UDP as a transport, and also is designed to operate using IPv6. Like HTTP, SIP is a request/response protocol. Requests are generated by UAs, and sent either to a server or directly to one or another. The UAs themselves or the server may generate response messages.

There are many types of requests in SIP. A few examples of request messages include:

- REGISTER, used to inform a server of the location of a UA, thus providing a location where a particular user can be reached.
- INVITE, to establish a session between two parties.
- BYE to terminate a session that has been established.

Response messages use a numerical status code, some borrowed directly from the HTTP protocol, others specific to SIP. A few examples of response messages include:

- 100 TRYING and 180 RINGING, which are used to provide provisional information about the progress of an attempt to establish a connection.
- 200 OK, indicating that a session has been established successfully.
- 302 MOVED TEMPORARILY, which is used to redirect a caller to a different location or user.

- 404 NOT FOUND, indicating that the requested user cannot be located to establish a session.

SIP/SIMPLE is structured so that two UAs can theoretically connect directly, but the proxy is generally needed to locate the remote party. In other words, two devices can be configured to directly contact each other and will function perfectly without the proxy if some other mechanism to locate users is provided.

In the SIP specification, the functions of a server are broken into several logical network elements, including a registrar capability to store the location of users, and two main techniques for routing, proxy (recursive) and redirection (iterative). In proxy, messages are passed from device to device on behalf of the user, and in redirection, a device replies with a suggested location to try and locate the user or resource. These capabilities are often incorporated into one physical box, which we refer to here as a server or proxy for convenience, even though it may have capabilities beyond simply proxy.

Note that the SIP notion of a proxy is a more sophisticated network element than a web proxy, maintaining location information about users and UAs, as well as routing signaling traffic between UAs. Proxies also provide authentication and username administration. The registration capability allows a UA to manage information about its location with the central server, while proxy (and redirect) capability allow the server to assist UAs in locating each other.

The initial signaling portion of the call consists of SIP messages flowing through the (one or more) central proxies connecting UAs. Media flows directly between UAs, and optionally some signaling may flow in this way as well. Media typically is encoded in the form of Real Time Protocol (RTP) or the secure variant SRTP [9, 89] packets. This separation is somewhat analogous to file sharing, where a central server (tracker) might

be used to store a list of which files each peer stores, while the files are exchanged directly between peers.

2.4.2.1 Registration

Users are uniquely identified by a SIP URI (Uniform Resource Identifier), such as "sip:bryan@cs.wm.edu". Users make themselves available to receive calls at a particular UA by sending a REGISTER message to the central server, which maintains a mapping between an address of record, or AoR, in the form of a SIP URI, and a contact, usually the user's UA IP address. REGISTER messages includes an expiry time and expire unless refreshed. Typical lengths of expiry are around one hour. Users remove themselves by registering an expiry time of zero, and a registrar can be queried for a particular user's registrations by sending a REGISTER with no expiry time in the message. Registrars can require security credentials to validate that the requester is authorized to modify the registration information.

Multiple registrations are allowed. In such cases, SIP utilizes a mechanism called "forking" to attempt to find the user. Different types of forking can be used. Sequential forking will try multiple registrations, such as a desk and mobile phone one after the other, while parallel forking would try both simultaneously. This mechanism allows a user to have two devices (for example a home and work phone) or for unanswered calls to forward to a receptionist or voicemail server after a pre-determined time or number of rings.

2.4.2.2 Establishing and Terminating Calls

The SIP INVITE message is generated by a UA and sent to a proxy to establish a new connection. Incoming messages for a registered user arrive at the proxy, which forwards

them to the UA.

For unknown or unregistered users, local rules may be consulted to determine how to route the call. The call may be rejected (for example by sending a response of 404 NOT FOUND), the URI may be examined and the request proxied or redirected to a different domain to be processed, or the call may be gated to the public phone system. Access to the public phone system requires gateways and phone lines, and is often a pay service.

In Figure 2.10 we show a basic successful proxy-brokered call flow between two users. While the INVITE is being forwarded to the destination, the proxy generates a 100 TRYING response message and returns it to the calling UA. This message indicates to the calling UA that the message has been received by the proxy, preventing retransmissions, but indicates nothing with respect to the remote party. The INVITE is passed to the remote party by the proxy, which generates a 180 RINGING response message, passed back to the proxy and on to the calling UA, indicating that the message has reached the remote party and is being acted on. These 1xx class messages are referred to as provisional responses, in the sense that they do not indicate a transaction has been completed, but provide information to the parties that processing is taking place.

Once the remote party answers, a 200 OK message is generated, indicating that the call has been accepted. Finally, the original caller generates an ACK message, indicating that the 200 has been received. This is one way in which SIP differs from protocols such as HTTP. Because some period of time can pass while the device is alerting the human user and waiting for a response, it is possible that one or more of the devices or the underlying network connection could fail between the sending of the INVITE and the sending of the 200 OK. This time may be only a few seconds, but on the time scale of the devices, this is considerable. As a result, SIP uses a “triple-handshake” connection, including this final ACK message.

Following the ACK message, a media session is established between the devices,

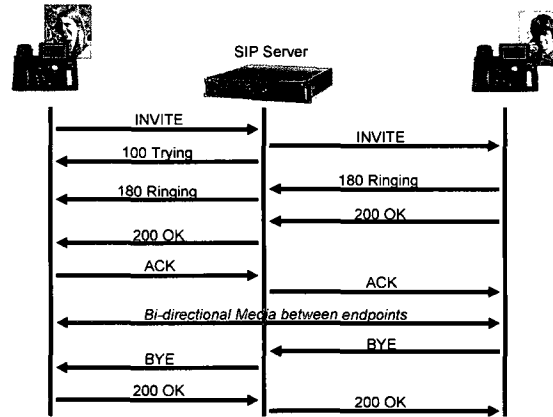


Figure 2.10: Call Flow for a Basic, Successful, Proxy-brokered SIP Call

and media flows. Note that the format of the media is specified in the embedded SDP in the SIP messages, and may include audio, video, or any other type of streaming information the user requests. Multiple channels can be established. SIP also offers an offer-answer paradigm [83], by which the devices can negotiate with each other, discovering the capabilities of the remote party and determining the best media connection possible.

At the end of the session, one or the other party (in this example, the remote party) decides to cancel the connection by sending a BYE message. This message is acknowledged with a 200 OK response, the media connection is torn down, and the session is complete.

2.4.2.3 SIMPLE: SIP for IM and Presence

SIMPLE is a set of extensions for SIP to add both IM and Presence capabilities. SIMPLE sends text IMs using the MESSAGE message. Unlike a conventional SIP session es-

establishment, no session or media stream is established to send the text messages. The text of the messages is carried directly in the SIP MESSAGE message, and so passes through the intermediaries that make up the signaling path. SIMPLE messages are not correlated into a session — each is a separate transaction and any correlation between messages to form a conversation is an artifact of the application.

SIMPLE also adds SUBSCRIBE/NOTIFY capabilities, allowing for the implementation of presence. Users can use a SUBSCRIBE message, sent to a presence server, indicating they would like to be notified when some subscribed variable changes, for example when a remote user comes online or when a voicemail arrives. NOTIFY messages are sent from the server to indicate the new value of the variable. The party being monitored (in other words, the user that comes online or the voicemail server that receives a new message) sends PUBLISH messages to the server to post changes in the variable.

Because of the close relationship between SIMPLE and SIP, a SIMPLE client that wishes to enable voice chat can do so very easily using the established SIP protocol. All of the SIMPLE messages and SIP messages use a common rule set for parsing, message construction and routing, meaning one stack can be used to enable both applications.

2.5 Related Work

While this is a very new area of research, there are a number of instances of related work to discuss. There have been a number of earlier efforts looking at using SIP to enable other P2P applications (rather than our work on running SIP in a P2P fashion), largely for file sharing. Some efforts have focused on developing P2P communications systems, but in a closed, non-standard way. Finally, since our work began, several others have

begun working on similar projects, some commercial and largely non-standard, but a few attempting to produce standards-based communications, similar to our work.

Note that many researchers have used P2P for distributed lookup, much as we have, but for purposes other than communications. We have briefly touched on this work in Section 2.3.

2.5.1 Using SIP for Other P2P Applications

A number of efforts have looked at using SIP to enable other P2P applications. In some ways, this can be looked at as being the opposite of our work. Rather than taking SIP and using P2P to enable a fully distributed version of it, these efforts focus on using SIP to enable P2P.

- **EarthLink SIPshare:** The EarthLink R&D group has created an application called SIPshare [91], using SIP as the underlying transport protocol for a file sharing P2P system. SIPshare uses UDP directly between hosts for the transfer of files, analogous to the way media sessions in SIP are streamed using UDP. Messages for search, as well as requesting the particular content requested are sent using SIP messages with specialized content. At the time, this work was quite revolutionary. Prior to this work, no one had suggested a standard mechanism for encoding the messages between P2P clients.
- **NUTSS:** NUTSS [43] is a project exploring how using SIP and related technologies such as the family of NAT traversal mechanisms developed for SIP — STUN, TURN and ICE — can improve P2P performance. The primary focus of the work is on the NAT traversal efforts. This work provided some guidance for the NAT traversal work in our P2PSIP solution.

2.5.2 Non-standards Based P2P Communications

- **Waste:** WASTE [101] was a short-lived P2P system for file sharing. Waste was briefly released as open-source in 2003 but later rescinded. What made it interesting from the perspective of our research was that group chat was supported, apparently the first instance of a P2P application incorporating personal communications.

For a new peer to join, some other peer needed to grant permission to that peer. All peers shared one encryption key and used it to encrypt all traffic. All messages are sent using a Gnutella-like flood mechanism. Periodic beacon broadcasts and flood polling are used by peers to identify other peers in the overlay. The group chat messages and search requests are also sent using broadcast, and passed through all peers.

WASTE had a number of shortcomings. WASTE was non-standard, requiring a special client, and use a proprietary IM protocol. Traffic within the overlay was encrypted against outside snooping, but all traffic flooded all peers, and all peers shared the same encryption key, requiring complete trust of users within the network. Since all messages were broadcast by flood, IM and broadcast traffic explode as size increases. This architectural decision limited WASTE networks to 50 peers. There was also no mechanisms to verify individual sender identity or ensure peers forwarded messages as requested.

- **Skype:** Skype [92] allows users to make free phone calls between Internet users, and offers public phone connectivity for a fee. The system is proprietary, and is not compatible with non-Skype devices. Skype provides VoIP and IM capabilities. While partially P2P, username control, authentication, and some other services are controlled by Skype's central servers.

While the Skype protocol is closed and details are thin, research indicates that

Skype works in somewhat the same way as SIP systems, with some critical differences. According to [7, 12], there are a number of important properties to the Skype protocol.

Like SIP, Skype uses a central server to authenticate and verify that users are allowed to use the system, but Skype uses a proprietary protocol for this service. Unlike SIP, each Skype peer may be used as a media relay to assist with NAT traversal. This, along with a very effective and streamlined configuration process, is part of what allows Skype to work so smoothly.

If a media relay is required for NAT traversal, Skype easily finds one within the system. This reduces the load on Skype's own servers, which only need to be used for authentication and lookup. The downside, at least from a network administrator's point of view, is the unpredictable traffic load on Skype peers within the network that are functioning as media gateways.

Since any peer meeting a set of requirements identified by Skype (presumably, having available bandwidth and a public address) can easily be turned into a media relay for many other calls, participating in a Skype network leaves one open to being used as a relay, with the associated bandwidth problems, whether an administrator desires it or not.

Similarly, a user's calls may be routed through some random location during a Skype call, although the encryption used on Skype calls makes this a less serious concern.

With these two exceptions – using peers as media relays and the proprietary protocols – Skype is very similar in end user experience to other solutions, such as the Google Talk service, although Skype has taken a non-standard approach, while Google Talk is largely standards-based.

- **vop2p:**

Another early project that seemed somewhat similar to Skype was a non-standard P2P communications project called vop2p. Vop2p was available open-source, but seemed to never reach a fully functional state. The project appears to have been shut down over five years ago, and the site has since been taken down.

2.5.3 P2PSIP Research

Extensive non-commercial efforts have been focused on extending SIP to use a pure P2P-based mechanism to locate other users, including work by the authors, researchers at Columbia University, Ericsson, Alcatel-Lucent, and others.

P2PSIP has drawn healthy participation at the IETF. Much of this work is documented on [30].

Kundan Singh and Henning Schulzrinne at Columbia University have been active in exploring research similar to ours [8, 90]. The current protocol draft for P2PSIP being advanced by the IETF, RELOAD, is based on a merge of our RELOAD protocol with the Columbia P2PP protocol [6] (as well as the ASP protocol [54] from Cisco). The group at Columbia was the first to propose a binary protocol, following on to our earlier proposal of the first P2PSIP solution using SIP messages. They have also proposed a simplified system by using OpenDHT [68, 76]. While this is a simple approach for experimentation, we feel it cannot be used for deployed standardized systems, and has the limitation of preventing experimentation with DHT structures optimized for communications.

In the five years since we have introduced the topic of P2PSIP, there have been many papers published in the area, including [10, 11, 46, 61], and the subject has become a frequent topic in P2P conference calls for papers. Our original paper [22] has been cited in more than 90 papers in the area.

2.6 Terminology and Glossary

For convenience, we define a number of terms here that are used in the subsequent sections:

- **Overlay Network:** An overlay network is a computer network which is built on top of another network. Peers in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet. As an example, dial-up Internet is an overlay upon the telephone network.
- **P2P Network:** A peer-to-peer (or P2P) computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting peers via largely ad-hoc connections. Such networks are useful for many purposes. Sharing content files containing audio, video, data or anything in digital format is very common, and real-time data, such as telephony traffic, is also exchanged using P2P technology.
- **P2PSIP:** The combination of the Session Initiation Protocol (SIP) [RFC3261] with peer-to-peer techniques for resolving the targets of SIP requests, providing SIP message transport, and providing other SIP-related functions with minimal need for servers.
- **User:** A human that interacts with the overlay through peers and clients.
- **Peer:** A node participating in an overlay that provides storage and transport services to other nodes in that overlay. Each Peer has a unique identifier, known as a Peer-ID, within the Overlay. In P2PSIP, each Peer may be coupled to one or more

SIP entities. Within the Overlay, the peer is capable of performing several different operations, including: joining and leaving the overlay, transporting SIP messages within the overlay, storing information on behalf of the overlay, putting information into the overlay, and getting information from the overlay.

- **Client:** A node participating in a overlay that is less capable than a Peer in some way. Clients have the ability to add, modify, inspect, and delete information in the overlay, but generally do not participate in routing or storing information. Clients usually connect to a peer to access the services of the overlay.
- **Peer-ID:** Information that uniquely identifies each Peer within a given overlay. This value is not human-friendly – in a DHT approach, this is a numeric value in the hash space. These Peer-IDs are completely independent of the identifier of any user of a user agent associated with a peer. (Note: This is sometimes also called a “Node-ID” in the P2P literature).
- **Username:** A human-friendly name for a user. This name must be unique within the overlay, and may be unique in a wider scope.
- **Service:** A capability contributed by a peer to an overlay or to the members of an overlay. Services might include routing of requests, storing of routing data, storing of other data, STUN discovery, STUN relay, and many other things.
- **Resource:** Anything about which information can be stored in the overlay. Both Users and Services are examples of Resources.
- **Resource-ID:** A non-human-friendly value that uniquely identifies a resource and which is used as a key for storing and retrieving data about the resource. One way to generate a Resource-ID is by applying a mapping function to some other unique name (e.g., Username) for the resource. The Resource-ID is used by the

distributed database algorithm to determine the peer or peers that are responsible for storing the data for the overlay.

- **Adapter Peer:** An adapter peer is a node in the overlay that acts as an adapter for other non-P2P enabled SIP entities, allowing them to access the resources of the overlay. The adapter peer participates actively in the overlay network, while the non-P2P enabled SIP entities it provides
- **Chord:** A particular algorithm/approach to implementing a DHT. Uses a circular arrangement for the namespace.
- **Finger Table:** The list of peers that a peer uses to send messages to. The finger table contains many entries about peers with similar IDs, and fewer entries about more remote IDs. The same (or similar) concept is sometimes called a neighbor table or connection table in other contexts.
- **Neighbors:** A collection of peers that a particular peer can reach in one hop. In general, note that a peer's set of neighbors is equivalent to the entries in that peer's finger table. In our DHT structure (Chord), neighbor relations are NOT symmetric, but may be in other DHTs such as Kademlia.
- **Successor Peer and Predecessor Peer:** A term borrowed from Chord. These terms refer to the peer directly after (before) a particular peer in the address space. This does not mean the successor/predecessor peer's ID is one greater/less than the peer, it simply means that there are no other peers in the namespace between the peer and the successor/predecessor. Note that the first peer in a finger table is typically also the first successor peer.
- **Responsible Peer:** The Peer that is responsible for storing the Resource Record for a Resource. In the literature, the term "Root Peer" is sometimes also used for this concept.

- **Joining Peer:** A peer that is attempting to become a Peer in a particular Overlay.
- **Bootstrap Peer:** A Peer in the Overlay that is the first point of contact for a Joining Peer. It selects the peer that will serve as the Admitting Peer and helps the joining peer contact the admitting peer.
- **Admitting Peer:** A Peer in the Overlay which helps the Joining Peer join the Overlay. The choice of the admitting peer may depend on the joining peer (e.g., depend on the joining peer's Peer-ID). For example, the admitting peer might be chosen as the peer which is "closest" in the logical structure of the overlay to the future position of the joining peer. The selection of the admitting peer is typically done by the bootstrap peer. It is allowable for the bootstrap peer to select itself as the admitting peer.
- **Peer Protocol:** In P2PSIP, the protocol spoken between overlay peers to share information and organize the P2PSIP Overlay Network.
- **Client Protocol:** In P2PSIP, the protocol spoken between Clients and Peers (if needed). It is used to store and retrieve information from the P2P Overlay.
- **Peer Admission:** The act of admitting a peer (the "Joining Peer") into an Overlay as a Peer. After the admission process is over, the joining peer is a fully-functional peer of the overlay. During the admission process, the joining peer may need to present credentials to prove that it has sufficient authority to join the overlay.

Chapter 3

The Design of the Peer-to-Peer SIP (P2PSIP) Protocol

In developing the first P2PSIP system and our protocols for P2PSIP, we encountered a number of challenges. In this chapter we discuss a number of the more interesting problems faced in the design of the protocol, and present our solutions to these challenges. In particular, we discuss the issues raised by Peer-ID generation, resource replication, and distributed identity, and show how an enrollment-time certificate server can address these and other problems. We also provide guidance for providing some level of security in systems where a certificate server cannot be used.

In Chapter 4 we discuss the specific details of our two protocol design implementations, and in Chapter 5 we discuss our detailed research into optimizing routing techniques for DHTs used in a P2PSIP solution.

3.1 Choosing a Mechanism to Distribute Information

The first step in the design of a fully distributed system is selecting a mechanism to distribute the (previously centralized) information among the participants. A number of mechanisms exist for this, including various types of P2P (structured, unstructured), as well as broadcast replication methods and the use of “cloud” services such as DNS.

In Section 1.3.1 we discussed why we chose to use a P2P mechanism, rather than broadcast or dynamic DNS mechanisms.

3.1.1 Requirements for Communications

In Section 1.2, we outlined the requirements from a user and functionality perspective of developing a distributed personal communications system. Here we once again list some requirements, but these differ in that they are specific requirements not for the capabilities of the system, but rather requirement for the underlying programmatic structures used to distribute the information.

In file sharing applications, many users may offer copies or versions of a file, and the identity of the provider is often deliberately obfuscated or at best unimportant. Except for special purpose systems, communications systems typically require that a unique individual be contacted. An obvious example is that when a user calls their bank, they expect to reach their bank, not someone offering a similar service, or worse still, a random user purporting to be the bank. Similarly, with multiple copies in a file sharing application, it is often acceptable to query some fraction of the peers, rather than all of them, and assume it is statistically likely the file will be found. Obviously, if each resource (user to communicate with, in our context) is unique, such an approach is not viable to locate a user.

The lack of anonymity and the availability of the desired resource (person) at only a single location, coupled with a need for high reliability for communications, leads to our first requirements:

- The distributed algorithm must return the location of a resource, if present. False negatives are unacceptable.
- The user must be able to verify the integrity of the resource and thus the identity of the remote party.

The requirement to avoid false negatives eliminates many unstructured techniques, strongly suggesting that if P2P is used, a DHT or other structured P2P algorithm be used. Furthermore, the algorithm must contain sufficient protection against attackers manipulating the overlay routing to gain control of a particular portion of identifier space to prevent DoS attacks against targeted users. While also a problem, false positives due to stale registrations are more easily handled at the application (SIP) level, and do not require DHT support.

Meeting the integrity and identity requirement requires a security mechanism that allows peers to sign resources, verify the signatures on resources, and the identity of remote parties.

- The system must provide connectivity between NATed peers and include them in the DHT when possible.

Deploying an Internet-scale P2P communications systems dictates that it must support the full range of deployed networking technologies. This includes consumer broadband network users and businesses and universities located in many countries. Unlike hosts at Western universities, these endpoints are frequently behind Network Address Translators (NATs), introducing the problem of non-symmetric (and sometimes

non-transitive) connectivity. Freedman et al. [39] explored these issues in a PlanetLab based system, where the cause of these connection problems is generally ephemeral problems or difficulties in Internet1 systems communicating with Internet2 systems. In our Internet-scale system, non-symmetric connectivity caused by NATs is the rule, rather than an exceptional failure condition. Because we support deployment scenarios where the majority of devices are behind NATs, we must utilize them as peers in the DHT rather than relying on a sufficient number of super-peers on the public Internet.

- The DHT algorithm must not amplify DoS attacks.
- The DHT algorithm must prevent an attacker from gaining control of a particular portion of the identifier space by manipulating Peer-ID assignment.

As with any other publicly deployed P2P network, the overlay must guard against malicious peers and be resilient against DoS attacks. In particular, a P2P communications network is subject to both general DoS attacks against the entire overlay and DoS attacks targeting specific users.

In the next section we explore the need for preventing the user from manipulating the Peer-IDs in detail.

3.2 Assignment of Peer-IDs

3.2.1 Why Users Cannot Control Peer-ID Assignment

Many attacks on P2P systems involve the user being able to place themselves in a particular location in the overlay. In particular, many require identifying a particular portion of the ID space and acquiring control of that space by becoming the responsible peer. In the

case of a P2PSIP system, one potential attack would be to control the peer storing a particular user's location registration. In such a case, callers wishing to reach the attacked user would request the registration information from the compromised peer, which could redirect the caller to a different party or simply deny the attacked user is registered with the system, constituting a man in the middle or DoS attack, respectively. We illustrate a DoS version of such an attack on the left side of Figure 3.1. Here we show an arbitrary hypothetical topology, where an attacker has compromised peers 18 and 20, and in so doing, effectively severed peer 19 from the overlay.

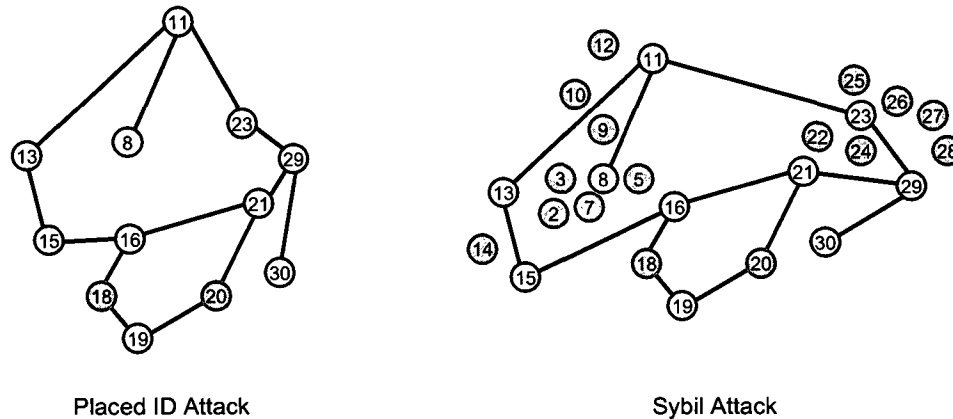


Figure 3.1: Examples of Placed ID and Sybil Attacks

As a result, it is very important for P2P systems that the Peer-ID be assigned in some way that is beyond the control of the user. Any mechanism that generates Peer-IDs based on hashing of a value that can be controlled in any way by the end user is inherently insecure. Additionally, one needs to limit the number of Peer-IDs a user can create or use. On the right side of Figure 3.1 we show a similar attack where the attacker has prevented connectivity to peer 19 simply by randomly generating many Peer-IDs until they are able to create peers with the IDs required to compromise peer 19. This attack is often referred to as a Sybil attack [34] after a 1973 book "Sybil" documenting a woman

with 16 personalities in that the attacker takes on many identities,

3.2.2 Problems with IP-based Peer-IDs

The majority of early DHT designs required that in order to produce unique Peer-IDs, the IP address (neglecting port) of the peer was hashed to produce the Peer-ID. This approach makes sense on the surface. The IP address of a computer seems to be a unique property, and as a result, hashing the IP address produces a unique Peer-ID. In our initial work, we looked to use an existing algorithm such as Chord unmodified, and thus explored hashing IP addresses as a mechanism for creating Peer-IDs. Unfortunately, we very quickly determined that such an approach was not feasible for an Internet-scale deployment for the following reasons:

- **Multi-user machines:** A multi-user machine may have more than one user with running instances of the P2P application, each of which is technically a separate peer. Multiple peers clearly cannot have the same Peer-ID, despite running on the same host.
- **Two peers running on different private subnets,** for example two home users behind different residential NATs, may believe themselves to have the same (private) IP address, typically 192.168.1.2. While this problem can be partially addressed by using STUN to determine the outside (public) address of the NAT, there is still the issue of multiple users located behind the same NAT, who would share an external IP address (although a different port). In both cases, duplicate Peer-IDs will be produced.
- **Access to multiple machines and IPv6 issues:** For an attacker who might have access to many IP addresses (for example the operator of a botnet, or even a

student at a large university with many machines he can log into), an IP hash-based approach may not prevent a Sybil attack. In IPv6, there are many, many more IP addresses. As a result, in some cases an ISP will issue a range of addresses for use by a particular user. In such a case, a single user may effectively have access to many IP addresses, again allowing them to mount a Sybil attack.

Combining the public address and a port number can ameliorate the problem, but does not protect against malicious peers. Appending the port number to the address *before* hashing produces a different Peer-ID for each of several peers behind a NAT, but enables an attack against the overlay, as an attacker can simply hash each of the 64K host/port pairs of a particular IP address in advance, and select the closest Peer-ID to mount the attack (technically a Sybil attack, but with such large numbers available, effectively a placed ID attack).

Replacing the least significant bits with the port number *after* hashing forces the possible IDs to be contiguous, reducing the area of DHT space that can be attacked, but has another problem. Chord places redundant storage of information on sequential peers, meaning an attacker can compromise all replicas of a particular resource. Many other DHTs take a similar approach. (We show a solution that reduces this concern in Section 3.5) As a result, we determined that straight IP address hashing techniques are not sufficient for assigning Peer-IDs, and that even supplementing with ports is of limited effectiveness.

3.3 Distributed Identity

Perhaps the largest open question in securing a fully distributed is how to verify the identity of and communicate securely with a user. In practice there are two aspects to

this problem. When contacting a user with a particular name, can one be sure that the user is the same user we contacted previously with this name? In other words, can we detect two users attempting to use the same name at different times. A very closely related, but subtly different problem is ensuring that a user is the globally authorized instance of a particular username, or actually preventing two users from using the same name in the first place.

Once the user is identified, we also need the best way to secure the connection with the remote party. SIP/SIMPLE offers private-key mechanisms for authentication between UAs and proxies, and for end-to-end encryption [53]. A P2P architecture, however, requires a system that assumes no trust between the user and the system with which they are authenticating.

3.3.1 Problems with Early Proposals

During the development of P2PSIP, at least four mechanisms to assert identity and secure connections between users were proposed, but each ultimately had sufficient drawbacks to exclude it from consideration:

- **Storing the public portion of a self-generated user's key in the overlay:** This mechanism provides no guarantee as to the ultimate identity of a user, but once the certificate has been obtained one can use it to determine that future conversations are with the same individual as previous conversations, ensuring detection of, if not prevention of, multiple users using the same name. Users need to use the certificates and awareness of previous conversations to distinguish users. This mechanism allows communication between peers to be encrypted using this public certificate.

- **Email verification of user identities by the admitting peer:** This requires usernames be valid email addresses. A user attempting to register is challenged by the peer handling the registration to provide a key, which the admitting peer sends to the user in email. This test can be performed each time the user joins, or the public key of the accepted user can be added to the overlay and used on subsequent joins. While with no compromised peers this approach ensures the uniqueness of usernames, it has a number of shortcomings. Most notably, it requires that the admitting peer not be compromised. A corrupted peer in an email challenge system can mount a DOS attack against users for which they are responsible. Another serious drawback is that this system requires central equipment in the form of email access, and requires that identities be in the form of email addresses to ensure uniqueness.
- **A PGP [108] style web of trust model:** In this model, a user wishing to join the overlay creates a certificate and ask one or more other users to “verify” the new user’s identity by signing the new certificate. By following this path of signed certificates, the peer receiving the register can ascertain that the registering user really is who they say they are. The system could be constructed so that only signatures from peers inside the overlay are considered, or conventional PGP mechanisms could be used, which could require access to systems outside the overlay. While this approach works, it requires the often cumbersome step of asking other users to verify the identity of the joining peer before admitting it to the network. Once a user is signed, it can sign others, meaning a compromised user, once admitted, can allow additional bad users into the system. While revocation policies based on who signed the certificate can fix this problem, we feel this approach is only appropriate for certain deployments, and is not a general purpose solution.
- **Computational puzzles to determine the owner of a particular name:** In such

a mechanism, a user asserts that they own a particular username. Their public key, as well as a record of the number of puzzles they have computed under the particular name is stored in the overlay. Users claiming to own the same name must compute more puzzles to “capture” the ID. If the ID is not used for some period of time, the key would expire. While this mechanism prevents users from easily claiming to be another user, there are issues relating to how difficult the puzzles should be. Puzzles need to be simple enough that low processing power entities such as IP or mobile handsets can compute the puzzles, but hard enough that a dedicated attacker with access to a small cluster cannot overtake the user. In general, there is nothing to prevent someone with access to many machines (again, a botnet or student with many department computers) from taking over any desired name. Additionally, issues arise relating to the integrity of the balances when they are maintained by a possibly corrupt peer.

3.4 Securing the System with an Enrollment-time Certificate Server

To secure Internet-scale deployments, we propose a system where both Peer-IDs and user identities are secured using a public-key certificate model, with certificates issued by a central certificate authority (CA) contacted at enrollment (but not peer join) time. Small-scale ad-hoc systems, particularly those limited to a small number of trusted users, may still use an approach incorporating hash-based Peer-ID generation and a shared secret, but our deployments have shown that such systems are not practical for Internet-scale deployments. In Section 3.6 we provide some advice for making ad-hoc deployments as secure as possible.

The proposed certificate-based mechanism used in our system offers the following advantages:

- Prevents collision of usernames, ensuring that a particular username is only used once in a deployment.
- Ensures that users can verify the identity of a remote party each time they connect with that party.
- Provides a PKI framework for encryption of peer and user messages, as well as offline messages and data stored in the overlay. Such messages are protected from eavesdropping, even though they may be stored on random peers.
- Prevents duplication of Peer-IDs, and decouples the issuing of them from any controllable value associated with the user.
- Enables rate limiting of Peer-ID requests, reducing (although not eliminating) the threat from Sybil attacks.
- Offers secure signatures for messages, ensuring that peers can identify the Peer and/or user that messages originated from.

3.4.1 Architecture of an Enrollment-time Certificate Server System

This CA based approach differs significantly from a hybrid P2P approach. In a hybrid P2P approach, the server is required for run-time operation of the system. In tracker-based file sharing applications, this means a central server stores a list of the files available and which peers can provide them. In a hybrid communications system such as Skype, this means the authentication server is contacted frequently (typically either when a new peer

joins or when a user places a call) to verify users. The centralized certificate authority we propose is not involved in every transaction, or even in the operation of the overlay.

In our architecture, the centralized server issues certificates to peers and users at the time that they enroll in the overlay. Once the certificate is issued, there is no need for the server to be contacted again. The overlay can operate properly, users can communicate, and peers can come and go, all without the server, although no new users or peers can enroll until the server is available. In the case of certificates that expire (detailed below), the server may need to be contacted periodically to obtain a new certificate.

The server generates two types of certificates:

- **User certificates** assert that a user is valid and owns the specified username. Through this mechanism, the server guarantees the uniqueness of usernames.
- **Peer certificates** asserts that a peer is authorized to join the system and has the Peer-ID specified in the certificate. This Peer-ID is a unique, random number generated by the server, and is not tied to IP or any other attribute of the peer itself.

A user must obtain both a user and peer certificate from the CA before joining the system. We allow the certificates to be separate, since a particular device, for example an IP phone, might have multiple users or “lines,” requiring several User-IDs to be asserted, but only one Peer-ID. Similarly, a user may be present on multiple devices, requiring multiple Peer-IDs but only one User-ID. Finally, devices such as gateways, which allow VoIP calls to terminate onto the PSTN, might be peers but not be associated with a particular user.

In addition, each peer in the system obtains a copy of the CA's root certificate directly from the CA and stores it locally to prevent tampering. The public portion of the certificate for each user is stored persistently in the overlay to make them available even if the

user is offline. The user's certificate is used to encrypt media sessions as well as to secure messages (voicemail) left for the user. Because the chain for each certificate can be traced to the well-known, CA root, the authenticity of any certificate can be verified without contacting the central server.

3.4.2 DHT Operations with Certificates

When a peer attempts to join, it must present a valid certificate for the asserted Peer-ID, which is verified by the peer currently responsible for that region of the overlay. The certificate must be signed by the CA, as verified by the copy of the CA's root certificate each peer has.

