P2PSIP (proposed)
Internet-Draft
Expires: April 25, 2007

D. Bryan
SIPeerior; William & Mary
B. Lowekamp
William & Mary; SIPeerior
C. Jennings
Cisco Systems
October 22, 2006

A P2P Approach to SIP Registration and Resource Location
draft-bryan-sipping-p2p-03

Status of this Memo

Copyright Notice

Abstract

This document outlines the motivation and requirements for a Peer-to-Peer (P2P) based approach for SIP registration and resource discovery using distributed hash tables and presents the architectural design for such a system. This design removes the need for central servers from SIP, while offering full backward compatibility with SIP,

allowing reuse of existing clients, and allowing P2P enabled peers to communicate with conventional SIP entities. A basic introduction to the concepts of P2P is presented, backward compatibility issues addressed, and the security considerations are considered.

This work is one possible implementation of the the protocol being discussed for creation in the proposed P2PSIP WG. In the context of the work being proposed, this draft would represent a concrete proposal for the P2PSIP Peer Protocol, using modified SIP as the underlying protocol. In this architecture, no P2PSIP Client Protocol is needed, rather unmodified SIP is used for access by non-peers.

This is early work, and is less secure in many ways than the traditional approach to SIP, but has certain other interesting characteristics that may make it desirable in some situations. This work is being discussed on the p2psip@cs.columbia.edu mailing list.

Table of Contents

1. Introduction

As SIP [1] and SIMPLE based Voice over IP (VoIP) and Instant Messaging (IM) systems have increased in popularity, situations have emerged where centralized servers are either inconvenient or undesirable. For example, a group of users wishing to communicate between each other, but using machines that are not consistently connected to the network, are often forced to use a central server that is outside the control of the group. Similarly, groups wishing to establish ephemeral networks for use in meetings, conferences, or classes often do not wish to configure a centralized server. Organizations may also want to allow their members to communicate with each other without traffic flowing to third parties, but may not have the staff or equipment to maintain a server.

Peer-to-Peer (P2P) computing has emerged as a mechanism for completely decentralized, server-free implementations of various applications. In particular, many recent efforts have focused on applying P2P to SIP within the IETF, starting with the forerunner of this document. Since then a substantial usecases document [10] has emerged and, most recently, a concepts and terminology [2] document has helped define a common set of terms. This iteration of this document attempts to incorporate the terminology proposed in the terminology draft.

This draft presents a SIP based system that uses P2P mechanisms to remove the need for central servers in SIP and SIMPLE based communications systems. This draft evolved from early work done on the SoSIMPLE [11] P2PSIP project, but has changed extensively. This works reflects experience gained in actually building a full commercialy available P2PSIP product based on this draft, as well as from extensive work/insight gleaned from the P2PSIP mailing list.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC2119 [3].

Terminology defined in RFC 3261 [1] is used without definition.

We use the terminology and definitions from the Concepts and Terminology for Peer to Peer SIP [2] draft extensively in this document. Other terms relating to P2P or new to this document are defined when used and are also defined in Definitions (Section 4). If this is your first time reading this document, we suggest reading

that draft and the Definitions (Section 4) section first.

In many places in this document, 10 hexadecimal digit values are used in examples as SHA-1 hashes. In reality, these hashes are 40 digit. They are shortened in this document for clarity only.

3. Background

3.1. Peer-to-Peer Fundamentals

The fundamental principle behind Peer-to-Peer (P2P) Architectures is that each and every member of the network has equal importance in the transactions that take place on the network, and that these peers communicate with each other to accomplish tasks. Contrast this with the more traditional Client-Server Architecture in which a large number of clients communicate only with a small number of central servers responsible for performing tasks. Each entity that participates in a P2P system, usually called a node or peer, provides server-like functionality and services as well as being a client within the system. In this way, the services or resources that would be provided by a centralized entity are instead available from the peers of the system. Note that a particular peer may or may not provide a particular service, but some peer does, ensuring that collectively the peers can provide that particular service. The logical network connecting the peers to one another is referred to as an overlay network or overlay, as it is in some sense a new, small sub-network at a higher logical level than lower level network connections.

Some P2P networks have certain peers that provide a higher level of functionality. Often these peers form a P2P network and connect to each other, then serve a number of true clients. These more powerful peers are often referred to as super-peers. This approach is often used to traverse NATs, with peers residing outside of the NATs serving as super-peers, and to allow peers with more bandwidth to serve as concentrators for information.

Many P2P systems further assume that peers are ephemeral in nature. A peer may join or leave the overlay at any time. The design of algorithms for P2P architectures take this into account. Information is often replicated, and the topology of the overlay can be quickly adapted as peers enter and leave.

Likely the best known (or perhaps most infamous) use of P2P technology is file sharing. In these systems, individual users store files, and join the overlay network by connecting to a small number of peers already in the overlay. When the user wishes to locate a

particular file they don't have, they contact these neighbors. Several alternatives exist for this query. In early systems, a peer searching for a file would ask their neighbors if they had the file. If one of these peers had the file, it would respond telling the requester they had the file. If not, they passed the request on to their neighbors. The search was limited to a particular depth using a Time to Live (TTL) mechanism, but since peers had no idea what other peers were doing, queries continued until the TTL was reached, even if some peer had already replied. This approach, often called the flood search approach, proved inefficient.

3.2. Distributed Hash Table (DHT) Systems

To improve the efficiency, most newer systems locate resources using a Distributed Hash Table, or DHT. Peers are organized using a Distributed Hash Table (DHT) structure. In such a system, every resource has a Resource-ID, which is obtained by hashing some keyword or value that uniquely identifies the resource. Resources can be thought of as being stored in a hash table at the entry corresponding to their Resource-ID. The peers that make up the overlay network are also assigned an ID, called a Peer-ID, which maps to the same hash space as the Resource-IDs. A peer is responsible for storing all resources that have Resource-IDs near the peer's Peer-ID. 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 may change. 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. Various DHT implementations may visualize the hash space as a grid, circle, or line.

Peers keep information about the location of other peers in the hash space and in general know about most peers nearby in the hash space, and progressively fewer more distant peers. When a user wishes to search, they consult the list of peer they are aware of and contact the peer with the Peer-ID nearest the desired Resource-ID. If that peer does not know how to find the resource, it either suggests the closest peer it knows about, or asks that peer itself and returns the result. In this fashion, the request eventually reaches the peer responsible for the resource, which then replies to the requester.

3.3. Chord

The Chord [12] system is one particular popular DHT algorithm. 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. In Chord, resource with Resource-ID k will be stored by the first peer with

Peer-ID equal to or greater (mod the size of the namespace) than k, ensuring that every Resource-ID is associated with some peer.

If the hash has 2^n bits in the range, each peer will keep a "finger table" of pointers to at most n other peers. The ith entry in the finger table contains a pointer to a peer at least 2^(i) units away in the hash space. The highest finger table entry thus point to a range 1/2 of the way across the hash space, the next highest 1/4, the next 1/8, and the smallest entry points to a range only 1 away in the hash space. The set of peers pointed to by these finger table entries are referred to as the neighbors of the peer, since they can be reached directly.

Searching in Chord is accomplished by sending messages to the peer in the finger table that is closest to the destination address. That neighbor will have finer resolution detail about the area and can route the message closer to the desired peer. This process is repeated until the message reaches the peer responsible for the destination, which can determine if the resource searched for is present.

This draft uses an algorithm derived from the Chord algorithm to communicate between peers in the DHT. Specifically, the algorithm detailed in Chord Overlay Algorithm (Section 10) has been adapted to use iterative searches rather than recursive searches in order to minimize the potential for Denial-of-Service (DOS) attacks. Chord is selected because of its simplicity, convergence properties, and general familiarity within the P2P community. We anticipate that other IETF working groups will be standardizing other DHT protocols and expect that the protocol and principles described in this draft will easily transfer to an alternative DHT algorithm. We have included specification of the specific DHT algorithm being used to support such transitions.

3.4. Issues for P2P Systems

All P2P systems need to solve the problem of locating some initial peer in the overlay, often called a bootstrap peer, in order to join. Some approaches taken to solving this problem include using some set of fixed peers, requiring that a peer be located using an offline mechanism, or using a broadcast/multicast mechanism.

P2P architectures offer several advantages over centralized architectures. P2P systems distribute resources across multiple machines, greatly reducing the potential of failure due to a single peer failing. This results in increased robustness, as well as some measure of protection from Denial-of-Service (DOS) attacks. P2P systems also have the advantage of scaling more easily as the number

of peers increases, since each new peer offers additional server-like functionality when it joins. P2P systems have their own class of problems, however. In particular, malicious peers can provide incorrect information, possibly denying access to resource in the system. Additionally, users can sometimes create many peers in the system, possibly using this as a mechanism for hijacking the system. These type of attack is often referred to as a Sybil [13] attack.

When referring to P2P systems in this document, we are referring to what are called true P2P systems in the literature. Some systems, such as the original Napster system, as well as many existing SIP deployments (which are occasionally referred to as P2P), are more properly referred to as hybrid systems. In hybrid systems, peers communicate with each other to exchange information, but resource location is still handled with a centralized server. Our goal in this document is a system that requires no central server of any type.

4. Definitions

Please also see the P2PSIP concepts and terminology [2] draft for additional terminology. We do not redefine terms from that draft here.

Peer-to-Peer (P2P) Architecture: An architecture in which peer nodes cooperate together to perform tasks. Each peer has essentially equal importance and performs the same tasks within the network. Additionally, peers communicate directly with one another to perform tasks. Contrast this to a Client-Server architecture.

Client-Server Architecture: An architecture in which some small number of nodes (servers) provide services to a larger number of nodes (clients). Client nodes initiate connections to servers, but typically do not communicate among themselves.

Distributed Hash Table (DHT): A mechanism in which resources are given a unique key produced by hashing some attribute of the resource, locating them in a hash space (see below). Peers located in this hash space also have a unique id within the hash space. Peers store information about resources with keys that are numerically similar to the peer's ID in the hash space.

Namespace or hash space: The range of values that valid results from the hash algorithm fall into. For example, using the SHA-1 algorithm, the namespace is all 40 digit hexadecimal identifiers. This namespace forms the set of valid values for Peer-IDs and Resource-IDs (see below).

Chord: A particular algorithm/approach to implementing a DHT, described by Stoica et al. [12]. 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.

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. Note that in our particular DHT structure, neighbor relations are NOT symmetric.

Adapter Peer: An adapter peer is a peer 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 service to DO NOT participate directly in the overlay. Compare these to the term "super peer" in the P2P community, although adapter peers may be thin software shims intended for only one client.

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.

Peer Admission: The act of a peer joining the overlay. Registration allows a peer to communicate with other peers, and requires (allows?) it to take on some server-like responsibilities such as maintaining resource location information. It DOES NOT register the user so that they can receive phone calls, which is the traditional SIP use of the word registration. We refer to traditional SIP registration as "user registration".

User Registration: The act of a user registering themselves with a SIP network. User registration creates a mapping between a SIP URI and a contact for a user to be created. This is the traditional meaning of registration in SIP. For a P2PSIP peer, this action MUST occur after peer registration.