In addition to preventing a peer from claiming an arbitrary Peer-ID by issuing randomly placed Peer-IDs, a CA can reduce the effectiveness of Sybil attacks by limiting the rate at which certificates are generated. Rate limiting mechanisms can include a minimal charge or requiring a valid credit card number that while not charged, can only be used to obtain an ID once in a given period. Other possible rate limiting mechanisms include verification of human presence (via CAPTCHA or email), or a computational puzzle. As a result, obtaining the very large number required to corrupt an overlay would be costly or impossible.

In addition to needing the certificate when joining the overlay, all requests by a particular peer are signed using the peer's certificate. This provides protection against two forms of attack. If an attacker does obtain a legitimate ID and launches a flooding attack, the sender's signature allows the attack to be detected, the malicious peer to be identified, and messages from that peer discarded (or the peer ejected). In addition, a peer sending a request can confirm that a response is delivered from the responsible peer, rather than a compromised peer along the route, by verifying the signature of the response. Similarly,

the recipient can verify the sender initiated the message. This allows “man-in-the-middle” attacks to be identified, and alternate routing paths (for example, sending using a different finger) can be used to circumvent them. Signing of the entire message, including timestamp but excluding portions that require modification on-path, such as via-list, also prevents replay attacks and tampering with the message while it traverses other peers.

Finally, in addition to signing the message, the portions of the message that are not required for routing can be encrypted using the public key of the destination peer, obtained from the overlay. This prevents intermediate peers from examining the contents of the messages.

3.4.3 User Operations with Certificates

The certificates issued for each username are used both to sign user information stored in the overlay and to encrypt media and messages between users.

A user registering a contact address used to reach them must sign the registration, incorporating an expiration time. A peer retrieving the registration does not need to trust the peer that stored the message, as they can directly verify the signed registration. An attacker, either the storing peer or a compromised peer along the query’s route can fail to return a registration or provide an outdated (but validly signed) registration, but cannot spoof one or take over a particular username by issuing registrations for names they don’t “own”. By simultaneously querying several replicas of the resource as discussed in Section 3.5, and employing a Byzantine comparison algorithm as discussed in [66, 78], the severity of these types of attack can be reduced.

The user certificates are also used to encrypt message payloads and media streams. Each user’s public key is stored in the overlay and the recipient’s public key is retrieved

and used to encrypt the text payload of IM messages sent to other users. These operations involve only retrieval from the overlay, and do not require communication with a central server. Additionally, the sender signs the message using their own public key. This ensures that the message is not intercepted and read on route, and ensures that the original message — not a spoofed payload — is received.

The public portions of the user certificates are also used by each side to encrypt media flows between users. If a recipient is offline, the caller records a message, encrypts it using the recipient's public key, and signs the message. The message is placed in the overlay for the recipient to retrieve later (keyed by their username), safe from both interception and tampering.

3.5 Providing Secure Redundancy

As discussed in Section 3.2.2, the default replication scheme for Chord places replicas on the peers succeeding the responsible peer. Because some hash-based ID generation schemes incorporating port numbers produce sequential IDs, it may be possible to attack sequential peers more easily. Even in systems where the IDs are assigned, the “half the distance to the target each step” approach used in a DHT means that the last few steps of a DHT search are often sequential. As a result, a corrupt peer may be able to deny easy access to a replica.

Instead, in our protocols we use a shared set of replica keys, as shown in Figure 3.2. As before, the same, well-known keyword is used to identify a resource. The keyword has a replica key appended to it, and the resulting string is hashed, producing a unique Resource-ID. For replication purposes, multiple shared replica keys are used, with the resource stored at each location. When a resource is stored, the storing party stores at all n locations. When retrieving, a peer can simply retrieve one, using others if they feel

the information is corrupt, or retrieve all n and compare the results using a Byzantine verification mechanism.

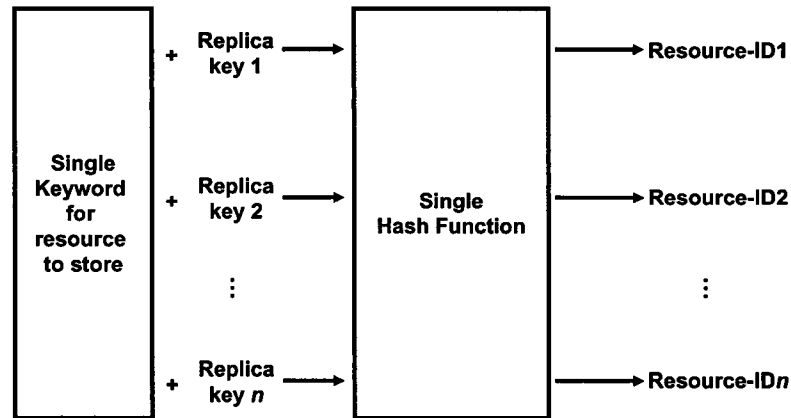


Figure 3.2: Using Replica Keys for Distributed Replication

This replication scheme can also be used as a very simple shared secret security mechanism. If only admitted peers know the key, it is difficult for an outside attacker to store or retrieve information. The keys can be changed periodically, further increasing the security of this mechanism. While no substitute for a CA-based system, it does provide some minimal security when a server is unavailable.

3.6 Recommendations for “Best Effort” Security in Ad-hoc Systems

While the certificate mechanism described in Section 3.4 provides very strong security and asserted identity without run-time servers, the system cannot be used for all deployments. For deployments where the use of a certificate model is not possible, for example in ad-hoc deployments, we provide the following advice:

- Use a key-based replication model such as the one described in Section 3.5 rather than a store on successor peer model.
- Use Peer-IDs produced by a hash of a public IP address with an unhashed port number appended. When combined with the replication scheme above, this will provide some protection against Sybil attacks.
- Use self-signed certificates. While such an approach won't prevent a user from claiming to be another user, it will allow a user to detect that they are talking to a different person. Use these certificates to secure all messages between users, preventing interception of messages.
- If the users of the system are able to share information offline, replica keys should be a shared secret. Any outsider attempting to use the overlay will be unable to store information to the correct location, or determine proper keys to find and retrieve information. While this provides only very basic security, it is superior to a completely open system.

3.7 Summary

In this chapter we have outlined the reason to use a DHT approach for P2PSIP to distribute registrations, store offline messages, and store public certificates (when used). We have explored the security implications of Peer-ID assignment and assuring distributed identity.

To address these problems, we have presented a solution based on an enrollment-time certificate server, and shown how this can be used to secure Peer-ID assignment, provide secure identity, authenticate peers and users, and encrypt messages (offline and live) and media sessions.

We then discussed the need to use alternate replication schemes for overlays, and show how a replica key system can not only help with replication, but provide a simple shared secret security mechanism.

Finally, we have provided advice on the “best efforts” that can be taken when a certificate server is unavailable, for example in ad-hoc deployments. In the next chapter, we present the details of two complete protocol designs we have created for P2PSIP: SOSIMPLE/dSIP and RELOAD.

Chapter 4

The SOSIMPLE/dSIP and RELOAD Protocols

Over the course of our research, we took two different approaches to solving the problem of creating a distributed personal communications system. While some of the interesting research problems, such as securing the system, identifying optimal routing techniques and ensuring namespace coherence are not impacted by the specific protocol design, others are affected by the choice. The two approaches are:

- **SOSIMPLE/dSIP solution:** This approach is an example of *P2P-over-SIP*, meaning the messages for all aspects of the solution (communications and overlay management) are carried using SIP messages. In this approach existing messages (specifically, the SIP REGISTER message) are used to carry the information required to establish and maintain the DHT, as well as to perform the traditional functions of a SIP client. We began with SOSIMPLE [21, 22], the first proposed P2PSIP solution (January 2005), which led to one of the first published paper on P2PSIP (June 2005). SOSIMPLE was defined as a complete protocol specification and as

part of our research we implemented a prototype of a SOSIMPLE instant message client based on the ReSIProcate open source SIP stack [74] . Later, SOSIMPLE grew into the dSIP protocol proposal [23, 28, 105, 106] (February 2007), which we again fully documented in protocol specifications. Our research on dSIP was conducted using an implementation developed by SIPeerior Technologies.

- **RELOAD solution:** As work on P2PSIP continued, much of the research moved toward a system where the P2P component of a P2PSIP solution was carried over a separate P2P protocol, termed *SIP-using-P2P*. The primary motivation behind this approach was reuse — allowing other applications that are not communications systems to reuse the P2P solution. We still believe P2P-over-SIP is the superior architecture when the single application a P2P communications system, but to participate in the research in SIP-using-P2P, we developed the RELOAD binary P2P protocol [20] for SIP-using-P2P (June 2007). This work was later merged with the work of several other authors to form the RELOAD protocol now being standardized for P2PSIP by the IETF [51, 52]. The discussion of RELOAD in this document refers to the earlier RELOAD we designed, not the later merged product of the working group.

In this chapter, we provide a description of each approach. Detailed formal protocol specifications for each design are attached as appendices.

4.1 SOSIMPLE/dSIP: A SIP-Based Protocol for P2PSIP

SOSIMPLE and dSIP are protocols for decentralized communications using SIP messages to carry P2P overlay information (P2P-over-SIP). Because the protocols developed over time, the discussion below applies to the newest version of dSIP unless

otherwise noted. The formal specification of the dSIP protocol, including protocol syntax/productions and network element processing instructions is included in Appendix A.

4.1.1 Why P2P-over-SIP?

There are several compelling reasons to use a P2P-over-SIP approach for interpersonal communications systems:

- In order to provide the features of SIP IM/VoIP, applications will need to incorporate a SIP stack. To prevent applications from having to add a second stack, SIP messages can be used to carry the P2P traffic.
- The design allows for compatibility with and reuse of existing SIP/SIMPLE network elements (IP phones, gateways, IM clients, etc.), protocols, and source code, meeting our requirement for Compatibility and Reuse.
- Because the P2P overlay's purpose is to support SIP communications, and many existing network shaping, monitoring, and management tools recognize SIP traffic, such a design allows for unified management of the SIP application and the DHT supporting it.
- An important property for a P2PSIP system is scalability. A P2P-over-SIP design preserves the SIP architecture's approach of sending only signaling through the SIP infrastructure (in this case the peers), while the media flows directly between UAs, improving scalability.

4.1.2 The SOSIMPLE/dSIP P2PSIP Architecture

Because our design has no central servers and peers communicate directly with one another, there are some differences between our approach and traditional SIP. Peers connect to a number of other peers in the overlay network and use these as the destination for most SIP messages. Peers act both as UAs and as SIP servers simultaneously. Specifically, the peers collectively replace the functionality of SIP registrars and proxies, with each peer responsible for providing these services to some portion of the overlay.

From a network architecture perspective, the code implementing the additional DHT and server functions can be located in several places in the network. The simplest is to have peers that are endpoints and which directly join the overlay. In this case the peers provide the basic functionality of any SIP endpoint and additionally implement dSIP to enable self-organization and provide SIP server-like functionality.

The distributed functionality can also be located in an adapter peer, which allows one or more non-P2P aware SIP UAs (UAs that do not speak dSIP) to interact with the P2P overlay network. These adapters perform dSIP functions on behalf of the UA or UAs they support. In this case, only the adapter peer is a peer in the overlay, the UAs are not peers themselves. Architecturally the adapters speak the P2PSIP Peer Protocol (dSIP in this case) and participate as full peers, where the UAs speak conventional SIP and participate as clients. The adapter essentially acts as a proxy server for the unmodified SIP UAs. The adapter can take the form of a small software shim on a host or dedicated box, or may be code within a conventional SIP server.

4.1.3 Protocol Design of SOSIMPLE/dSIP

4.1.3.1 Use of SIP Messages

Where possible, the semantics of existing conventional SIP messages are preserved in dSIP, even when used for a slightly different purpose. All messages needed to maintain the DHT as well as those needed to query for information are implemented using SIP messages. Fundamentally, messages are being exchanged for two purposes. The purpose of the first class of messages is to maintain the DHT and transfer information between peers. An example of this type is the set of messages needed to join or leave the overlay. We refer to these as peer operations. The second use of messages is in the more conventional SIP sense — registering users, inviting other users to a session, etc. We refer to these operations as user operations. Because the DHT is used as a distributed registrar, registrations and searches for resources are performed using the DHT. Once the target resource has been located, further communication proceeds directly between the UAs in the same way as conventional SIP.

The messages used to manipulate the DHT and convey peer-level operations are SIP REGISTER messages. The SIP specification states that REGISTER messages are used to “add, remove, and query bindings.” Accordingly, we have selected REGISTER methods to use to add, remove, and query the information the peers need to maintain the DHT. REGISTER is used to bind both locations in the overlay to hosts as neighbors (entries in the routing table) as well as resource names to locations that are commonly maintained by SIP registrars. From the perspective of the user-level SIP functionality, the only difference is that these operations occur within the overlay, rather than on the conventional server.

4.1.3.2 DHT Algorithms and IDs

dSIP is modular, allowing for the use of multiple DHT Algorithms. The first peer to start an overlay selects the DHT algorithm to be used, and later peers obtain this information when they attempt to join. For compatibility, support for the Chord DHT algorithm is required for all implementations. Additional DHT algorithms can be added and supported. In our work, we presented solutions using both Chord and Bamboo, and others have designed an extension for dSIP using Kademlia [27].

Each peer is assigned a Peer-ID, and each resource that is stored in the overlay is assigned a Resource-ID. These values must map to the same name space or range. dSIP allows various hash algorithms to be used to produce these values, although all members of the overlay must use the same algorithm. In the Chord DHT implementation, SHA-1 is used to produce 160 bit values for both the Peer-ID and Resource-ID.

The Peer-ID assigned to each peer determines the peer's location in the DHT and the range of Resource-IDs for which it will be responsible. The exact mapping between these is determined by the DHT algorithm used by the overlay. Our early SOSIMPLE work used a simple hash of IP address/port, but we encountered the problems discussed in Section 3.2.2 when deploying on the open Internet. dSIP allows for flexibility in generating Peer-IDs. A simple SHA-1 hash of the IP address and the port of the peer could be used to generate the Peer-IDs for an ad-hoc solution running on a single network (i.e., no NATs). For systems on the open Internet a certificate based system with CAs assign Peer-IDs as discussed in Section 3.4 is recommended.

Every resource has a Resource-ID. In the case of users, the unique keyword is the username and the resource is the registration — a mapping between the username and a contact. Resources can be thought of as being stored in the distributed hash table at a location corresponding to their Resource-ID. Because entities searching for resources

must be able to locate them given the unique keyword, Resource-IDs are produced by hashing, and are never assigned, regardless of the DHT and security mechanism being used.

Because dSIP allows for pluggable DHT algorithms, and each DHT is defined and functions differently, we generically refer to the table of peers that the DHT maintains and uses to route requests (i.e., neighbors) as a routing table. dSIP defines the syntax for the headers used by peers to exchange these entries, but leaves the exact form of the data each DHT stores in the table to the DHT implementation. Peers may maintain a list of peers to which they maintain connections for purposes other than routing, for example NAT traversal or caching. This larger table is a superset of the routing table and is referred to in dSIP as the connection table. Peers may make routing decisions using either the routing table or the larger connection table, and we use the terms interchangeably.

4.1.3.3 Routing

When locating a resource, a peer will first calculate the appropriate Resource-ID by hashing the keyword. The peer will then send the request to the routing table entry with the Peer-ID closest to the desired Resource-ID as defined by the particular DHT algorithm being used. The target could be another user, a particular resource, or a peer (including itself) for DHT maintenance purposes. Because DHT algorithms must converge on the resource, the peer receiving the request is assumed to know of a peer with a Peer-ID closer to the Resource-ID, and responds by suggesting or forwarding the message to this peer, depending on the routing mechanism being used.

The earliest work on SOSIMPLE used only an iterative routing mechanism, which we later determined to be sub-optimal. dSIP allows iterative or recursive (symmetric response or direct response) routing to be used. The specifics of selecting an appropriate

routing mechanism became a key component of our research and this topic is discussed in extensive detail in Chapter 5

dSIP allows various routing mechanisms to be used within the same overlay and even within the same search on a hop-by-hop basis. For example, a search may start as iterative, but if a particular peer receiving the request knows that the requester cannot reach the next hop directly (perhaps due to NAT issues), it may route the hop recursively.

4.1.3.4 Peer Operations

When a peer (called the joining peer) wishes to join or register with the overlay, it determines or obtains its Peer-ID and sends a REGISTER message to a bootstrap peer already in the overlay, requesting to join. Any peer in the DHT may serve as a bootstrap peer. The mechanism for selecting bootstrap peers is application dependent, and could include using a well-known peer, trying peers cached from the last time the joining peer connected, or using a broadcast mechanism to locate a peer on the local subnet participating in the overlay.

If an iterative routing scheme is being used, the bootstrap peer is asked to look up the peer it knows nearest to the Peer-ID of the joining peer and responds with a SIP 302 redirect response message directing the joining peer to this closer peer. The joining peer will repeat this process until the message reaches the admitting peer, which is the peer currently responsible for the space it will occupy.

Once the admitting peer is located, the joining peer then exchanges DHT state information with this peer allowing the joining peer to learn about other peers in the overlay (neighbors) and to obtain the resources the joining peer will now be responsible for maintaining. Other DHT maintenance messages will be exchanged later to maintain the overlay as other peers enter and leave, as well as to periodically verify the information

about the overlay, but once this set of initial messages are exchanged, a peer has joined the overlay.

Further messages will be sent by the joining peer to other peers in the overlay to update their finger tables to point to the new peer. At this point the peer has joined the overlay and peer registration is complete. Any users associated with the peer are not yet registered at this point, but user operations can now take place. We discuss the user operations in the next section.

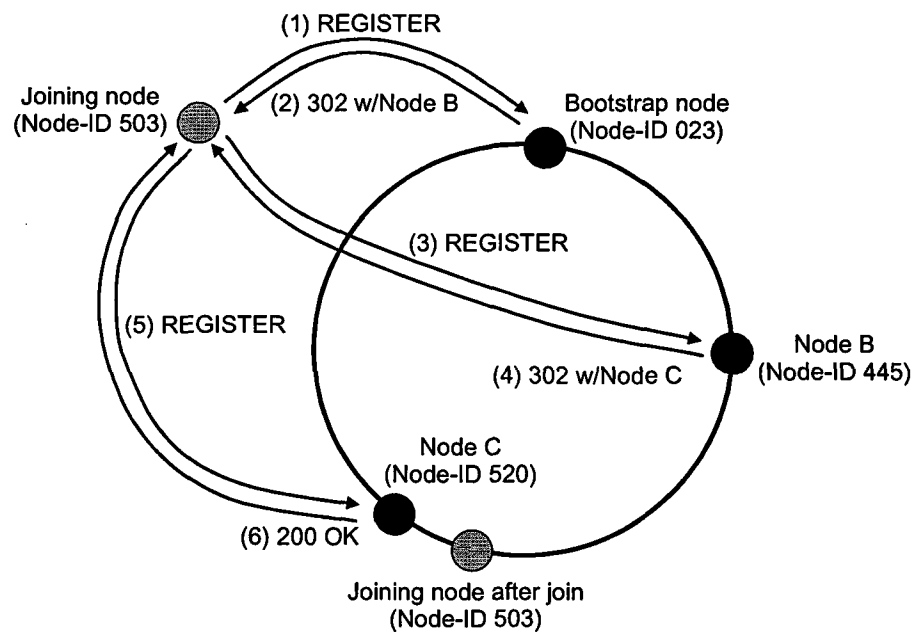


Figure 4.1: An Example of a New Peer With Peer-ID 503 Joining the Overlay

Figure 4.1 shows an example of a peer with Peer-ID 503 joining an overlay using iterative routing. The peer sends a REGISTER to the bootstrap peer, which has Peer-ID 023 (1). Assuming that the bootstrap peer is not the peer currently responsible for this region, it responds with information about the peers it knows nearest to where the joining peer will be placed in the overlay—in this case, Peer B, with Peer-ID 445. This information is passed back in our new headers within a SIP 302 Moved Temporarily response (2).

The joining peer repeats the process (3-4). Finally, the joining peer reaches the admitting peer, In this case Peer C, with Peer-ID 520. Peer C responds with a SIP 200 OK response including detailed information about nearby neighbors in the headers (5-6), allowing the joining peer to insert itself into the overlay.

If recursive routing is being used, the process is similar. The bootstrap peer looks up the peer it knows nearest to the Peer-ID of the joining peer but forwards the REGISTER message to that peer on behalf of the joining peer, rather than replying with a 302 redirection directing the joining peer there. This process of forwarding the message repeats until the admitting peer is found.

In order to keep the overlay stable, peers must periodically perform bookkeeping operation to take into account peer failures. These DHT maintenance messages are sent using REGISTER messages and the particular overlay algorithm being used dictates how often and where these messages are sent.

DHT maintenance messages are routed similarly to peer registrations. The peer calculates the Peer-ID of the peer it wants to exchange DHT information with, and sends the message, routed in either an iterative or recursive fashion, allowing the peers to exchange DHT maintenance information.

4.1.3.5 User Operations

The peer registration does not register the peer's user(s) or other resources with the P2PSIP network: It simply inserts the peer into the overlay. The user still must be registered in the overlay, after which the the overlay may be used to locate users, establish calls, transmit instant messages, etc.

User registration and resource location (finding users) are decentralized to meet our requirements for *No Central Server* and *No Central Naming Authority* . In order to register,

or to contact another user, the peer responsible for storing that user's information must be located. We will discuss registration and establishing a session individually, but first will address resource location in general as each requires it. We illustrate the resource location process in Figure 4.2. In this example, an iterative routing technique is shown.

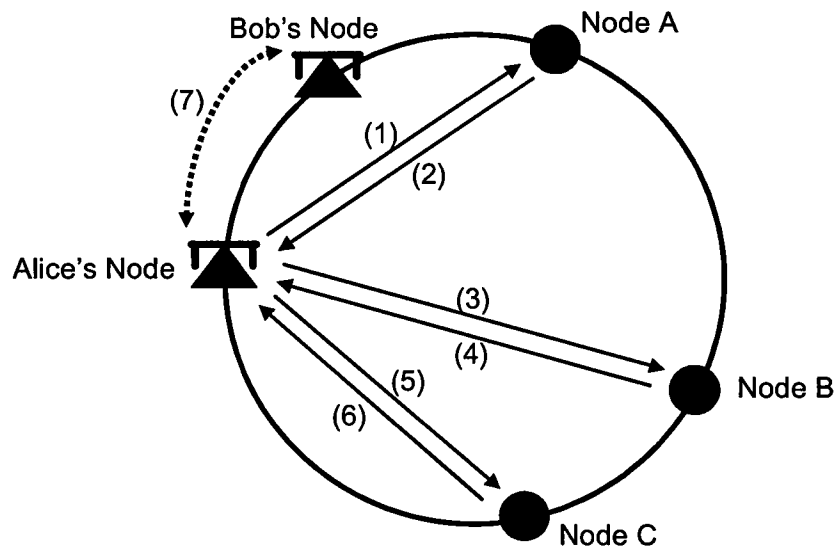


Figure 4.2: Alice Locates User Bob and Establishes a Communications Session (Iterative Routing)

When a user wishes to find the peer responsible for a particular user they begin by hashing the username to produce a Resource-ID. Because the user's peer is already properly in the overlay (peer registration has already occurred), the peer has a number of finger table entries pointing to peers within the overlay. In our example, Alice attempts to look up a resource (Bob). Alice's peer looks in its finger table and finds the peer with Peer-ID nearest the Resource-ID to be located. In this case, Peer A. Alice's peer sends a message to Peer A (1). The exact type of message sent depends on why Alice wants to locate the resource, as we'll see later in this section.

Because we are using an iterative routing approach in this example, and because

Peer A is not responsible for that Resource-ID, Peer A sends a SIP 302 reply, including the peer it thinks is closest, Peer B, in the headers (2). If recursive or direct response routing were being used, Peer A would forward the message to Peer B on Alice's behalf instead of sending a 302. After receiving the 302, Alice's peer tries Peer B and again receives a reply with another peer to try, in this case Peer C (3-4). Finally, Alice's peer tries Peer C, which is responsible for that resource (5). The result of that request varies depending on why Alice wished to locate the resource.

Once a peer has joined the overlay, the user(s) the peer hosts must be registered with the system. This process is referred to as resource registration. This registration is analogous to the conventional SIP registration, in which a message is sent to the registrar creating a mapping between a SIP URI and a user's contact. The only difference is that since there is no central registrar, some peer in the overlay will maintain the registration on the users behalf. In our example, if Alice was registering herself as a user within the overlay (user registration), the messages Alice's UA sent (1, 3, 5) would be of type REGISTER. When Peer C receives this message, it realizes that it is responsible for storing the registration on Alice's behalf. Peer C enters a mapping between Alice's username and her IP address into its registration table and replies to Alice's peer with a 200 OK (6). At this point, Alice is registered with the system.

If instead Alice was trying to call Bob, the messages Alice's UA sent (1, 3, 5) were either SIP INVITE messages to establish a VoIP call or SIP MESSAGE messages containing IM data. When these messages reach the desired peer, in this case Peer C, it looks in its registration table, determines it is responsible for Bob's registration, and looks to see if a registration is present. If there is not, a 404 Not Found is returned (6), indicating the user is not in the system. If Bob is registered, Peer C sends a 302 Moved Temporarily reply, providing a contact directly for Bob (6).

Once Alice has the IP address of Bob's UA, a call between these UAs can be estab-

lished directly between the peers using conventional (non-P2P) SIP mechanisms without involving the overlay (7). Alice's peer caches the information received from Peer C and can use that for future communications with Bob. Unless Bob or Alice logs off or switches peers (prompting a new search) the overlay is not involved in any further messages or conversations between Alice and Bob. Once this location information is passed to the UA, the call establishment is unmodified SIP—no P2P functions are required. The resource location portion can be implemented in a thin adapter shim, leaving the SIP UA unmodified. As in traditional SIP, the media also flows directly between the endpoints.

Note that because Alice and Bob's registration data would likely be stored on different peers (i.e, their usernames are likely to hash to Resource-IDs that map to different peers), "Peer C" in the register and session establishment examples above would most likely be different peers.

For redundancy, resources should also be registered at additional peers within the overlay. As discussed in section 3.5 there are many approaches that can be taken to provide redundancy. In dSIP these replicas are located by adding a replica number to the resource name and hashing to identify a new Resource-ID for each replica. In this way, replicas are located at unrelated points around the DHT, minimizing the risk of an attacker compromising more than one registration for a single resource. In the earlier SOSIMPLE design, the traditional Chord approach of storing replicas on the successors of the responsible peer was used, as shown in Figure 4.3.

4.1.3.6 NAT Traversal

Because dSIP uses conventional SIP messages, the mechanisms used for NAT traversal by the SIP protocol, including STUN, TURN and ICE are reused. As a consequence, many peers are able to participate in the overlay even when behind NATs. For those that

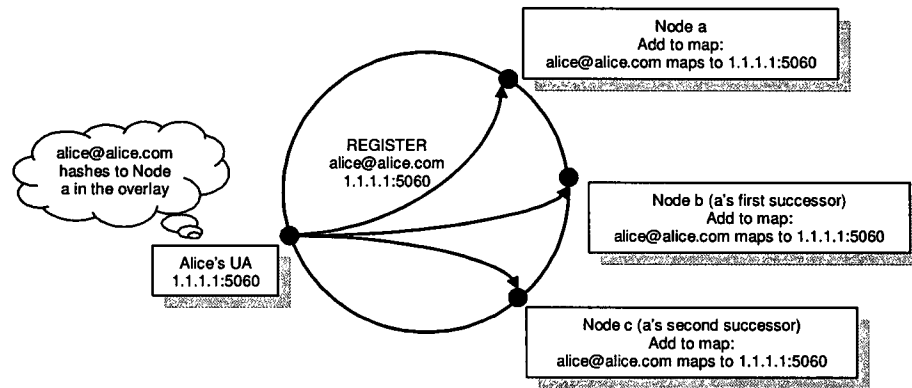


Figure 4.3: Example Redundant Storage of Registrations Using Successor Method

cannot for some reason, conventional SIP can be used, and these peers can connect using adapter peers, as described in Section 4.1.2. Since conventional SIP is used for this, there is no need for a P2PSIP Client Protocol (a particular protocol used for clients to connect to peers serving as super-peers), and therefore dSIP defines no such protocol. To the non-dSIP device, the adapter peer simply appears to be and functions as a conventional SIP proxy.

4.1.3.7 Offline Message Storage, IM and Presence

Essentially, offline messages in dSIP are stored by the peer responsible for hosting registration, and a PKI mechanism is used to secure the traffic and messages. A user leaving a voicemail or IM for an unavailable user records the message, signs it with their own certificate, and encrypts it with the remote party's public certificate. The peer that stores the registrations for the recipient also stores the offline messages (or a pointer to them), and they are retrieved the next time the recipient performs a user registration with the overlay. Note that other items such as buddy lists, configurations etc. can also be stored in the overlay to address the requirement of User Mobility Support.

IM is supported via the SIP MESSAGE method and routed like any other message using the P2P mechanisms described above. For presence, When a P2P UA starts, the user's buddy or contact list is consulted. The SIP SUBSCRIBE/NOTIFY mechanism (using P2P mechanisms to locate the resources) is used to attempt to establish subscriptions with these users. These subscriptions track the availability of the buddies. If the buddy is not connected, the subscription will be attempted later. The address of the peer responsible for the user's information (not the address of the UA itself) will also be cached, and if that peer supports the functionality, it can serve as a presence server for the offline peer.

4.2 RELOAD: A Binary Protocol for P2PSIP

RELOAD is a binary protocol, designed to be used alongside SIP to maintain a DHT and perform lookup tasks (SIP-using-P2P). Message encoding derives from the encoding used in the STUN protocol [82]. The formal specification of the RELOAD protocol, including protocol syntax/productions and network element processing instructions is included in Appendix B.

4.2.1 Why SIP-using-P2P

While we believe P2P-over-SIP to be a superior technical solution for a pure P2PSIP solution, there are a few advantages to a separate protocol. The most obvious are that a P2P-over-SIP protocol can be used for other applications, and the protocol can be designed as a clean sheet. Additionally, there are a few less obvious advantages to a distinct protocol, particularly one based on an existing binary encoding such as STUN:

- A fixed-length header can be used for routing the message through the overlay without the contents needing to be parsed (or even visible) by intermediate peers.
- The header includes no information about specific IP addresses because none are needed to route along an overlay. The header only includes source and destination Peer-IDs as well as some minor option and version flags. Clearly separating the header components necessary for routing from the message contents simplifies processing and increases security.
- Although the header uses fixed-length, fixed-offset fields, the content of the message is in STUN-based TLV (Type-Length-Value) format, extending a syntax for which many implementers (and many NAT-capable applications) will already have a parser. Furthermore, this extensible design allows great flexibility in supporting a variety of DHT and security algorithms using the same base peer protocol.
- A clean design allows for NAT traversal operations as part of the base design, including establishing new RELOAD connections and tunneling SIP or other application protocols required by P2PSIP.
- This design offers support for security primitives that separate the roles of message sender and resource owner, allowing a single message to contain multiple signatures authenticating different portions of the message content.

4.2.2 The RELOAD P2PSIP Architecture

In general, the architecture of RELOAD is quite similar to the architecture of SOSIMPLE/dSIP described in Section 4.1.2. However, there are a number of differences.

Strictly speaking, a RELOAD peer has no requirement to implement SIP. While a UA intended for interpersonal communication would clearly do so, it is possible that RELOAD

could be used to implement other applications, including file sharing. Additionally, even in a P2PSIP deployment, there might be reasons to have additional peers participating in the overlay, storing resources, etc., but not actually participating as SIP devices.

As a result of this disconnect between the P2P and SIP functionalities, the behavior of adapter peers is somewhat different in a RELOAD deployment. An adapter peer must implement both RELOAD and SIP and act as a SIP proxy on behalf of the conventional SIP clients it serves, sending requests to store, retrieve, and locate resources as required on behalf of these clients.

4.2.3 Protocol Design of RELOAD

4.2.3.1 Binary Messages

RELOAD uses binary message with a fixed length header. Fixed headers can be faster to process than messages without a fixed header because the headers can be processed by the stack and redirected to the appropriate peer without processing the entire message.

RELOAD provide three classes of messages: Peer, Resource, and Transport. Peer messages are designed to enable peer joins, or registrations, and peer searches, and perform a role roughly analogous to the peer operations in dSIP (Section 4.1.3.4) in that they are used for operation of the DHT. Resource messages include the GET, PUT, and TRANSFER messages and generally provide similar functions to the user operations in dSIP (Section 4.1.3.5) in that they allow information to be stored and retrieved to enable the application. Transport messages are designed to help with NAT traversal and the tunneling of both P2PSIP and SIP connections between machines, and do not have a direct dSIP analog, as dSIP relied on the NAT traversal mechanisms provided by the underlying SIP protocol.

RELOAD message bodies can contain information about multiple resources. RELOAD message bodies can contain the keys and bodies for multiple resource registrations, each signed by the person creating the resource. This enables information about multiple resources that a peer is responsible for to be sent in one message.

The transport methods are used to facilitate NAT and firewall traversal. The OPEN transport method provides a control connection that can be used by ICE to establish a direct connection between two peers for future RELOAD messages or messages in the form of the application's protocols. The TRANSPORT-TUNNEL method is used to encapsulate SIP (or other protocol, for other applications) messages in a RELOAD message that is routed across the overlay. TRANSPORT-TUNNEL is most useful for non-recurring messages such as SIP's INVITE, whereas adding a new peer to the routing table or using SIP MESSAGE or SUBSCRIBE/NOTIFY messages will benefit from using OPEN to establish a direct connection. Mechanisms are provided for encoding the SIP client's Peer-ID, allowing SIP messages to be routed across the overlay using P2PSIP primitives.

RELOAD messages can contain multiple bodies. For example, a transport OPEN can be sent in the same message as a resource PUT saving both time and the expense of sending an OPEN after the resource registration has completed.