Joining Peer: During the peer registration process, this is the peer that is attempting to register -- that is, the peer that is attempting to join the overlay network.

Bootstrap Peer: During the process of peer registration, the bootstrap peer is the peer that the joining peer contacts. This peer may be a well-known peer, a peer located using a broadcast method, a peer that the joining peer previously knew about, or a peer that another bootstrap peer referred the joining peer to. Often, the only role the bootstrap peer plays in the peer registration is to direct the joining peer to the admitting peer.

Admitting Peer: During the process of peer registration, this is the peer that is currently responsible for the portion of the namespace the new peer will eventually reside in. This peer is responsible for generating many of the messages exchanged during peer registration.

5. Overview

In this section we provide an overview of how P2PSIP works. Protocol details are provided in the remainder of the document.

Unlike a conventional SIP architecture, P2PSIP systems require no central servers. In a traditional SIP architecture many UAs connect to a central proxy server. In a P2PSIP network the peers connect directly to a few other peers, forming a virtual overlay network of peers that communicate with each other to provide services in the overlay. The peers participating in the overlay not only act as traditional SIP UAs, allowing their users to place and receive calls, but, when viewed collectively with the other peers, perform the roles of registrars and proxies in traditional SIP networks. These roles include resource location, maintaining presence information, and call routing. Each participating peer will maintain some fraction of the information that would normally be maintained by the proxy and/or registrar in a conventional SIP network.

5.1. Peer Functions and Behavior

P2PSIP peers provide many functions, more than any single entity in a traditional SIP architecture. Minimally, a participating peer must be an active member of the overlay and must provide some SIP "server-like" behaviors as well. The code that implements the additional server-like and DHT behavior can be located in several places in the network. The simplest is to have peers that are endpoints directly joining the overlay as peers. In this case, these peers provide the basic functionality of any SIP endpoint, but additionally implement the operations described in this document to enable self-organization and provide SIP functionality.

The behavior can also be located in an adapter peer, which allows one or more non-P2P aware SIP UAs to interact with the P2P overlay network. These adapters perform the additional self-organizing and SIP server-like behavior on behalf of the UA or UAs it supports. In this case, only the adapter peer is a peer in the overlay, the UAs are not peers themselves. All interaction with the P2P network is carried out by the adapter peer. The adapter essentially acts as a proxy server for the unmodified SIP UAs. The adapter can take the form of a small software shim or may be code within a traditional RFC

3261 server.

In most places in this document, which type of peer we are discussing won't affect the discussion. In those cases where it will, we have noted the differences.

5.2. P2P Overlay Structure

The P2P overlay consists of peers, which collectively serve as a directory service for locating resources (users, voicemail messages, etc.). Peers are organized using a Distributed Hash Table (DHT) P2P structure based on Chord. Like Chord, the system uses consistent hashing to a one dimensional namespace, conceptually in the form of a circle. Unlike Chord, all the messages needed to maintain the DHT are implemented as SIP messages. We use many Chord-like terms, which are defined in the section Definitions and Terminology. (Section 4)

Each peer is assigned a Peer-ID that determines the peer's location in the DHT ring and the range of resources for which it will store location information. Peer-IDs are created by hashing the IP address and port of the peer providing service. We allow for different algorithms to be used to calculate these hashes, but all members of the overlay MUST use the same algorithm.

Every resource has a Resource-ID, obtained by hashing some keyword that identifies the resource. The Resource-IDs map to the same space as the Peer-IDs. In the case of users, the unique keyword is the userid and the resource is the registration -- a mapping between the user name and a contact. Resources can be thought of as being stored in the distributed hash table at a location corresponding to their Resource-ID.

Like Chord, a resource with Resource-ID k will be stored by the first peer with Peer-ID equal to or greater (mod the size of the namespace) than k, ensuring that every Resource-ID is associated with some peer. 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 peer keeps information about how to 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 peer keeps track of its immediate predecessor peer, as well as one or more successor peers. The peer also keeps a table of information about other neighbors called a finger table, consisting of peers distributed around the overlay.

Messages are routed by taking advantage of a key property of these finger tables. 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 with a particular Resource-ID, the peer will send the request to the finger table entry with the Peer-ID closest to the desired Resource-ID. Because 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 Resource-ID, and suggests this peer to in response. The request is then resent to this closer peer. The process is repeated until the peer responsible for the Resource-ID is located, which can then determine if it is storing the information.

6. General Architecture

6.1. Use of SIP Messages

This draft documents a possible P2PSIP peer protocol using SIP messages. Our motivation throughout has been to preserve the semantics of standard SIP messages to the extent possible. All of the messages that are 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, such as the messages needed to join or leave the overlay, and to transfer information between peers. The second type of message are the type most SIP users will be familiar with -- registering users, inviting other users to a session, etc. -- basic session establishment. As the DHT is used as a distributed registrar, the registration and other searches are performed within the DHT. Once the target resource has been located, further communication proceeds directly between the UAs (or designated adapter peers) as with traditional SIP communications.

The messages used to manipulate the DHT are SIP REGISTER messages. RFC 3261, Section 10.2, specifies that REGISTER messages are used to "add, remove, and query bindings." Accordingly, we have selected REGISTER methods to use to add, remove, and query bindings. We use REGISTER both for the bindings of hosts as neighbors in DHT maintanence operations as well as the bindings of resource names to locations that are commonly maintained by SIP registrars.

Earlier versions of this draft utilized INVITE to perform searches within the DHT, with the guiding principle being that the INVITE would be redirected until it reached that actual UA of the desired contact. After further consideration, this use of INVITE raised the problem that it might result in a DHT-based SIP operation being sent

to a simpler SIP device unaware of the SIP extensions. By explicitly requiring DHT operations to be performed using REGISTER operations, and the final end-to-end connection made in the traditional SIP manner, we allow P2P-aware agents to deliberately separate their interactions with other P2P-aware peers from those interactions that require only traditional SIP messages, and can be performed with non-P2P agents.

6.2. Pluggable Overlay Algorithms

While this draft explores a protocol that was designed with a modification of the Chord algorithm in mind, we have attempted to make the SIP communication general-purpose, such that it can be used to implement a variety of overlay algorithms. Following these intentions, we have separated the overlay-specific algorithms into a Chord Overlay Algorithm (Section 10) section, which describes how the basic P2PSIP operations can be used to implement an iterative Chord algorithm. Our intention and hope is that others will design other overlay algorithms that rely on the same basic operations and are perhaps even optimizations of the basic Chord algorithm so that compatibility can be maintained. Furthermore, as other IETF working groups explore ideas for standardizing P2P algorithms, we hope that the basic SIP messages can remain consistent while only the overlay "module" needs to be redesigned to reflect the new protocols.

7. 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, obtained by hashing its own IP address, or a user's registration, for which it hashes the user's URI to obtain the appropriate Resource-ID. It then consults its finger table for the closest peer it is aware of to the target ID. In the trivial case of initial startup, the application may know only of a single bootstrap peer. The message is sent to that peer, which performs the requested operation if it is responsible for that ID. If the contacted peer is not responsible for the target ID, then the contacted peer issues a 302 redirect pointing the searching peer toward the best match the contacted peer has for the target ID. The searching peer then contacts the peer to which it has been redirected and the process iterates until the responsible peer is located.

7.1. 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 above 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. The joining peer then exchanges additional REGISTER messages 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 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.

7.2. 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 traditional 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 iterates using 302 redirects 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.

7.3. 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.

8. Message Syntax

8.1. 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 processed in a P2P method 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.

8.2. 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, from RFC 2104 [4] 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. See Extensions to sip-identity (Section 12) for information on how to protect against replay attacks.

8.2.1. Peer-IDs

Peer-IDs are determined by the algorithm being used. In the case of SHA-1, <40 hex digit hash>. The Peer-ID MUST be formed by taking the IP address of the peer, without the colon or port, and hashing this string with the appropriate algorithm. Then the least significant sixteen bits of the hash are replaced by the port used by the peer. For SHA-1, the resulting Peer-ID looks like a04d371e3f4078a7a8c49bb7a4ea6199fc9d5c77. 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 [5] and use this address for their Peer-ID.

The string hashed to obtain the PeerID is formally defined below as ipaddress.

ipaddress = IPV4address / IPv6reference

PeerID is formally defined as:

PeerID = token

When using SHA-1:

PeerID = 40LHEX

8.2.2. Resource-IDs and the replica URI parameter

No special restrictions, beyond those imposed by RFC 3261, are imposed on the resource URIs in a P2PSIP system. Note that various security schemes, two of which are discussed in Protecting the Namespace (Section 14.2) may place restrictions of their own on the user's URIs.

For reliability, redundant registrations are made for resources to avoid certain forms of DOS attacks and guard against the loss of information in case of peer failure. The primary registration is made using the canonical form of the resource's URI. Replica registrations are made by attaching a replica URI parameter to the URI. The value of this replica parameter is left to the implementor, but all peers in the overlay MUST use the same set of values, and there MUST be a different, unique value for each level of redundancy. The replica parameter MUST NOT be included for the primary registration. The replica URI parameter is of type other-param as defined in RFC 3261.

replica-param = "replica=" token

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 = token

When using SHA-1:

ResourceID = 40LHEX

8.3. P2PSIP URIs

Because hashing URIs to produce identifiers is a non-trivial cost, P2P SIP 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.

8.3.1. Peer URIs and the user=peer URI Parameter

A P2PSIP peer is represented by constructing a URI with the PeerID as the userinfo portion. Additionally, the URI parameter "user=peer" MUST be used.

PeerURI = PeerID "@" hostport ";" user-param uri-parameters

Formally, the user=peer parameter is defined by using the keyword "peer" of type token, serving as "other-user" in the definition of user-param from RFC 3261. A peer receiving a PeerURI MUST verify the hash before using it to update its neighbors or finger table.

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, 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-addrs for peers with display-name information MUST ignore the information.

Examples (using shortened Peer-ID 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:

To: <sip:ed57487add@10.6.5.5;user=peer>

8.3.2. 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 ResourceID is a significant expense, the optional URI parameter resource-ID=<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=" ResourceID
```

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

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

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

8.4. 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 required user=peer, algorithm, and overlay as well as the optional expires header parameter.

```
DHT-PeerID: <sip:a04d371e@192.168.1.1;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
```

8.4.1. 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 PeerID and ResourceID 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 RFC4474. [6] Currently, those consist of 'sha1', indicating SHA-1 as defined in RFC 3174 [7] and 'hmac-sha1', indicating HMAC-SHA1 as defined in RFC2104 [4]. Implementations MUST support the SHA-1 algorithm.

A peer should 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 of 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 provide a normal 302 forwarding response 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.

8.4.2. 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 should reject a message with 488 Not Acceptable here if it specifies an overlay in which the peer is not participating. An

initial contact of 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 provide a normal 302 forwarding response 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.

8.4.3. 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. 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 should be returned.

8.4.4. PeerID Expires header parameter

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

8.5. 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 the predecessor, successors, and finger 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 linktype-token depth-token
depth-token     = 1*DIGIT
linktype-token  = "P" / "F" / "S" / other-token
```

expires-param = "expires" EQUAL delta-seconds

and an example, the header might look like (using a shortened 10 digit Peer-ID for clarity):

DHT-Link: <sip:671a65bf22@192.168.0.1;user=peer>;link=S1;expires=600

8.5.1. The linktype and depth values

The linktype and depth values are dependent on the DHT routing algorithm employed by the peer. For the algorithm described in Section Chord Overlay Algorithm (Section 10), the linktype MUST be one of three single characters, P, S, or F. P MUST be used to indicate that the information provided describes a predecessor of the sending peer. S MUST indicate that the information describes a successor peer, and F MUST indicate that it is a finger table peer from the sending peer.

For the algorithm in Chord Overlay Algorithm (Section 10), the depth MUST be a non-negative integer representing which predecessor, successor, or finger table entry is being described. For predecessors and successors, this MUST indicate numeric depth. In other words, "P1" indicates the peers immediate predecessor, while "S5" would indicate the fifth successor. "P0" or "S0" would indicate the sending peer itself. In the case of finger table entries, the depth MUST indicate the exponent of the offset. Since finger tables point to ranges in the hash table that are offset from the current peer in the hash space by a power of two. That is, finger table entry i points to a range that begins with a peer 2^i away in the hash space, and there are a maximum of k finger table entries, where k is the size of the hash result in bits. For an finger table entry, the depth corresponds to this exponent i. In other words, "F0" would correspond to a finger table entry pointing to the peer for a range starting a distance 2^0 = 1 from the Peer-ID in the hash space, while "F6" would point to peer used to search for resources in a range starting 2^6 = 64 away from the Peer-ID in the hash space.

8.5.2. Expires Processing

Each DHT-Link header MUST contain an expires parameter. Each peer maintains an expiration time for each of its neighbor and finger 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.

9. 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 11.1) section of this document. Additionally, SIP REGISTER messages are used to register a new peer with the DHT and to transmit the information needed to maintain the DHT.

9.1. Bootstrapping

When a peer wishes to join an existing overlay, it must first locate some peer that is already participating in the overlay. referred to as the bootstrap peer. Peers MAY use any method they choose to locate the initial bootstrap peer. The following are a few of the many methods that may be used:

Static Locations: Some number of peers in the overlay may be persistent and have well know addresses. These addresses could be configured into the peer application or obtained using an out-of-band mechanism such as a web page.

Cached Peers: While this mechanism cannot be used the first time that a peer runs, on subsequent attempts to join the overlay a peer might attempt to use a previously contacted peer as a bootstrap peer.

Broadcast mechanisms: Peers can use a broadcast mechanism to locate the initial peer, for example by sending the first REGISTER message to the SIP multicast address.

In the rest of this section, we assume that the joining peer is not the first peer and that a bootstrap peer has been located.

9.2. 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.

9.2.1. 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 subsection Peer URIs (Section 8.3.1) in the Message Syntax section.

While using the IP address of the sender for To and From is different than traditional 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 user name 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 10.4.1.2:5060 hashes to 463ac4b449 under 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:10.7.8.129 SIP/2.0
To: <sip:463ac4b449@10.4.1.2;user=peer>
From: <sip:463ac4b449@10.4.1.2;user=peer>
Contact: <sip:463ac4b449@10.4.1.2;user=peer>
Expires: 600
DHT-PeerID: <sip:463ac4b449@10.4.1.2;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

9.2.2. 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 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. 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 traditional C/S SIP message.

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

9.2.2.1. Routing the Peer Registration

The presence of user=peer 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 hashed Peer-ID corresponds to the IP address specified in the URI by hashing the IP address and port and comparing it to the Peer-ID. 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, it will provide the joining peer with information about a peer closer to the area of the overlay where the joining peers Peer-ID is stored.

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. Peers SHOULD NOT proxy the request, as described in RFC3261:10.3, item1. (although they could, it would place undue burden on a peer to ask it to do so, so we advise against it) The 302

is constructed according the rules of RFC 3261 with the following rules. The receiving peer MUST look up the peer in its finger table nearest the 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 8.3.1) section of this document, including appropriate URI parameters. The response MUST contain a valid DHT-PeerID header. This response is sent to the joining peer.

A peer MUST NOT add a new peer to its finger 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 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:463ac4b449@10.4.1.2;user=peer>
From: <sip:463ac4b449@10.4.1.2;user=peer>
Contact: <sip:47e46fa2cd@10.3.1.7;user=peer>
Expires: 600
DHT-PeerID: <sip:084d299ff2@10.7.8.129;user=peer>;algorithm=sha1;
            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.

9.2.2.2. Admitting the Joining Peer

The admitting peer MUST verify that the Peer-ID hash of the IP address is valid, as described above. If these do not match, the message should 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 falls between the admitting peer's predecessor's Peer-ID and the admitting peer's 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 require an authentication challenge to the REGISTER message. Once any challenge has been met, the admitting will reply with a 200 OK message to the joining peer. As in a traditional 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 joining peer has a Peer-ID between the admitting peer's predecessor's Peer-ID and the admitting peer's Peer-ID. The admitting peer must populate the DHT-Link headers with all values required by the routing protocol so that the joining peer can initialize its neighbors and finger table entries. Additionally, the admitting peer MUST include its DHT-PeerID header containing the admitting peer's Peer-ID and IP.

In the special case where the admitting peer's predecessor is NULL (as can happen if the admitting peer is the peer that started the overlay), that peer MUST reply with a 200 response to any peer validly attempting to join the system, regardless of PeerID. See the Chord Overlay Algorithm (Section 10) section for more information on NULL.

For further details of the contents of the link headers and the joining peer processing, see Chord Admission Processing (Section 10.4).

Continuing the example register from the previous sections, assume now that the peer with Peer-ID 47e46fa2cd and IP address 10.3.1.7 is currently responsible for 463ac4b449 in the namespace. The admitting peer here does send the fingertable, but we show only the first entry entry for clarity. We also omit the additional successors used to support redundancy for clarity. The response would look something like:

```
SIP/2.0 200 OK
To: <sip:463ac4b449@10.4.1.2;user=peer>
From: <sip:463ac4b449@10.4.1.2;user=peer>
Contact: <sip:463ac4b449@10.4.1.2;user=peer>
Expires: 600
DHT-PeerID: <sip:463ac4b449@10.4.1.2;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:4201034a89@10.233.4.1;user=peer>;link=P1;expires=412
DHT-Link: <sip:574fb2d34a@10.0.233.227;user=peer>;link=S1;expires=816
DHT-Link: <sip:5f8dd34100@10.44.76.67;user=peer>;link=F2;expires=121
Require: dht
```

Supported: dht

Both the admitting peer and joining peer SHOULD immediately perform both a stabilize and fix fingers operation (Section 10.8) to stabilize the overlay.

9.3. Peer Query

As with traditional SIP, REGISTER messages that are sent without a Contact: header are assumed to be queries, as described in Section 10 of RFC3261. This corresponds to the find_successor operation in Chord.

9.3.1. Constructing a Peer Query Message

The peer looks for the finger table entry that covers the range they wish to search. If the finger table entry has not yet been filled (and the peer was not provided another finger table to use to get started), then the peer may send the request to any peer it has available, including their successor, predecessor, 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:4823affe45@0.0.0.0;user=peer>
From: <sip:463ac4b449@10.4.1.2;user=peer>
DHT-PeerID: <sip:463av4b449@10.4.1.2;user=peer>;algorithm=sha1;
            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 (Section 8.3.1) in the Message Syntax section. If a specific peer is being sought, the PeerURI must specify that IP address. 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.

9.3.2. 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.

9.3.2.1. Routing the Peer Query Message

The presence of a user=peer URI parameter 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 (Section 9.2.2.1) until the responsible peer is reached.

9.3.2.2. 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.

The reply that is constructed MUST provide information about the receiving peer's neighbors and finger table entries. For further details of the contents of the link headers and the joining peer processing, see Chord Query Processing (Section 10.5).

9.4. Populating the Joining Peer's Finger Table

Once admitted, the joining peer MUST populate its finger table by issuing queries for peers with the appropriate identifiers (see see Chord Finger Table (Section 10.6), Section Section 10.6). If the admitting peer provided finger table information, the joining peer MAY use this information to construct a temporary finger table and use this temporary table in the queries to populate the table.

9.5. Transfering 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 Registering and Removing User Registrations (Section 11.1), except that as they are third party registration from a peer, that is, the From header contains the PeerURI of the admitting peer.

9.6. Peers Leaving the Overlay Gracefully

Peers MUST send their registrations to their successor before leaving the overlay, as described in the section above. Additionally, peers MUST unregister themselves with both their successor and predecessor. This REGISTER is 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 Chord Graceful Leaving (Section 10.7).

9.7. 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 occuring. The IETF has developed STUN [5] and ICE [8] to address these issues. When contacting a new peer learned through a 302, the contacting peer should first send it the message using normal direct routing. If a timeout occurs, then it sends another message to the new peer it is trying to contact, but this time using loose routing to send it through the redirecting peer. The redirecting peer MUST route this message to the destination peer, which it has already asserted it has connectivity with by issuing the redirect. This connection is then

used as the control connection through which ICE negotiation is performed to establish a direct connection between the contacting and new peers.

9.8. Handling Failed Requests

When a request sent to another peer fails, the peir MUST perform searches to update its pointers. If the failed request was sent to a peer in the finger table, then the searches discussed in Populating the Joining Peer's Finger Table (Section 9.4) should be performed for all intervals that rely on the failed peer. If the predecessor or successor peer fails, a search for the predecessor's or successor's ID should be performed, and requests should be repeated, based on the predecessors and successors returned by these, until the correct successor or predecessors are determined. Multiple repeats may be needed until the failed peer's neighbors recognize that the peer is dead and update their own predecessor/successor.

OPEN ISSUE: should it be possible to trigger another peer to check its predecessor?

10. Chord Overlay Algorithm

The DHT routing algorithm used in this protocol is based on Chord, with adaptations to rely on iterative operations rather than recursive operations. As this places the burden of an operation on the searching or joining peer, rather than on intermediate peers, it is more appropriate for an Internet protocol.

We anticipate other routing algorithms being developed that may improve the performance, locality, or security of this algorithm. For this reason, the Chord-specific portions of the protocol are confined to this Section. The other elements of the protocol should be equally relevant to any DHT-based P2P routing algorithm.

10.1. DHT Name Parameter

For this protocol, the dht-param token must be set to "ChordIter1.0"

10.2. Starting a New Overlay

A peer starting an overlay for the first time need not do anything special in order to construct the overlay. The peer MUST initalize its finger table such that all entries point to itself. The peer MUST set its successor (which is also the first entry of the finger table) and all other finger table entries to itself, and MUST set its predecessor to NULL.

10.3. Finger Table

Chord recommends keeping a number of finger table entries equal to the size in bits of the hash space, for example 160 for SHA-1. These entries point to the first peer at least 2^i away from the peer, for 0 <= i <= 159, mod 2^160. Essentially, the peer divides the overlay hash circle up into segments, the first being the segment from [2^0-2^1) away from the peer, the second being from [2^1-2^2), the third being from [2^2-2^3), etc., all the way to the segment from [2^158-2^159) away from peer. It then stores an entry in the finger table for the first peer 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 less and less entries about more remote peers. These tables are populated when the peer joins the overlay, and are kept up to date by periodically updating them.