RELOAD responses contain a numeric status response code, many of which (for example 302 to redirect to a new location) are identical to dSIP responses (which are identical to SIP responses, themselves derived from HTTP).

4.2.3.2 DHT Algorithms and IDs

Like dSIP, RELOAD allows multiple DHT algorithms to be implemented, allows the peers to indicate which is being used, and specifies Chord as a mandatory to implement com-

mon DHT. The creation and use of Peer-IDs and Resource-IDs in RELOAD is very similar to that in dSIP, as described in Section 4.1.3.2.

4.2.3.3 Routing

Like dSIP's routing (Section 4.1.3.3), various routing techniques, including iterative, symmetric recursive, direct response, and strict forwarding are supported. Various mechanisms may be used within the same overlay within the same search.

4.2.3.4 P2PSIP Resources and Resource Registration

RELOAD resources support a “one-to-many” mapping, with a single key or name (such as a SIP AoR) mapping to multiple resource bodies (registrations), each of which is annotated with a number of parameters indicating its type or other attributes. For example, a resource such as “bob@p2psip.example.org” can have registrations for types such as “voicemail” and “mobile”. Each resource body has its own expiration, parameters, and signature authenticating the registrations as genuine. A typical user's registration might also include a registration body with their certificate, allowing other users to validate the signatures as appropriate for the security specification used by the overlay.

Resource bodies are given separate expirations because they serve different purposes. An office phone, for example, might only be listed while the user is at their desk and is refreshed frequently by that device, whereas their voicemail registration should be long-lived to allow messages to be left when the user is away from the desk. RELOAD allows invalidation of individual resource bodies as well as mass invalidation of the resource values for a particular resource name.

4.2.3.5 Peer Operations

In general, the process for a peer to join the overlay is similar to dSIP's operation (Section 4.1.3.4). Peers wishing to join the overlay in a particular location send a RELOAD JOIN message to a bootstrap peer already in the overlay with the desired location. Bootstrap mechanisms are flexible, as in dSIP. For iterative routing, a series of 302 responses will be returned suggesting peers ever closer to the area the joining peer wishes to insert itself into, until the peer currently responsible for the area is reached. For iterative routing, intermediate peers will forward the JOIN message. Once the admitting peer is reached, messages are exchanged to admit the peer, and further messages transfer resources and later maintain the overlay.

4.2.3.6 Resource Operations

Resource operations are similar to user operation in dSIP (Section 4.1.3.5). Resources must be registered into the overlay using RELOAD messages, referred to as resource registration. Resource-IDs are calculated by hashing keywords, and the search messages to locate a peer to store or retrieve a resource can be routed either iteratively or recursively.

A resource registration maps from a key to a value, typically a URL. As different resources (a user, a voicemail, a movie) vary in the action taken to use them, and their size, the same transport or protocol may not be suitable for different resources. The specific mechanism being used may vary and need not be the same within the overlay. Essentially, the search for a keyword returns a URL-like object, specifying where the object can be retrieved, and what protocol is used to retrieve it. RELOAD can be used for some of these transfers directly, but is not optimized for large file transfer. An existing

mechanism defined for such a purpose, such as HTTP, scp, MSRP or FTP is instead used to store and retrieve larger objects.

For redundancy, a replica number hashing scheme, as used in dSIP and described in Section 3.5 is used.

SIP Sessions are established by contacting the UA identified by the registration in the DHT. The first step in establishing a session is locating the remote party, which is done by searching for a resource in the DHT (using RELOAD GET messages). If a registration is found, the session (using SIP messages) is then initiated directly with the UA.

4.2.3.7 Large Messages and Fragmentation

Practical issues of deploying RELOAD dictate that there is no one transport protocol that is appropriate for all situations. Device power, the number of connections that must be maintained, durations of typical connections, NAT and firewall policies and capabilities, and message size all influence the choice of transport protocol and frequently lead to different conclusions. Because stream-oriented protocols such as TCP cannot always be used, and because unlike dSIP the underlying protocol cannot be relied upon to provide these functions, the RELOAD protocol provides fragmentation support.

RELOAD messages that are very large can be sent in multiple fragments or chunks. Each fragment contains one part of the complete message. The Frag-Offset header field indicates the overall position of each fragment, and the Frag-Length header field indicates the total fragments in the complete message. The FRAG header field is a flag that indicates whether the RELOAD message is fragmented or not. Finally, the LFRG header field is a flag that indicates whether this is the last fragment of the complete message.

4.2.3.8 NAT Traversal and Transport Operations

RELOAD is designed with NAT traversal in mind. The Transport OPEN and TUNNEL messages are designed to be used by peers to ask neighbors for help opening NAT holes and establishing connections that can reliably be used to traverse NATs. OPEN is used to establish new RELOAD connections between peers and serves as a control connection for the exchange of ICE messages used to facilitate NAT traversal. OPEN is used when establishing new Routing Table entries by both iterative and recursive routing and may be needed for iterative routing when the searching peer is redirected to a peer it cannot directly contact. In this case, the redirecting peer serves as the intermediary, relaying the OPEN message between the two peers while they establish their own connection.

TRANSPORT-TUNNEL is used to encapsulate application messages, such as SIP messages, as they are routed across the overlay. Using encapsulation eliminates the cost of establishing a direct connection (of benefit to short-lived application connections) and provides a way to support applications that do not themselves support the capabilities required for using OPEN or ICE directly.

4.3 Summary

In this chapter, we have provided a thumbnail sketch of our design of two complete protocols, the SOSIMPLE/dSIP P2P-over-SIP protocol and the RELOAD SIP-using-P2P protocol. Out of necessity, many details have been omitted. More detail of the design of these protocols can be found in Appendix A and Appendix B, and the detailed full specifications (consisting, collectively of several hundred pages) are available online [20, 23, 28, 59, 105, 106]

Chapter 5

Message Routing for P2PSIP

In evaluating how best to optimize a P2PSIP network, one of the most critical factors is the routing of messages. P2PSIP systems have some unique properties, some of which are not always present in file sharing systems. In a P2PSIP system, the connection to search for a piece of information (for example the location of another user) may not require a persistent connection to that peer after the information is returned. In the case of locating a remote user, once the location is returned, further communication is with the remote user referenced, not the peer storing the information. Additionally, the requirements around NATs may differ considerably from those required for file sharing applications.

As discussed in Section 3.1.1, Internet-scale P2P communications systems must operate in diverse network environments that frequently have a majority of endpoints located behind NATs, and we wish to have many of these devices directly join the network, rather than connect via super-peers. The choice of routing algorithm—iterative, recursive, etc.—might at first appear to be merely a question of the number of hops taken by a message. However, if we choose to utilize peers behind NATs, we must select an algorithm that allows them to function efficiently in the overlay, while considering how the unique properties of a P2PSIP network may impact algorithm selection.

5.1 Narrowing the Field

Our initial step in selecting a routing technique is to identify and remove those that are clearly non-optimal before performing a more detailed analysis on the remaining candidates. To do this, we consider a number of factors, including the specific requirements imposed by personal communications, the impact of NATs, and the number of messages required.

To enable this comparison, in Table 5.1, we explore the basic worst-case costs of four metrics for the four mechanisms presented in Section 2.3.4.3:

- The total number of messages (request and response) exchanged between the peers for a transaction, neglecting NATs as well as the cost of any messages required to establish a secure connection.
- The number of new connections that must be established. We assume a stable overlay, meaning that a peer can reach other peers already in the finger table without having to establish a new connection, but peers not in the finger table require new connections.
- Given that some messages will require a new connection, and given that the peer may be behind a NAT, we calculate the total number of messages (request and response) including the cost for NAT traversal for each new connection.

Table 5.1: Comparison of costs associated with various routing algorithms

Algorithm	Messages	New Conn. Established	Msgs Including NAT Traversal	Msgs Processed by Interior Peers
Symmetric Recursive	$2 \lg(N)$	0	$2 \lg(N)$	4
Direct Response	$\lg(N) + 1$	1	$3 \lg(N) + 5$	2
Iterative	$2 \lg(N)$	$\lg(N) - 1$	$10 \lg(N) - 8$	2
Forward-only	$2 \lg(N)$	0	$2 \lg(N)$	2 to 4

All values are worst-case. We assume an overlay of size N peers and further assume an algorithm like Chord, with worst-case routing cost of $\lg(N)$.

As our protocols (SOSIMPLE, dSIP, and RELOAD) as well as the new RELOAD being developed in the IETF all use ICE to establish connections in the presence of NATs, we calculate costs for NAT traversal based on ICE. ICE relies on a rendezvous service to exchange information between the two devices attempting to form a direct connection. When used in a P2P overlay, the two peers can form a direct connection by relying on the overlay to route the ICE offer/answer messages that allow them to establish a direct connection. Using ICE to open a new connection requires the round-trip open request/response routed along the existing overlay connection plus at least one pair of STUN request/response messages sent directly between the two peers in each direction.

As already mentioned, for our analysis we assume that the connections to immediate neighbors are established in advance and keep-alive messages occur during periodic DHT maintenance, therefore we look only at the cost of contacting a peer not in our set of neighbors.

Using symmetric recursive routing, messages are proxied through the overlay and no new connections must be opened through NATs. Forward-only routing is similar, as only established connections are used. The round trip number of messages in both cases is thus $2 \lg(N)$. For symmetric recursive, each interior peer deals with four messages—receiving then sending the request on, and receiving and sending the response on. Forward-only is similar, but as the return path may be different, each peer may only process two messages (four if the return path is the same as the request path, which may occur either by coincidence or because an algorithm with symmetric neighbor tables such as Kademila is being used.)

Direct response routing attempts to deliver the response directly to the querying peer. If the peer is behind a NAT that allows the traffic through, then the direct response is more

efficient, but otherwise, the direct response requires an ICE exchange to open the connection. For our worst-case examination, we assume that the ICE exchange is required each time. In Section 5.2 we show a more detailed analysis that takes into account the number of peers that are unreachable behind NATs and gives a more favorable impression of the direct response routing technique, but we include it here for completeness. Given this assumption, the cost in column four of Table 5.1 is derived by counting the normal $\lg(N) + 1$ messages, plus a round-trip symmetric recursively routed ICE open request ($2 \lg(N)$ hops), plus the STUN connection checks (4).

For an iterative algorithm, many more connections must be established. While the first peer where a message is sent is the direct neighbor of the source, all subsequent connections may be (and in a large overlay are statistically likely to be) with non-neighbors. A new NAT connection must be opened for each of these connections. In all, as many as $\lg(N) - 1$ new connections may need to be established. By routing the ICE open request through the referring peer, each ICE request/response will take exactly 4 hops, with an additional 4 STUN connection check messages.

5.1.1 Eliminating Iterative Routing

Table 5.1 shows that Iterative is an extremely expensive routing technique in the presence of NATs. In the worst-case example presented here, the number of messages needed approaches $5\times$ the number needed for symmetric recursive routing for large overlays.

In Section 5.2 we explore a more detailed analysis of direct response routing that takes into account the probability that a peer is behind a NAT, illustrating that direct response is not always as expensive as it appears here. The reason that we did not perform a similar analysis for iterative routing is because of the greater number of new connections that need to be established. With direct response only one new connection must

be established. If 50% of the peers are behind NATs, direct response will succeed 50% of the time, and the tradeoff between better performance when NATs were not present and decreased performance in their presence seemed worth a closer look. With iterative routing however, $\lg(N) - 1$ new connections must be established. For an overlay of even moderate size, perhaps 1000 users, this implies approximately 9 new connections need to be established using iterative routing. Even with only 10% of the peers behind NATs, the chances of having all new connections established directly without needing NAT traversal is less than 40%.

Additionally, there is little gain to using iterative. Unlike direct response, the total number of message is the same as symmetric recursive. The primary advantage to an iterative approach is that interior peers need to provide less processing—that is, the effort is pushed to the requesting peer, rather than intermediary peers. While this is appealing in a P2P system where designing the system to cause the requester to do the bulk of the work (for their own requests) increases fairness, direct response offers the same advantage. In the environment where using iterative would suffer the least penalty (environments with no NATs), direct response also performs well, offering the same reduction in processing by intermediate peers while also reducing total message count significantly. As a result we eliminate iterative routing from our analysis and focus on more promising techniques.

With that said, there is one area where iterative routing offers an advantage. Iterative routing offers some protection against DoS attacks in that it is more difficult to use another peer as a DoS surrogate or multiplier when iterative routing is used. In a completely unsecured network, this may be a consideration. See section 5.4.1 for a more in-depth discussion of this issue.

5.1.2 Eliminating Forward-Only Routing

Forward-only offers performance very similar to symmetric recursive routing. The same number of messages are used, and like symmetric recursive routing, no new connections need to be established because all messages are routed through existing finger table entries. The advantage offered by forward-only routing is that (for larger overlays where the response path is unlikely to overlap the request path), the load imposed on interior peers is more evenly distributed.

Unfortunately, there are a number of significant drawbacks that cause us to eliminate forward-only routing as a recommended technique for P2PSIP.

First, forward-only cannot be used as the exclusive routing technique in an overlay. In the case of a peer just joining the overlay, the response from an admitting peer to the joining peer cannot reach the joining peer if it is forwarded around the overlay because the joining peer is not yet properly inserted and therefore not reachable via overlay routing. As a result, messages during a peer join would need to be routed differently from other messages. While this is obviously possible, it introduces additional complexity into the design of a system.

In addition, in overlays with unstable or incorrect routing information (for example in the presence of significant churn), the odds of a message being misrouted are higher using forward-only routing. With forward-only routing, on average twice as many finger table entries must be correct to enable the message to be routed properly— both the forward path and the forward-calculated return path must be correct. A similar argument applies for peer failure, as twice as many peers will be traversed, there is twice the chance that a failed peer will be encountered in routing the message.

As a result, we also eliminate forward-only routing from our evaluation. Given this, for the remainder of our analysis, we will analyze the two remaining existing proposals:

symmetric recursive response and direct response.

5.2 Return Routing and Asymmetric Connectivity

As ever increasing numbers of applications are delivered using Peer-to-Peer (P2P) architectures, the networks on which these applications must operate, and the resulting constraints, have become more diverse. In particular, these networks often exhibit asymmetric network connectivity problems, where a peer may be unreachable by some fraction of other peers in the overlay, or a peer may not be able to directly respond to a peer that can send it messages. As discussed above, these connectivity problems are most often caused by NATs or firewalls. However, these failures may also be caused by transient phenomena such as link failures or BGP updates [39].

Given that we eliminated iterative and forward-only routing from consideration in the sections above, and that the forward routing cost of symmetric recursive and direct response routing are the same, the critical remaining element in comparing P2P routing algorithms is the mechanism used to return responses to the sending peer. With some exceptions [17, 38] the literature on response routing for P2P algorithms, particularly DHTs, neglects the impact of asymmetric network connectivity problems on selecting a technique, advocating the most efficient solutions, even when these are impractical in real deployments. Some direct comparisons of these techniques in the literature [26] neglect transient network problems and the costs associated with opening new connections (for example the overhead to secure the connection). On the other hand, the Internet protocol standards community has defaulted to using techniques known to work in the presence of NATs, but which are often highly inefficient [52] for peers with adequate direct connectivity.

In this section we examine direct response and symmetric recursive response, and use an analytical model to evaluate their performance in the context of operation on the open Internet, considering the impact of asymmetric connectivity, as well as considering the costs incurred when new connections are established.

5.2.1 Deployment Environments

The environments in which P2P applications are deployed differ widely in terms of the fraction of peers that may have asymmetric connectivity. In some environments virtually all peers are able to reach each other directly, with no interference from NATs. Examples of this types of environment include LANs, overlays where the majority of peers are located in universities or large companies, and certain experimental deployments like Planetlab [71].

Other environments may have a majority of peers that have some connectivity issues, including peers behind one or more NATs. For example, overlays where many peers are located in homes, small corporate offices, or in countries where IPv4 addresses are in short supply fit this description. The fraction of such peers in an overlay may change based on time of day, changes in the geographical location or changes in the population from which the peers are drawn.

5.2.2 Defining An Analytical Model for Comparing Response Times

For our comparison, we make a number of assumptions about the overlay and the network that it is operating on. The motivation for these assumptions is to make the system very closely correspond to the emerging IETF DHT standard protocol RELOAD. Our assumptions in this section are:

- A DHT-based overlay using an algorithm (for example, Chord [94].) requiring $\lg(N)$ messages in the worst-case for routing, where N is the number of peers composing the overlay.
- As we wish to evaluate the impact of NATs and security messages on the cost of routing, and want to impose the maximum penalties when they are encountered, we use the worst-case value for routing cost here, rather than the average case $\frac{\lg(N)}{2}$ for a DHT. Either could be used. As a result, our model presents worst-case values for the cost of actually routing a message, although we arguably use average cost routing for deciding when a peer is behind a NAT (i.e., when we assume 50% of peers are behind NATs, we assume 50% of queries encounter a NAT, not a truly worst-case world where, by bad luck, 100% of requests encounter a NAT). If we did this, as we will see, recursive response would always provide the best results, and our model would provide no useful information. As a result, we obtain something that is probably best characterized as an “expected worst-case result”.
- As we eliminated iterative and forward-routing from consideration in Section 5.1, all the techniques being compared require the same $\lg(N)$ message to traverse the overlay from requester to responder. We analyze only the costs associated with the response message(s) in this section. **Note that this differs from section 5.1 where we considered total cost** (request and response). We also assume requests follow established, stable links between peers where necessary NAT traversal and security establishment negotiation are complete.
- We use a request-response transaction model for the analysis. Once the responder has provided an answer, the transaction is concluded. Further actions (e.g. NAT traversal) that might be required for additional or permanent connections between peers, such as establishing media streams are not considered. From the point of view of our application this makes sense. With P2PSIP, the overlay is used to locate

the remote party, and there is not always a need for a persistent connection to the remote party.

- In keeping with our overall design, this analysis evaluates the performance of techniques that maximize the number of peers able to fully participate in the overlay (full peers, or “super-peers”). While there may be other motivations for a peer to rely on a super-peer for services, we evaluate alternative routing techniques that do not require super-peers simply for reachability.

5.2.2.1 Modeling Symmetric Recursive Response Routing (SR)

In symmetric recursive response routing (Figure 2.7, Section 2.3.4.3), responses follow the original path of the request through the overlay. This approach has been shown to provide a number of advantages for systems running on the open Internet. Because messages retrace the established connections of the DHT, there is no need to negotiate new secure connection or perform NAT traversal, and therefore this return routing technique ensures responses are properly delivered, assuming the overlay is stable and the path still exists when the response is sent. However, this approach is expensive in terms of total message count and number of hops traversed in those cases where a direct responses would be possible.

For Symmetric Recursive, the message count for the response, MC_{SR} , is equal to the length of the forward path, $lg(N)$. To route the response, intermediate peers are required to process and forward the response. Excluding the sender and recipient, that number of intermediate peers that must receive and re-transmit a response on its way back to the sender, IP_{SR} is $lg(N) - 2$. Assuming mean time τ for messages to traverse one hop in the overlay, we can additionally define the mean propagation time T_{SR} required for the response to travel from the responder back to the requester. For SR, the key metrics are

defined as:

$$\begin{aligned} MC_{SR} &= \lg(N) \\ T_{SR} &= \tau \lg(N) \\ IP_{SR} &= \lg(N) - 2 \end{aligned}$$

5.2.2.2 Modeling Simple Direct Response Routing (SD)

In the simplest form of direct response routing, responses are sent directly to the sender in a single message. This approach is very efficient, and is frequently proposed as the mechanism for sending responses in a P2P network. However, if a direct connection is not possible or has not been pre-negotiated between the responder and requester, this approach will fail. Additionally, delivery failures may not be detectable when an unreliable transport is being used with this technique, as there is no way to determine if the message was sent properly. This mechanism is by far the most efficient in networks where connectivity is guaranteed (e.g. closed networks, managed networks, etc.), but in this basic form is unsuitable for networks where direct connectivity cannot be guaranteed.

While limited in where they can be used, the values for the message count, mean propagation time, and processing by intermediate peers when using the simplest direct response mechanism are useful as a baseline of the lowest possible cost, and can be defined as:

$$\begin{aligned} MC_{SD} &= 1 \\ T_{SD} &= \tau \\ IP_{SD} &= 0 \end{aligned}$$

Note for Simple Direct, no processing by intermediate peers is required to deliver responses.

5.2.2.3 Modeling Practical Direct Response Routing (DR)

To properly analyze the cost of a direct response technique in an open Internet deployment, it is necessary to augment the simple direct response routing method. For our analysis, we will define and use an augmented version of direct routing, which we refer to as augmented Direct Response (DR). Augmented DR is able to function in the presence of NATs due to a fall-back mechanism for SR response routing.

In this routing approach, additional messages exchanged between the requester and responder determine if a direct connection exists. If the reachability mechanism fails (after a timeout), we retransmit the response using Symmetric Response. (An alternative in the failure case would be to instead perform NAT traversal using some technique such as ICE and establish a connection, but such an exchange is far more expensive, and not required for simple request-response transactions).

The minimum message count of the augmented technique is two, assuming both requester and responder are directly reachable: the response and an acknowledgment returned from the requester. In the proposed RELOAD draft protocol being proposed by the IETF, TLS [33] or DTLS is used to secure the response. If the DTLS protocol [65, 73] is used to establish secure channels between peers, seven messages are exchanged between requester and responder. Assuming the initial message pair in the DTLS handshake is used as the initial reachability check between requester and responder, eight messages (seven to negotiate the connection as well as the response) will be used in the transaction. As symmetric response always works, the burden of proof is on the direct response to show that it can work. Therefore we use this worst-case value in our analysis.

In any case, the difference between the technique evaluated here, using eight messages, and a simpler approach using only two messages is a constant factor.

Further, because the DTLS protocol recommends an initial timeout on the security handshake of $\tau_S = 1s$, we use this value for our analysis as the timeout for a response. (For systems not using TLS, a shorter timeout, perhaps $3 \times$ mean RTT time, or $\tau_S = 450ms$ could be used, but we have chosen to use the more expensive cost in the model to capture worst-case behavior.) In the event no response is received to the initial message within 1s, the responder assumes the requester is not directly reachable, aborts the handshake, and resends the reply using Symmetric Response. Using τ to represent the mean latency on an overlay link, defining the probability any given peer is unreachable to be P , and including the seven message cost of establishing a DTLS connection prior to sending back the response itself, this gives us mean message count and propagation times of:

$$MC_{DR} = P (\lg(N) + 1) + (1 - P) 8$$

$$T_{DR} = P (\tau \lg(N) + \tau_S) + (1 - P)(8 \tau)$$

Finally, the cost for intermediate peers to route a response for DR depends on the probability that the sender is reachable. However, when the response can be directly routed, there is no cost to intermediate peers to process the response. IP_{DR} can thus be defined as:

$$IP_{DR} = P (\lg(N) - 2)$$

5.2.3 Analytical Model Results

5.2.3.1 Comparing Message Count

First we examine the mean message counts required for each type of response routing technique. Analytically, Simple Direct (SD) is always most efficient, using a low fixed number of messages $MC_{SD} = 1$, but is impractical in all but the few networks with perfect symmetric connectivity. when augmented Direct Response (DR) succeeds in sending a message directly, eight messages (in the case of DTLS security) will be used. This will be lower than the cost of a Symmetric Recursive (SR) response $MC_{SR} = \lg(N)$ in all overlays with $N > 2^8$. In the case that the direct connection of DR fails, it will need to fall back to and incur the full cost of SR, plus an additional message (the initial connectivity check that failed).

In an overlay where peers are unreachable directly with probability P , Figure 5.1 shows each mean message count MC for Symmetric Recursive (SR), Simple Direct (SD) and augmented Direct Response (DR) for overlays of size $N = 10K$ and $N = 1M$ (SD is identical for overlays of any size).

Comparing SR to DR, we first evaluate the conditions under which DR will outperform SR in terms of average message count. Setting MC_{DR} equal to MC_{SR} and solving for P , we find that the mean expected worst-case message count of these two techniques is equivalent when the following condition is met:

$$P_{EQ} = \frac{\lg(N) - 8}{\lg(N) - 7}$$

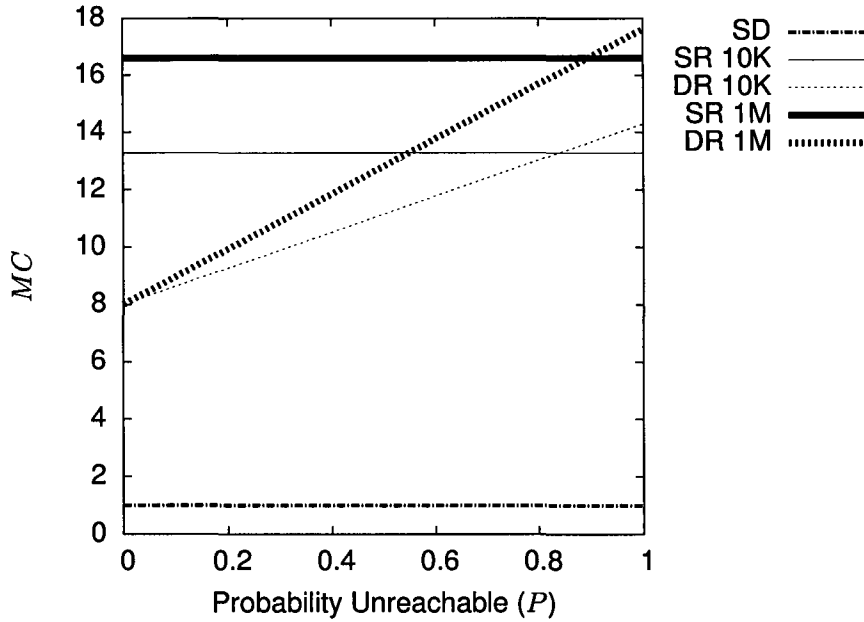


Figure 5.1: MC vs. P by routing type

When $P < P_{EQ}$ for a given size network, DR response routing will outperform SR response routing, and when $P > P_{EQ}$, SR will be more efficient in terms of mean message count. Graphically, we present this probability for overlays of size 10K to 1M peers in Figure 5.2. For an overlay of size $N = 10K$, DR will outperform SR when $P < .66$, and for $N = 1M$ DR will outperform SR for all $P < .92$. In terms of DR's performance, DR requires one additional message than SR in the worst-case. In the best case, DR requires a factor of $\frac{\lg(N)}{8}$ less messages than SR. While in many cases the number of peers likely to be unreachable cannot be estimated (particularly for a generic package or protocol intended for use in many environments), if it is static and can be measured or calculated, Figure 5.2 can be used to determine which technique is most efficient. For large overlays and unless nearly all peers are behind NATs, DR is usually more efficient than SR in terms of message count, even when the high additional overhead of establishing DTLS connections and needing to occasionally fall back to SR is considered.

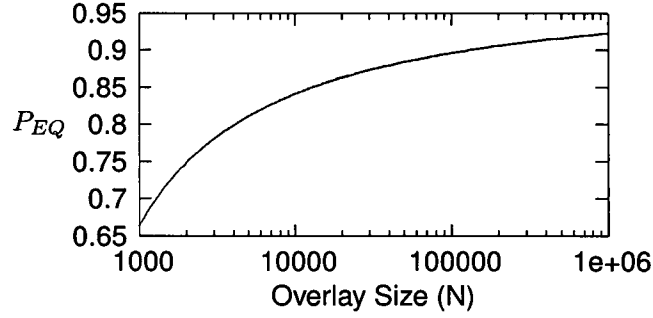


Figure 5.2: P_{EQ} vs. N for SR/DR routing

5.2.3.2 Comparing Propagation Time

To analyze the mean propagation time for responses, we need to determine a value to use for the average overlay link propagation time τ . An analysis of the PlanetLab environment in 2008 found that the average RTT latency between nodes was approximately 150ms [104]. The majority of PlanetLab nodes are located at universities and research institutes, meaning their RTT latency is lower than the average experienced by nodes in various environments and geographical locations, e.g. home users. For this analysis we use an average one-way time for a message to travel between peers of one half this value: $\tau = 75\text{ms}$.

Figure 5.3 shows the average propagation time T for each of the techniques with respect to P . For SR and DR, we illustrate these techniques for overlays of size 10K and 1M.

Here again SD exhibits the best performance, with constant time, but can only be used in environments where complete connectivity can be assured. Just as with total message count, whether SR or DR is more efficient depends on the fraction of peers that are not directly reachable. Again we compare these by setting T_{SR} equal to T_{DR} and solving for P . We find that the average time for these techniques is equal when the following condition is met:

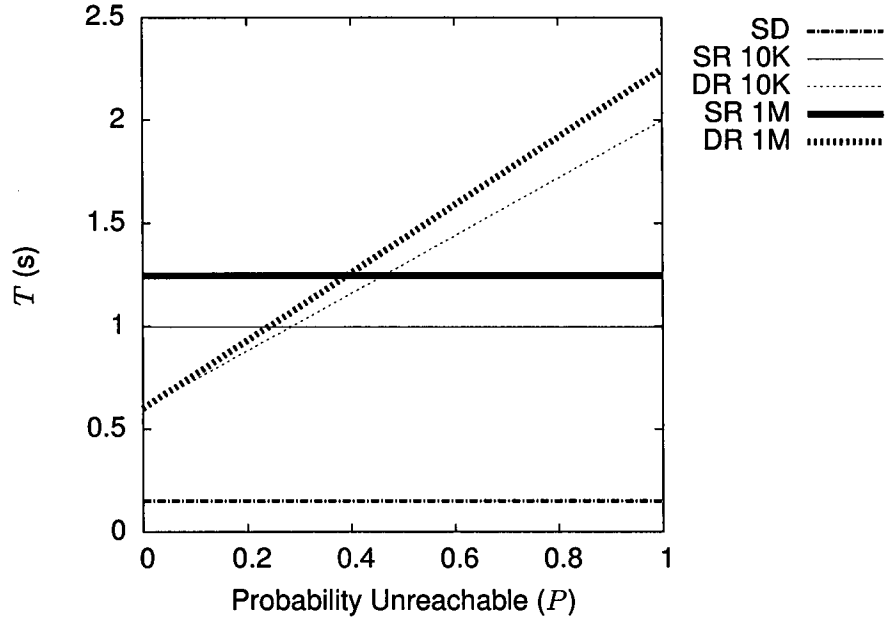


Figure 5.3: T vs. P by routing technique

$$P'_{EQ} = \frac{\lg(N) - 8}{\lg(N) + \frac{\tau_S}{\tau} - 8}$$

Because $\tau = 75\text{ms}$ and $\tau_S = 1\text{s}$, $\frac{\tau_S}{\tau} \approx 13.33$, and we have:

$$P'_{EQ} \approx \frac{\lg(N) - 8}{\lg(N) - 5.33}$$

Plotting P'_{EQ} vs. N , we can see in Figure 5.4 that, as with message count, DR becomes more efficient as N increases. However in terms of mean response time, the fraction of peers that must be unreachable for SR to outperform DR is considerably lower. In the case of time, DR will outperform SR only when $P < .128$ for $N = 1\text{K}$ and when $P < .47$ for $N = 1\text{M}$. In terms of DR's performance, in the worst-case, when a direct response is attempted and fails, DR will require an additional $\tau_S = 1\text{s.}$ more time than SR to route the response. This delay is due to waiting for the direct message to time out, and

then retrying using SR. In the best case, DR again improves performance over SR by a factor of $\frac{\lg(N)}{8}$ in terms of average time. For smaller overlays, SR almost always outperforms DR in terms of propagation time, but for larger overlays, systems with a moderate number of NATs will achieve lower propagation times using DR.

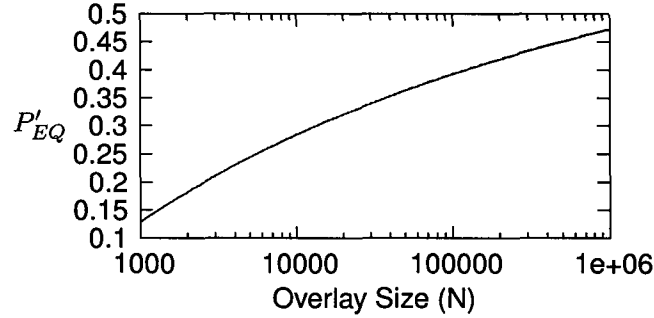


Figure 5.4: P'_{EQ} vs. N for SR/DR routing

5.2.3.3 Comparing Processing by Intermediate Peers

The final metric to compare these techniques is in terms of the mean number of messages intermediate peers must process to deliver the message. As with all comparisons in our analysis, in the event a network with perfect connectivity can be guaranteed, SD is optimal, requiring no processing by intermediate peers, $IP_{SD} = 0$.

For SR and DR, DR provides a more efficient routing in terms of the average cost incurred by intermediate peers to process messages for any case where $P < 1$. Whenever some peers are publicly reachable, at least some of the messages in a DR system will be routed directly. Similarly, in the worst-case $P = 1$ when the entire overlay consists of mutually unreachable peers, the failed direct attempt will require no processing by intermediate peers, and the fall-back to SR routing results in equivalent processing by intermediate peers. In general, DR always performs at least as well, and for a given

value of P , $\frac{\lg(N)}{P \lg(N)} = \frac{1}{P}$ times more intermediate peers on average will need to process messages for an overlay using SR routing instead of DR routing.

5.2.3.4 Selecting a Technique

If a system does not need to consider non-symmetric connectivity, for example a system running strictly in a data center, within one LAN, etc., using a simple, direct response mechanism (SD) provides the highest performance by any metric. However, in the open Internet, few if any deployments will be free of NATs. As a result, SD is only practical for specialized deployments.

Deciding if SR or DR is most appropriate depends on two factors: determining what metric the system is optimizing for, and determining how many peers are not directly reachable (P).

For systems optimizing for average message count (e.g. systems where transmission cost is high, such as battery powered RF systems and pay-per-bit connections), DR will provide significantly better performance for systems with all but the highest values of P , with the advantage increasing as N increases.

In addition, processing by intermediate peers is always less in a DR than an SR deployment. For systems attempting to minimize the work being performed by intermediate peers (e.g. systems where participation requires incentives, and reducing workload is a priority), DR will always provide equivalent or superior performance to SR.

In deployments optimizing for average propagation time, the case for DR is less clear. Only a small number of peers need to be unreachable for SR to perform better for systems with a small number of peers. Again, as the system gets larger, DR performs better relative to SR, both in terms of the number of peers that can be unreachable and in terms

of impact on RTT. For a large system of 1M, $\lg(N) \approx 20$, so if $\tau = 75\text{ms}$ and $\tau_s = 1000\text{ms}$, RTT for SD is 3s, and ranges from 1.575-4s for DR.

One could reach the conclusion that the best choice is application-specific. For example, in a time-insensitive application such as performing a lookup of a user's location in a VoIP system, ring time will usually exceed the possible 1s cost for a failed connection attempt with DR, and DR might be chosen. For very time-sensitive applications, where the cost of a failed attempt (1s of lost time) is unacceptable, for example control events in a video games, SR might be used. While for very special purpose applications and deployments this is true (and we provide Figures 5.2 and 5.4 for guidance in these situations), we are interested in the development of general-purpose protocols and applications, and in the next section propose an adaptive approach and show the approach is superior for general purpose protocols.

5.3 Adaptive Response (AR)

When developing general purpose libraries and protocols for deployment in multiple network environments, selecting an optimal technique is not possible. In such a case, we instead recommend using an adaptive approach. If a peer finds that most responses are reaching it directly, it requests that DR be used to route responses, while peers that observe that few if any messages reach them directly would request SR be used. We now present a design for such an algorithm we call Adaptive Response (AR).

The Adaptive Response algorithm allows each peer to determine experimentally (as best as possible) whether or not it believes itself to be reachable directly from the outside. If the peer believes it is directly reachable, it requests that DR response routing is used to route its responses. As always, DR will resort to an SR technique if a direct response fails, ensuring reachability. If the peer believes it is not reachable directly, it requests SR

be used instead, eliminating the cost of attempting a direct response. To support this technique, a flag is used by the requesting peer to indicate a preferred mechanism to use when sending it a reply, and by the responding peer to indicate which technique was used to send a response.

A similar approach was informally considered in [60] as a possible use for the NAT discovery algorithm presented there, but no actual mechanism was proposed or analyzed, and no applications discussed.

When connecting to the network, the requester begins by assuming they are reachable, and requests that responses are sent directly using DR. The responder attempts to honor this request by first sending the response directly, but if this fails, reverts to SR (essentially following the rules for DR). In either case, the responder marks the response indicating which mechanism (SR or DR) was ultimately used to transmit the response. After some number of requests, the sender evaluates the percentage that have succeeded to determine if they should continue using DR or request that future responses be transmitted using SR.

Depending on the importance of minimizing message count, these “probe” messages could be attempted as part of ordinary searches as the application runs, or could be performed when the peer joins the overlay. Alternately, the peer could use the techniques described in [60] to determine if it is reachable.

5.3.1 Evaluating Adaptive Response Analytically

To explore the performance of AR, we first evaluate the worst-case performance using the model from Section 5.2.2. As with that model, we will use worst-case values for the routing costs of the overlay ($\lg(N)$) in the case that messages are routed recursively. In

Section 5.3.2, we present the results of a simulation which calculates the actual (average) cost of routing for an overlay.

While a real system will not be 100% correct in determining reachability, if we imagine a stable system using this approach, having detected perfectly or having been provided the information via an oracle, the performance is determined strictly by the fraction of reachable peers. For those that are reachable, the performance of a Perfect Adaptive Response system (ARP) will incur the cost of a successful direct response for reachable peers and the cost of SR for unreachable peers. As with DR, the cost for intermediate peers to route a response is dependent on P . There is no processing required by intermediate peers when direct routing succeeds. Following the analytical model defined in Section 5.2.2, the average message counts, response time (again, including setup time for DTLS connections), and processing by intermediate peers is:

$$\begin{aligned} MC_{ARP} &= P (\lg(N)) + (1 - P) 8 \\ T_{ARP} &= P (\tau \lg(N)) + (1 - P)(8 \tau) \\ IP_{ARP} &= P (\lg(N) - 2) \end{aligned}$$

A real system is unlikely to have perfect visibility. Some peers will incorrectly determine or will not yet have accurately determined if they are reachable. To model such a scenario, we introduce C , the probability that the peer has correctly determined if it is unreachable. For simplicity, we further assume that the peers guess incorrectly proportionate to the fraction that are unreachable. In other words, if 90% of the peers are reachable, than 90% of the incorrect guesses will be reachable peers believing they are unreachable. We will augment this simplistic model with simulation in Section 5.3.2.

Given these assumptions, and assuming the system has at least $N > 2^8$ peers, we can calculate the expected mean message count and propagation time for a more realistic Adaptive Response system AR as follows. First we define MC_{err} and T_{err} as the values when the peer guesses incorrectly:

$$\begin{aligned} MC_{err} &= P (\lg(N) + 1) + (1 - P) \lg(N) \\ T_{err} &= P (\tau (\lg(N)) + \tau_S) + (1 - P)(\tau \lg(N)) \end{aligned}$$

When the peer guesses correctly, the values for ARP are used. We can then define the combined mean message counts, response transit times and intermediate processing for a more realistic system MC_{AR} , T_{AR} and IP_{AR} as:

$$\begin{aligned} MC_{AR} &= C MC_{ARP} + (1 - C) MC_{err} \\ T_{AR} &= C T_{ARP} + (1 - C) T_{err} \\ IP_{AR} &= C (P (\lg(N) - 2)) + (1 - C)(\lg(N)) \end{aligned}$$

5.3.1.1 Analytical Results for Adaptive Response

In Figures 5.5 and 5.6, we evaluate the performance of the AR routing algorithm in three forms for overlays of size $N = 10K$ and $N = 1M$ and various values of P : theoretically perfect (ARP), as well as 90% and 75% correct at determining if the peer is unreachable (AR90 and AR75, respectively).

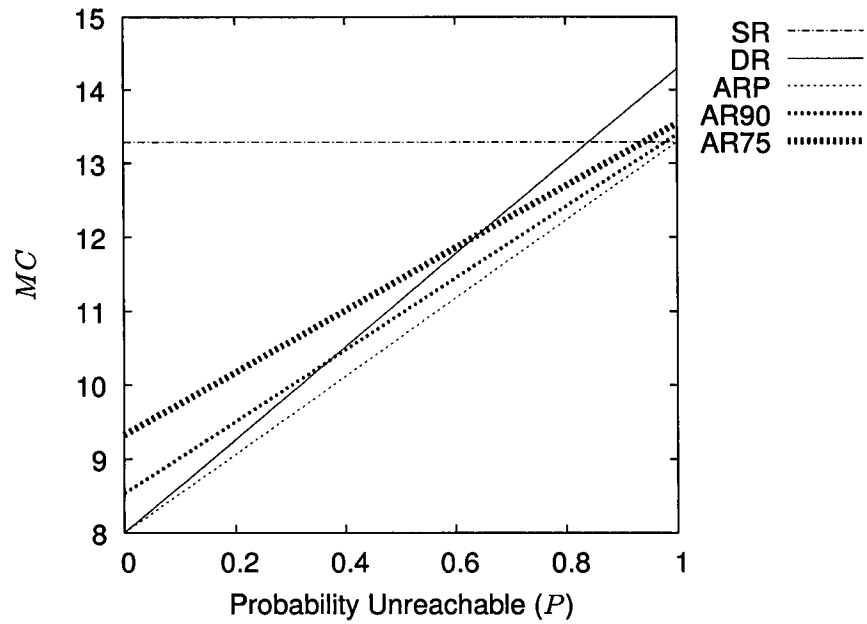


Figure 5.5: MC vs P ($N = 10K$)

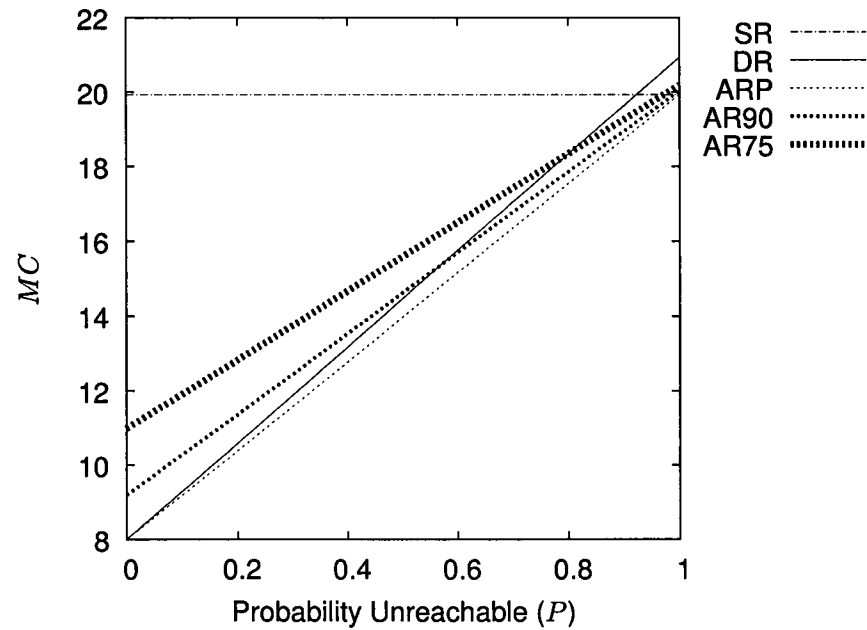


Figure 5.6: MC vs. P ($N = 1M$)

As with DR, all variations of the AR algorithm outperform SR for nearly all values of P except overlays where virtually all peers are unreachable. ARP, unsurprisingly, outperforms or equals both DR and SR for all overlays. However, even in the worst environment, AR75, where only 75% of peers have determined their reachability correctly, AR improves performance relative to DR for high values of P , while incurring little additional cost for low values of P . As expected, the performance of the various AR algorithms, like the DR algorithm, improves compared to SR as the size of the overlay N increases.

Figure 5.7 presents the results slightly differently. Again, we plot for overlays of size $N = 10K$ and $N = 1M$, but with a fixed $P = .5$, a value of P for which DR outperforms SR. Now, we plot mean message count against accuracy C . We plot results for values of C from .5 (the expected value for simple guessing) to 1, indicating perfect accuracy. Figure 5.8 is similar, but for a fixed $P = .9$, a value of P for which the performance of DR and SR are more similar.

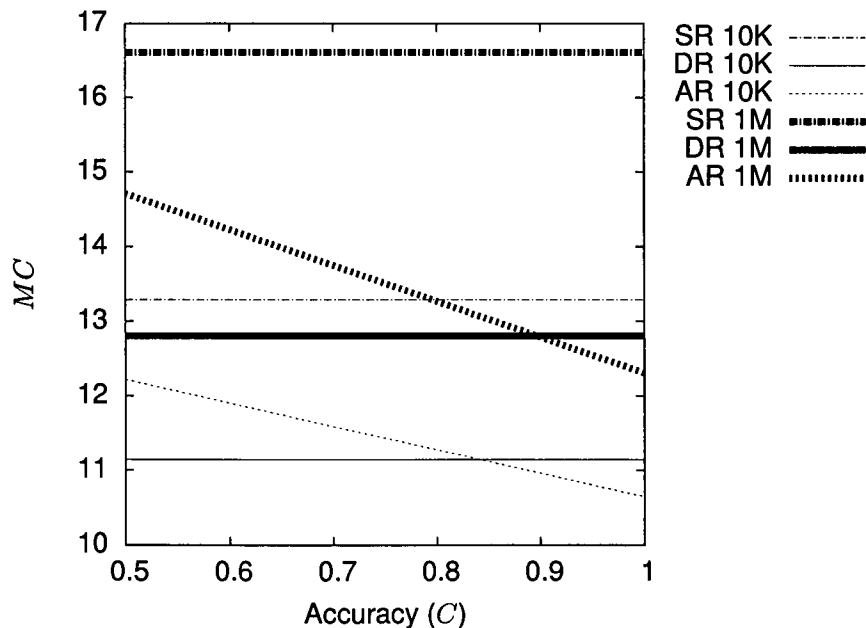


Figure 5.7: MC vs C ($P = .5$)

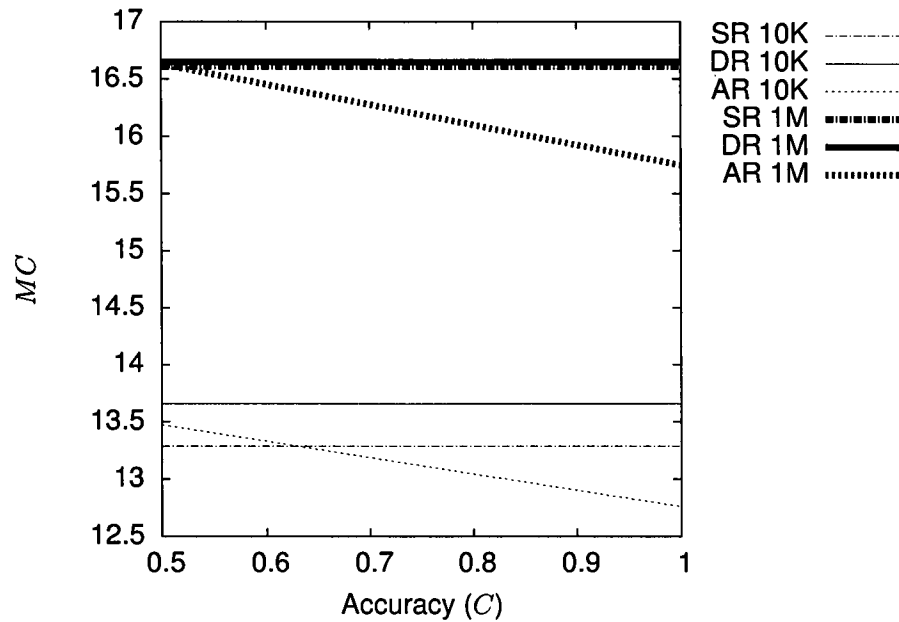
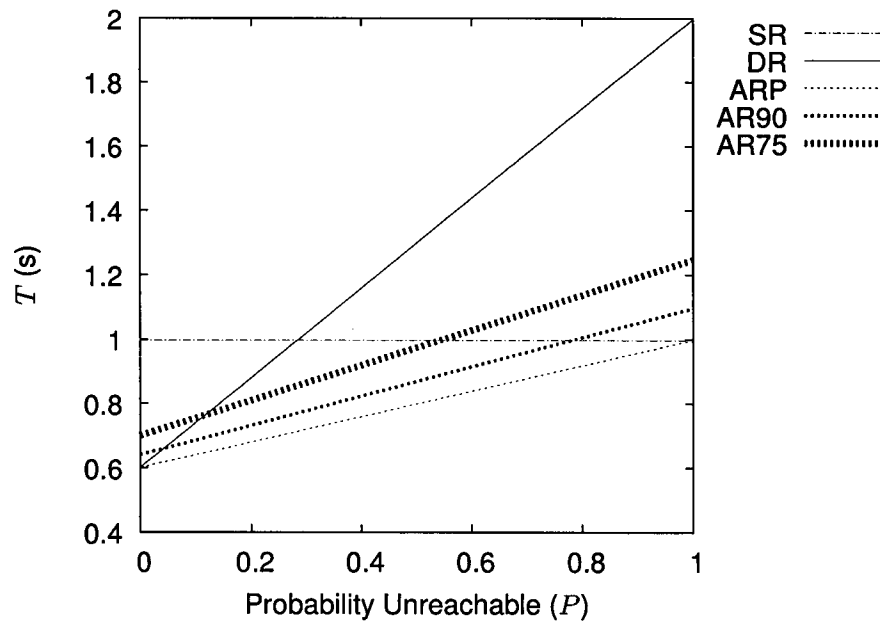
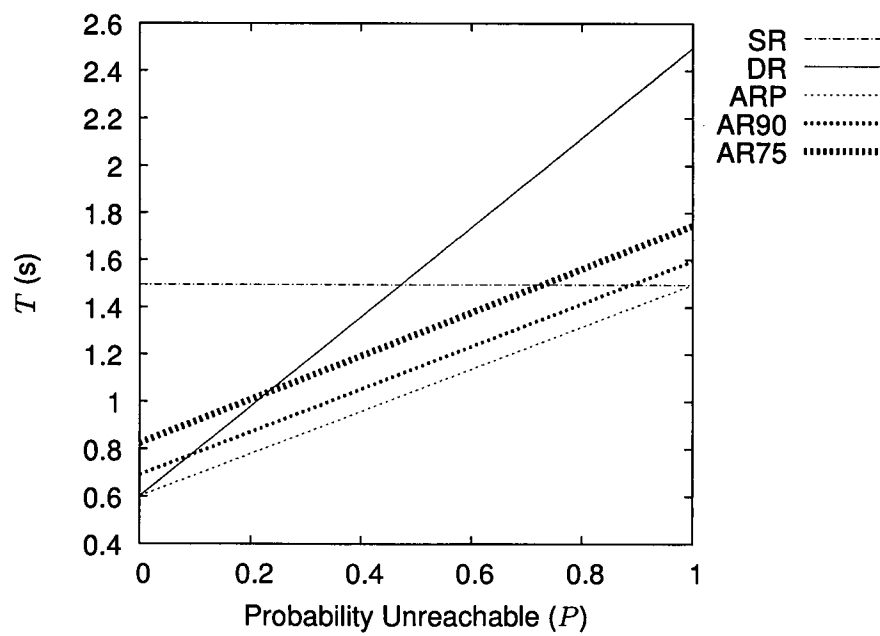


Figure 5.8: MC vs C ($P = .9$)

Here we see that, as expected, as the accuracy of determining if the peer is reachable increases, the overall message count decreases. When many peers are directly reachable ($P = .5$), AR significantly outperforms SR even when effectively guessing ($C = .5$), and at 70% accuracy offers performance very close to DR. When many peers are unreachable ($P = .9$), SR and DR are flipped, with SR performing better, but AR's performance advantage is even more dramatic. When $N = 1M$, AR outperforms both SR and DR in terms of mean message count for nearly any value of C . The advantages of being able to eliminate both the unneeded use of SR when peers are directly reachable and the overhead of trying direct response when they are not results in reduced message counts over SR (and over DR for environments with many NATs) in many deployment scenarios.

Looking at average propagation time, we again compare ARP, AR90 and AR75 to SR and DR in Figures 5.9 and 5.10 for overlays of size $N = 10K$ and $N = 1M$.

Figure 5.9: T vs. P ($N = 10K$)Figure 5.10: T vs. P ($N = 1M$)

Here the results are more dramatic. Except for overlays where virtually all peers are reachable (very small P), AR significantly outperform DR. Like DR, AR outperform SR for small P , but most importantly, AR offers a significant performance improvement over DR for larger values of P . AR offers average propagation times better than either SR or DR for values of P less than approximately .5 for $N = 10K$ and .7 for $N = 1M$. Just as significantly, even for very large P , AR75 increases response propagation time by only 25% for $N = 10K$ and less than 20% for $N = 1M$.

Viewing the performance relative to accuracy (Figures 5.11 and 5.12), one can see that for mean response time, where SR offers better performance than DR, AR offers a dramatic improvement over DR, approaching and eventually exceeding the performance of SR as accuracy improves.

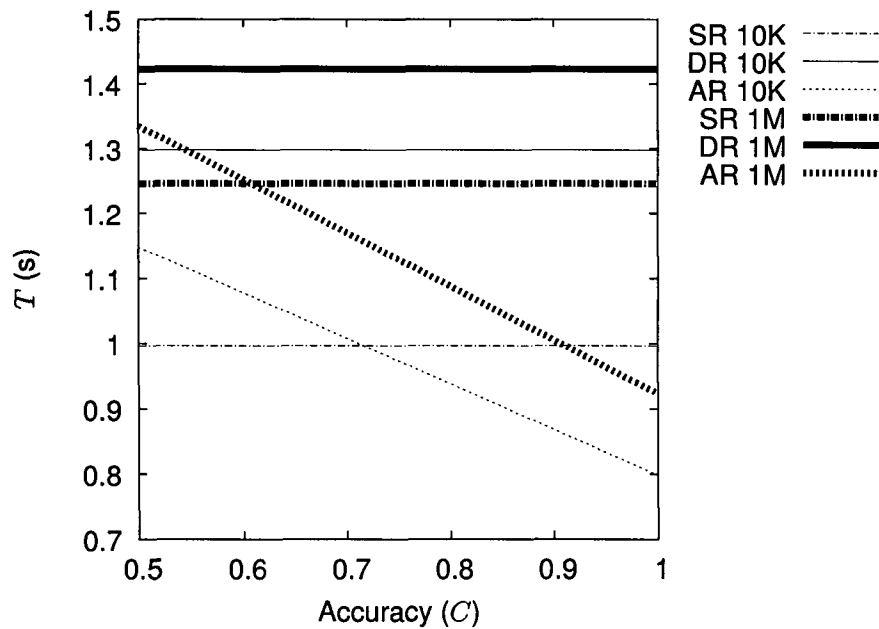
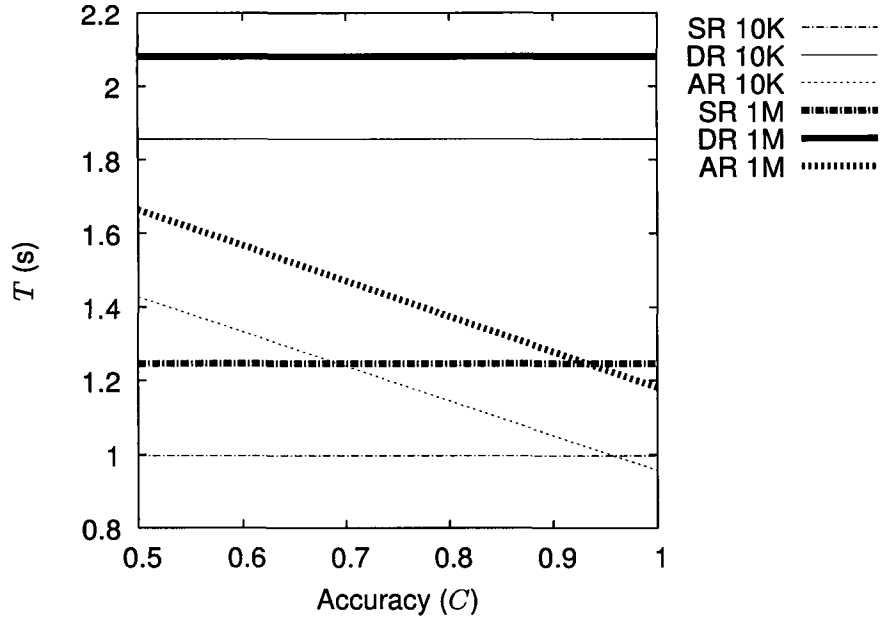


Figure 5.11: T vs C ($P = .5$)

Figure 5.12: T vs C ($P = .9$)

5.3.2 Evaluating Adaptive Response through Simulation

While our analytical model provides information to compare the techniques when worst-case routing cost is considered, a proper evaluation of the techniques needs to provide more information about average case performance. Our analytical model could certainly be extended for average cost routing in a DHT ($\frac{\lg(n)}{2}$), but such a model would still fail to capture the effects of churn. To obtain this information, we developed a simulator to evaluate the techniques in the average case, and in the presence of churn. The simulator uses Chord as the underlying DHT algorithm. For the simulation, we made a number of assumptions:

- We simulate for 24 hours in one minute intervals. We simulate using 10000 peers in the overlay.

- We assume that some number of peers are behind NATs. We performed simulations for 10%, 30%, 60%, 90% and 95% of the peers behind NATs.
- We performed simulation runs for a number of different average peer lifetimes (churn): 30 minutes, 1 hour, 2 hours, 4 hours, 6 hours, 8 hours, 10 hours and 12 hours.
- We assume each peer executes an average of two searches each minute. This number was calculated by assuming two phone calls per hour, 20 instant messages per hour, queries for the status of 40 buddies, and an overlay stability operation once per minute. While these numbers are somewhat arbitrary, as actual call rate information and IM information is often a closely guarded secret, from anecdotal evidence these are reasonable values. In the future we would like to obtain more realistic data. As a result, we perform $2 \times N$ searches each time cycle, however the selection of the particular peers performing the call is random for each search.
- To simulate churn, we calculate a probability that any given peer leaves in a simulation step based on the average lifetime of the peer for that simulation run. We use this probability to randomly select a corresponding number of peers each interval. These peers leave and are immediately replaced, resulting in a constant number of peers at all time, $N = 10000$. For simulator efficiency reasons, the location in the overlay of the new peer is identical to the old peer, but the determination if the peer is actually behind a NAT, and any information it has determined about that state is reset to ensure appropriate impact on the performance of the AR and DR algorithms. As will call distributions, information on real-world peer lifetimes is difficult to come by, and one future goal of our research is to obtain better information as to average peer lifetime in a communications system.
- For each peer using AR, we assume that they will try the first five messages as DR to determine if they are reachable, and pay the corresponding cost in terms of

having to timeout and retransmit as SR if they are behind a NAT. Even if the first message fails, they will continue to perform five tests, as in a real deployment other factors (e.g. packet loss) may be at work, and a peer deciding if it is behind a NAT when the first message fails is unreasonable. Once a peer determines it is behind a NAT, it will continue to use SR for its lifetime.

The results of our simulation are presented below, and show that AR performs very well in the average case, in all deployments except those where nearly all peers are behind NATs and those displaying a great deal of churn.

Because of the nature of the simulation, there is a high degree of variability between call hops and routing times. When being routed back recursively, responses may take anywhere from 1 to $\lg(N)$ hops to traverse the overlay. As with the analytical model above, we assume a 1s timeout when an attempt to route directly fails, and a time of 75ms for each hop. As a result, time for DR and AR can range from 75ms to $1000\text{ms} + 75\text{ms} \times h$, where h is the number of hops. Due to this, we have not plotted standard deviations on the graphs for clarity, but full results including standard deviations are presented in Appendix C.

As one would expect, the deviations are large but constant for symmetric recursive (illustrating that messages take from 1 to $\lg(N)$ hops, but do so uniformly and independently of churn or the presence of NATs). However for AR, the deviations are minimized when there are few NATs or when there are many NATs, and are most variable with an intermediate number, when some messages are routed using each technique. DR similarly shows the least variability at the extremes of NAT density, when it either always succeeds or always fails and must fall back to SR.

5.3.2.1 Simulating Hop Counts

In Figure 5.13, we present the mean hop counts measured for each routing technique. For clarity, runs for each technique (SR, AR, DR) use the same color, regardless of NAT density, and runs with the same NAT density uses the same symbol, regardless of technique. SR provides identical results for any NAT density, as the symmetric recursive mechanism is unaffected by NATs, and is plotted only once. For DR and AR, the number following each technique indicates the NAT density. For example, AR10 is the result for AR when an average of 10% of the peers are behind NATs. **Note that this is different from the graphs for the analytical section, where the number in the label indicated the accuracy C of determining the peer was behind a NAT.** In this case, the peers in the simulation determine their NAT state by sending messages. As a result, longer lived peers are more likely to have accurate information, while peers with short lifetimes are more likely to have incorrect information.

We can see that for hop count AR outperforms SR in all cases, with the performance improvement being more marked when fewer peers are behind NATs. DR also tends to outperform SR when there are few NATs, but for higher NAT densities, the cost of the additional message sent in failed attempts to route directly adds up, and the average cost for DR90 and DR95 exceeds the cost of SR. AR reduces this cost, as many of the peers behind NATs learn and send messages using SR, and the few that learn they are not behind NATs gain an advantage, resulting in slightly more efficient average routing cost, even with many NATs.

While the effect is subtle, one can also see that ARs performance decreases when churn is higher (peer lifetime lower). This is expected, as short-lived peers spend more of that lifetime testing if they are behind a NAT, and thus effectively performing as DR peers, and less of the lifetime with an accurate picture of their NAT situation.

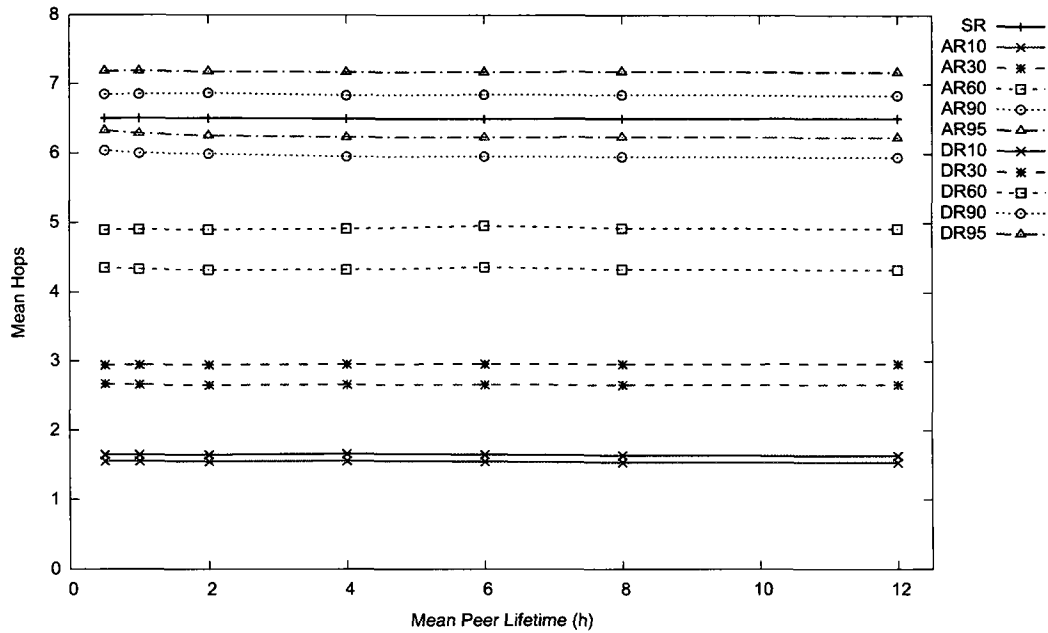


Figure 5.13: Simulation Results, Hops

5.3.2.2 Simulation Routing Times

While the hop count results are similar to the worst-case results from our analytical model, the results for time show that in the average case, the performance of AR is typically significantly better than SR and DR. In Figure 5.14 we present the results for each technique. In general, we see that AR techniques perform better than SR, as do DR10 and DR30, however when the NAT density exceeds 60%, DR's performance becomes considerably worse, as a result of frequently paying the assumed 1s timeout.

As a result, in Figure 5.15 we exclude these three results (DR60, DR90, DR95), allowing us to more closely examine the results for the remaining routing techniques. As with the number of hops above, we see that performance worsens as the NAT density increases for both AR and DR, and that the performance of AR worsens for higher churn. Despite this, AR always outperforms DR for the same NAT density, and in all but the

worst cases—NAT densities above 90% and peer lifetimes of under 2 hours—significantly outperforms SR.

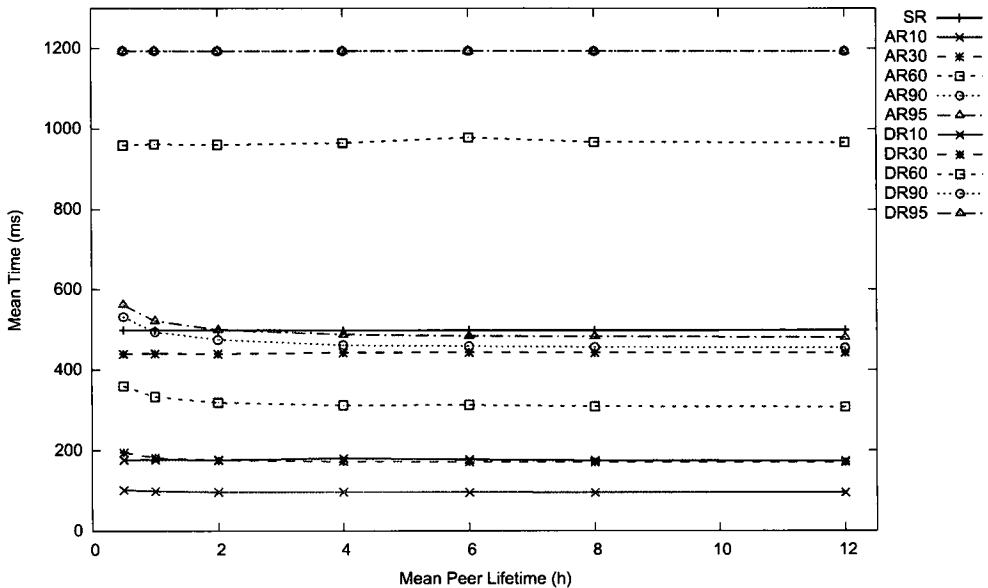


Figure 5.14: Simulation Results, Time

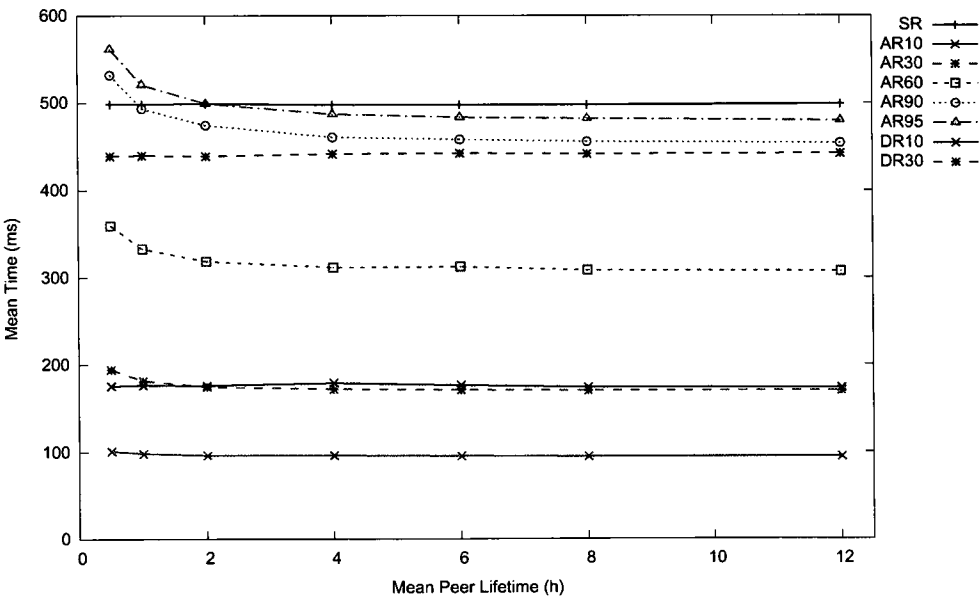


Figure 5.15: Simulation Results, Time (Detail)

5.3.3 Results of the Analysis

In conclusion, the environment in which a P2P application is deployed has a tremendous impact on which response routing technique is most appropriate. The portion of peers with asymmetric connectivity, most frequently caused by NATs, heavily impacts which techniques offer the best performance. Given a-prior knowledge of the fraction of peers reachable in a specific deployment and an understanding of what it is that the application is optimizing for (message count or propagation time), one can use Figures 5.2 and 5.4 to determine whether direct response or symmetric recursive routing will provide better results.