We recommend that, while using the full SHA-1 hash algorithm, peers maintain less than the full 160 entries in the finger table, perhaps 16 entries for small networks, 32 for larger networks. As this affects only the efficiency of the client, it is left to the implementor to determine a useful value. Note that a client can easily store enough finger table entries to exceed the maximum MTU size when transmitting the full finger table. In this case, a client may need to reduce the number of finger table entries reported in DHT-Link headers.

10.4. Peer Admission

When handling an initial join from a peer, the admitting peer MUST reply with a 200 response if the joining peer has a Peer-ID between the admitting peer's predecessor's Peer-ID and the admitting Peer-ID, or the admitting peer's predcessor is NULL. If the admitting peer's predecessor is not NULL, it MUST provide the joining peer with its current predecessor and successor in the 200. If the predecessor is NULL, the 200 SHOULD NOT include a value for the predecessor but MUST include a value for the successor. (Note: we may define a value to pass this, and require it be passed, in a future version) These MUST be placed placed in DHT-Link headers, as described in The DHT-Link Header (Section 8.5) section of this document. The predecessor MUST be transmitted in a DHT-Link header using a type of P and a depth of 1. The successor MUST be transmitted in a DHT-Link header using a type of S and a depth of 1.

The joining peer obtains the Peer-ID and address of the admitting peer from the DHT-Peer header, and the information about the admitting peer's predecessor from the DHT-Link P 1 header. The joining peer MUST set its successor to be the admitting peer and its predecessor to be the admitting peer's predecessor. If the admitting

peer did not provide a predecessor (which MUST only occur if the admitting peer's predecessor is NULL), the joining peer should leave their predecessor as NULL.

After the admitting peer sends the 200 reponse, it MUST set its predecessor to be the joining peer, and MUST obtain the information from the DHT-Peer header in the register request. It MUST NOT change the value of the predecessor prior to sending the 200. The admitting peer's successor is unchanged. Note that at this point the joining can also set all fingers pointing to intervals before the successor in the finger table to point to the successor.

The admitting peer SHOULD send a copy of the entries in their finger table to the joining peer, using DHT-Link headers of the F type. As the joining peer will likely be nearby the admitting peer in the hash space (at least for an overlay with a reasonable number of peers), this finger table information can likely improve the performance of the queries required to obtain a correct finger table information. It is the responsibility of the joining peer to calculate and reconstruct the intervals that the admitting peer would have based on the F parameters and the Peer-ID supplied in the 200. Note that providing the first finger is optional (and unwise since packet length is an issue), as it is (by definition) identical to the required successor field.

Following the admission, the joining peer MUST run periodic stabilization as described in Chord Periodic Stabilization (Section 10.8). The admitting peer should run periodic stabilization as well.

10.5. Chord Query Processing

A reply that is constructed to a query by the responsible peer MUST provide the current predecessor (if not NULL) and successor in the 200 or 404 message. These MUST be placed placed in DHT-Link headers, as described in The DHT-Link Header (Section 8.5) section of this document. If the predecessor is not NULL it MUST be transmitted in a DHT-Link header using a type of P and a depth of 1. It must be omited if NULL. The successor MUST be transmitted in a DHT-Link header using a type of S and a depth of 1. The 200 or 404 SHOULD contain the next 4 successor peers, for use in redundancy. Additionally, the replying peer MUST include its DHT-PeerID header.

10.6. Chord Finger Table

To populate the finger table, a peer must take its Peer-ID and, by applying the exponential offsets for each finger, calculate the Resource-IDs corresponding to the start of each finger interval. See

the P2P Overlay Structure (Section 5.2) subsection in the Overview section of this document. The joining peer then performs a search for each of these start intervals, as described above. The resulting Peer-IDs/IPs are entered into the corresponding finger table entries. This is analogous to the fix_fingers procedure in Chord.

10.7. Chord Graceful Leaving

When a peer sends its unregister message to its successor and predecessor, it MUST include DHT-Link headers listing its predecessor and successor peers. This allows the peers receiving the requests to obtain the information needed to correct their predecessor and successor peers, as well as keep their successor lists needed for redundancy current.

10.8. Chord Periodic Stabilization

In order to keep the overlay stable, peers must periodically perform book keeping operations to take into account peer failures. Periodically (we suggest 60-360 seconds), peers MUST perform an arbitrary query for their current successor's Peer-ID. The peer should examine the response from their successor. The predecessor reported should be the peer that made the request. If it is not, the peer MUST update their own successor with the predecessor returned, and additionally MUST send a REGISTER to this peer, structured as if the stabilizing peer had just entered the system. However, the peer sending this message MUST not process the response, but simply discard it, as this is intended only to pass information. (Note: Should we use a SIP unsolicited NOTIFY here instead?) This will serve to properly update the overlay. This is analogous to the notify procedure in Chord.

OPEN ISSUE: this operation is identical to the original chord operation, but it seems like we can pay attention to the response and observe if there have been multiple peers inserted and the peer we sent the REGISTER to knows a better successor peer, but this goes away with the next stabilize, anyway.

Additionally, when this periodic stabilization takes place, the peer should perform searches as discussed in Populating the Joining Peer's Finger Table (Section 9.4) to ensure that the finger table is up to date.

10.9. Peer Failure

Peer failure is handled by the periodic stabilization and responses to failed requests discussed above. Redundancy prevents against lost registrations.

10.10. Resource Replicas

When a resource is registered, the registering peer SHOULD create at least 2 redundant replicas to ensure the registry information is secure in the DHT. The registering peer is responsible for maintaining these replicas along with the primary entry.

11. Resource Operations

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

11.1. 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 (with a specific IP address) 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 Resource URIs (Section 8.3.2). 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 ResourceID 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 (Section 9.2). 302 messages are sent to indicate that the message is to be redirected to another peer (this contact should contain the URI parameter user=peer). 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 user=peer parameters. 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.

11.2. 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, 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 successors provided in the 200 to send these updates to the redundant peers as well.

11.3. 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.

11.4. Querying Resource Registrations

Resource queries are constructed as described in RFC 3261, Section 10. Querying peers should send a registration with no contact header. As described in Peer Search (Section 9.3.1), 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.

11.5. 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. (Section 11.4)

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.

11.6. Presence

We use SUBSCRIBE/NOTIFY for this. We subscribe to every user on our buddy list when we come online. If the buddies are online, that means that we know exactly where they are. Peers MAY use the PeerIDs of their buddies peers as additional "finger table" entries (essentially, cached values), consulting these first, as connections are likely to be made to people on the user's buddy list. These should also be periodically checked, as described in the Periodic Stabilization (Section 10.8) section.

If buddies are offline, one should periodically try to make the connection. However, if a UA receives a SUBSCRIBE from a buddy that it believes to be offline, it SHOULD attempt to subscribe to that buddy. This will allow people that are reciprocally on each other's buddy lists to rapidly be notified when one or the other comes

online, therefore the retry interval for subscribing to offline buddies can be fairly long because it is only necessary in the case of race conditions or other temporary failures in resource location.

11.7. 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. [14] describe an approach that separates the storage of resource location information from the actual storage of the offline research. We believe that this approach is in agreement with the approach taken by the rest of this document, which relies on the DHT overlay to store the registrar's location information, but relies on external, traditional 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.

12. Extensions to sip-identity

Cryptographically securing P2PSIP messages is required for many environments. P2PSIP builds on the existing sip-identity [6] architecture, however because there are no proxies or other servers, the authentication service must run on each individual UA. Two techniques are provided for authenticating messages and should be chosen based on the security requirements of the particular overlay.

TO DO: the following section is primarily descriptive, should be normative.

12.1. Shared Secret

To secure a small-scale network, perhaps an office environment or small group of people who wish to communicate together, a shared secret will suffice. Rather than relying on a domain certificate, therefore, the Identity field consists of an HMAC-SHA1 [4] hash of the canonical string defined by RFC4474 for use as the digest-string. In this use, ident-info-alg is "hmac-sha1". Identity-Info contains an ident-info-extension consisting of the string "p2psip:hmac-sha1".

Note that this technique authenticates peers within the overlay, but it does not authenticate individual users within the overlay, nor does it provide any form of authentication for communication with nodes outside the overlay.

12.2. User certificates

Because there are no servers in a P2PSIP environment, user-certificates must be used rather than domain certificates. Rather than supplying an Identity-Info header with an http URL to fetch the domain key, the peer SHOULD supply an Identity-Info header with a SIP URI through which a subscription request can be made to obtain the user's certificate, using the "certificate" SIP Event Package defined in sip-certs [9]. A peer MAY provide an http URL if it knows that NAT traversal techniques are not required to reach it. In the typical case, the UA provides access to its own certificate and cannot assume that it is not behind a NAT or firewall. Therefore, SIP subscriptions will be more reliable than http. Typically all peers in an overlay would already have the certificate of the CA for the overlay. However, the Issuer SHOULD encode a URL for obtaining its certificate in issuerAltName, thereby supporting scenarios where a peer does not have the overlay CA's certificate, such as communication between P2PSIP overlays and overlays with multiple CAs.

OPEN ISSUE: I cannot find a specification for fetching a certificate chain at runtime, the assumption seems to be that each peer should send its user certificate as well as the signing overlay certificate, which is presumably signed by a trusted CA. This seems like a bit of a waste in most cases. Should we define this mechanism for fetching signing certificates or just leave it at each peer sending the whole chain each time?

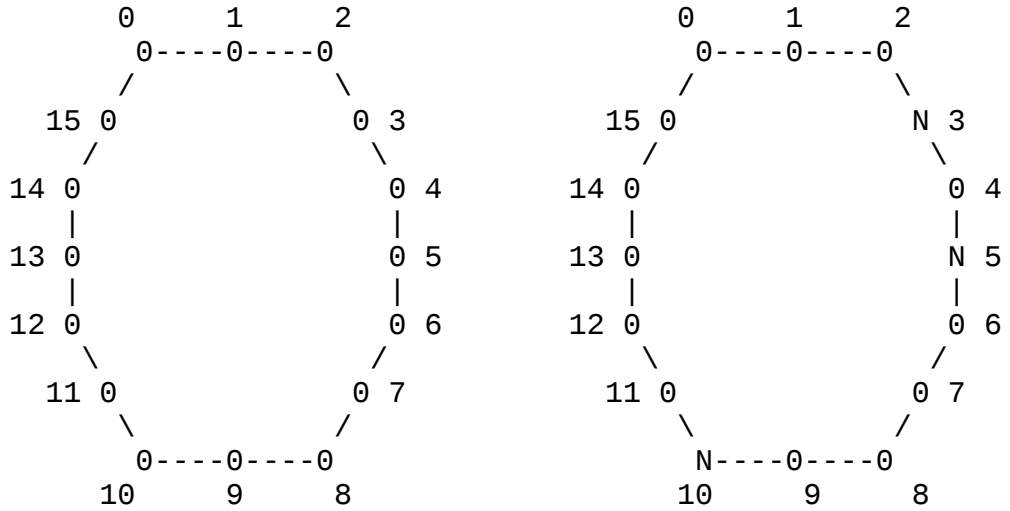
This specification is compatible with C/S sip-identity [6] implementations if they are configured to trust certificates issued by the overlay's CA. However, as sip-identity does not require supporting the use of certificate chains for user-certificates, this implementation does not provide compatibility with all non-P2P UAs. Overlays desiring such compatibility must provide their own authentication service configured with a domain certificate for communication with non-overlay nodes.