However in the majority of cases when the network environment cannot be accurately characterized in advance, when multiple metrics need to be optimized, or for general purpose libraries or protocols, we have shown that the Adaptive Response algorithm presented in Section 5.3 offers an attractive middle ground, outperforming both DR and SR in all but the worst network environments. When the mean hop counts and propagation times for responses are considered, AR offers superior performance in nearly all network situations. While worst-case time cost of AR may exceed SR, as we have seen in the analytical model, simulations indicate that if the goal is to optimize mean performance over all environments, AR is the superior choice.

5.4 Other Concerns and Topics in P2P Routing

5.4.1 DoS Attack and Parallelization Risks

The various recursive routing mechanisms allow for peers behind NATs to participate in the overlay, but introduces the hazard of DoS amplification attacks. Recursive algorithms

allow peers to serve as surrogates (although not amplifiers) in DoS attacks against a particular target. In Figure 5.16, we assume an attacker A can send requests to intermediate peers I searching for target peer T . With iterative routing, the intermediate peer I replies with a closer peer to use to attempt to reach T , but the attacker must directly send to T to attack it. In the recursive approach, attacker A can send to many intermediate peers I , all of which relay the message to T . The intermediate peers essentially serve as DoS surrogates for A and may assist in obfuscating the source of the attack. The number of peers A can use as surrogates is limited only by the number of intermediate peers I that are reachable from A . If we take Chord as an example, each peer has $\lg(m)$ immediate neighbors, where $m = 160$ is the size of the address space. If the attacker sends to every neighbor, $\lg(m)$ messages will reach the target. (Ironically, while this capability provides a way for an attacker to obfuscate the sender, sending parallel messages to a single peer using different paths also can be used as a defense against routing attacks, since if a bad peer on path attempts to drop the message, a parallel message will likely arrive [93]).

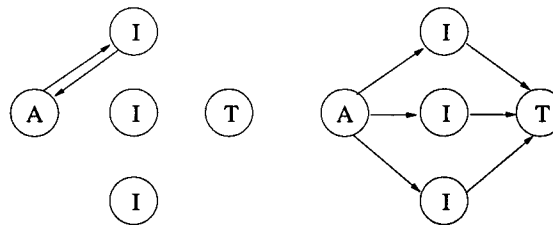


Figure 5.16: Surrogate Attackers in a Recursive Approach

Another problem with recursive algorithms is that combining them with parallelization can further aggravate the DoS Attack, turning intermediate peers into attack amplifiers as well as surrogates. Some iterative DHT algorithms such as EpiChord or Kademlia improve average case performance using parallel search, where each initiator sends the request to k intermediate peers, and the algorithm parallelizes by sending the request to an average of p neighbors at each level of the search. Combining parallelization with a recursive algorithm creates an explosive DoS multiplication attack risk. In a DHT where

queries have a worst-case depth of $\lg(N)$ (where N is the number of peers in the system), and all messages are delivered to the target, the number of messages M_T arriving at the target T can in the worst-case be $M_T = k p^{\lg(N)}$. For even small values of k and small branching factors such as the $p = 3$ used by EpiChord, taking a recursive approach is disastrous. If we assume $N = 100,000$, and $k = 3$, (not even a particularly large DHT), we have $M_T \approx 3 \cdot 3^{16.6} \approx 250$ million. Techniques can be applied to detect and drop duplicate messages, but the target may still receive N messages, far more than a direct attack could produce.

In a generic DHT without certified identity, the fact that intermediate peers have relayed the messages can make determining who sent the message impossible, leaving no simple way to find the attacker. Signing of the overlay messages, including signing each entry in the “Via-List” can prevent obfuscation, but cannot prevent the attack completely. In essence, the target of the attack would be able to determine who had attacked them, but not prevent the attack. Additionally, such an approach is computationally expensive, requiring that certificates are checked far more frequently than we propose in our system. Exploring how distributed reputation services and certificates might be combined to reduce the impact of this problem is an area of future work.

At present, the solutions being utilized for P2PSIP use a recursive routing technique, but with signed messages and no parallelization, the total number of messages cannot be increased beyond those the sender could create themselves, and signed messages eliminate any ability for the sender to disguise the source of the messages. In such a system, the capability to attack other peers via the overlay degenerates to the same capability as directly attacking. Clearly, in unsecured systems (or more precisely systems where the identity of the sender is not verified) or systems using parallelization to increase lookup performance, this is a serious concern that must be considered for recursive routing approaches.

5.4.2 A (seemingly) Dead End: Social Network Routing

A very interesting addition to routing that we explored early in our research is to augment traditional DHT based routing table entries with social links. The basic idea is to take advantage of the availability of the buddy list or phone directory on a communications device to provide improved routing.

Six degrees of separation, a common theory among social scientists, postulates that all individuals are separated by 6 connections. While buddy networks are unlikely to be rich enough to capture this property (every person one knows is not in their buddy list), we felt that an opportunity for significant performance improvement in routing by using social networking. Routing techniques that capitalize on this additional information could greatly increase the efficiency of searches by searching for users via buddies, rather than the network as a whole. Since many people have friends with similar interests, statistically speaking people they want to contact are more likely to be buddies of their buddies than buddies of random peers. To contact another computer scientist, for example, it is likely someone else in the computer science department may know them. By searching their buddies' buddy lists, could we produce a more efficient technique for routing the call than a traditional DHT which does not capture this additional information?

One possible approach is to simply use the buddies as additional fingers. The creators of SPROUT [62] first proposed such a system as a mechanism for getting "more trusted" results in a file sharing system, and we worked to extend this work to apply to communications systems. We adapted our simulator to use the SPROUT technique, which simply uses buddies as additional fingers, but while it is possible that a remote party could be reached easily in such a way, the fact that the fingers are randomly spaced means the impact of these new fingers proved statistically no better than simply adding more fingers.

We attempted a more advanced technique based on social networking [3, 47, 67, 96,

99], but unfortunately we ran into two (currently seeming) insurmountable obstacles to evaluating such an approach. To make such a model meaningful, this approach requires several things we were not able to obtain. We would need good information on the distribution of most people's buddy lists, and would need to be able to determine what the average overlap between friends is. Essentially, we would need access to buddy list information (anonymized data would be sufficient to construct such a model), but this data is viewed as extremely sensitive by the organizations that have it.

Additionally, information that would be difficult to obtain in any circumstances is required. Having a metric of which buddies are a good match to you would be the first requirement. It is likely that a close friend and a parent, while both in a user's buddy list, would be of radically differing value in reaching another friend, for example. While one could imagine sharing a hashed list of buddies with buddies, and determining which have the most overlap, some users would be unlikely to share buddy lists, even in a hashed form. In any case, without out-of-band information about who one is trying to reach (a friend, a work colleague, a family member), and similar information about the current set of friends, choosing which friends are best to try is a very difficult problem. Except for extremely large networks, $\lg(N)$ search begins to quickly look more efficient when the cost of dead end searches is so high.

In summary, while a considerable amount of time was spent on this effort, at present we cannot say that it has yielded any useful results. We will continue to explore possibilities for using social networking to augment search, when more practical solutions arise.

5.5 Summary

In this section we have provided a detailed discussion of the appropriate routing technique to use for a communications system. We first explored the differences in overall routing techniques, showing that iterative and strict-forward techniques are unattractive for a number of reasons, and limiting the pool of options to symmetric recursive and direct response techniques. We then presented an analytical model comparing the expected worst-case performance of these approaches, and provided guidance on when each is most appropriate. Finally, we presented an adaptive technique, AR, which optimizes between SR and DR techniques, and showed analytically and through simulation that for general deployments, AR provides superior average-case mean performance for both hop count and response propagation time in the majority network environments.

Chapter 6

Conclusion

In this chapter we outline the contributions presented in each chapter, and discuss future directions for our work.

6.1 Results and Contributions

The design of a P2P system optimized for interpersonal communications differs significantly from a system designed for applications such as file sharing. In Chapter 3, we explore the unique requirements imposed by a P2P communications system, and their impact on system design. We explain why the simplistic mechanisms for IP-based Peer-ID assignment are vulnerable to attack in networks with NATs, and explore the problems of asserting identity when there is no central server available at run-time. To address these issues, we present an enrollment-time certificate authority model, and show how it can be used to secure both user and DHT operations. We further explain how this approach improves security and strengthens identity assertions without requiring the server to be contacted at peer-join or call time.

In Chapter 4 and Appendices A and B, we present two complete designs for the Internet protocol needed for P2PSIP:

- SOSIMPLE/dSIP provides a complete P2P-over-SIP solution, defining the mechanisms and SIP messages required for a peer to perform both P2P and SIP operations using a single protocol, SIP.
- RELOAD is a SIP-using-P2P binary protocol designed to encode P2P messages. We illustrate how this protocol is used in tandem with SIP to build a P2PSIP system.

This work led to the development of the research area of P2PSIP, including a working group at the IETF, now extending a descendent of RELOAD. Additionally, this work provided the impetus for a number of conferences and publications which address or document the work performed in P2PSIP.

As a result of the different requirements on P2P imposed by communications, and because of a need to operate in the open Internet where NATs are common, we show in Chapter 5 that many commonly proposed routing techniques, including iterative and forward-only, are sub-optimal. After narrowing the viable routing techniques to direct response and symmetric recursive, we introduce an analytical model showing that the NAT density in the network environment is the key deciding factor in determining which technique is optimal for a given deployment. We discuss which approach is most effective in different network environments, and provide guidance for determining which technique should be used by applications. Additionally, we show that neither technique is optimal for all network environments, and that each strongly favors optimization of a particular parameter, either message count or latency.

Motivated by the strong impact of NAT density on optimal technique selection, we go on in Chapter 5 to present an adaptive response mechanism, AR. AR infers NAT

connectivity and adapts accordingly, switching dynamically between symmetric recursive or direct response routing. Using both an analytical model and simulation, we show that in all but the worst environments—those with high churn and nearly all peers behind NATs—our technique offers a significant performance advantage in terms of both mean hop count and mean response propagation time. Additionally, we show that AR's performance is less sensitive to NAT density, making it well suited to general-purpose protocol design.

6.2 Future Work

While this work has made a large impact in the P2P communications area, there remain interesting questions to answer and further work to be done. Areas for future work include:

- We have explored the use of P2P systems for interpersonal communications, and developed protocols that are optimized for this type of P2P application, however many other applications of P2P beyond interpersonal communications and file sharing exist. In particular, the use of P2P to improve the performance of streaming media has been a topic of considerable interest in the literature, but to date no standard protocol for messages between peers for such a system exists. While many aspects of the protocol implications of this application are poorly understood, and would certainly impose new requirements, we feel our work could be extended to help design a protocol for use in these scenarios.
- While our work has shown adaptive response to outperform other response techniques in most environments, the impact of presence on the speed with which a peer can determine its NAT status has not been fully explored. As most users have a number of buddies, and these are contacted immediately upon joining an overlay,

this initial search may prove to be sufficient to quickly identify the peer's NAT status, further improving the performance of AR. We would like to further explore this question, either through simulation or direct implementation of the AR technique in a P2PSIP client.

- On the other hand, our work has currently explored only simple NAT scenarios. In our simulation a NAT, once characterized, does not change behavior, NAT behavior is relatively simplistic and uniform across NATs, and peers are behind only a single NAT. While these are reasonable assumptions for most networks, in the open Internet some more unusual cases exist. We would like to further explore these considerations in the future.
- At present, our models and simulations evaluating the AR algorithm use a fixed, average value for the time required to traverse a link. We would like to obtain information about the distribution of peers behind various types of access links (mobile, various types of home broadband, enterprise, etc.), and the average propagation time for messages using each type of link. Given this information, we would like to include these characteristic in our simulation in an effort to better take into account the performance of AR in the open Internet. This result could also be achieved by instrumenting and measuring the performance of an actual deployed P2PSIP system.
- Finally, if we are able to obtain better information on average peer lifetime, on the number of peers located behind NATs and the NAT type, or on average call patterns in a P2P communications system, we will be better able to refine our model. Such additional information would allow us to better determine if AR truly outperforms other routing techniques in actual deployments.

6.3 Conclusion

We have shown the need for a standardized, decentralized system for interpersonal communications, and have presented our solution, P2PSIP. We have designed two protocols and corresponding systems to implement P2PSIP. We have shown how to secure these protocols with a minimal amount of interaction with a central server, offering both user and DHT-level security. We have shown that the need for security and the presence of NATs in the open Internet dictates routing technique, and have introduced a novel adaptive technique, Adaptive Response. We have shown through analytical models and simulation that Adaptive Response is better suited than existing approaches for environments requiring security and where NAT density cannot be determined in advance.

Appendix A

dSIP Protocol Specification

A.1 Message Syntax

Much of this text appears in [23], and as such is a snapshot of the work on dSIP at that date. Protocol development on dSIP was not required for further research, and has not been continued, although our work has directed the work in [52] and others are continuing to use our research to develop that protocol in the IETF.

This section provides normative text explaining the syntax of the extensions we use for SIP messages.

A.1.1 Option Tags

We create a new option tag “dht” as described in RFC 3261. This option tag indicates support for DHT based P2PSIP. Peers **MUST** include a Require and Supported header with the option tag dht for all messages that are intended to be processed using dSIP or include P2P extensions. Clients supporting P2P and contacting another SIP entity using

a non-P2P mechanism for a transaction that may or may later be P2P SHOULD include a Supported header with dht. For a typical session establishment the search within the DHT MUST specify Require dht, whereas the actual contact with the resource's UA SHOULD include a Supported header with dht but SHOULD NOT include a Require header with dht.

A.1.2 Hash Algorithms and Identifiers

All IDs used for an overlay must be calculated using the same algorithm. Implementations MUST support the SHA-1 algorithm, which produces a 160 bit hash value. The hash algorithm used is specified in the DHT-PeerID header, described below. An implementation MAY rely on a secret initialization vector, key, or other shared secret to use the identifier as an HMAC [56] such that no peer may join the overlay without knowledge of the shared secret, however this technique by itself does not protect the overlay against replay attacks.

Both Peer-IDs and Resource-IDs MUST have the same range of values (map to the same space). Formally:

P2PID = token

When using SHA-1:

P2PID = 40LHEX

A.1.2.1 Peer-IDs

The particular DHT algorithm being used MAY specify an alternate mechanism for determining Peer-ID. Similarly, some security models may assign Peer-IDs from a central

authority. In the event that neither of these mechanisms are being used, the Peer-ID MUST be formed by taking the IP address of the peer, without the colon or port, and with no leading zeros, and hashing this string with the hash algorithm. Then the least significant sixteen bits of the hash are replaced by the port used by the peer. For peers behind a NAT participating in an overlay on the public Internet, they must identify their address on the public Internet through a protocol such as STUN and use this address for their Peer-ID.

The string hashed to obtain the Peer-ID is formally defined below as `ipaddress`.

`ipaddress` = `IPV4address` / `IPv6reference`

`PeerID` is formally defined as:

`PeerID` = `P2PID`

A.1.2.2 Resource-IDs and the Replication

Resource-IDs MUST be formed by hashing the resource URI after converting it to canonical form. To do that, all URI parameters MUST be removed (including the user-param) except for the replica URI parameter, Any escaped characters MUST be converted to their unescaped form. Formally:

`ResourceID` = `P2PID`

A.1.3 P2PSIP URIs

Because hashing URIs to produce identifiers is a non-trivial cost, dSIP messages are constructed including these values already calculated. This is strictly as a courtesy to

peers processing messages for this peer, as it prevents them from having to hash the URI again before routing. Identifiers provided in a message are a courtesy only and **MUST NOT** be used when making any changes to the data stored in an overlay, as they may be spoofed or incorrect. If the hash parameter is used incorrectly for routing, this only affects the transmitting peer's user. If it is used to insert or modify stored information, it can affect the system's integrity. Peers **MUST** verify the hash of all URIs before making changes that affect the overlay.

A.1.3.1 Peer URIs

A P2PSIP peer is represented by constructing a SIP-URI (or SIPS-URI) with the keyword "peer" or a short form of "P" for the userinfo portion. The URI parameter "peer-ID", or the short form "pID" **MUST** be used.

```
PeerURI      = ("peer@" / "P@") hostport ";" PeerID-Param ";"
               uri-parameters
```

Formally, the peerID uri-parameter is defined as type other-param from RFC 3261 with a pname of "peerID" or "pID" for short form, and a pvalue which is of type PeerID. A peer receiving a PeerURI **MUST** verify the hash value of the PeerID-Param before using it to update its routing table.

```
PeerID-Param  = ("peer-ID" / "pID") EQUAL PeerID
```

For search operations, where an identifier is being searched for, but the host responsible for that identifier is unknown, hostport **MUST** be set to "0.0.0.0". All non-search operations **MUST** specify a valid hostport.

P2P Peer URIs MUST NOT include the resource-ID URI parameter (below), as it is intended to define information about resources that are stored in the overlay, not information about the peers making up the overlay. P2P Peer URIs used in name-addr SHOULD NOT include any display-name information, and peers receiving name-addr for peers with display-name information MUST ignore the information.

Examples, using a shortened hash for clarity: The URI for a peer using the SHA-1 hash algorithm, with hashed ID ed57487add matching an IP address 10.6.5.5 used in a To header. Uses the short forms:

To: <sip:P@10.6.5.5;pID=ed57487add>

The URI for a peer using the SHA-1 hash algorithm, with hashed ID ed57487add matching an IP address 10.6.5.5 used in a To header. Uses the long forms:

To: <sip:peer@10.6.5.5;peer-ID=ed57487add>

A.1.3.2 Resource URIs and the resource-ID URI Parameter

Resource URIs are no different for P2PSIP resources than for non-P2P SIP applications. However, because calculating the Resource-ID is a significant expense, the optional URI parameter resource-ID=*Resource-ID* or the short form rID=*Resource-ID* SHOULD be provided. This parameter is a courtesy only and MUST NOT be used when making any changes to the data stored in an overlay without being recalculated, as it may be spoofed or incorrect. The resource-ID URI parameter is of type other-param as defined in RFC 3261.

resourceID-param = ("resource-ID" \ "rID") EQUAL ResourceID

P2P Resource URIs MUST NOT include the PeerID-Param URI parameter, because this indicates that the target of the URI is a peer. P2P Resource URIs MAY include other user-parameters such as user=phone.

Examples (again using shortened hashes for clarity): The URI for a user resource with username bob@p2psip.org using the SHA-1 hash algorithm, with hashed Resource-ID 723fedaab1. The optional resource-ID URI parameter is included, using the long form:

```
sip:bob@p2psip.org;resource-ID=723fedaab1
```

The URI for a user resource with username bob@p2psip.org using the SHA-1 hash algorithm, with hashed Resource-ID 723fedaab1. The optional resource-ID URI parameter is included, using the short form:

```
sip:bob@p2psip.org;rID=723fedaab1
```

The URI, used in a To header for user Alice White, with username alice@p2psip.org. This example omits the optional resource-ID URI parameter:

```
To: "Alice White" <sip:alice@p2psip.org>
```

A.1.4 The DHT-PeerID Header and Overlay Parameters

We introduce a new SIP header called the DHT-PeerID header. This header is used to express the Peer-ID of the sending peer as well as to identify the name and parameters of the overlay. The format of the DHT-PeerID header is as follows:

```
DHT-PeerID = "DHT-PeerID" HCOLON PeerURI SEMI algorithm SEMI  
            dht-param SEMI overlay-param *(SEMI generic-param)
```

Examples: A peer with an SHA-1 hashed Peer-ID of a04d371e on IP 192.168.1.1. We include the PeerURI, algorithm, dht-param, and overlay as well as the optional expires header parameter. In this example, the overlay name is chat and the DHT algorithm being used is dhtalg1.0

```
DHT-PeerID: <sip:peer@192.168.1.1;peer-ID=a04d371e>;algorithm=sha1;  
            dht=dhtalg1.0;overlay=chat;expires=600
```

A.1.4.1 Hash Algorithms and the algorithm Parameter

The hash algorithm used for the overlay is specified as a parameter of the DHT-PeerID header. This parameter **MUST** appear in the DHT- PeerID header. It **MUST** be the algorithm used to calculate all Peer-ID and Resource-ID values used in the message. It **SHOULD NOT** appear in other headers in the message, but if it does it **MUST** match the value in the DHT-PeerID header.

The hash algorithm is specified using the algorithm parameter from RFC3261. The tokens used to identify the algorithm **MUST** be the same as those used in other SIP documents such as [70]. Currently, those consist of 'sha1', indicating SHA-1 as defined in [1] and 'hmac-sha1', indicating HMAC-SHA1 [56]. Implementations **MUST** support the SHA-1 algorithm.

A peer **MUST** reject a message with 488 Not Acceptable here if it specifies a different hash algorithm than that used by the peer's overlay. An initial contact to a bootstrap peer may specify the hash algorithm as the wildcard "*", in which case the joining peer

indicates its willingness to use whatever hash algorithm the bootstrap peer identifies in its response. A peer responding to such a request **MUST** route the message according to the rules described in the Message Routing Section if all other elements of the message are correct and the routing algorithm indicates such a response is appropriate. If the normal response would be to allow the join with a 200 OK, the receiving peer **MAY** respond with a 302 redirect to itself and specifying the algorithm used in this overlay, in which case the joining peer should reissue the message with the proper hash algorithm specification.

A.1.4.2 Overlay Names and the overlay Parameter

Each overlay is named using a string, which **SHOULD** be unique to a particular deployment environment. Peers will use this value to identify messages in cases where they may belong to multiple overlays simultaneously. These are defined formally simply as a token:

`overlay-name = "*" / token`

The overlay-param parameter **MUST** appear in the DHT-PeerID header. It **SHOULD NOT** appear in other headers in the message, but if it does it **MUST** match the value in the DHT-PeerID header. This parameter is defined formally as:

`overlay-param = "overlay" EQUAL overlay-name`

A peer **MUST** reject a message with 488 Not Acceptable here if it specifies an overlay in which the peer is not participating. An initial contact to a bootstrap peer **MAY** specify overlay-name as the wildcard "*", in which case the joining peer indicates its willingness

to join whatever overlay the bootstrap peer identifies in its response. A peer responding to such a request **MUST** route the message according to the rules described in the Message Routing section if all other elements of the message are correct and the routing algorithm indicates such a response is appropriate. If the normal response would be to allow the join with a 200 OK, the receiving peer **MAY** respond with a 302 redirect to itself, in which case the joining peer should reissue the message with the proper overlay specification.

A.1.4.3 DHT Algorithms and the dht Parameter

The routing algorithm used to implement the overlay is specified using a dht-param in the DHT-PeerID header. It **SHOULD NOT** appear in other headers in the message, but if it does it **MUST** match the value in the DHT-PeerID header. This parameter is defined formally as:

dht-name	= token
dht-param	= "dht" EQUAL dht-name

The behavior of a peer receiving a message with a dht-param specifying a routing algorithm other than that which it is following is dependent on the routing algorithm. An initial contact of a bootstrap peer **MAY** specify dht-param as the wildcard "*", in which case the joining peer indicates its willingness to use whatever DHT algorithm the bootstrap peer identifies in its response. A peer responding to such a request **MUST** route the message according to the rules described in the Message Routing section if all other elements of the message are correct and the routing algorithm indicates such a response is appropriate. New routing algorithms **SHOULD** be designed to maintain backward compatibility with previous algorithms where possible. If the routing algorithm specified is incompatible, a 488 Not Acceptable Here response **MUST** be returned.

A.1.4.4 PeerID Expires header parameter

The DHT-PeerID header MAY include an Expires parameter indicating how long a recipient may keep knowledge of this peer. If not present, a default of 3600 is assumed. Mobile peers may wish to specify a shorter interval.

A.1.5 The DHT-Link Header

We introduce a new SIP header called the DHT-Link header. The DHT-Link header is used to transfer information about where in the DHT other peers are located. In particular, it is used by peers to pass information about its neighbor peers and routing table information stored by a peer.

```
DHT-Link          = "DHT-Link" HCOLON PeerURI SEMI link-param SEMI
                    expires-param *(SEMI generic-param)
link-param         = "link" EQUAL link-value
expires-param      = "expires" EQUAL delta-seconds
```

The value of linkvalue – that is, how you represent what type of link this is, is defined by the DHT algorithm specification. The generic-param leaves flexibility for an algorithm to add additional parameters if needed.

As an example, the header might look like (using a shortened 10 digit Peer-ID for clarity). The value *** here is intended to represent a value determined by the particular DHT:

```
DHT-Link: <sip:peer@192.168.0.1;Peer-ID=671a65bf22>;link=***;expires=600
```


A.1.5.1 Expires Processing

Each DHT-Link header **MUST** contain an expires parameter. Each peer maintains an expiration time for each of its neighbor and routing table entries. These expiration times are updated whenever the peer receives a response with a longer expiration time than it currently maintains, most commonly in the PeerID header of a response to a join or search. A peer **MUST NOT** report an expired entry in a DHT-Link header. A peer **MUST** update the expires parameter with the current value, adjusted for passed time, each time it generates a DHT-Link header.

A.2 Message Routing

When a peer sends a message within the DHT, it begins by calculating the target ID it is attempting to locate, which might be its own location in the DHT, or a user's registration, for which it hashes the user's URI to obtain the appropriate Resource-ID. It then consults its routing table, and its other neighbor peers, for the closest peer it is aware of to the target ID.

The messages in the overlay **MAY** be routed either iteratively or recursively. The Request-Disposition header as described in [85] **SHOULD** be used to indicate if the next node should process the message using a recursive or iterative mechanism. If the header is omitted, the receiving node may process the message either recursively or iteratively.

If the Request-Disposition header is iterative, the contacted peer **MUST** determine if it is responsible for that target ID. If it is not, then the contacted peer **MUST** issue a 302 redirect pointing the search peer toward the best match the contacted peer has for the target ID. The searching peer then contact the peer to which it has been redirected and the process iterates until the responsible peer is located.

In recursive routing, the peer sends a message to the peer it knows that is nearest to the target. If the contacted peer is not responsible for the target ID, it **MUST** forward the query to the nearest peer to the target that it knows, and the process repeats until the target is reached. This process follows standard proxy behavior in RFC 3261.

A.2.1 Peer Registration

When a peer (the joining peer) wishes to join the overlay, it creates its Peer-ID and sends a REGISTER message to a bootstrap peer already in the overlay, requesting to join. Any peer in the DHT may serve as a bootstrap peer, although we expect that most UAs will be configured with a small number of well-known peers.

Following the iterative routing scheme, the bootstrap peer looks up the peer it knows nearest to the Peer-ID of the joining peer and responds with 302 redirect to this nearer peer. The joining peer will repeat this process until it reaches the peer currently responsible for the space it will occupy.

If recursive routing is being used, the bootstrap peer looks up the peer it knows nearest to the Peer-ID of the joining peer and forwards the REGISTER message to that peer. This process of forwarding the message repeats until the peer currently responsible for the space the joining peer will occupy is found.

Once the peer responsible for the joining peer's portion of the namespace is located, the joining peer then exchanges DHT state information with this peer, called the admitting peer, to allow the joining peer to learn about other peers in the overlay (neighbors) and to obtain information about resources the joining peer will be responsible for maintaining. Other DHT maintenance messages will be exchanged later to maintain the overlay as other peers enter and leave, as well as to periodically verify the information about the overlay, but once the initial messages are exchanged, a peer has joined the overlay.

A.2.2 Resource Registration

The peer registration does not register the peer's user(s) or other resources with the P2PSIP network – it has only allowed the peer to join the overlay. Once a peer has joined the overlay, the user that peer hosts must be registered with the system. This process is referred to as resource registration. This registration is analogous to the conventional SIP registration, in which a message is sent to the registrar creating a mapping between a SIP URI and a user's contact. The only difference is that since there is no central registrar, some peer in the overlay will maintain the registration on the users behalf.

Resource registrations are routed similarly to peer registrations. The resource's peer calculates the resource-ID and contacts the peer it is aware of closest to the resource-ID. This search process continues in either an iterative or recursive manner until the responsible peer is located. This peer then stores the registration for that user and returns a 200 response.

For redundancy, resources should also be registered at additional peers within the overlay. These replicas are located by adding a replica number to the resource name and hashing to identify a new resource-ID for each replica. In this way, replicas are located at unrelated points around the DHT, minimizing the risk of an attacker compromising more than one registration for a single resource.

A.2.3 Session Establishment

Sessions are established by contacting the UA identified by the registration in the DHT. The first step in establishing a session is locating this peer, which is done by searching for a resource in the DHT. The name of the target resource is used to calculate a resource-ID and a REGISTER message with no Contact information (a conventional SIP search) is

sent to the closest known peer to that resource-ID. The search iterates until the responsible peer is located. The responsible peer then returns either a 200 OK with the Contact information for the resource or a 404 Not Found. The session is then initiated directly with the resource's UA.

If the peer needs to have the session establishment routed through the overlay, it MAY use the Request-Disposition header with a value of proxy to request that intermediate nodes proxy the invite over the overlay on their behalf. This is particular critical for NAT traversal.

A.2.4 DHT Maintenance

In order to keep the overlay stable, peers must periodically perform book keeping operations to take into account peer failures. These DHT maintenance messages are sent using REGISTER messages and the overlay algorithm being used will dictate how often and where these messages are sent. DHT maintenance messages are routed similarly to peer registrations and resource registrations. The peer calculates the Peer-ID of the peer it wants to exchange DHT information with and contacts the peer it is aware of closest to that Peer-ID. This search process continues until the current closest peer to the target Peer-ID is located at which point the peers exchange DHT maintenance information.

A.3 Peer/DHT Operations

The SIP REGISTER message is used extensively in this system. REGISTER is used to register users, as in conventional SIP systems, and we discuss this further in the Resource Registration section. Additionally, SIP REGISTER messages are used to register a new peer with the DHT and to transmit the information needed to maintain the DHT.

A.3.1 Peer Registration

After a peer has located an initial bootstrap peer, the process of joining the overlay is started by constructing a REGISTER message and sending it to the bootstrap peer. Third party registration MAY NOT be used for registering peers into the overlay, and attempts to do so MUST be rejected by the peer receiving such a request (although third party registrations are used for other purposes, as described below). The peer MUST construct a SIP REGISTER message following the instructions in RFC3261, Section 10, with the exceptions/rules outlined below.

A.3.1.1 Constructing a Peer Registration

The Request-URI MUST include only the IP address of the peer that is being contacted (initially the bootstrap peer). This URI MUST NOT include any of the P2P defined parameters. For example, a request intended for peer 10.3.44.2 should look like: "REGISTER sip:10.3.44.2 SIP/2.0".

The To and From fields of the REGISTER message MUST contain the URI of the registering peer constructed according to the rules in the section on Peer URIs in the Message Syntax section.

While allowing the IP address of the sender for To and From is different than conventional SIP registers, there are two reasons for this. First, in a P2P network, which peer the request is sent to, and thus the domain for which the registration is intended, is not important. Any peer can process the information, and the username is not associated with a particular IP address or DNS domain, but rather with the overlay name, which is encoded elsewhere. In that sense, the IP address used is irrelevant. Choosing the domain of the sender ensures that if a request is sent to a non-P2P aware RFC 3261

compliant registrar, it will be rejected. RFC 3261 (section 10.3) states that a registrar should examine the To header to determine if it presents a valid address-of-record for the domain it serves. Since the IP address of the sending peer is unlikely to be a valid address for a non-P2P aware registrar, the message will be rejected, eliminating possibly erroneous handling by the registrar.

The registering peer **MUST** also list its PeerURI in the contact field when registering so that this may be identified as a registration/ update, rather than a query. The peer **MUST** provide an expires parameter or expires header with a non-zero value. As in standard SIP registrations, Expire headers with a value of zero will be used to remove registrations.

The registering peer **MUST** provide a DHT-PeerID header field. It **MAY** leave the overlay parameter set to "" for its initial registration message, but **MUST** set this parameter to the name of the overlay it is joining as soon as it receives a response from the bootstrap peer.

The registering peer **MUST** include Require and Supported headers with the option tag "dht".

Assume that a peer running on IP address 10.4.1.2 on port 5060 attempts to join the network by contacting a bootstrap peer at address 10.7.8.129. Further assume that 10.4.1.2 hashes to 463ac4b449 under SHA-1 (using a 10 digit hash for example simplicity), and the least significant bits are replaced with the port number, yielding 463ac413c4 and that the overlay name is chat and the dht-param is dhtalg1.0. An example message would look like this (neglecting tags):

```
REGISTER sip:10.7.8.129 SIP/2.0
To: <sip:peer@10.4.1.2;peer-ID=463ac413c4>
```

```
From: <sip:peer@10.4.1.2;peer-ID=463ac413c4>  
Contact: <sip:peer@10.4.1.2;peer-ID=463ac413c4>  
Expires: 600  
DHT-PeerID: <sip:peer@10.4.1.2;peer-ID=463ac413c4>;algorithm=sha1;  
            dht=dhtalg1.0;overlay=chat;expires=600  
Require: dht  
Supported: dht
```

A.3.1.2 Processing the Peer Registration

The receiving peer determines that this is a P2PSIP message based on the presence of the dht Require and Supported fields. In the event that the peer does not support P2P extensions, it **MUST** reply with a 420 Bad Extension response. If the peer examines the overlay parameters and determines that this is not an overlay the peer participates in, the peer **MUST** reject the message with a 488 Not Acceptable Here response. Likewise if the peer examines the dht- param and determines that the algorithm specified is not compatible with its algorithm, the peer **MUST** reject the message with a 488 Not Acceptable Here response. If a P2P peer receives a non-P2P request it **MAY** reject it with a message such as 421 Extension Required or it **MAY** process it as a conventional SIP message.

An implementation may support both P2P and conventional SIP messages. In that case, it **MAY** include the dht Supported field with all messages but **MUST NOT** include it with messages intended for conventional nodes.

Routing the Peer Registration

The presence of peer-ID URI parameter in the To and Contact headers and a valid expiration time indicate that this message is a peer registration and the receiving peer **MUST** process this as a DHT level request. The bootstrap peer **SHOULD** verify that the Peer-ID corresponds to peer listed in the URI by validating the hash or the peers credentials. If these do not match, the message **SHOULD** be rejected with a response of 493

Undecipherable. The bootstrap peer examines the Peer-ID to determine if it corresponds to the portion of the overlay the bootstrap peer is responsible for. If it does, the peer will handle the REGISTER request itself. If not, the bootstrap peer will either provide the joining peer with information about a peer closer to the area of the overlay where the joining peers Peer-ID is stored (iterative routing) or forward the request along the closest peer it knows about (recursive routing). If a Request- Disposition header is present and set to proxy, the peer MUST use a recursive routing mechanism, and if it is present and set to redirect, the peer MUST use an iterative routing mechanism. In the event that the Request-Disposition header is not present, the peer may choose either mechanism.

In the case of iterative routing, if the receiving peer is not responsible for the area of the hash table where Peer-ID should be stored, the peer SHOULD generate a 302 message. The 302 is constructed according the rules of RFC 3261 with the following rules. The receiving peer MUST look in its list of neighbors or in the routing table to find the peer with Peer-ID nearest the to joining peer's Peer-ID, and use it to create a contact field in the form of a peer URI, as specified in the P2P Peer URIs section, including appropriate URI parameters. The response MUST contain a valid DHT-PeerID header. This response is sent to the joining peer.

In the case of recursive routing, if the receiving peer is not responsible for the area of the hash table where the Peer-ID should be stored, the receiving peer should forward the request to the peer it knows about that is closest to the Peer-ID.

A peer MUST NOT add a new peer to its routing table or redirect requests to that new peer until it has successfully contacted that peer itself. By redirecting a message to another peer, the contacted peer indicates that it believes that peer to be alive and that it is willing to route messages to it for NAT and Firewall traversal purposes.

Using our example register from the previous section, assume that iterative routing is being used and that the bootstrap peer 10.7.8.129 receives the message, determines it

is not responsible for that area of the overlay, and redirects the joining peer to a peer with Peer-ID 47e46fa2cd at IP address 10.3.1.7. The 302 response, again neglecting tags, is shown below. Note that the peer creating the response uses its information to construct the DHT-PeerID header.

```
SIP/2.0 302 Moved Temporarily
To: <sip:peer@10.4.1.2;peerId=463ac413c4>
From: <sip:peer@10.4.1.2;peerId=463ac413c4>
Contact: <sip:peer@10.3.1.7;peerId=47e46fa2cd>
Expires: 600
DHT-PeerID: <sip:@10.7.8.129;peerId=084d299ff2>;algorithm=sha1;
            dht-param=dhtalg1.0;overlay=chat;expires=600
Require: dht
Supported: dht
```

Upon receiving the 302, the joining peer uses the contact address as the new bootstrap peer. The process is repeated until the peer contacted is currently responsible for the area of the DHT in which the new peer will reside. The receiving peer that is responsible for that portion of the overlay is referred to as the admitting peer.

Admitting the Joining Peer

The admitting peer **MUST** verify that the Peer-ID is valid, as described above. If these do not match, the message **MUST** be rejected with a response of 493 Undecipherable. The admitting peer recognizes that it is presently responsible for this region of the hash space – that is, it is currently the peer storing the information that this Peer-ID will eventually be responsible for. The admitting peer knows this because the joining peer's Peer-ID is closest to its own Peer-ID. The admitting peer is responsible for helping the joining peer become a member of the overlay. In addition to verifying that the Peer-ID was properly calculated, the admitting peer **MAY** perform additional security checks. Once any challenge has been met, the admitting will reply with a 200 OK message to the joining peer.

As in a conventional registration, the Contact in the 200 OK will be the same as in the request, and the expiry time MUST be provided.

The admitting peer MUST reply with a 200 response if the admitting peer's Peer-ID is the closest to the joining peer's Peer-ID. Each DHT algorithm MAY choose to define closest however they want, but the DHT algorithm MUST be able to deterministically find the closest Peer-ID. The admitting peer must populate the DHT-Link headers with all values required by the DHT routing protocol so that the joining peer can initialize its neighbors and routing table entries. Additionally, the admitting peer MUST include its DHT-PeerID header containing the admitting peer's Peer-ID and IP.

A.3.2 Peer Query

As with conventional SIP, REGISTER messages that are sent without a Contact: header are assumed to be queries, as described in Section 10 of RFC3261.

A.3.2.1 Constructing a Peer Query Message

The peer looks for the routing table entry or neighbor peer that is closest to the ID they are searching for. If the routing table has not yet been filled, then the peer may send the request to any peer it has available, including their other neighbor peers or even some bootstrap peer. While these initial searches may be less efficient, they will succeed. The Request-URI MUST include only the IP address of the peer that the search is intended for. This URI MUST NOT include any of the P2P defined parameters. For example, a request intended for peer 10.3.44.2 should look like: "REGISTER sip:10.3.44.2 SIP/2.0".

Because this is a query, the sending peer MUST NOT include a contact header. The sender MUST NOT include an expires header.

The peer **MUST** provide a DHT-PeerID header.

The peer **MUST** include Require and Supported headers with the option tag "dht".

Assume that a peer running on IP address 10.4.1.2 on port 5060 wants to determine who is responsible for Peer-ID 4823affe45, and asks the peer with IP address 10.5.6.211. Further assume that the peer uses SHA-1 (using a 10 digit hash for example simplicity), and that the overlay name is chat. An example message would look like this (neglecting tags):

```
REGISTER sip:10.5.6.211 SIP/2.0
To: <sip:peer@0.0.0.0;peerId=4823affe45>
From: <sip:peer@10.4.1.2;peerId=463ac413c4>
DHT-PeerID: <sip:peer@10.4.1.2;peerId=463av413c4>;algorithm=sha1;
           dht-param=dhtalg1.0;overlay=chat;expires=600
Require: dht
Supported: dht
```

The To field of the REGISTER message **MUST** contain the PeerURI of the identifier being search for, constructed according to the rules in the subsection P2P peer URIs in the Message Syntax section. If a specific peer is being sought, the PeerURI must specify that hostport. If only the identifier is being searched for, then hostport **MUST** be set to "0.0.0.0". The From URI **MUST** use the searching peer's PeerURI.

A.3.2.2 Processing Peer Query Message

The receiving peer determines that this is a P2PSIP message based on the presence of the dht Require and Supported fields. In the event that the peer does not support P2P extensions, it **MUST** reply with a 5xx class response such as 501 Not Implemented. If the peer examines the overlay parameters and determines that this is not an overlay the

peer participates in, the peer **MUST** reject the message with a 488 Not Acceptable Here response. In the event a P2P peer receives a non-P2P request, it **SHOULD** reject it with a message such as 421 Extension Required.

Routing the Peer Query Message

The presence of a PeerURI and lack of an expiration time indicate that this message is a peer query and the receiving peer **MUST** process this as a DHT level request. The receiving peer **SHOULD NOT** alter any of its internal values such as successor or predecessor in response to this message, since it is a query. Otherwise, the message is processed and routed as a peer registration until the responsible peer is reached.

Responding to the Peer Query Message

If the receiving peer is responsible for the region that the search key lies within, it **MUST** respond to the query. If the receiving peer's Peer-ID exactly matches the search key, it **MUST** respond with a 200 OK message. If it is responsible for that region, but its Peer-ID is not the search key, it **MUST** respond with a 404 Not Found message. The peer **MAY** verify the Peer-ID and IP address presented by the querying peer in the message. If these do not match, the message should be rejected with a response of 493 Undecipherable.

A.3.3 Populating the Joining Peer's Routing Table

Once admitted, the joining peer **SHOULD** populate its routing table and locate neighbors by issuing queries for peers with the appropriate identifiers. If the admitting peer provided neighbor or routing table information in its response, the joining peer **MAY** use this information to construct a temporary routing table and neighbor information and use this temporary table in the queries to populate the table.

A.3.4 Transferring User Registrations

When a new peer joins, it splits the area in the hash space the admitting peer is responsible for. Some portion of the user registrations the admitting peer was responsible for may now be the responsibility of the joining peer, and these user registrations are handed to the joining peer by means of third party user registrations. Third party registrations are allowed for user registrations and arbitrary searches, but are not allowed for peer registrations. These registrations are exactly the same as those discussed in the Registering and Removing User Registrations section, except that as they are third party registration from a peer, that is, the From header contains the PeerURI of the admitting peer.

A.3.5 Peers Leaving the Overlay Gracefully

Peers **MUST** send their registrations to the closest peer before leaving the overlay, as described in the section above. Additionally, peers **MUST** unregister themselves with their symmetric neighbors (if the DHT routing algorithm uses symmetric neighbors in any form). These graceful exit REGISTER messages are constructed exactly the same as one used to join, with the following exceptions. The expires parameter or header **MUST** be provided, and **MUST** be set to 0. DHT-Link headers must be provided, as specified in DHT routing algorithm

A.3.6 NAT and Firewall Traversal

The filtering properties of NATs and firewalls can lead to non- transitive connectivity. Typically this will manifest itself in a peer receiving a 302 redirecting it to another peer that it cannot contact, most likely because address dependent filtering is occurring.

A.3.7 Handling Failed Requests

When a request sent to another peer fails, the peer **MUST** perform searches to update its pointers. If the failed request was sent to a peer in the routing table or a neighbor peer, then the searches discussed in the Populating the Joining Peer's Routing Table section should be performed.

A.4 Resource Operations

The most important element of resource operations within the P2PSIP DHT is that they are performed exactly as if using a conventional SIP registrar, except that the registrar responsibilities are distributed among the DHT members.

A.4.1 Resource Registrations

When a peer is in the overlay, it must register the contacts for users and other resources for which it is responsible into the overlay. This differs from the registrations described above in that these registrations are responsible for entering a URI name to URI location mapping into the overlay as data, rather than joining a peer into the overlay. These registrations are very similar to those outlined in section 10 of RFC3261.

The Request-URI that is constructed for the REGISTER **MUST** be addressed to the peer the request is sent to. The To and From fields of the REGISTER message **MUST** contain the Resource URI of the resource being registered, as described in the Resource URIs section. The request **MUST** include the value dht in Require and Supported headers. The request **MUST** include a DHT-PeerID header and **MAY** include one or more DHT-Link headers.

The resource registration **MUST** include at least one Contact header containing a location of the resource and allowing this to be identified as a registration/update, rather than a query. The peer **MUST** provide an expires parameter or an Expire header with a non-zero value. As in standard SIP registrations, Expires parameters with a value of zero will be used to remove registrations. Any valid Contact for RFC 3261 is valid Contact for P2PSIP. Most users will register a Contact with the address of the user's UA (which may or may not be the IP address of the peer, since the peer could be an adaptor peer). The Contact URI does not need to include the Resource-ID or other P2PSIP parameters as it is stored in the DHT but not processed or routed by it in any way.

The message is routed in a fashion exactly analogous to that described in the section on peer registration. In iterative routing algorithms, 302 messages are sent to indicate that the message is to be redirected to another Peer URI. In recursive routing algorithms, the receiving peer **SHOULD** forward the request to the peer in its connection table that is closest to the Resource-ID. Once the message arrives at a destination that is responsible for that portion of the hash namespace, the peer recognizes it as a resource registration, rather than a peer wishing to join the system, based upon the fact that the To and From fields do not contain a Peer URI. The peer responds with a 200 indicating a successful registration. The response is constructed as dictated by RFC3261.

The registering peer **SHOULD** construct and register replica registrations using the same Contact headers, but with the replica URI parameter used in the To and From headers.

A.4.2 Refreshing Resource Registrations

Resource registrations are refreshed exactly as described in RFC 3261, Section 10. Responsible peers should send a new registration with a valid expiration time prior to the

time that the registration is set to expire.

Agents MAY cache the address where they previously registered and attempt to send refreshes to this peer, but they are not guaranteed success, as a new peer may have registered and may now be responsible for this area of the space. In such a case if iterative routing is being used, the peer will receive a 302 from the peer with which they previously registered, and should follow the same procedure for locating the peer they used in the initial registration.

As with initial registrations, the sending peer should use the neighbor peer or routing table information provided in the 200 to send these updates to the redundant peers as well.

A.4.3 Removing Resource Registrations

Resource registrations are removed exactly as described in RFC 3261, Section 10. Responsible peers MUST send a registration with expiration time of zero.

As with initial registrations, the sending peer MUST construct replica unregister messages and use these to unregister the replicas.

A.4.4 Querying Resource Registrations

Resource queries are constructed as described in RFC 3261, Section 10. Querying peers should send a REGISTER message with no contact header. As described in the Peer Search section, this mechanism can also be used to locate the peer responsible for a particular Resource-ID.

A P2P environment can do little to protect against an individual peer compromising the registrations it is responsible for. Accordingly, a UA cannot trust a response from a single peer, whether it indicates a successful search or an error. In the absence of other methods of verifying the response (such as having a certificate of the user being searched for and a signed registration that can be verified with the certificate) a UA should search for the primary registration and at least one replica. Because the locations the replicas are stored are unrelated to the location of the primary registration, a single attacker is unlikely to be able to compromise both entries. As the overlay gains more peers and more replicas are searched for, the odds of a compromise are reduced.

A.4.5 Session Establishment

When a caller wishes to send a SIP message (such as an INVITE, MESSAGE or SUBSCRIBE), the caller must first locate the peer where this callee's information resides using the resource search procedure described in the section titled Resource Location.

Establishing a session is done entirely in the normal SIP fashion after the user is located using the P2P resource query. Once the peer responsible for the Resource-ID is located, it will provide either a 200, providing a contact for the users UA, or will provide a 404 if the user is not registered. If a 200 with a valid contact is received, the call will then be initiated directly with the UAS of the called using the standard RFC 3261 fashion for methods such as INVITE or MESSAGE, or the INVITE can be processed by routing it through the overlay if necessary for NAT traversal.

A.4.6 Presence

We use SUBSCRIBE/NOTIFY for this. We subscribe to every user on our friend list when we come online. If the friends are online, that means that we know exactly where they

are. Peers MAY use the Peer-IDs of their friends' peers as additional routing table entries or neighbor peers (essentially, cached values), consulting these first, as connections are likely to be made to people on the user's friend list. These should also be periodically checked, as described in the DHT Maintenance section.

If friends are offline, one should periodically try to make the connection. However, if a UA receives a SUBSCRIBE from a friend that it believes to be offline, it SHOULD attempt to subscribe to that friend. This will allow people that are reciprocally on each other's friend lists to rapidly be notified when one or the other comes online, therefore the retry interval for subscribing to offline friends can be fairly long because it is only necessary in the case of race conditions or other temporary failures in resource location.

A.4.7 Offline Storage

Delivery of messages to offline users, or voicemail for voice applications, requires storing that information for later retrieval. Storing user configuration information in a format accessible from the network also will allow a user to retrieve their profile from any computer. Cao et al. [25] describe an approach that separates the storage of resource location information from the actual storage of the offline resource. We believe that this approach is in agreement with the approach taken by the rest of this appendix, which relies on the DHT overlay to store the registrar's location information, but relies on external, conventional methods for the actual connection. For offline storage, it also allows the use of other standard protocols to store and retrieve the offline information, keeping the P2PSIP scope restricted to storing resource mappings.

A.5 Pluggable DHT Algorithm Requirements

All dSIP peers **MUST** support the Chord pluggable DHT algorithm for compatibility. They **MAY** support additional pluggable algorithms. The requirements for new pluggable algorithms are defined in this section.

Pluggable algorithm **MUST** use Peer-IDs and Resource-IDs as defined in the Hash Algorithms and Identifiers section. Pluggable algorithms are free to define what hash algorithms they support, but they **MUST** clearly specify what they are.

A resource with Resource-ID k will be stored by the peer with Peer-ID closest to the Resource-ID. The definition of closeness may vary in different DHT algorithms, but each DHT algorithm **MUST** guarantee Resource-ID searches converge to exactly one peer responsible for that portion of the namespace. As peers enter and leave, resources may be stored on different peers, so the information related to them is exchanged as peers enter and leave. Redundancy is used to protect against loss of information in the event of a peer failure.

Each new DHT algorithm **MUST** define a value for the `dht-name` parameter to be used in the `dht-param` parameter of the DHT-PeerID header, as defined in the DHT Algorithms section and the `dht` parameter.

Each new DHT algorithm **MUST** define the valid BNF for the link-value used in the DHT-Link header, as defined in The DHT-Link header section.

A.6 Security Considerations

The goal of P2PSIP is to scale gracefully from ad hoc groups of a few people to an overlay of millions of peers across the globe. As such, there is no one security model

that fits the needs of all envisioned environments; for the small network establishing a certificate chain is ludicrously difficult, while for a global network the unrestricted ability to insert resources and devise useful Peer-IDs is a clear invitation to insecurity. Any P2PSIP protocol must offer a range of security models that can be selected according to the needs of the overlay.

A.6.1 Threat Model

Without other security, the attacker is able to generate an ID and become a valid peer in the system. They can see other peers and process certain queries. Attackers may wish to receive communications intended for other participants, prevent other users from receiving their messages, prevent large portions of the users from receiving messages, or send messages that appear to be from others. Users would like to be sure they are communicating with the same person they have previously talked to, to be able to verify identity via some out of band mechanism. Attackers may try to squat on all the good names. Users would like names that are meaningful to them. Attackers may have computers that are many times faster than the average user's. Attackers may be able to DOS other particular peers and make them fail. To make a robust DHT, many peers need to store information on behalf of the community. Peers may lie about this and not store the information. Attackers may wish to see who is communicating with whom and how much data is getting communicated.

Many of the threats to P2PSIP are also threats to conventional SIP. As such, P2PSIP imports much of its security from conventional SIP. However, because conventional SIP generally relies on secure servers to maintain the integrity of the system, modifications to those techniques are required to maintain the same level of security.

A.6.2 Protecting the ID Namespace

The fundamental protection that P2PSIP relies on is protecting the ID namespace. In particular, many of the attacks on P2PSIP require identifying a particular portion of the ID space and acquiring control of that space. This is a common vector both for attacks on a particular user, by obtaining control of the location in the overlay where the user is registered, and on the overlay itself, by means of a Sybil attack when one is able to insert multiple identities at different locations on the ring.

The P2PSIP ID Namespace is considered protected when an attacker is not able to select an arbitrary Peer-ID and insert a peer at the location by convincing other peers to route traffic to them. This protects against hijacking and DoS attacks. The ID Namespace may also be protected by restricting admission to the overlay to some authorized (and trusted) set of individuals.

A.6.2.1 Protection Using ID Hashing

The default base security for P2PSIP determines Peer-IDs by hashing the peer's IP address and appending the port number. The security of this scheme depends on the ease with which an attacker can choose their own Peer-ID. Because the port number is only appended to the Peer-ID, an attacker gains nothing by selecting different ports on the same node. Assuming that the SHA1 hash used to calculate the Peer-ID is reliably random, the attacker's ability to succeed depends on the number of separate IP addresses that they are able to obtain from which to launch their attacks.

In the current predominantly IPV4 Internet, few attackers have access to more than a handful of IP addresses, perhaps a few hundred at worst. For a large-scale P2P system, this is unlikely to provide the ability to hijack a particular user ID or control a sufficient

portion of the network to affect other peers, in particular when registrations are replicated at independent peers. Ultimately, however, a sufficiently skilled and provisioned attacker can compromise this scheme.

As the Internet migrates to IPV6, however, it is unclear that the assumption that few attackers have access to a significant range of IP addresses will remain true. Therefore, hashing IP addresses to Peer-IDs is assumed to provide a diminishing amount of security in the future.

A.6.2.2 Cryptographic Protection

Stronger protection guarantees are possible by relying on cryptographic techniques to restrict the generation of peer-IDs, either through requiring knowledge of a shared secret to calculate a valid hash or by issuing certificates through a central authority.

A.6.3 Protecting the Resource Namespace

The two primary vectors of attacks on resources in a P2PSIP overlay are inserting illegitimate resources into the overlay and corrupting the registrations for which a compromised peer is responsible.

For overlays that do not rely on certificates, once a peer has joined the overlay there are no restrictions on its ability to register resources. In an unsecured network, multiple peers can register the same resource (username) in the overlay. However, self-signed certificates can be used to authenticate a user as the same user previously contacted with that certificate. Unless a conventional SIP authentication server is available, however, establishing identity upon initial contact is still a problem. One potential solution is for an overlay that is expected to persist over long time-frames to store the credentials of

previous users for verification of a new registration. These techniques are beyond the scope of this specification.

The second form of resource attack, which is really an ID attack, concerns the attacks that are possible when a peer has legitimately inserted itself into the overlay and is now responsible for storing resource registrations. Such an attack could occur through a corrupted peer or by an attacker who convinces the CA to issue them a certificate for a Peer-ID. In this case, the peer can corrupt any resource that is assigned to it. In the absence of certificates, the primary means of defense of such attacks is relying on the replication described in the Resource-IDs and Replication section. By storing replicas of each registration on multiple peers and performing parallel searches for resource lookup, the searching peer protects itself from a single peer trying to corrupt the namespace.

Further protection from each attack vector is achieved by relying on certificates for resource authentication.

A.6.4 Protecting the Routing

The DHT forms a complex routing table. When a peer joins, it may contact a subversive peer that lies about the finger table information it provides. The subversive peer could do this to try to trick the joining peer to route all the traffic to a subversive group of peers. Prevention of this attack relies on protecting the namespace and (for hashed namespaces) identifying trusted bootstrap peers to use when joining.

Resource searches are protected from a single subversive peer through the use of parallel searches on replicated registrations. Similar protection could be achieved through performing parallel searches using multiple bootstrap peers for initial join, but such specification is beyond the scope of this specification. When possible, securing the namespace is a better solution.

A.6.5 Protecting the Signaling

The goal here is to stop an attacker from knowing who is signaling what to whom. An attacker being able to observe the activities of a specific individual is unlikely given the randomization of IDs and routing based on the present peers discussed above.

A.6.6 Protecting the Media

As with conventional SIP, all the media SHOULD be encrypted. Negotiating encryption for an end-to-end media session should be performed in the same manner for P2PSIP communications.

Appendix B

RELOAD Protocol Specification

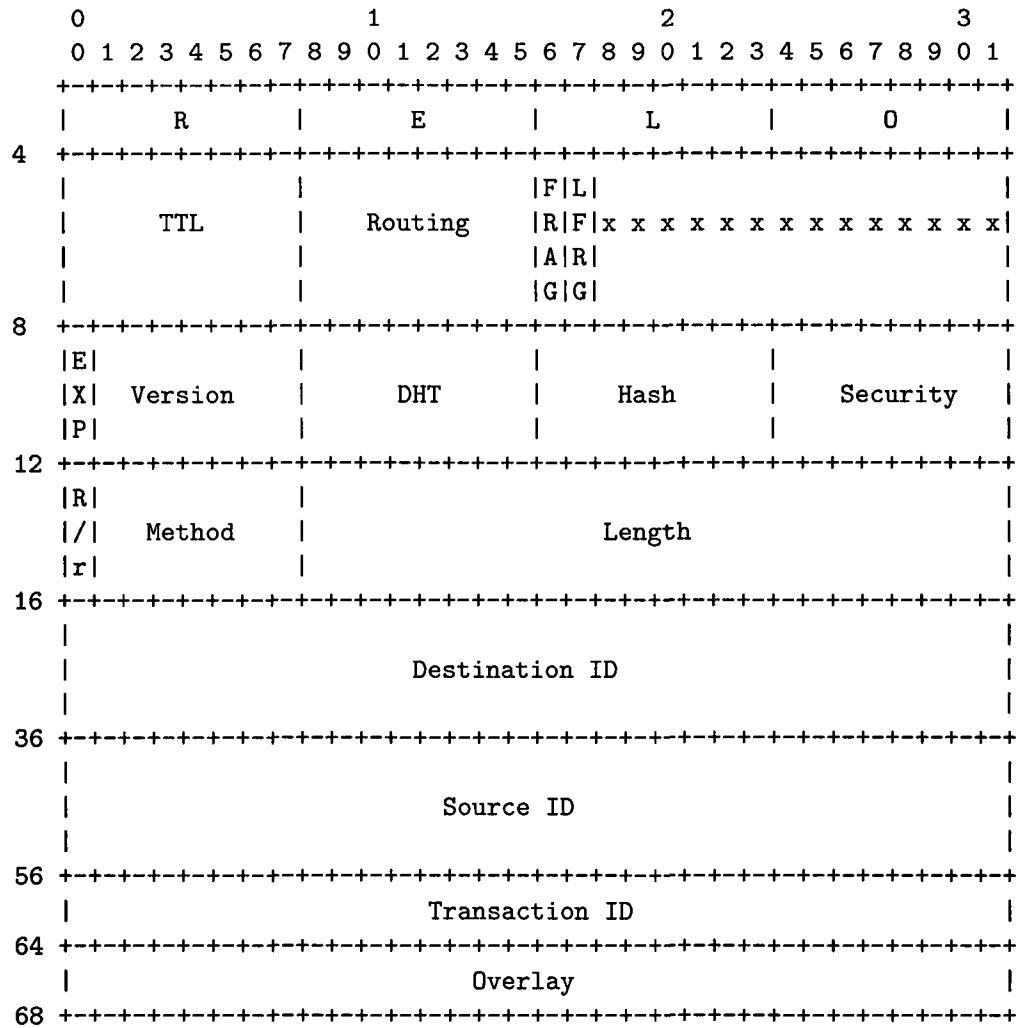
B.1 Message Syntax

Much of this text appears in [20], and as such is a snapshot of the work on RELOAD at that date. Since then, this work has been merged into [52] and others are continuing to develop the protocol in the IETF with the help of some of our later research.

This section provides normative text explaining the syntax of the RELOAD messages and provides the details of the binary encoding. The RELOAD protocol is a request-response protocol. The messages are encoded using binary fields and is modeled after the binary STUN protocol. All integers are represented in network byte order and are present in base 10, decimal, unless otherwise noted.

B.1.1 Message Header

The RELOAD message contains a 68 byte header.



The first four bytes identify this message as a RELOAD message.

TTL (time-to-live) is an 8 bit field indicating the number of iterations, or hops, a message can experience before it is discarded. The TTL `uint8` number **MUST** be stored in network byte order and **MUST** be decremented by one at every hop along the route the message traverses. If the TTL is 0, the message **MUST NOT** be propagated further and **MUST** be discarded.

Routing is an 8 bit field that specifies the type of routing the requester would like for the message. The following Routing options **MUST** be understood:

UNSPECIFIED	: 0x00
PROXY	: 0x01
REDIRECT	: 0x02

If a peer is unable or unwilling to perform the type of routing requested, the peer **MUST** respond with a 499 error message that indicates its unwillingness to process the message.

FRAG is a 1 bit field used to specify if this message is a fragment.

NOT-FRAGMENT	: 0x0
FRAGMENT	: 0x1

LFRG is a 1 bit field used to specify whether this is the last fragment in a complete message.

NOT-LAST-FRAGMENT	: 0x0
LAST-FRAGMENT	: 0x1

EXP is a 1 bit field that specifies if this protocol is experimental or not. The EXP bit can be set to denote that this version of the protocol is private, in-house. This makes it possible to have private protocol versions that don't collide with IETF standards.

Version is a 7 bit field that indicates the version of the P2PSIP protocol being used.

Version1.0 : 0x1

DHT is an 8 bit field that specifies the DHT algorithm being used. Because of its simplicity, implementations **MUST** support the Chord DHT algorithm, though other DHT algorithms **MAY** be used (and likely will be)

Chord1.0 : 0x01
Bamboo1.0 : 0x02

HASH is an 8 bit field that specifies the hash algorithm used to generate IDs. All IDs used for an overlay must be calculated using the same algorithm. Implementations **MUST** support the SHA-1 algorithm, which produces a 160 bit hash value, though other hash algorithms **MAY** be used.

SHA-1 : 0x01

Security is an 8 bit field that indicates the security mechanism being used by participants in the overlay.

NONE : 0x01

R/r is a one bit field used to specify if this is a request or a response.

REQUEST : 0x0
RESPONSE : 0x1

Method is a 7 bit field that indicates the message method. There are four types of methods: peer, resource, transport, and DHT specific. DHT specific methods are defined by the DHT algorithm being used. The class of methods are specified by the first two bits:

PEER-METHODS	: 0x0
RESOURCE-METHODS	: 0x1
TRANSPORT-METHODS	: 0x2
DHT-SPECIFIC-METHODS	: 0x3

The remaining four bits are used to specify the peer, resource, and transport methods. The following list of methods **MUST** be understood and are shown with their full 6 bit representations (first two bits are method type, last four specify method):

PEER-JOIN	: 0x00
PEER-SEARCH	: 0x01
RESOURCE-GET	: 0x10
RESOURCE-PUT	: 0x11
RESOURCE-TRANSFER	: 0x12
TRANSPORT-OPEN	: 0x20
TRANSPORT-TUNNEL	: 0x21

The message Length is the count in bytes of the size of the message, not including the header. 24 bits.

The Destination ID is a 160 bit identifier (either Peer-ID or Resource-ID) that uniquely identifies destination peer or resource.

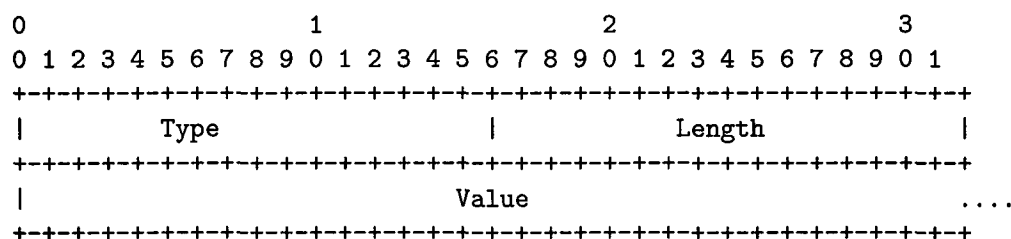
The Source ID is a 160 bit identifier (either Peer-ID or Resource-ID) uniquely identifying the sender.

The Transaction ID is a unique 64 bit number that identifies this transaction and also serves as a salt to randomize the request and the response. Responses use the same Transaction ID as the request they correspond to.

The Overlay field is the 32 bit checksum/hash of the overlay being used. The CRC-32 checksum MUST be used to convert the variable length string representing the overlay name into a 32 bit value. The 32bit Overlay field does not provide a security check, but rather an integrity check. The CRC-32 checksum Overlay field provides a “digital signature” representing the overlay name so that peers can verify that they are interacting with overlay they are intending to interact with.

B.1.2 Message Attributes

Following the fixed length header, the message contains 0 or more attributes. Message attributes are TLV (type-length-value) encoded, with 16 bit type, 16 bit length, and variable value:



The following message attribute types are defined:

```
RESPONSE-CODE      : 0x0001
SOURCE-INFO        : 0x0002
```

```

RESOURCE           : 0x0003
ROUTE-LOG          : 0x0004
REDIRECT           : 0x0005
SIGNATURE          : 0x0006
SDP                : 0x0007
APPLICATION        : 0x0008

```

The SIGNATURE attribute MUST be the last attribute within a message when originally sent. ROUTE-LOG attributes may be added following the SIGNATURE.

The attribute type field provides information about how to parse the value. Many of the attribute types listed here are composite types, i.e. the value is itself a set of TLV-encoded attributes. In this case, the length of the parent attribute, such as SOURCE-INFO, is the length of the entire set of member attributes to that SOURCE-INFO, and a parser can skip decoding the entire composite attribute if it is not interested in its contents.

The following tables indicate which attributes appear in which messages. An M indicates that inclusion of the attribute in the message is mandatory, O means its optional, C means it's conditional based on some other aspect of the message, and N/A means that the attribute is not applicable to that message type.

Attr.	Peer Join Req	Peer Join Resp.	Peer Search Req.	Peer Search Resp.
-----	-----	-----	-----	-----
RESPONSE-CODE	N/A	M	N/A	M
SOURCE-INFO	O	M	O	M
RESOURCE	N/A	N/A	N/A	N/A
ROUTE-LOG	O	O	O	O
REDIRECT	N/A	O	N/A	O
SIGNATURE	O	O	O	O
SDP	O	C	O	C
APPLICATION	N/A	N/A	N/A	N/A

Attr.	Res. Get Req.	Res. Get Resp.	Res. Put Req.	Res. Put Resp.	Res. xfer Req.	Res. xfer Resp.
RESPONSE-CODE	N/A	M	N/A	M	N/A	M
SOURCE-INFO	O	M	O	M	O	M
RESOURCE	N/A	M	M	N/A	M	N/A
ROUTE-LOG	O	O	O	O	O	O
REDIRECT	N/A	O	N/A	O	N/A	O
SIGNATURE	O	O	O	O	O	O
SDP	O	C	O	C	O	C
APPLICATION	N/A	N/A	N/A	N/A	N/A	N/A

Attr.	Transport OPEN Req.	Transport OPEN Resp.	Transport tunnel Req.	Transport tunnel Resp.
RESPONSE-CODE	N/A	M	N/A	M
SOURCE-INFO	O	M	O	M
RESOURCE	N/A	N/A	N/A	N/A
ROUTE-LOG	O	O	O	O
REDIRECT	N/A	O	N/A	O
SIGNATURE	O	O	O	O
SDP	M	M	N/A	N/A
APPLICATION	N/A	N/A	M	O

The Length refers to the length of the actual useful content of the Value portion of the attribute, measured in bytes. Since STUN aligns attributes on 32 bit boundaries, attributes whose content is not a multiple of 4 bytes are padded with 1, 2 or 3 bytes of padding so that they are a multiple of 4 bytes. Such padding is only needed with attributes that take freeform strings, such as USERNAME and PASSWORD. For attributes that contain more structured data, the attributes are constructed to align on 32 bit boundaries. The value in the Length field refers to the length of the Value part of the attribute prior to padding - i.e., the useful content. Consequently, when parsing messages, implementations will need to round up the Length field to the nearest multiple of four in order to find the start of the next attribute.