13. Examples

For our examples, we use a simplified network. Rather than use a full SHA-1 hash, and the resulting 2^160 namespace, we instead use a smaller 4 bit hash, leading to a namespace of size 16. All hash results in our examples are contrived. We list the Peer-ID and Resource-IDs as xx, where xx is a number between 0 and 15 (2^4 namespace). In a real situation, the full 40 hex chars would be used. Additionally, because the number of finger table entries is so small in this case, we use the full 4 entries, where in a real case we suggest that one uses less than the number of bits in the

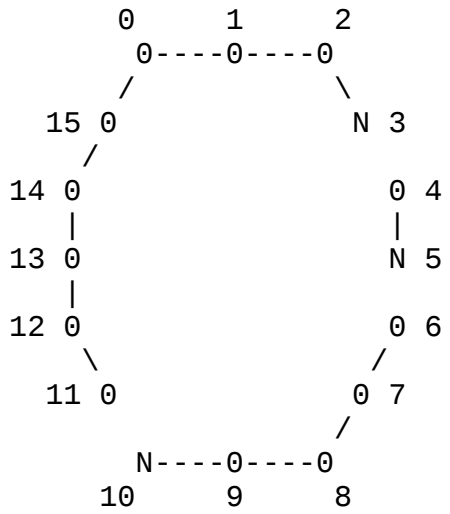
namespace.

The empty overlay can be visualized as a circle with 16 possible vacant points, each corresponding to one possible location in the hash space. On the left, we have labeled these locations in the hash space as 0-15, starting in the upper left, and have used 0s to indicate vacant spaces in the hash space. On the right, we show the same network with 3 operating peers, denoted by capital Ns, with Peer-IDs of 3, 5, and 10. We will use this sample network state as the starting point for all our networks:



Further, for the sake of example simplicity, assume the peer Peer-ID 3 has IP address 10.0.0.3, the peer peer with Peer-ID 5 has IP address 10.0.0.5, etc.

Data that hashes to a Resource-ID is stored by the next peer whose Peer-ID is equal to or larger than the Resource-ID, mod the size of the hash. As such, Peer 3 is responsible for any resources hashing from 11-15, as well as 0-3. Peer 5 is responsible for resources with Resource-IDs from 4-5, and Peer 10 is responsible for resources with Resource-IDs from 6-10. From this illustration, you follow a location clockwise until you encounter a peer, and this is the peer responsible for storing the information. This is illustrated below:



Finger tables give pointers to nearby peers. For our system, with 4 bit identifiers, we have 4 finger table entries. These finger tables point to the peer nearest to Peer-ID + 2^0, Peer-ID + 2^1, Peer-ID + 2^2 and Peer-ID + 2^3. If no peer is present at that location, the next available peer will be used. Thus, for our 3 peers, the finger tables look like the following, with ranges (indicated in traditional mathematical form) mapping to the peer those requests will be sent to:

	Peer 3	Peer 5	Peer 10
2^0 Entry	[4,5) -> 5	[6,7) -> 10	[11,12) -> 3
2^1 Entry	[5,7) -> 5	[7,9) -> 10	[12,14) -> 3
2^2 Entry	[7,11) -> 10	[9,13) -> 10	[14,2) -> 3
2^3 Entry	[11,3) -> 3	[13,5) -> 3	[2,10) -> 3

Assume further our sample network is called sipchat, and that 2 users are currently registered. User alice has a Resource-ID of 5, so her registration information is stored at peer 5. User bob is also registered, and has a Resource-ID of 12, so his registration information is stored by peer 3. Assume further that bob's UA is co-located with Peer 10, so his contact is sipchat/bob@10.10.10.10, and that alice is running a UA on a completely separate IP of 10.99.99.99, but is using an adapter peer running on Peer 3, therefore Peer 3 will send messages on alice's behalf, but alice's contact is sipchat/alice@10.99.99.99.

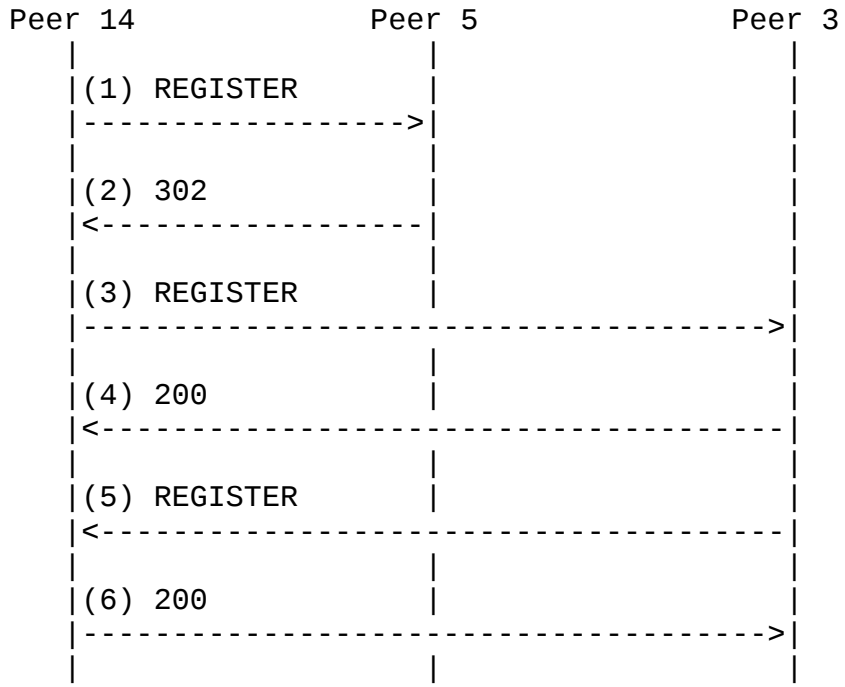
In each of the examples below, we assume we start from the network described above. Changes to the example network from previous examples are discarded.

Note that for simplicity we do not show user registration redundancy in any examples. This includes responses -- we only send predecessor and successor, as well as finger table -- not redundant successors.

13.1. Example of a Peer Registration

Assume a new peer wishes to join the system. The peer has an IP address of 10.0.0.14, which we shall assume hashes to a Peer-ID of 14. From an out of band mechanism, this peer discovers peer 5. This peer constructs a REGISTER as described in Peer Registration (Section 7.1), and sends it to peer 5. Peer 5 verifies that 10.0.0.14 hashes to 14, then checks to see if it controls that portion of the namespace. Since it does not, it looks up in its finger table where it would route a search for 14, and determines it would send it to peer 3. The peer then sends a 302 back to peer 14, with a contact of peer 3.

Peer 14 the constructs a new REGISTER and sends it to Peer 3. Again, Peer 3 verifies the hash, and determines it is currently responsible for 14 in the hash space. After an optional challenge, it replies with a 200 OK message to admit the peer to the system. Finally, Peer 3 sends a third party registration on behalf of bob to Peer 14, transferring bob's registration to the new peer.



Peer 14 -> Peer 5

```
REGISTER sip:10.0.0.5 SIP/2.0
To: <sip:14@10.0.0.14;user=peer>
From: <sip:14@10.0.0.14;user=peer>
Contact: <sip:14@10.0.0.14;user=peer>
Expires: 600
DHT-PeerID: <sip:14@10.0.0.14;user=peer>;algorithm=sha1;overlay=chat;
            expires=600
Require: dht
Supported: dht
```

Peer 5 -> Peer 14

```
SIP/2.0 302 Moved Temporarily
To: <sip:14@10.0.0.14;user=peer>
From: <sip:14@10.0.0.14;user=peer>
Contact: <sip:3@10.0.0.3;user=peer>
DHT-PeerID: <sip:5@10.0.0.5;user=peer>;algorithm=sha1;overlay=chat;
            expires=1200
DHT-Link: <sip:3@10.0.0.3;user=peer>;link=P1;expires=427
DHT-Link: <sip:10@10.0.0.10;user=peer>;link=S1;expires=387
Require: dht
Supported: dht
```

Peer 14 -> Peer 3

```
REGISTER sip:10.0.0.3 SIP/2.0
To: <sip:14@10.0.0.14;user=peer>
From: <sip:14@10.0.0.14;user=peer>
Contact: <sip:14@10.0.0.14;user=peer>
Expires: 600
DHT-PeerID: <sip:14@10.0.0.14;user=peer>;algorithm=sha1;overlay=chat;
            expires=600
Require: dht
Supported: dht
```

Peer 3 -> Peer 14

```
SIP/2.0 200 OK
To: <sip:14@10.0.0.14;user=peer>
From: <sip:14@10.0.0.14;user=peer>
Contact: <sip:14@10.0.0.14;user=peer>
Expires: 600
DHT-PeerID: <sip:3@10.0.0.3;user=peer>;algorithm=sha1;overlay=chat;
            expires=600
DHT-Link: <sip:10@10.0.0.10;user=peer>;link=P1;expires=125
```

```
DHT-Link: <sip:5@10.0.0.5;user=peer>;link=S1;expires=919
DHT-Link: <sip:5@10.0.0.5;user=peer>;link=F0;expires=919
DHT-Link: <sip:5@10.0.0.5;user=peer>;link=F1;expires=919
DHT-Link: <sip:10@10.0.0.10;user=peer>;link=F2;expires=125
DHT-Link: <sip:3@10.0.0.3;user=peer>;link=F3;expires=600
Require: dht
Supported: dht


Peer 3 -> Peer 14

REGISTER sip:10.0.0.14 SIP/2.0
To: <sip:bob@p2psip.org;resourceID=12>
From: <sip:3@10.0.0.3;user=peer>
Contact: <sip:bob@10.0.0.10>
Expires: 201
DHT-PeerID: <sip:3@10.0.0.3;user=peer>;algorithm=sha1;overlay=chat;
            expires=600
Require: dht
Supported: dht


Peer 14 -> Peer 3

SIP/2.0 200 OK
To: <sip:bob@p2psip.org;resourceID=12>
From: <sip:3@10.0.0.3;user=peer>
Contact: <sip:bob@10.0.0.10>
Expires: 201
DHT-PeerID: <sip:14@10.0.0.14;user=peer>;algorithm=sha1;overlay=chat;
            expires=600
Require: dht
Supported: dht
```

13.2. Example of a User Registration

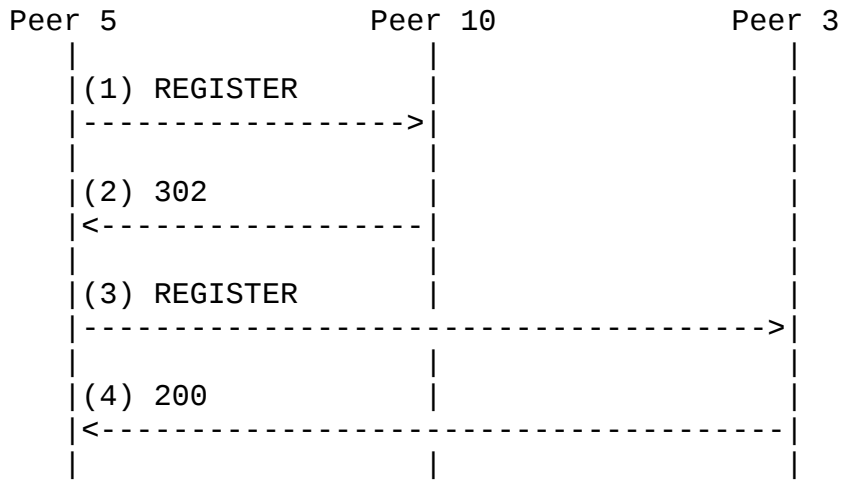
Assume user Carl starts a UA co-located with peer 5. Carl's contact will be carl@10.0.0.5, and his user name will be carl@p2psip.org. Carl's Peer hashes his user id and determines that the corresponding Resource-ID will be 11 -- that is, Carl's registration will be stored by the peer responsible for Resource-ID 11 -- ultimately Peer 3 in our example.