The RESPONSE-CODE is based on STUN's ERROR-CODE attribute, and provides the response status code for the message. The RESPONSE-CODE attribute value is a numeric value in the range of 100 to 600 plus a textual reason phrase encoded in UTF-8. The length of the reason phrase is the length prior to padding.

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               XXXXX                |Class|      Number       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          Reason Phrase (variable)                                     ..
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

The class represents the hundreds digit of the response code. The class value **MUST** be between 1 and 7. The number represents the response code modulo 100, and its value **MUST** be between 0 and 99.

- **200 (OK):** Indicates a successful request. The information returned in the response will depend on the request method.
- **302 (Moved Temporarily):** The requesting peer **SHOULD** retry the request at the new address specified in the 302 response message.

- 404 (Not Found): The resource or peer cannot be found or does not exist.
- 408 (Request Timeout): A response to the request has not been received in a suitable amount of time. The requesting peer MAY resend the request at a later time.
- 498 (Incompatible with Overlay) A peer receiving the request is using a different overlay, DHT algorithm, or hash algorithm.
- 499 (UnWilling To Proxy) A peer receiving the request is unwilling to support the Routing mechanism specified in the Routing field of the message header.

B.1.2.2 SOURCE-INFO and PEER-INFO

The SOURCE-INFO attribute provides information about the (source) peer sending the request or response.

The SOURCE-INFO attribute is of PEER-INFO type, which itself contains several attributes. Because a number of other attributes are also of type PEER-INFO, they share the same sub-attributes. PEER-INFOs contain the following attributes:

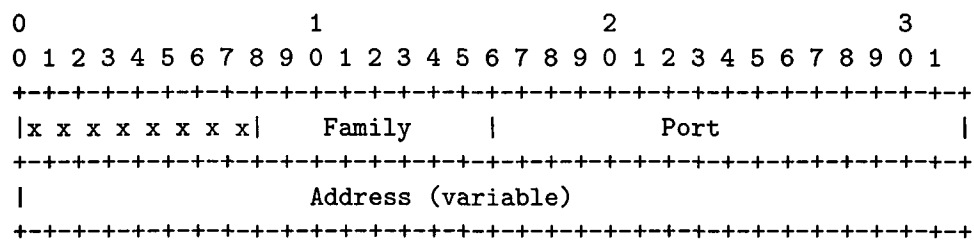
PEER-ID	: 0x0101
PEER-NAME	: 0x0102
PEER-IP-PORT	: 0x0103
PEER-EXPIRATION	: 0x0104
PEER-CERTIFICATE	: 0x0105
PEER-SIGNATURE	: 0x0106

PEER-ID

PEER-NAME

A PEER-INFO may contain multiple PEER-NAME attributes.

PEER-IP-PORT is based on STUN's MAPPED-ADDRESS attribute, and provides IP address and port pairs that can be used to contact the peer. PEER-IP-PORTS consist of an eight bit address family, and a sixteen bit port, followed by a variable length value representing the IP Address.



The first 8 bits are used for alignment and are ignored. The family is 0x01 for IPv4 and 0x02 for IPV6. The port is the mapped port represented by a 16 bit network byte order number. If the family is IPV4, the Address is 32 bits. For IPV6, the address is 128 bits (again, in network byte order).

A PEER-INFO may contain multiple PEER-IP-PORT values. The meanings and uses of these values are not defined by this specification. Reliable communication in environments with NATs or firewalls can only be established through the use of ICE as provided by the OPEN method.

PEER-EXPIRATION

The PEER-EXPIRATION attribute value provides the amount of time in seconds the peer information should be considered valid. The value **MUST** be a numeric value representing the relative time in seconds at which the peer will expire. The value **SHOULD** be 32bits and **MUST** be encoded in network byte order.

PEER-CERT

The PEER-CERT attribute contains the security certificate used to authenticate the peer. This optional attribute is defined by the security mechanism being used.

PEER-SIGNATURE

The PEER-SIGNATURE attribute value contains the signature of the message. This optional PEER-INFO attribute is defined by the security mechanism being used. The PEER-SIGNATURE **MUST** be the last attribute in the PEER-INFO and is applied to all preceding attributes in the PEER-INFO.

B.1.2.3 RESOURCE and RESOURCE-INFO

The RESOURCE attribute provides information about resources and resource registrations.

The RESOURCE attribute type is a RESOURCE-INFO, which itself contains several attributes. RESOURCE-INFO contain the following attributes.

RESOURCE-INFO.KEY : 0x0201
RESOURCE-INFO.BODY : 0x0202

RESOURCE-INFO.KEY

The RESOURCE-INFO.KEY attribute represents the hashable string that uniquely identifies a resource. It MUST be present in an RESOURCE- INFO attribute. The RESOURCE-INFO.KEY value is variable length value and the encoded length is its length prior to padding.

RESOURCE-INFO.BODY

The RESOURCE-INFO.BODY attribute provides information about all of the resource types. At least one RESOURCE-INFO.BODY attribute MUST be present in a RESOURCE-INFO. Additional RESOURCE-INFO.BODY attributes MAY be added to the RESOURCE-INFO. The RESOURCE-INFO.BODY attribute value consists of several attributes.

RESOURCE-INFO.BODY.ENTRY : 0x0303
RESOURCE-INFO.BODY.PARAMETER : 0x0304
RESOURCE-INFO.BODY.EXPIRATION : 0x0305
RESOURCE-INFO.BODY.SIGNATURE : 0x0306
RESOURCE-INFO.BODY.PEER-INFO : 0x0307
RESOURCE-INFO.BODY.CERTIFICATE : 0x0308

RESOURCE-INFO.BODY.ENTRY

The RESOURCE-INFO.BODY.ENTRY attribute contains the resource information. It MUST be present in the RESOURCE-INFO.BODY attribute and only one RESOURCE-INFO.BODY.ENTRY can be specified in each RESOURCE-INFO.BODY. The RESOURCE-INFO.BODY.ENTRY attribute is variable length and its encoded length is its length prior to padding..

RESOURCE-INFO.BODY.PARAMETER

RESOURCE-INFO.BODY.PARAMETER provides type parameters for each RESOURCE-INFO.BODY. For example, a RESOURCE-INFO.BODY.PARAMETER might indicate that the RESOURCE-INFO.BODY.ENTRY is of type "voicemail". 0 or more RESOURCE-INFO.BODY.PARAMETERs MAY be specified in a RESOURCE-INFO.BODY. RESOURCE-INFO.BODY.PARAMETER specifies characteristics of the RESOURCE-INFO.BODY. RESOURCE- INFO.BODY.PARAMETER consist of a name, an operator, and a value.

```
RESOURCE-INFO.BODY.PARAMETER.NAME      : 0x0309
RESOURCE-INFO.BODY.PARAMETER.OP         : 0x030A
RESOURCE-INFO.BODY.PARAMETER.VALUE      : 0x030B
```

RESOURCE-INFO.BODY.PARAMETER.NAME is the textual description of the resource type encoded in UTF-8. The length of the description is encoded as its length prior to padding. The RESOURCE- INFO.BODY.PARAMETER.NAME MUST be specified in the RESOURCE- INFO.BODY.PARAMETER..

RESOURCE-INFO.BODY.PARAMETER.OP is the operator defining the relationship between the RESOURCE-INFO.BODY.PARAMETER.NAME and the RESOURCE-INFO.BODY.PARAMETER.VALUE. It MUST be specified in the RESOURCE- INFO.BODY.PARAMETER. It is an 8 bit numeric value with the following operators defined:

```
=      : 0x1
!=     : 0x2
>      : 0x3
>=     : 0x4
```

< : 0x5
<= : 0x6

RESOURCE-INFO.BODY.PARAMETER.VALUE is the textual right hand side of the operator encoded in UTF-8. It MUST be specified in the RESOURCE- INFO.BODY.PARAMETER attribute.

RESOURCE-INFO.BODY.EXPIRATION

The RESOURCE-INFO.BODY.EXPIRATION attribute value provides the amount of time in seconds the RESOURCE-INFO.BODY information should be considered valid. The value MUST be a numeric value representing the relative time in seconds at which the RESOURCE-INFO.BODY will expire. The value SHOULD be 32bits and MUST be encoded in network byte order

RESOURCE-INFO.BODY.SIGNATURE

The RESOURCE-INFO.BODY.SIGNATURE attribute contains the signature of the message. This optional RESOURCE-INFO.BODY attribute is defined by the security mechanism being used.

RESOURCE-INFO.BODY.PEER-INFO

The RESOURCE-INFO.BODY.PEER-INFO attribute contains information about the peer that is storing the resource and how to access the physical resource stored on that peer. RESOURCE-INFO.BODY.PEER-INFO is a type PEER-INFO attribute, see the SOURCE-INFO and PEER-INFO sections for syntax details, and it is an optional RESOURCE- INFO.BODY attribute.

PEER-INFO might be included in a resource registration to provide information on how to contact a particular device through the overlay (using TRANSPORT-TUNNEL)

rather with a traditional URL. This information might also be included as a uri-param or other protocol- specific method embedded in the RESOURCE-INFO.BODY.ENTRY.

B.1.2.4 ROUTE-LOG

ROUTE-LOG provides diagnostic information about the path taken by the request so far and in this manner it is similar in function to SIP's Via header field. At each hop peers **MUST** append their P2PSIP implementation version, transport protocol, and PEER-INFOs in a new ROUTE-LOG attribute. The order of the ROUTE-LOGs in the message is used to determine the order of the peers were traversed along the path. The first ROUTE-LOGs attribute corresponds to the peer at the first hop along the path, and each subsequent ROUTE-LOG attribute corresponds to the peer at the next hop along the path.

ROUTE-LOG.P2PSIP-VERSION	: 0x0400
ROUTE-LOG.TRANSPORT	: 0x0401
ROUTE-LOG.PEER-INFO	: 0x0402

ROUTE-LOG.P2PSIP-VERSION

ROUTE-LOG.P2PSIP-VERSION contains the protocol and version of the P2PSIP implementation used at each hop along the path. The value of ROUTE-LOG.P2PSIP-VERSION is a variable length value. Its encoded length is its length prior to padding.

ROUTE-LOG.TRANSPORT

ROUTE-LOG.TRANSPORT specifies the transport protocol being used at each hop along the path. ROUTE-LOG.TRANSPORT values are integers numbers and the following transport protocol values **MUST** be understood:

ROUTE-LOG.TRANSPORT.TCP	: 0x0006
ROUTE-LOG.TRANSPORT.UDP	: 0x0011
ROUTE-LOG.TRANSPORT.TLS	: 0x0038
ROUTE-LOG.TRANSPORT.SCTP	: 0x0084

TLS means, TLS over TCP. The hex numbers specified are taken from IANA's assigned Internet Protocol Numbers, which are used to fill in the "Protocol" and "Next Header" fields in IPv4 and IPv6, respectively.

ROUTE-LOG.PEER-INFO

ROUTE-LOG.PEER-INFO provides information about the peer at each hop along the path. The syntax of ROUTE-LOG.PEER-INFO is identical to the PEER-INFO syntax described above in the SOURCE-INFO and PEER-INFO section.

B.1.2.5 REDIRECT

The REDIRECT attribute is intended to be used to specify a new peer to contact and is only present in response messages. The REDIRECT attribute value is a PEER-INFO, the syntax of which is described in the SOURCE-INFO and PEER-INFO section.

B.1.2.6 SIGNATURE

SIGNATURE attribute is a signature of the header fields. The SIGNATURE is applied to the entire message from the 8th byte of the header to the end of the attribute preceding the SIGNATURE. The SIGNATURE MUST be the last attribute of the message except for ROUTE- LOGs appended subsequently. The SIGNATURE attribute value is dependent on the security mechanism being used as indicated in the Security header field.

B.1.2.7 SDP

The SDP attribute is used in a TRANSPORT-OPEN to establish a direct connection between peers. All RELOAD PEERs **MUST** provide an ICE implementation. This implementation **SHOULD** be a Full ICE implementation, rather than a Lite implementation. The SDP describes the type of connection that will be established. “application/reload” and “application/sip” are the two primary connections that a P2PSIP peer will need, but others such as “application/http” and “application/msrp” may be used as well. Both regular ICE and ICE-TCP [69] may be used for RELOAD connections.

A TRANSPORT-OPEN message indicates a desire for a direct connection between the source and destination peers. The TRANSPORT-OPEN message itself is routed along the established overlay network and is used as a control connection for the ICE negotiation. The peers may already be exchanging messages over the overlay, and one may decide to open the connection to improve efficiency. Many times the TRANSPORT-OPEN will be used when a peer is updating its routing table and discovers a peer that should be in its routing table based on its DHT algorithm.

TRANSPORT-OPEN is not required to be used for every connection. A PEER-INFO may contain multiple PEER-IP-PORT attributes for a particular peer and a connection may be attempted with those without use of TRANSPORT-OPEN. The exact decision of when to use TRANSPORT-OPEN is not specified by this specification.

Support for ICE mandates that all RELOAD peers **MUST** implement STUN on their ports and multiplex between RELOAD and STUN traffic. Peers must also support keepalives on their ports.

The straightforward mechanism described here works quite well for a single offer-answer exchange. OPEN ISSUE: Because RELOAD does not require the various mid-

session media changes that are required for SIP, we may not need to support subsequent offer/answer exchanges to update or improve the same session.

Peers may continue to exchange RELOAD messages over the overlay until the connection is established. Note that this only applies to recursive routing.

B.1.2.8 APPLICATION

The APPLICATION attribute is used to tunnel application-specific messages between peers in TRANSPORT-TUNNEL messages. APPLICATION attribute has three sub-attributes, each of which MUST be included in an APPLICATION attribute:

APPLICATION.NAME	: 0x0403
APPLICATION.BODY	: 0x0404
APPLICATION.DIALOG	: 0x0405

Each has a type of string. APPLICATION.NAME should contain the name of the protocol, i.e. "sip", "http", or "reload".

APPLICATION.BODY contains the actual message to be delivered to the application.

APPLICATION.DIALOG contains a 32-bit integer that uniquely identifies the dialog the current message is a part of. It is intended to help identify request/response pairs and its specific meaning is particular to the protocol being tunneled.

Multiple methods are possible for electing when to tunnel SIP traffic. In particular, the SIP UA portion of the P2PSIP peer may not wish to be aware of whether its traffic is being sent directly, routed over a TUNNEL, or sent through a connection opened with TRANSPORT-OPEN.

The response to a TRANSPORT-TUNNEL message MAY contain an APPLICATION attribute. If an application does not produce a response message or takes too long, the peer SHOULD produce a response with a 200 response code indicating the message was received. Subsequent responses will be sent in their own TRANSPORT-TUNNEL request with the same APPLICATION.DIALOG as previous messages.

B.1.3 Adding New Attributes

This specification is designed to allow future revisions of this protocol and pluggable DHT algorithms to add new attributes. DHT algorithms SHOULD define attributes that specify how to exchange DHT Routing Table, Connection Table, and Neighbor information. The exact details and syntax of these attributes is left to the DHT algorithms.

Attributes numbering from 0x0600 to 0x06FF are reserved for the pluggable DHT algorithms. Each DHT algorithm may use any attributes in this range. Because the DHT algorithm in use is specified in the header, there are no concerns about collisions between two different DHT algorithms: they may each use the same attribute for different purposes. The pluggable security algorithms may use attributes from 0x0800 to 0x08FF subject to the same rules.

B.2 Hash Algorithms and Identifiers

B.2.1 Hash Algorithms

All IDs used for an overlay must be calculated using the same hash algorithm. Implementations MUST support the SHA-1 algorithm, which produces a 160 bit hash value. The

hash algorithm used is specified in the HASH field of the message header, described in the Message Header section.

B.2.1.1 Peer-IDs

Peer-IDs **MUST** be acquired or generated in the same manner for all peers in the overlay. Peer-IDs **MAY** be generated by a peer hashing information specific to that peer, such as the IP address and port number, a peer creating a UUID, or a central authority assigning Peer-IDs to peers. For example, the particular DHT algorithm being used **MAY** specify an alternate mechanism for determining Peer-ID. Similarly, some security models may assign Peer-IDs from a central authority.

Hashing the IP address and Port number to create a Peer-ID provides minimal security, but only if the Peer-ID can be used to verify the IP address of the peer. This is limited security in that it limits the number of Peer-IDs a malicious user could obtain to the number of IP addresses that user controls. In the case of an administrator or IPv6 addresses a user could obtain a large number of Peer-IDs. Furthermore, NATs make it challenging to verify the IP addresses of peers. Hashing IP addresses and port numbers (possibly along with some other peer specific information) **MAY** be used to create unique Peer-IDs, but **SHOULD NOT** be used to provide security for the overlay.

If the security mechanism does not define how Peer-IDs are generated, UUIDs **SHOULD** be created by peers and used as their Peer-IDS. The system constructs a version 1 UUID according to RFC 4122 [57]. If the application cannot guarantee that the timestamp obtained for the UUID is globally unique (i.e. there is no system-wide shared stable store for the generator state), it **MUST** append to the UUID “;”, the protocol (“udp” or “tcp”), “;”, and the port number allocated for use by the application. The resulting

string is then hashed to generate a Peer-ID. UUIDs provide unique identifiers, but with no pretense of security.

The security mechanism being used by the overlay MAY define how Peer-IDs are generated or acquired. For example, a certificate authority can be responsible for assigning Peer-IDs to all of the peers in the overlay. Security algorithms are not discussed in detail in this specification.

B.3 Message Routing

When a peer sends a message within the DHT, it begins by calculating the target ID it is attempting to locate, which might be its own location in the DHT, or a user's registration, for which it hashes the user's URI to obtain the appropriate Resource-ID. It then consults its routing table, and its other neighbor peers, for the closest peer it is aware of to the target ID.

The messages in the overlay MAY be routed either iteratively or recursively. The Routing as described above SHOULD be used to indicate if the next node should process the message using a recursive or iterative mechanism. If the header is omitted, the receiving peer may process the message either recursively or iteratively. If the receiving peer cannot or does not want to route the message in the manner specified in the Routing header field, a 499 Unwilling to Proxy message SHOULD be returned to the requesting peer.

If the Routing header is iterative, the contacted peer MUST determine if it is responsible for that target ID. If it is not, then the contacted peer MUST issue a 302 redirect pointing the search peer toward the best match the contacted peer has for the target ID.

The searching peer then contact the peer to which it has been redirected and the process iterates until the responsible peer is located.

In recursive routing, the peer sends a message to the peer it knows that is nearest to the target. If the contacted peer is not responsible for the target ID, it **MUST** forward the query to the nearest peer to the target that it knows, and the process repeats until the target is reached. This process follows standard proxy behavior in RFC 3261.

At each step, the contacted peer must only examine the first 12 bytes to ensure version compatibility, type of routing requested, and R/r, then look at the source or destination ID to determine how to route the message. Parsing the body of the message or the other headers is not required unless the contacted peer is responsible for the destination ID.

[28] provides an analysis of the tradeoffs between iterative and recursive routing. Although that analysis is based on dSIP, the functionality and message count of RELOAD is very similar.

B.3.1 SIP Session Establishment

If the peer needs to have the SIP session establishment routed through the overlay, it **MAY** use the **TRANSPORT-OPEN** or **TRANSPORT-TUNNEL** messages to request either that a new connection be established or that intermediate nodes proxy the **INVITE** over the overlay on their behalf.

B.4 Peer and DHT Operations

B.4.1 Peer Registration

After a peer has located an initial bootstrap peer, the process of joining the overlay is started by constructing a PEER-JOIN message and sending it to the bootstrap peer. Third party registration MAY NOT be used for registering peers into the overlay, and attempts to do so MUST be rejected by the peer receiving such a request (although third party registrations are used for other purposes, as described below). The peer MUST construct according to the rules outlined below.

B.4.1.1 Constructing a Peer Registration

The Method field in the message MUST be set to PEER-JOIN, 0x00, and the R/r bit field in the message header MUST be set to 0x0.

The Source ID field of the header MUST include only the 160bit identifier that specifies the joining peer's Peer-ID. The Destination ID header for a PEER-JOIN MUST also be set to the joining peer's Peer-ID.

The joining peer MAY leave the Overlay header field set to 0x0 for its initial registration message, but MUST set this parameter to the checksum of the name of the overlay it is joining as soon as it receives a response from the bootstrap peer.

The joining peer MUST set the DHT header field to the name of the DHT it will be using and MUST set the Version header field to be 0x01 for this version of the protocol.

The joining peer MUST set the Hash header field to the name of the Hash algorithm it will use and as soon as it receives a response from the bootstrap peer.

The joining peer *MAY* leave the Security header field set to 0x0 for its initial registration message, but *MUST* set this parameter to the security type of the overlay it is joining as soon as it receives a response from the bootstrap peer.

The joining peer *SHOULD* also list its peer information in a SOURCE- INFO attribute in the body of the message. See the sections on SOURCE-INFO and PEER-INFO for details on the syntax of this attribute.

Assume that a peer running on IP address 10.4.1.2 on port 5060 attempts to join the network by contacting a bootstrap peer. Further assume that the overlay is using IP:port hashing to determine Peer-IDs and that 10.4.1.2 hashes to 463ac4b449 under SHA-1 (using a 10 digit hash for example simplicity), and the least significant bits are replaced with the port number, yielding 463ac413c4 and that the overlay name is chat and the DHT is dhtalg1.0. An example message would look like this:

```
Version: 0x01
Method: 0x00
DHT: 0x0
Security: 0x0
Hash: 0x0
Source ID: 463ac413c4
Destination ID: 463ac413c4
-----Attributes-----
SOURCE-INFO:
    PEER-ID: 463ac413c4
    PEER-IP-PORT: 10.4.1.2:5060
    PEER-EXPIRATION: 300
```

B.4.1.2 Processing the Peer Registration

If the peer examines the overlay parameters and determines that this is not an overlay the peer participates in, the peer *MUST* reject the message with a 497 Incompatible

With Overlay response. Likewise if the peer examines the DHT field determines that the algorithm specified is not compatible with its algorithm, the peer **MUST** reject the message with a 488 Incompatible With Overlay.

Routing the Peer Registration

The bootstrap peer **SHOULD** verify that the Peer-ID corresponds to the peer listed in the URI by validating the peer's credentials, if any are present. If these do not match, the message **SHOULD** be rejected with a response of 493 Undecipherable. The bootstrap peer examines the Peer-ID to determine if it corresponds to the portion of the overlay the bootstrap peer is responsible for. If it does, the peer will handle the PEER-JOIN request itself. If not, the bootstrap peer will either provide the joining peer with information about a peer closer to the area of the overlay where the joining peer's Peer-ID is stored (iterative routing) or forward the request along the closest peer it knows about (recursive routing). If a Routing header is set to proxy, the peer **MUST** use a recursive routing mechanism, and if it is set to redirect, the peer **MUST** use an iterative routing mechanism. If the peer is unwilling to use this routing mechanism, it **MUST** reject the message with a 499 response. In the event that the Routing header is set to unspecified, the peer may choose either mechanism.

In the case of iterative routing, if the receiving peer is not responsible for the area of the hash table where Peer-ID should be stored, the peer **SHOULD** generate a 302 message. The receiving peer **MUST** look in its list of neighbors or in the routing table to find the peer with Peer-ID nearest the to joining peer's Peer-ID, and use it to create a Redirect attribute in the form of a PEER-INFO, as specified in the REDIRECT section. The response **MUST** also contain a SOURCE-INFO attribute. The Source ID and Destination ID of the reply **MUST NOT** be changed. This response is sent to the joining peer.

In the case of recursive routing, if the receiving peer is not responsible for the area of the hash table where the Peer-ID should be stored, the receiving peer should forward the

request to the peer it knows about that is closest to the Peer-ID. When using recursive routing, the reply from the admitting (responsible) peer is routed back through the overlay to the bootstrap peer, which then forwards the reply to the joining peer. A receiving peer receiving a join request from the joining peer and proxying that request **MUST** route that message to the admitting peer in a TRANSPORT-TUNNEL message with an APPLICATION attribute whose APPLICATION.NAME is "reload". This technique is necessary because RELOAD must support both reliable authentication and NAT-traversal for routing its initial join messages. If the initial join is routed directly to the admitting peer, there is no way for the reply to be routed back to the joining peer because the joining peer is not yet properly placed in the overlay. The reply must be routed through the bootstrap peer, because it is the only peer known to have a connection to the joining peer. Using a TRANSPORT-TUNNEL for the transaction allows the joining peer to authenticate the message properly but encapsulates it so that a reply can be routed back to the bootstrap peer. A recursive join request **MUST** also include a TRANSPORT-OPEN to establish a direct connection with the admitting peer.

A peer **MUST NOT** add a new peer to its routing table or redirect requests to that new peer until it has successfully established a connection with that peer itself. By redirecting a message to another peer, the contacted peer indicates that it believes that peer to be alive and that it is willing to route messages to it for NAT and Firewall traversal purposes.

Admitting the Joining Peer

The admitting peer recognizes that it is presently responsible for this region of the hash space – that is, it is currently the peer storing the information that this Peer-Id will eventually be responsible for. The admitting peer knows this because the joining peer's Peer-ID is closest to its own Peer-ID. The admitting peer is responsible for helping the joining peer become a member of the overlay and the admitting peer **MAY** perform additional security checks. Once any challenge has been met, the admitting will reply

with a 200 OK message to the joining peer.

The admitting peer **MUST** reply with a 200 response if the admitting peer's Peer-ID is the closest to the joining peer's Peer-ID. Each DHT algorithm **MAY** choose to define closest however they want, but the DHT algorithm **MUST** be able to deterministically find the closest Peer-ID. The admitting peer must populate any DHT-SPECIFIC attributes with all values required by the DHT routing protocol so that the joining peer can initialize its neighbors and routing table entries or sent additional DHT-SPECIFIC messages. Additionally, the admitting peer **MUST** include its peer information in the SOURCE-INFO attribute of the response.

B.4.2 Peer Query

B.4.2.1 Constructing a Peer Query Message

The peer looks for the routing table entry or neighbor peer that is closest to the ID they are searching for. If the routing table has not yet been filled, then the peer may send the request to any peer it has available, including their other neighbor peers or even some bootstrap peer. While these initial searches may be less efficient, they will succeed.

The Method field in the message **MUST** be set to PEER-SEARCH, 0x01, and the R/r bit field in the message header **MUST** be set to 0x0.

The Source ID field of the header **MUST** include only the 160bit identifier that specifies the searching peer's Peer-ID. The Destination ID header for a PEER-SEARCH **MUST** be set to the Peer-ID that is being searched for.

The searching peer **MUST** set the Overlay header field to the name of the overlay it is participating in.

The searching peer **MUST** set the DHT header field to the name of the DHT it is using and **MUST** set the Version header field to be 0x01 for this version of the protocol.

The searching peer **MUST** set the Hash header field to the name of the Hash algorithm it is using.

The searching peer **MUST** set the Security field to the security type of the overlay.

The searching peer **SHOULD** also list its peer information in a SOURCE- INFO attribute in the body of the message. See sections SOURCE-INFO and PEER-INFO for details on the syntax of this attribute.

Assume that a peer wants to determine who is responsible for Peer-ID 4823affe45. Further assume that the peer uses SHA-1 (using a 10 digit hash for example simplicity), and that the overlay name is chat. An example message would look like this (Note that there is no indication of the source or destination IP addresses or ports, which allows this message to be routed, unchanged, to its destination):

```
Version: 0x01
Method: 0x01
Source ID: 463ac413c4
Destination ID: 4823affe45
-----Attributes-----
SOURCE-INFO:
    PEER-ID: 463ac413c4
    PEER-IP-PORT: 10.4.1.2:5060
    PEER-EXPIRATION: 300
```

B.4.2.2 Processing Peer Query Message

If the peer examines the overlay, hash, and DHT fields overlay the peer participates in or that the algorithms are incompatible, the peer **MUST** reject the message with a 498

Incompatible With Overlay response.

Routing the Peer Query Message

The receiving peer SHOULD NOT alter any of its internal values such as successor or predecessor in response to this message, since it is a query. This decision was made for security reasons – the peer should only learn about other peers it has directly queried. Otherwise, the message is processed and routed as a peer registration until the responsible peer is reached.

Responding to the Peer Query Message

If the receiving peer is responsible for the region that the search key lies within, it MUST respond to the query. If the receiving peer's Peer-ID exactly matches the search key, it MUST respond with a 200 OK message. If it is responsible for that region, but its Peer-ID is not the search key, it MUST respond with a 404 Not Found message.

B.4.3 Populating the Joining Peer's Routing Table

Once admitted, the joining peer SHOULD populate its routing table and locate neighbors by sending DHT-SPECIFIC messages or by issuing queries for peers with the appropriate identifiers. If the admitting peer provided neighbor or routing table information in its response, the joining peer MAY use this information to construct a temporary routing table and neighbor information and use this temporary table in the queries to populate the table.

B.4.4 Transferring User Registrations

When a new peer joins, it splits the area in the hash space the admitting peer is responsible for. Some portion of the resources the admitting peer was responsible for may now

be the responsibility of the joining peer, and these resources are handed to the joining peer by means of RESOURCE-TRANSFER messages.

B.4.5 Peers Leaving the Overlay Gracefully

Peers **MUST** send their registrations to their closest peer before leaving the overlay, as described in the section above. Additionally, peers **MUST** unregister themselves with their symmetric neighbors (if the DHT routing algorithm uses symmetric neighbors in any form). These graceful exit messages are constructed exactly the same as one used to join, with the following exceptions. The PEER- EXPIRATION of the SOURCE-INFO **MUST** be set to 0.

B.4.6 NAT and Firewall Traversal

The filtering properties of NATs and firewalls can lead to non- transitive connectivity. Typically this will manifest itself in a peer receiving a 302 redirecting it to another peer that it cannot contact, most likely because address dependent filtering is occurring. In that case, the peer **SHOULD** use a TRANSPORT-OPEN message routed to the target peer through an established RELOAD connection to open a RELOAD connection with the target peer.

B.4.7 Handling Failed Requests

When a request sent to another peer fails, the peer **MUST** perform searches to update its pointers. If the failed request was sent to a peer in the routing table or a neighbor peer, then DHT-SPECIFIC message should be sent or the searches discussed in the Populating the Joining Peer's Routing Table section should be performed.

B.5 Resource Operations

The most important element of resource operations within the P2PSIP DHT is that they are performed essentially the same as if using a conventional SIP registrar, except that the registrar responsibilities are distributed among the DHT members using the RELOAD protocol.

B.5.1 Resource Registrations

When a peer is in the overlay, it must register the contacts for users and other resources for which it is responsible into the overlay. This differs from the registrations described above in that these registrations are responsible for entering a resource name to resource body mapping into the overlay as data, rather than joining a peer into the overlay. Resources are registered in the overlay by sending a RESOURCE-PUT message.

The Method field in the message MUST be set to RESOURCE-PUT, 0x11, and the R/r bit field in the message header MUST be set to 0x0.

The Destination ID field of the header MUST be set to the 160bit identifier that specifies the Resource-ID of the resource body being registered. The Source ID header MUST be set to the Peer-ID of the peer doing the RESOURCE-PUT.

The RESOURCE-PUT request SHOULD also include a SOURCE-INFO attribute in the body of the message. See the sections on SOURCE-INFO and PEER-INFO for details on the syntax of this attribute.

The resource registration MUST include at least one RESOURCE-INFO attribute, with at least one RESOURCE-INFO.BODY attribute in the body of the RESOURCE-PUT message. The RESOURCE-INFO.KEY specifies the name of the resource and the

RESOURCE-INFO.BODY specifies one of the key to resource body (registrations) mappings. If more than one RESOURCE-INFO.BODY is in the RESOURCE-INFO attribute, there is a one to many mapping between resource key and resource body values. Each RESOURCE-INFO.BODY MUST have an expires value specified in the RESOURCE-INFO.BODY.EXPIRATION attribute. EXPIRATION set to zero will be used to remove resource bodies.

The message is routed in a fashion exactly analogous to that described in the section on peer registration. In iterative routing algorithms, 302 messages are sent to indicate that the message is to be redirected to another peer. In recursive routing algorithms, the receiving peer SHOULD forward the request to the peer in its routing table that is closest to the Resource-ID. Once the message arrives at a destination that is responsible for that portion of the hash namespace, the peer responds with a 200 indicating a successful registration.

The registering peer SHOULD construct and register replica registrations.

B.5.2 Refreshing Resource Registrations

Resource registrations are refreshed exactly as described in RFC 3261, Section 10. Responsible peers should send a new registration with a valid expiration time prior to the time that the registration is set to expire.

Agents MAY cache the address where they previously registered and attempt to send refreshes to this peer, but they are not guaranteed success, as a new peer may have registered and may now be responsible for this area of the space. In such a case if iterative routing is being used, the peer will receive a 302 from the peer with which they previously registered, and should follow the same procedure for locating the peer they used in the initial registration.

B.5.3 Removing Resource Registrations

Resource registrations are removed exactly as described in RFC 3261, Section 10. Responsible peers **MUST** send a registration with expiration time of zero.

If it registered replica registrations, the sending peer **MUST** construct replica unregister messages and use these to unregister the replicas.