Carl's UA begins by constructing a SIP REGISTER message as described in Resource Registrations (Section 11.1). Carl's UA consults its finger table, and determines that it should route requests pertaining to a Resource-ID of 11 to Peer 10. The REGISTER is sent to Peer 10,

which observes that it is not responsible for that portion of the namespace, and consults the finger table, finding Peer 3 in the appropriate entry. Peer 10 sends a 302 containing Peer 3 as a contact.

Peer 5 constructs a new REGISTER on behalf of carl, and sends it to Peer 3. Peer 3 recognizes that it is responsible for storing this registration, and replies with a 200 OK (although in reality it might challenge in some way). The 200 contains some number of successor peers -- in this case 2 (although in our contrived example, one is peer 5 itself) that Carl's peer could send redundant registrations to. In our example, we do not show these. The 200 also (like 302s) must contain successors/predecessors in case the request is being used for stabilization. Again, in the tiny contrived example it looks odd since the second successor is the same as the predecessor. In a larger example this would not be the case.

[To Do: Maybe use a bigger example to fix these problems? That might be to big and ugly. Need a good way to show this]



Peer 5 -> Peer 10

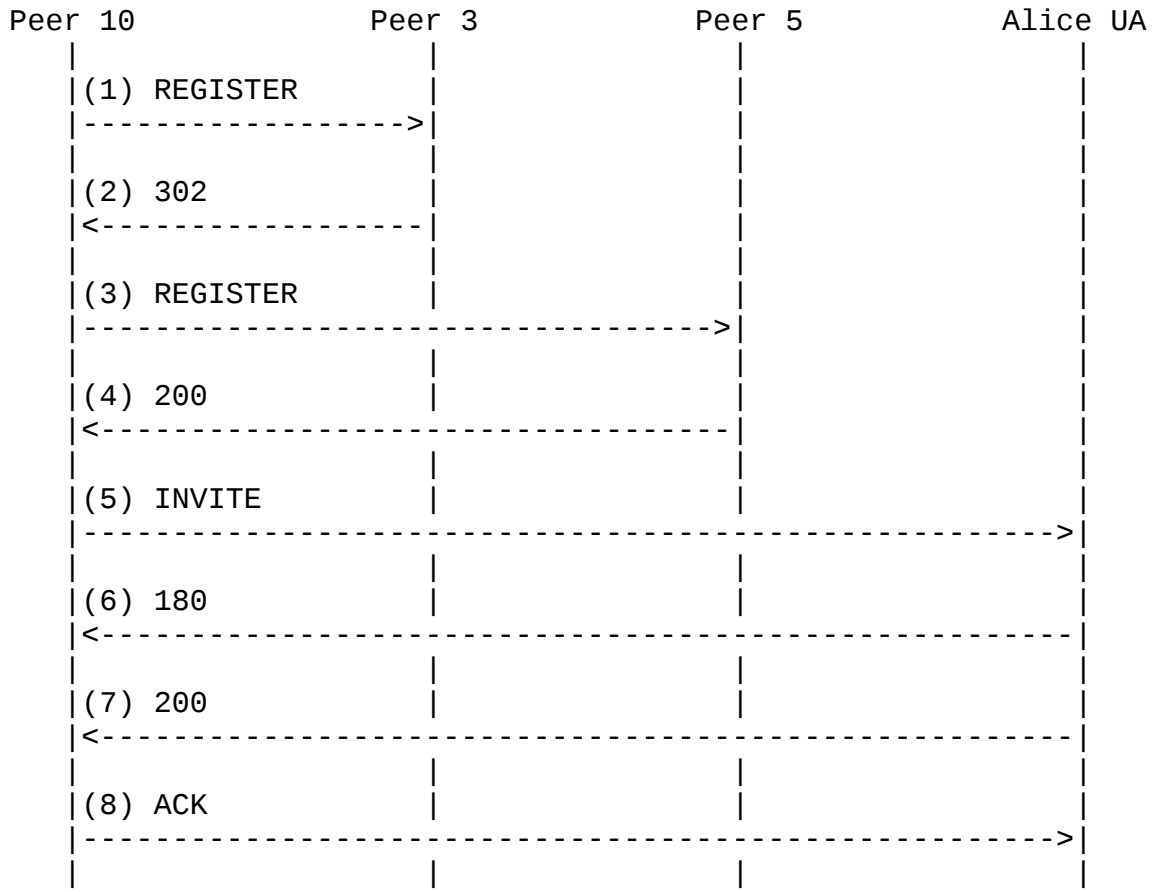
```
REGISTER sip:10.0.0.10 SIP/2.0
To: <sip:carl@p2psip.org;resourceID=11>
From: <sip:carl@p2psip.org;resourceID=11>
Contact: <sip:carl@10.0.0.5>
Expires: 600
DHT-PeerID: <sip:5@10.0.0.5;user=peer>;algorithm=sha1;overlay=chat;
            expires=1200
Require: dht
```

```
   Supported: dht


   Peer 10 -> Peer 5

   SIP/2.0 302 Moved Temporarily
   Contact: <sip:3@10.0.0.3;user=peer>
   DHT-PeerID: <sip:10@10.0.0.10;user=peer>;algorithm=sha1;overlay=chat;
               expires=800
   DHT-Link: <sip:5@10.0.0.5;user=peer>;link=P1;expires=1200
   DHT-Link: <sip:3@10.0.0.3;user=peer>;link=S1;expires=412
   Require: dht
   Supported: dht


   Peer 5 -> Peer 3

   REGISTER sip:10.0.0.3 SIP/2.0
   To: <sip:carl@p2psip.org;resourceID=11>
   From: <sip:carl@p2psip.org;resourceID=11>
   Contact: <sip:carl@10.0.0.5>
   Expires: 600
   DHT-PeerID: <sip:5@10.0.0.5;user=peer>;algorithm=sha1;overlay=chat;
               expires=1200
   Require: dht
   Supported: dht


   Peer 3 -> Peer 5

   SIP/2.0 200 OK
   To: <sip:carl@p2psip.org;resourceID=11>
   From: <sip:carl@p2psip.org;resourceID=11>
   Contact: <sip:carl@10.0.0.5>
   Expires: 600
   DHT-PeerID: <sip:3@10.0.0.3;user=peer>;algorithm=sha1;overlay=chat;
               expires=600
   DHT-Link: <sip:10@10.0.0.10;user=peer>;link=P1;expires=405
   DHT-Link: <sip:5@10.0.0.5;user=peer>;link=S1;expires=1200
   DHT-Link: <sip:10@10.0.0.10;user=peer>;link=S2;expires=405
   Require: dht
   Supported: dht
```

13.3. Example of a Session Establishment

Assume user Bob wishes to call user Alice. Bob's peer hashes Alice's user id, resulting in a Resource-ID of 5. Bob's peer (recall that

Bob's UA is co-located with peer 10) consults its finger table, and determines that a request for Resource-ID 5 should be routed to Peer 3. A REGISTER query message is constructed and routed to Peer 3. Peer 3 determines it is not responsible for a Resource-ID of 5, looks up the ID in its finger table and determines it should be routed to Peer 5, so it returns a 302 referring to Peer 5. Bob's peer resends the REGISTER to Peer 5, which stores Alice's information. It sends a 200 with Alice's contact -- sipchat/alice@10.99.99.99. Bob finally sends an INVITE to Alice's UA, and session establishment is completed as normal.



Peer 10 -> Peer 3

```
REGISTER sip:10.0.0.3 SIP/2.0
To: <sip:alice@p2psip.org;resource-ID=5>
From: <sip:bob@p2psip.org;resource-ID=12>
DHT-PeerID: <sip:10@10.0.0.10;user=peer>;algorithm=sha1;overlay=chat;
            expires=800
```

```
   Require: dht
   Supported: dht


   Peer 3 -> Peer 10

   SIP/2.0 302 Moved Temporarily
   To: <sip:alice@p2psip.org;resource-ID=5>
   From: <sip:bob@p2psip.org;resource-ID=12>
   Contact: <sip:5@10.0.0.5;user=peer>
   DHT-PeerID: <sip:3@10.0.0.3;user=peer>;algorithm=sha1;overlay=chat;
               expires=600
   DHT-Link: <sip:10@10.0.0.10;user=peer>;link=P1;expires=421
   DHT-Link: <sip:5@10.0.0.5;user=peer>;link=S1;expires=1004
   Require: dht
   Supported: dht


   Peer 10 -> Peer 5

   REGISTER sip:10.0.0.5 SIP/2.0
   To: <sip:alice@p2psip.org;resource-ID=5>
   From: <sip:bob@p2psip.org;resource-ID=12>
   DHT-PeerID: <sip:10@10.0.0.10;user=peer>;algorithm=sha1;overlay=chat;
               expires=800
   Require: dht
   Supported: dht


   Peer 5 -> Peer 10

   SIP/2.0 200 OK
   To: <sip:alice@p2psip.org;resource-ID=5>
   From: <sip:bob@p2psip.org;resource-ID=12>
   Contact: <sip:alice@10.99.99.99>
   DHT-PeerID: <sip:5@10.0.0.5;user=peer>;algorithm=sha1;overlay=chat;
               expires=1200
   DHT-Link: <sip:3@10.0.0.3;user=peer>;link=P1;expires=108
   DHT-Link: <sip:10@10.0.0.10;user=peer>;link=S1;expires=492
   Require: dht
   Supported: dht


   Peer 10 -> Alice UA

   INVITE sip:alice@p2psip.org SIP/2.0
   To: <sip:alice@p2psip.org>
   From: <sip:bob@p2psip.org>
```

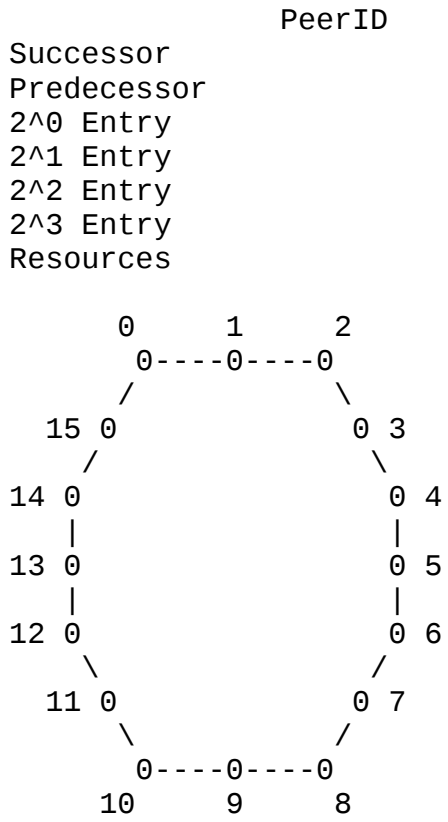
```
Contact: <sip:bob@10.0.0.10>
DHT-PeerID: <sip:10@10.0.0.10;user=peer>;algorithm=sha1;overlay=chat;
            expires=800
Supported: dht
```

The remainder of the call is completed as any other SIP call. Note that if Alice's UA is DHT-compliant, then it will recognize the Supported field and DHT-PeerID header, and may respond with similar fields. However, if it does not support DHT extensions, it will simply ignore those values and complete the call as any normal non-P2P SIP UA.