B.5.4 Querying Resource Registrations

Querying peers should send a RESOURCE-GET message to the peer in its routing table with Peer-ID closest to the Resource-ID of the resource being searched for.

The 200 response to a RESOURCE-GET **MUST** provide a RESOURCE-INFO with information about the requested resource. The RESOURCE-INFO.BODY **MAY** specify how to acquire the resource instead of containing the resource itself. For example, if the resource being searched for is a large voicemail file, the RESOURCE-INFO.BODY might contain a URL that can be used to fetch the resource. After obtaining the RESOURCE-INFO.BODY from the RESOURCE-GET response message, it is the responsibility of the UA to acquire the resource from the peer specified in the RESOURCE-INFO.BODY.

A P2P environment can do little to protect against an individual peer compromising the registrations it is responsible for. Accordingly, a UA cannot trust a response from a single peer, whether it indicates a successful search or an error. In the absence of other methods of verifying the response (such as having a certificate of the user being searched for and a signed registration that can be verified with the certificate) a UA should search for the primary registration and at least one replica. Because the locations the replicas are stored are unrelated to the location of the primary registration, a single attacker is

unlikely to be able to compromise both entries. As the overlay gains more peers and more replicas are searched for, the odds of a compromise are reduced.

B.5.5 SIP Session Establishment

When a caller wishes to send a SIP message (such as an INVITE, MESSAGE or SUBSCRIBE), the caller must first locate the peer where this callee's information resides using the resource search procedure described in the section titled Resource Location.

Establishing a session is done entirely in the normal SIP fashion after the user is located using the P2P resource query. Once the peer responsible for the Resource-ID is located, it will provide either a 200, providing a contact for the users UA, or will provide a 404 if the user is not registered. If a 200 with a valid contact is received, the call will then be initiated directly with the UAS of the called using the standard RFC 3261 fashion. If necessary for NAT traversal, the INVITE can be processed by routing it through the overlay using TRANSPORT-TUNNEL or through a connection established with TRANSPORT-OPEN.

B.5.6 Offline Storage

Delivery of messages to offline users, or voicemail for voice applications, requires storing that information for later retrieval. Storing user configuration information in a format accessible from the network also will allow a user to retrieve their profile from any computer. Cao et al. [25] describe an approach that separates the storage of resource location information from the actual storage of the offline resource. We believe that this approach is in agreement with the approach taken by the rest of this specification, which relies on the DHT overlay to store the registrar's location information, but relies on external, conventional methods for the actual connection. For offline storage, it also allows the use of

other standard protocols to store and retrieve the offline information, keeping the P2PSIP scope restricted to storing resource mappings.

B.6 Pluggable DHT Algorithm Requirements

All RELOAD peers **MUST** support the Chord pluggable DHT algorithm for compatibility. They **MAY** support additional pluggable algorithms. The requirements for new pluggable algorithms are defined in this section.

Pluggable algorithm **MUST** use Peer-IDs and Resource-IDs as defined in the Hash Algorithms and Identifiers section. Pluggable algorithms are free to define what hash algorithms they support, but they **MUST** clearly specify what they are.

A resource with Resource-ID *k* will be stored by the peer with Peer-ID closest to the Resource-ID. The definition of closeness may vary in different DHT algorithms, but each DHT algorithm **MUST** guarantee Resource-ID searches converge to exactly one peer responsible for that portion of the namespace. As peers enter and leave, resources may be stored on different peers, so the information related to them is exchanged as peers enter and leave. Redundancy is used to protect against loss of information in the event of a peer failure.

Each new DHT algorithm **MUST** define a value for the DHT 8 bit field to be used in the DHT field of the RELOAD message header, defined in Message Header Syntax section.

Each new DHT algorithm **MUST** define DHT-SPECIFIC Methods and any DHT-SPECIFIC attributes that are to be used and understood in RELOAD messages.

B.7 Security Considerations

Please see the security considerations in A.6 which apply here, in addition to the new (or superseded) sections below:

B.7.1 Protection Using ID Hashing

(this Section supersedes Section A.6.2.1 with respect to RELOAD.)

One method for determining Peer-IDs is hashing the peer's IP address and appending the port number. The security of this scheme depends on the ease with which an attacker can choose their own Peer-ID. Because the port number is only appended to the Peer-ID, an attacker gains nothing by selecting different ports on the same node. Assuming that the SHA1 hash used to calculate the Peer-ID is reliably random, the attacker's ability to succeed depends on the number of separate IP addresses that they are able to obtain from which to launch their attacks.

In the current predominantly IPV4 Internet, few attackers have access to more than a handful of IP addresses, perhaps a few hundred at worst. For a large-scale P2P system, this is unlikely to provide the ability to hijack a particular user ID or control a sufficient portion of the network to affect other peers, in particular when registrations are replicated at independent peers. Ultimately, however, a sufficiently skilled and provisioned attacker can compromise this scheme.

As the Internet migrates to IPV6, however, it is unclear that the assumption that few attackers have access to a significant range of IP addresses will remain true. Therefore, hashing IP addresses to Peer-IDs is assumed to provide a diminishing amount of security in the future.

Even in the IPV4 Internet, however, it is frequently impossible to verify another peer's IP address. Because of the common use of NAT in the modern Internet, IP addresses are not necessarily transitive or reflexive. Therefore, a peer cannot generate a valid Peer-ID for its IP address as seen by all other peers on the Internet. This problem makes hashed Peer-IDs essentially unverifiable in many situations.

Due to these shortcomings, we recommend UUIDs as the default technique for generating Peer-IDs in the absence of other security schemes. While not offering any level of security, they do a better job of ensuring uniqueness than allowing multiple peers behind NATs to claim the Peer-ID corresponding to 192.168.1.100:5060.

B.7.2 Protecting the Signaling

(this Section supersedes Section A.6.5 with respect to RELOAD.)

The goal here is to stop an attacker from knowing who is signaling what to whom. An attacker being able to observe the activities of a specific individual is unlikely given the randomization of IDs and routing based on the present peers discussed above. Furthermore, because messages can be routed using only the header information, the actual body of the RELOAD message can be encrypted during transmission.

B.7.3 Replay Attacks

Defense against replay attacks is discussed in [59].

Appendix C

Tables of Simulation Results

Table C.1: 10% NAT Density Results, Hops

Peer Lifetime (h)	SR Hops	SR SD	AR Hops	AR SD	DR Hops	DR SD
.5	6.50385	1.76157	1.55744	1.76899	1.64732	2.02494
1	6.50388	1.76449	1.55439	1.75743	1.64912	2.02767
2	6.50417	1.76109	1.55051	1.74739	1.64733	2.02482
4	6.50799	1.76207	1.56360	1.76442	1.66420	2.04889
6	6.50527	1.76446	1.55310	1.74894	1.65225	2.03242
8	6.50427	1.76482	1.54043	1.73020	1.63755	2.01145
12	6.50355	1.75821	1.53703	1.72418	1.63385	2.00548

Table C.2: 10% NAT Density Results, Time

Peer Lifetime (h)	SR Time (ms)	SR SD	AR Time (ms)	AR SD	DR Time (ms)	DR SD
.5	498.917	132.585	101.2910	185.286	175.852	426.854
1	498.942	132.806	98.4012	162.722	176.459	427.401
2	498.976	132.552	96.4549	149.128	175.977	426.848
4	499.382	132.636	96.2551	143.269	179.239	431.743
6	499.116	132.810	95.5596	139.571	176.641	428.278
8	498.962	132.829	94.9240	136.936	174.118	423.930
12	498.798	132.326	94.8111	135.094	173.612	422.841

Table C.3: 30% NAT Density Results, Hops

Peer Lifetime (h)	SR Hops	SR SD	AR Hops	AR SD	DR Hops	DR SD
.5	6.49940	1.76723	2.67211	2.74257	2.94197	3.12799
1	6.50743	1.75905	2.66162	2.72272	2.94539	3.13031
2	6.50580	1.76554	2.65252	2.71101	2.94310	3.12971
4	6.50098	1.76589	2.66021	2.71001	2.95678	3.13533
6	6.50325	1.76435	2.66102	2.70967	2.95913	3.13714
8	6.50266	1.75981	2.65728	2.70565	2.95549	3.13409
12	6.50181	1.76632	2.65906	2.70682	2.95816	3.13631

Table C.4: 30% NAT Density Results, Time

Peer Lifetime (h)	SR Time (ms)	SR SD	AR Time (ms)	AR SD	DR Time (ms)	DR SD
.5	498.493	133.001	194.226	294.647	439.439	653.114
1	499.240	132.402	181.562	256.432	439.974	653.431
2	499.099	132.885	174.612	233.819	439.415	653.239
4	498.598	132.900	171.938	221.655	442.214	654.575
6	498.848	132.791	171.030	217.191	442.751	654.800
8	498.816	132.453	170.176	214.697	442.043	654.417
12	498.695	132.935	170.137	212.612	442.650	654.679

Table C.5: 60% NAT Density Results, Hops

Peer Lifetime (h)	SR Hops	SR SD	AR Hops	AR SD	DR Hops	DR SD
.5	6.49887	1.76476	4.35354	3.07188	4.89498	3.46512
1	6.50345	1.76057	4.33448	3.04819	4.90436	3.46519
2	6.50475	1.76177	4.31725	3.03694	4.90074	3.46669
4	6.50404	1.76077	4.32194	3.02878	4.91503	3.46447
6	6.50739	1.76107	4.35833	3.02458	4.96041	3.46057
8	6.50743	1.76918	4.32680	3.02903	4.92452	3.46761
12	6.50333	1.75862	4.32194	3.02313	4.92083	3.46309

Table C.6: 60% NAT Density Results, Time

Peer Lifetime (h)	SR Time (ms)	SR SD	AR Time (ms)	AR SD	DR Time (ms)	DR SD
.5	498.392	132.811	359.306	354.274	958.468	700.610
1	498.786	132.503	332.946	302.436	960.977	700.496
2	499.028	132.604	318.542	270.479	959.862	700.741
4	498.906	132.524	311.626	252.006	964.146	700.156
6	499.219	132.552	312.537	245.480	977.954	698.644
8	499.393	133.172	308.576	242.431	966.677	700.188
12	498.771	132.356	307.221	238.661	966.040	699.945

Table C.7: 90% NAT Density Results, Hops

Peer Lifetime (h)	SR Hops	SR SD	AR Hops	AR SD	DR Hops	DR SD
.5	6.50527	1.76215	6.04064	2.38783	6.85348	2.57086
1	6.50737	1.76068	6.00182	2.37000	6.85588	2.57009
2	6.50955	1.76384	5.99576	2.35233	6.87357	2.55801
4	6.49950	1.76467	5.95439	2.36317	6.83948	2.57899
6	6.50115	1.76210	5.95897	2.35468	6.84906	2.57037
8	6.50178	1.76267	5.95344	2.35776	6.84467	2.57536
12	6.50630	1.77064	5.95067	2.36760	6.84282	2.58706

Table C.8: 90% NAT Density Results, Time

Peer Lifetime (h)	SR Time (ms)	SR SD	AR Time (ms)	AR SD	DR Time (ms)	DR SD
.5	499.037	132.630	531.907	343.038	1193.05	468.177
1	499.193	132.520	493.477	280.274	1193.05	468.163
2	499.557	132.773	474.607	238.038	1193.05	465.397
4	498.415	132.803	461.002	213.454	1193.05	469.620
6	498.635	132.618	458.293	203.579	1193.05	468.055
8	498.642	132.658	456.005	198.859	1193.05	469.007
12	499.352	133.284	454.256	194.484	1193.05	470.528

Table C.9: 95% NAT Density Results, Hops

Peer Lifetime (h)	SR Hops	SR SD	AR Hops	AR SD	DR Hops	DR SD
.5	6.51029	1.76470	6.32866	2.12729	7.18721	2.22679
1	6.50853	1.76412	6.28603	2.10943	7.18848	2.22244
2	6.50357	1.76367	6.25672	2.10276	7.18138	2.22447
4	6.50259	1.76186	6.23507	2.10656	7.16981	2.23570
6	6.50533	1.75987	6.23512	2.10244	7.17404	2.23269
8	6.50098	1.76283	6.23961	2.09227	7.18242	2.21899
12	6.50412	1.76074	6.23269	2.09878	7.17604	2.22938

Table C.10: 95% NAT Density Results, Time

Peer Lifetime (h)	SR Time (ms)	SR SD	AR Time (ms)	AR SD	DR Time (ms)	DR SD
.5	499.586	132.835	561.840	334.467	1193.05	401.867
1	499.345	132.783	521.244	269.554	1193.05	401.109
2	498.950	132.747	499.258	225.487	1193.05	401.416
4	498.890	132.613	487.175	198.465	1193.05	403.583
6	498.963	132.453	483.571	187.914	1193.05	403.351
8	498.671	132.678	482.324	181.902	1193.05	400.411
12	498.958	132.526	479.986	176.691	1193.05	402.606

Bibliography

- [1] D. EASTLAKE 3RD AND T HANSEN. RFC 4634 - US Secure Hash Algorithms (SHA and HMAC-SHA). <http://www.ietf.org/rfc/rfc4634.txt>, July 2006.
- [2] A. ACHARYA, S. BERGER, AND C. NARAYANASWAMI. Unleashing the Power of Wearable Devices in a SIP infrastructure. In *3rd IEEE International Conference on Pervasive Computing and Communications (PerCom) 2005*, Kauai, Hawaii, USA, March 2005.
- [3] L. ADAMIC, R. LUKOSE, A. PUNIYANI, AND B. HUBERMAN. Search in Power-Law Networks. *Phys. Rev. E*, September 2001.
- [4] AOL INC. AOL AIM Software Website. <http://www.aim.com>.
- [5] APPLE INC. Apple Bonjour Developer Website. <http://developer.apple.com/networking/bonjour/index.html>.
- [6] S. BASET, H. SCHULZRINNE, AND M. MATUSZEWSKI. Peer-to-Peer Protocol (P2PP), draft-baset-p2psip-p2pp-01 (work in progress). <http://www.ietf.org/internet-drafts/draft-draft-baset-p2psip-p2pp-01.txt>, November 2007.
- [7] S. BASET AND H. SHULZRINNE. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *INFOCOM2006*, April 2006.
- [8] S. BASET AND H. SHULZRINNE. Using an External DHT as a SIP Location Service. Technical report, Columbia University, February 2006.
- [9] M. BAUGHER, D. MCGREW, M. NASLUND, E. CARRARA, AND K. NORRMAN. RFC 3711 - The Secure Real-time Transport Protocol (SRTP). <http://www.ietf.org/rfc/rfc3711.txt>, March 2004.
- [10] I. BAUMGART. P2PNS: A Secure Distributed Name Service for P2PSIP. In *Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008)*, Hong Kong, China, March 2008.
- [11] I. BAUMGART, B. HEEP, AND S. KRAUSE. A P2PSIP Demonstrator Powered by OverSim. In *Proceedings of 7th IEEE International Conference on Peer-to-Peer Computing (P2P2007)*, Galway, Ireland, September 2007.

- [12] P. BIONDI AND F. DESCLAUX. Silver Needle in the Skype. Presentation at Black Hat Europe 2006, February 2006.
- [13] BITTORRENT.ORG. BitTorrent.org. <http://www.bittorrent.org>.
- [14] BLUETOOTH SIG. Bluetooth Special Interest Group Website. <http://www.bluetooth.com/bluetooth>.
- [15] D. BRYAN. P2PSIP: On the Road to a World Without Servers. *Business Communications Review*, April 2007.
- [16] D. BRYAN AND B. LOWEKAMP. Decentralizing SIP. *ACM Queue*, March 2007.
- [17] D. BRYAN, B. LOWEKAMP, AND M. ZANGRILLI. The Design of a Versatile, Secure P2PSIP Communications Architecture for the Public Internet. In *Proceedings of the 2008 Workshop on Hot Topics in P2P (Hot-P2P '08)*, April 2008.
- [18] D. BRYAN, P. MATTHEWS, E. SHIM, D. WILLIS, AND S. DAWKINS. Concepts and Terminology for Peer to Peer SIP, draft-ietf-p2psip-concepts-02 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-p2psip-concepts-02.txt>, July 2008.
- [19] D. BRYAN, E. SHIM, B. LOWEKAMP, AND S. DAWKINS. Application Scenarios for Peer-to-Peer Session Initiation Protocol (P2PSIP) draft-bryan-p2psip-app-scenarios-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-bryan-app-scenarios-00.txt>, November 2007.
- [20] D. BRYAN, M. ZANGRILLI, AND B. LOWEKAMP. REsource LOcation And Discovery (RELOAD), draft-bryan-p2psip-reload-01 (work in progress). <http://www.ietf.org/internet-drafts/draft-bryan-p2psip-reload-01.txt>, July 2007.
- [21] D. A. BRYAN AND C. JENNINGS. A P2P Approach to SIP Registration, draft-bryan-sipping-p2p-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-bryan-sipping-p2p-00.txt>, January 2005.
- [22] D. A. BRYAN, B. B. LOWEKAMP, AND C. JENNINGS. SOSIMPLE: A Serverless, Standards-based, P2P SIP Communication System. In *Proceedings of the 2005 International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA 2005)*. IEEE, June 2005.
- [23] D. A. BRYAN, BRUCE B. LOWEKAMP, AND C. JENNINGS. dSIP: A P2P Approach to SIP Registration and Resource Location, draft-bryan-p2psip-dsip-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-bryan-p2psip-dsip-00.txt>, February 2007.
- [24] B. CAMPBELL, J. ROSENBERG, H. SCHULZRINNE, C. HUITEMA, AND D. GURLE. RFC 3428 - Session Initiation Protocol (SIP) Extension for Instant Messaging. <http://www.ietf.org/rfc/rfc3428.txt>, December 2002.

- [25] F. CAO, D. A. BRYAN, AND B. B. LOWEKAMP. Providing Secure Services in Peer-to-Peer Communications Networks with Central Security Servers. In *Proceedings of the 2006 International Conference on Internet and Web Applications and Services (ICIW'06)*, February 2006.
- [26] Y. CHENG, X. WEN, AND Y. SUN. Simulation and Analysis of Routing Schemes in Structured P2P Systems. In *ISECS 2008*, August 2008.
- [27] S. CIRANI AND L. VELTRI. A Kademlia-based DHT for Resource Lookup in P2PSIP, draft-cirani-p2psip-dsip-dhtkademlia-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-cirani-p2psip-dsip-dhtkademlia-00.txt>, October 2007.
- [28] E. COOPER, P. MATTHEWS, B. LOWEKAMP, AND D. BRYAN. NAT Traversal for dSIP, draft-matthews-p2psip-dsip-nat-traversal-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-matthews-p2psip-dsip-nat-traversal-00.txt>, February 2007.
- [29] R. COX, A. MUTHITACHAROEN, AND R. MORRIS. Serving DNS using a Peer-to-Peer Lookup Service. In *Proceedings of IPTPS02*, March 2002.
- [30] D. BRYAN. P2PSIP.org. <http://www.p2psip.org>.
- [31] F. DABEK, E. BRUNSKILL, M. FRANS KAASHOEK, D. KARGER, R. MORRIS, I. STOICA, AND H. BALAKRISHNAN. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS)*, pages 81–86, May 2001.
- [32] K. DARILION, C. KURTH, W. KAMPICHLER, AND K. M. GOESCHKA. A Service Environment for Air Traffic Control Based on SIP. In *37th Hawaii International Conference on Systems Sciences*, Big Island, Hawaii, USA, January 2004.
- [33] T. DIERKS AND E. RESCORLA. RFC 2246 bis 13 - The TLS Protocol Version 1.1. <http://www.ietf.org/rfc/rfc2246.txt>, June 2005.
- [34] J. R. DOUCEUR. The Sybil Attack. In *Proceedings of the IPTPS02 Workshop*, Cambridge, MA, USA, March 2002.
- [35] R. DROMS. RFC 2131 - Dynamic Host Configuration Protocol. <http://www.ietf.org/rfc/rfc2131.txt>, March 1997.
- [36] R. FIELDING, J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH, AND T. BERNERS-LEE. RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1. <http://www.ietf.org/rfc/rfc2616.txt>, June 1999.
- [37] G. CAMARILLO ED. FOR THE IAB. RFC 5694 - Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. <http://www.ietf.org/rfc/rfc5694.txt>, November 2009.

- [38] B. FORD, P. SRISURESH, AND D. KEGEL. Peer-to-Peer Communication Across Network Address Translators. In *USENIX 2005*, April 2005.
- [39] M. FREEDMAN, K. LAKSHMINARAYANAN, S. RHEA, AND I. STOICA. Non-Transitive Connectivity and DHTs. In *WORLDS05*, December 2005.
- [40] GNUTELLA RFC. Gnutella RFC. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.
- [41] D. GOLDSCHLAG, M. REED, AND P. SYVERSON. Onion Routing for Anonymous and Private Internet Connections. *Communications of the ACM*, 42(2), February 1999.
- [42] GOOGLE INC. Google Talk Website. <http://www.google.com/talk/>.
- [43] S. GUHA, Y. TAKEDA, AND P. FRANCIS. NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity. In *Future Directions in Network Architecture (FDNA-04)*, SIGCOMM '04 Workshops, Portland, OR, August 2004.
- [44] E. GUTTMAN, C. PERKINS, J. VEIZADES, AND M. DAY. RFC 2608 - Service Location Protocol, Version 2. <http://www.ietf.org/rfc/rfc2608.txt>, June 1999.
- [45] H. HANDLEY AND V. JACOBSON. RFC 2327 - SDP: Session Description Protocol. <http://www.ietf.org/rfc/rfc2327.txt>, April 1998.
- [46] J. HAUTAKORPI AND G. CAMARILLO. Evaluation of DHTs from the Viewpoint of Interpersonal Communications. In *MUM '07: Proceedings of the 6th =International Conference on Mobile and Ubiquitous m=Multimedia*, December 2007.
- [47] A. IAMNITCHI, M. RIPEANU, AND I. FOSTER. Small-World File-Sharing Communities. In *The 23rd Conference of the IEEE Communications Society (InfoCom 2004)*, Hong Kong, March 2004.
- [48] INTERNATIONAL TELECOMMUNICATIONS UNION. International Telecommunications Union H.323 Recommendation. <http://www.itu.int/rec/T-REC-H.323/en>.
- [49] INTERNET ENGINEERING TASK FORCE. Internet Engineering Task Force Website. <http://www.ietf.org>.
- [50] C. JENNINGS AND D. BRYAN. P2P for Communications: Beyond File Sharing. *Business Communications Review*, February 2006.
- [51] C. JENNINGS, B. LOWEKAMP, E. RESCORLA, S. BASET, AND H. SCHULZRINNE. A SIP Usage for RELOAD, draft-ietf-p2psip-sip-03 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-p2psip-sip-03.txt>, October 2009.
- [52] C. JENNINGS, B. LOWEKAMP, E. RESCORLA, S. BASET, AND H. SCHULZRINNE. REsource LOcation And Discovery (RELOAD) Base Protocol, draft-ietf-p2psip-base-08 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-p2psip-base-07.txt>, March 2010.

- [53] C. JENNINGS, J. PETERSON, AND M. WATSON. RFC 3325 - Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks. <http://www.ietf.org/rfc/rfc3325.txt>, November 2002.
- [54] C. JENNINGS, J. ROSENBERG, AND E. RESCORLA. Address Settlement by Peer to Peer, draft-jennings-p2psip-asp-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-jennings-p2psip-asp-00.txt>, July 2007.
- [55] J. KENSIN. RFC 5321 - Simple Mail Transfer Protocol. <http://www.ietf.org/rfc/rfc5321.txt>, October 2008.
- [56] H. KRAWCZYK, M. BELLARE, AND R. CANETTI. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication. <http://www.ietf.org/rfc/rfc2104.txt>, February 1997.
- [57] P. LEACH, M. MEALLING, AND R. SALZ. A Universally Unique IDentifier (UUID) URN Namespace. <http://www.ietf.org/rfc/rfc4122.txt>, July 2005.
- [58] B. LEONG, B. LISKOV, AND E. DEMAINE. EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. In *Proceedings of the 12th International Conference on Networks (ICON 2004)*, November 2004.
- [59] B. LOWEKAMP AND J. DEVERICK. Security Extensions for RELOAD, draft-lowekamp-p2psip-reload-security-01 (work in progress). <http://www.ietf.org/internet-drafts/draft-lowekamp-p2psip-reload-security-01.txt>, July 2007.
- [60] D. MACDONALD AND B. LOWEKAMP. RFC 5780 : NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN). <http://www.ietf.org/rfc/rfc5780.txt>, March 2010.
- [61] J. MAENPAA AND G. CAMARILLO. Study on Maintenance Operations in a Chord-based Peer-to-Peer Session Initiation Protocol Overlay Network. In *Proceedings of the 2009 Workshop on Hot Topics in P2P (Hot-P2P '09)*, May 2009.
- [62] S. MARTI, P. GANESAN, AND H. GARCIA-MOLINA. SPROUT: P2P Routing with Social Networks. In *First International Workshop on Peer-to-Peer Computing and Databases (P2P&DB 2004)*, March 2004.
- [63] P. MAYMOUNKOV AND D. MAZIERES. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Proceedings of IPTPS02*, March 2002.
- [64] MICROSOFT CORPORATION. Microsoft Windows Live Messenger Website. <http://windowslive.com/desktop/messenger>.
- [65] N. MODADUGU AND E. RESCORLA. The Design and Implementation of Datagram TLS. In *11th Network and Distributed System Security Symposium (NDSS)*, February 2004.

- [66] A. MUTHITACHAROEN, S. GILBERT, AND R. MORRIS. Etna: A fault-tolerant Algorithm for Atomic Mutable DHT Data. Technical Report MIT-LCS-TR-993, MIT-LCS, June 2005.
- [67] M. E. J. NEWMAN. The Structure and Function of Complex Networks. *SIAM Review*, May 2003.
- [68] OPEN DHT. Open DHT. <http://www.opendht.org>.
- [69] S. PERREAULT AND J. ROSENBERG. TCP Candidates with Interactive Connectivity Establishment (ICE), draft-ietf-mmusic-ice-tcp-08 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-mmusic-ice-tcp-08.txt>, October 2009.
- [70] J. PETERSON AND C. JENNINGS. RFC 4474 - Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP). <http://www.ietf.org/rfc/rfc4474.txt>, August 2006.
- [71] PLANETLAB CONSORTIUM. PlanetLab. <http://www.planet-lab.org>.
- [72] Y. REKHTER, T. LI, AND S. HARES. RFC 4271 - A Border Gateway Protocol 4 (BGP-4). <http://www.ietf.org/rfc/rfc4271.txt>, January 2006.
- [73] E. RESCORLA AND N. MODADUGU. RFC 4347 - DTLS: Datagram Transport Layer Security. <http://www.ietf.org/rfc/rfc4347.txt>, April 2006.
- [74] RESIPROCATATE PROJECT. ReSIProcate Project. <http://www.resiprocate.org>.
- [75] S. RHEA, D. GEELS, T. ROSCOE, AND J. KUBIATOWICZ. Handling Churn in a DHT. In *USENIX Annual Technical Conference*, June 2004.
- [76] S. RHEA, B. GODFREY, B. KARP, J. KUBIATOWICZ, S. RATNASAMY, S. SHENKER, AND J. HELLERSTEIN. OpenDHT: A Public DHT Service and Its Uses. In *Proceedings of ACM SIGCOMM 2005*, August 2005.
- [77] M. RIPEANU. Peer-to-Peer Architecture Case Study: Gnutella Network. In *First International Conference on Peer-to-Peer Computing (P2P'01)*, pages 99–101, August 2001.
- [78] R. RODRIQUES AND B. LISKOV. Rosebud: A Scalable Byzantine-fault-tolerant Storage Architecture. Technical Report TR/932, MIT CSAIL, December 2003.
- [79] J. ROSENBERG. RFC 3856 - A Presence Event Package for the Session Initiation Protocol (SIP). <http://www.ietf.org/rfc/rfc3856.txt>, August 2004.
- [80] J. ROSENBERG. (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, draft-ietf-mmusic-ice-19 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-mmusic-ice-19.txt>, October 2007.

- [81] J. ROSENBERG, R. MAHY, AND P. MATTHEWS. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), draft-ietf-behave-turn-16 (work in progress). <http://www.ietf.org/internet-drafts/draft-ietf-behave-turn-16.txt>, July 2009.
- [82] J. ROSENBERG, R. MAHY, P. MATTHEWS, AND D. WING. RFC 5389 - Session Traversal Utilities for NAT (STUN). <http://www.ietf.org/rfc/rfc5389.txt>, October 2008.
- [83] J. ROSENBERG AND H. SCHULZRINNE. RFC 3264 - An Offer/Answer Model with the Session Description Protocol (SDP). <http://www.ietf.org/rfc/rfc3264.txt>, June 2002.
- [84] J. ROSENBERG, H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY, AND E. SCHOOLER. RFC 3261 - SIP : Session Initiation Protocol. <http://www.ietf.org/rfc/rfc3261.txt>, June 2002.
- [85] J. ROSENBERG, H. SCHULZRINNE, AND P. KYZIVAT. RFC 3841 - Caller Preferences for the Session Initiation Protocol (SIP). <http://www.ietf.org/rfc/rfc3841.txt>, August 2004.
- [86] A. ROWSTRON AND P. DRUSCHEL. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
- [87] P. SAINT-ANDRE. Extensible Messaging and Presence Protocol (XMPP): Core. <http://www.ietf.org/rfc/rfc3920.txt>, October 2004.
- [88] P. SAINT-ANDRE. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. <http://www.ietf.org/rfc/rfc3921.txt>, October 2004.
- [89] H. SHULZRINNE, S. CASNER, R. FREDERICK, AND V. JACOBSON. RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. <http://www.ietf.org/rfc/rfc1889.txt>, January 1996.
- [90] K. SINGH AND H. SCHULZRINNE. Peer-to-peer Internet Telephony using SIP. In *"Proceedings of the 2005 Network and Operating System Support for Digital Audio and Video (NOSSDAV 2005)"*, June 2005.
- [91] SIPSHARE PROJECT. SIPshare Project. <http://www.research.earthlink.net/p2p/>.
- [92] SKYPE LIMITED. Skype Website. <http://www.skype.org/>.
- [93] M. SRIVATSA AND L. LIU. Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. In *Proceedings of Computer Security Applications Conference*, December 2004.

- [94] I. STOICA, R. MORRIS, D. KARGER, M. KAASHOCK, FRANK DABEK, AND H. BALAKRISHNAN. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, February 2003.
- [95] I. STOICA, R. MORRIS, D. KARGER, M. FRANS KAASHOEK, AND H. BALAKRISHNAN. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160. ACM Press, August 2001.
- [96] Y. UPADRASHTA, J. VASSILEVA, AND W. GRASSMANN. Social Networks in Peer-to-Peer Systems. In *Proceedings of 38th Hawaii International Conference on System Sciences*, January 2005.
- [97] P. VIXIE, S. THOMSON, Y. REKHTER, AND J. BOUND. RFC 2136 - Dynamic Updates in the Domain Name System DNS UPDATE. <http://www.ietf.org/rfc/rfc2136.txt>, April 1997.
- [98] VONAGE MARKETING, LLC. Vonage Website. <http://www.vonage.org/>.
- [99] D. WATTS, P. SHERIDAN DOBBS, AND M. E. J. NEWMAN. Identity and Search in Social Networks. *Science*, May 2002.
- [100] B. WELLINGTON. RFC 3007 - Secure Domain Name System (DNS) Dynamic Update. <http://www.ietf.org/rfc/rfc3007.txt>, November 2000.
- [101] WIKIPEDIA FOUNDATION, INC. WASTE Wikipedia Page. <http://en.wikipedia.org/wiki/WASTE>.
- [102] J. M. WINTERS. Telerehabilitation Research: Emerging Opportunities. *Annual Review of Biomedical Engineering*, 4:287–320, 2002.
- [103] YAHOO INC. Yahoo Messenger Software Website. <http://messenger.yahoo.com>.
- [104] P. YALAGANDULA, S. LEE, P. SHARMA, AND S. BANERJEE. Correlations in End-to-End Network Metrics: Impact on Large Scale Network Monitoring. In *IEEE Global Internet Symposium*, April 2008.
- [105] M. ZANGRILLI AND D. A. BRYAN. A Bamboo-based DHT for Resource Lookup in P2PSIP, draft-zangrilli-p2psip-dsip-dhtbamboo-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-zangrilli-p2psip-dsip-dhtbamboo-00.txt>, February 2007.
- [106] M. ZANGRILLI AND D. A. BRYAN. A Chord-based DHT for Resource Lookup in P2PSIP, draft-zangrilli-p2psip-dsip-dhtchord-00 (work in progress). <http://www.ietf.org/internet-drafts/draft-zangrilli-p2psip-dsip-dhtchord-00.txt>, February 2007.
- [107] B. Y. ZHAO, L. HUANG, J. STRIBLING, S. C. RHEA, A. D JOSEPH, AND JOHN D. KUBIATOWICZ. Tapestry: A Resilient Global-scale Overlay for Rapid Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004. Special Issue on Service Overlay Networks.

- [108] P. ZIMMERMANN. *PGP Source Code and Internals*. MIT Press, 1995.

VITA

David Alan Bryan

David A. Bryan spent his earliest years in Virginia, but has also lived in Maryland, Georgia, Minnesota, New Jersey, Pennsylvania, California, and Alberta, Canada. He obtained two degrees from the Richard Stockton College of New Jersey, a B.S. in Computer Science with Program Distinction and High Honors in 1995, and a B.A. in Physics, Magna Cum Laude, in 1996. He received his M.S. in Computer Science from The College of William and Mary in 2000, and defended his Ph.D. Thesis in April of 2010.

In addition to his academic achievements, David is well known as an industry leader and entrepreneur in the Voice over IP space, having co-founded Jasomi Networks, where he served as CTO, and SIPeerior Technologies, where he served as CEO. He serves as Chair of the Peer-to-Peer SIP Working Group of the Internet Engineering Task Force, and is a frequent speaker at academic and industry conferences.