13.4. Example of Moving From Empty Overlay to Stable 3 Peer System

In this example, we track the system state from overlay creation to a stable 3 peer overlay with 2 resources (user registrations) registered and stored in the system. This example will show how successor, predecessor, and finger table entries are updated during each step as well as show the P2P SIP messages exchanged.

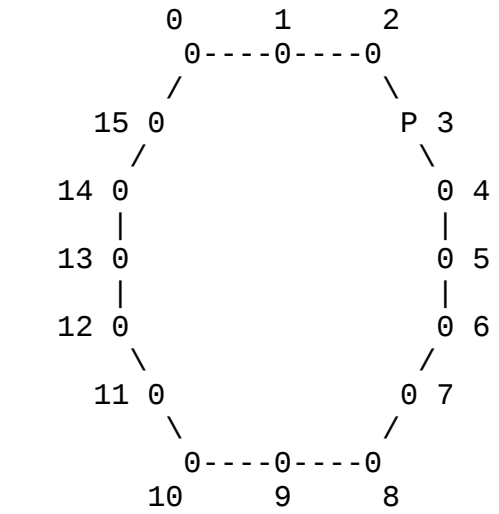
Assume we start with an empty overlay with a namespace of 16. Each peer in the system will have one successor, one predecessor and 4 finger table entries (2^4 namespace). Peer state will be shown in the following way:



Additionally, we will track the location of resources with a resource map.

A peer with PeerID 3, IP address 10.0.0.3, and port 5060 starts the overlay called chat. When a peer starts a new overlay, it sets its successor to be its PeerID and sets its predecessor to be NULL. Additionally, the peer initializes its finger table and sets all fingers to be its PeerID.

The resulting state of the system is:

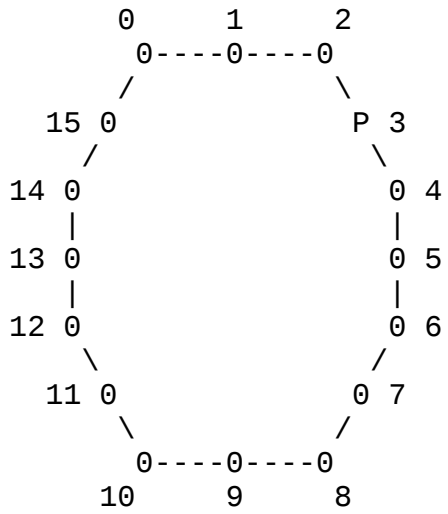


	Peer 3
Successor	3
Predecessor	NULL
2^0 Entry	[4,5) -> 3
2^1 Entry	[5,7) -> 3
2^2 Entry	[7,11) -> 3
2^3 Entry	[11,3) -> 3
Resources	

Resource Map

For peer 3, there is a resource called alice that must be registered with the system. alice's peer hashes her user id and determines that the corresponding resource will be 8. alice's contact will be sipchat/alice@10.0.0.3 and her user name will be alice@p2psip.org. Because there are no other peers in the system, peer 3 is responsible for all resources including alice's registration. Note that we use "resID" as a short form for "resourceID" to save room in the figure only -- resourceID is used in the actual messages.

As a result the system state changes to:



	Peer 3
Successor	3
Predecessor	NULL
2^0 Entry	[4,5) -> 3
2^1 Entry	[5,7) -> 3
2^2 Entry	[7,11) -> 3
2^3 Entry	[11,3) -> 3
Resources	alice;resID=8 -> 3

Resource Map

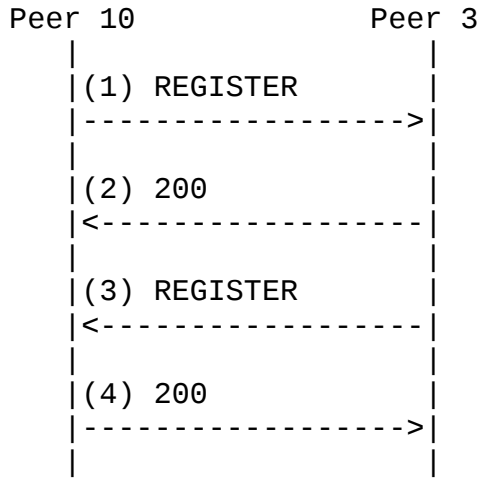
Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	3

Next a peer with PeerID 10, IP address 10.0.0.10, and port 5060 decides to join the overlay called chat. From an out of band mechanism, such as one of the ones listed in the Bootstraping section of this document, this peer discovers peer 3. This new peer 10, constructs a REGISTER as described in Peer Registration (Section 7.1) and sends it to peer 3.

Peer 3 verifies that 10.0.0.10 hashes to 10, then checks to see if it controls that portion of the namespace. Since Peer 3's predecessor is NULL and no other peers are in the system, Peer 3 determines that is currently responsible for peer 10 in the hash space. After an optional challenge, peer 3 replies with a 200 OK message to admit peer 10 into the system.

Peer 3 then sets its predecessor to be Peer 10. When Peer 10 receives the 200 OK message, it will set its successor to be Peer 3 and keeps its predecessor set to NULL (because no predecessor was in the 200 response).

Finally, Peer 3 sends a third party registration on behalf of alice to Peer 10, transferring alice's registration to the new peer.



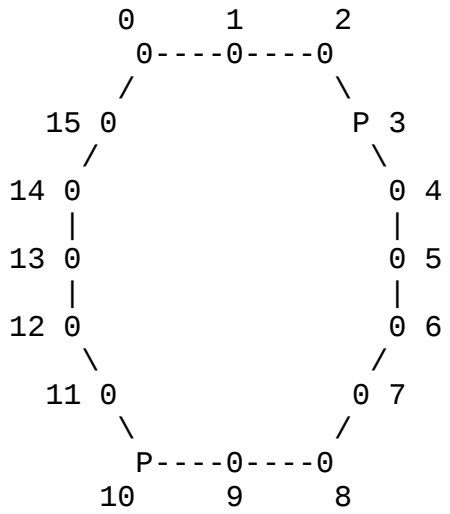
Peer 10 -> Peer 3

```
REGISTER sip:10.0.0.3:5060 SIP/2.0
To: <sip:10@10.0.0.10:5060;user=peer>
From: <sip:10@10.0.0.10:5060;user=peer>
Contact: <sip:10@10.0.0.10:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 3 -> Peer 10

```
SIP/2.0 200 OK
To: <sip:10@10.0.0.10:5060;user=peer>
From: <sip:10@10.0.0.10:5060;user=peer>
Contact: <sip:10@10.0.0.10:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
```

```
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
   Require: dht
   Supported: dht


   Peer 3 -> Peer 10

   REGISTER sip:10.0.0.10:5060 SIP/2.0
   To: <sip:alice@p2psip.org;resourceID=8>
   From: <sip:3@10.0.0.3:5060;user=peer>
   Contact: <sip:alice@10.0.0.3:5060>
   Expires: 201
   DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   Require: dht
   Supported: dht


   Peer 10 -> Peer 3

   SIP/2.0 200 OK
   To: <sip:alice@p2psip.org;resourceID=8>
   From: <sip:3@10.0.0.3:5060;user=peer>
   Contact: <sip:alice@10.0.0.3:5060>
   Expires: 201
   DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
   Require: dht
   Supported: dht


   The system state is now:
```



	Peer 3	Peer 10
Successor	3	3
Predecessor	10	NULL
2^0 Entry	[4,5) -> 3	[11,12) -> 3
2^1 Entry	[5,7) -> 3	[12,14) -> 3
2^2 Entry	[7,11) -> 3	[14, 2) -> 3
2^3 Entry	[11,3) -> 3	[2, 10) -> 3
Resources		alice;resID=8 -> 3

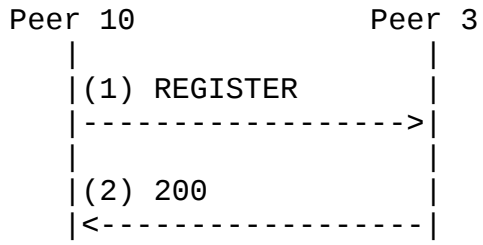
Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10

Peer 10 runs its periodic stabilization. It constructs a REGISTER as described in the Peer Query section, and sends it to its successor, Peer 3, querying for its successor's PeerID.

Peer 3 checks the query to determine if it is responsible for the region the search key lies within. Because Peer 3's PeerID directly matches the search key, it sends a 200 OK response message with its current successor and predecessor specified in the DHTLink headers.

Peer 10 examines the response from Peer 3. Because the predecessor in the response from Peer 3 is the same as Peer 10, the stabilizing peer, Peer 10 is not required to do any more work. Peer 10 should perform searches to update its finger table, but these are omitted for clarity.



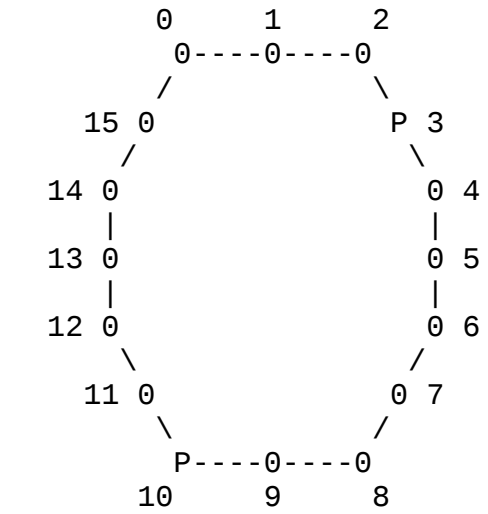
Peer 10 -> Peer 3

```
REGISTER sip:10.0.0.3:5060 SIP/2.0
To: <sip:3@0.0.0.0;user=peer>
From: <sip:10@10.0.0.10:5060;user=peer>
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 3 -> Peer 10

```
SIP/2.0 200 OK
To: <sip:3@0.0.0.0;user=peer>
From: <sip:10@10.0.0.0:5060;user=peer>
Contact: <sip:3@10.0.0.3:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:10@0.0.0.10:5060;user=peer>;link=P1;expires=0
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

The state stays unchanged after Peer 10's periodic stabilization.



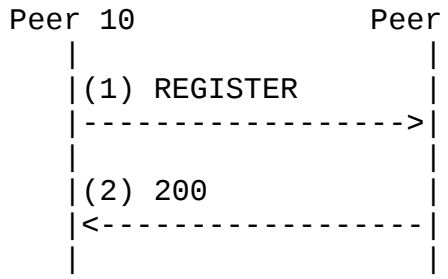
	Peer 3	Peer 10
Successor	3	3
Predecessor	10	NULL
2^0 Entry	[4,5) -> 3	[11,12) -> 3
2^1 Entry	[5,7) -> 3	[12,14) -> 3
2^2 Entry	[7,11) -> 3	[14, 2) -> 3
2^3 Entry	[11,3) -> 3	[2, 10) -> 3
Resources		alice;resID=8 -> 3

Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10

For peer 10, there is a resource called bob that must be registered with the system. bob's contact will be sipchat/bob@10.0.0.10 and his user name will be bob@p2psip.org. bob's peer hashes his user id and determines that the corresponding resource will be 11.

Bob's UA begins by constructing a SIP REGISTER message as described in Resource Registrations (Resource Registrations). Bob's UA consults its finger table, and determines that it should route requests pertaining to a Resource-ID of 5 to Peer 3. The REGISTER is sent to Peer 3, which observes that it is responsible for that portion of the namespace and replies with a 200 OK containing Peer 3's predecessor and successor information in the DHTLink headers.



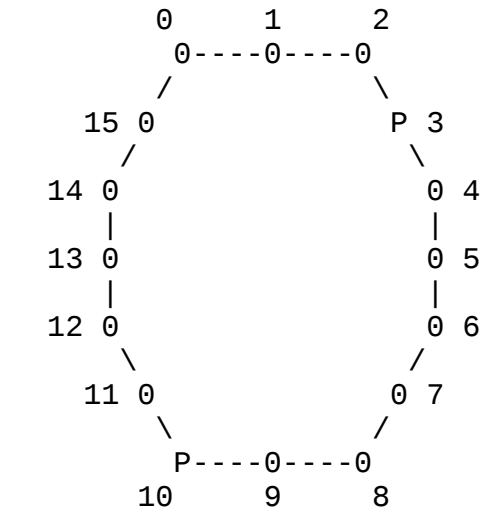
Peer 10 -> Peer 3

```
REGISTER sip:10.0.0.3:5060 SIP/2.0
To: <sip:bob@p2psip.org;resourceID=11>
From: <sip:bob@p2psip.org;resourceID=11>
Contact: <sip:bob@10.0.0.10:5060>
Expires: 231
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 3 -> Peer 10

```
SIP/2.0 200 OK
To: <sip:bob@p2psip.org;resourceID=11>
From: <sip:bob@p2psip.org;resourceID=11>
Contact: <sip:bob@10.0.0.10:5060>
Expires: 231
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:10@0.0.0.10:5060;user=peer>;link=P1;expires=600
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

After bob's registration, the system state is:



	Peer 3	Peer 10
Successor	3	3
Predecessor	10	NULL
2^0 Entry	[4,5) -> 3	[11,12) -> 3
2^1 Entry	[5,7) -> 3	[12,14) -> 3
2^2 Entry	[7,11) -> 3	[14, 2) -> 3
2^3 Entry	[11,3) -> 3	[2, 10) -> 3
Resources	bob;resID=11 -> 10	alice;resID=8 -> 3

Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10
bob	11	10	3

Peer 3 now runs its periodic stabilization. It constructs a REGISTER as described in the Peer Query section, and sends it to its successor, Peer 3, querying for its successor's PeerID.

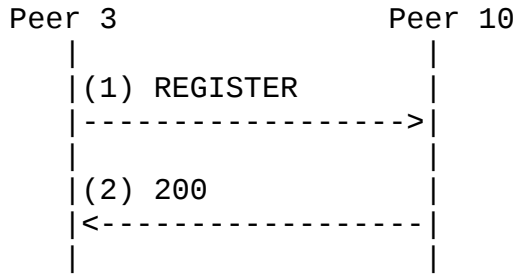
Peer 3 checks the query to determine if it is responsible for the region the search key lies within. Because Peer 3's PeerID directly matches the search key, it sends a 200 OK response message with its current successor and predecessor specified in the DHTLink headers.

Peer 3 examines the response from itself. Because the predecessor in the response from Peer 3 is Peer 10, Peer 3 updates its successor to be Peer 10 and sends a REGISTER to Peer 10, structured as if Peer 3 had just entered the system.

When Peer 10 receives the message, it then sends a 200 OK response to Peer 3. Then, Peer 10 sets its predecessor to be Peer 3 because Peer 10's predecessor was NULL.

Then Peer 3 should perform searches to update its finger table, but these are simple peer queries and are omitted in this example. We show the finger table as though these searches were performed.

Note that because Peer 3's successor is Peer 3, we do not show such a REGISTER message being sent because implementations may choose to remove this step for efficiency. Rather we show the message sent from Peer 3 to Peer 10 that notifies Peer 10 that Peer 3 is its predecessor.



Peer 3 -> Peer 10

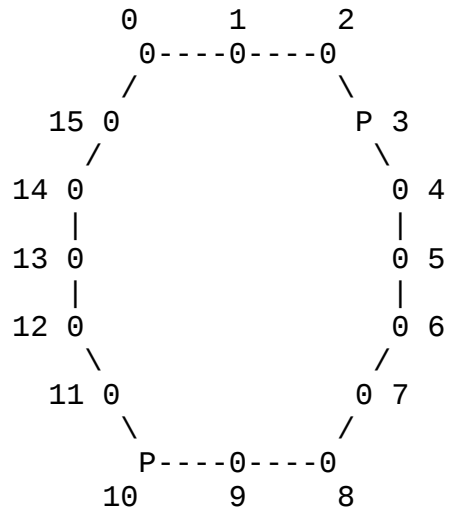
```
REGISTER sip:10.0.0.10:5060 SIP/2.0
To: <sip:3@10.0.0.3:5060;user=peer>
From: <sip:3@10.0.0.3:5060;user=peer>
Contact: <sip:3@10.0.0.3:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:3@3.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 10 -> Peer 3

```
SIP/2.0 200 OK
To: <sip:3@10.0.0.3:5060;user=peer>
From: <sip:3@10.0.0.3:5060;user=peer>
Contact: <sip:3@10.0.0.3:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
```

```
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

After Peer 3's stabilization, the system state is:



	Peer 3	Peer 10
Successor	10	3
Predecessor	10	3
2^0 Entry	[4,5) -> 10	[11,12) -> 3
2^1 Entry	[5,7) -> 10	[12,14) -> 3
2^2 Entry	[7,11) -> 10	[14, 2) -> 3
2^3 Entry	[11,3) -> 3	[2, 10) -> 3
Resources	bob;resID=11 -> 10	alice;resID=8 -> 3

Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10
bob	11	10	3

Next a peer with PeerID 2, IP address 10.0.0.2, and port 5060 decides to join the overlay called chat. From an out of band mechanism, such as one of the ones listed in the Bootstraping section of this document, this peer discovers peer 10. This new peer 2, constructs a REGISTER as described in Peer Registration (Section 7.1) and sends it to peer 10.

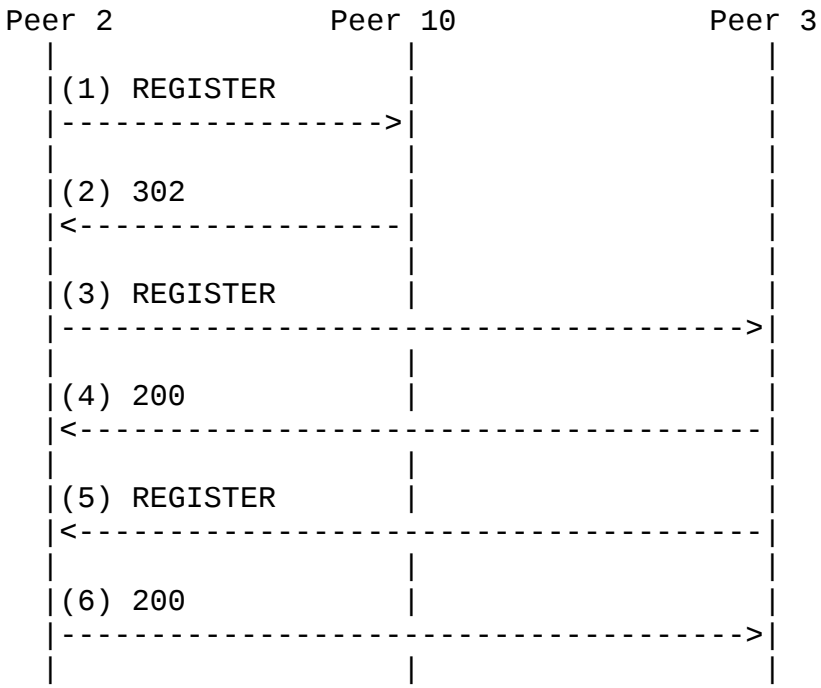
Peer 10 verifies that 10.0.0.2 hashes to 2, then checks to see if it

controls that portion of the namespace. Since it does not, it looks up in its finger table where it would route a search for 2, and determines it would send it to peer 3. The peer then sends a 302 back to peer 2, with a contact of peer 3.

Peer 2 then constructs a new REGISTER and sends it to Peer 3. Again, Peer 3 verifies the hash, and determines it is currently responsible for 2 in the hash space. After an optional challenge, it replies with a 200 OK message to admit the peer to the system.

After sending the 200 response, Peer 3 then sets its predecessor to be Peer 2. When Peer 2 receives the 200 OK message, it will set its successor to be Peer 3 and set its predecessor to be Peer 10 (because that was the predecessor in the 200 response). Peer 2 should also perform searches as discussed in Populating the Joining Peer's Finger Table (Section 9.4) to ensure that the finger table is up to date. These searches are omitted, but we update the finger table.

Finally, Peer 3 sends a third party registration on behalf of bob to Peer 2, transferring bob's registration to the new peer.



Peer 2 -> Peer 10

REGISTER sip:10.0.0.10:5060 SIP/2.0

```
   To: <sip:2@10.0.0.2:5060;user=peer>
   From: <sip:2@10.0.0.2:5060;user=peer>
   Contact: <sip:2@10.0.0.2:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:2@10.0.0.2:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   Require: dht
   Supported: dht


   Peer 10 -> Peer 2

   SIP/2.0 302 Moved Temporarily
   To: <sip:2@10.0.0.2:5060;user=peer>
   From: <sip:2@10.0.0.2:5060;user=peer>
   Contact: <sip:3@10.0.0.3:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   DHT-Link: <sip:3@0.0.0.3:5060;user=peer>;link=P1;expires=919
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
   Require: dht
   Supported: dht


   Peer 2 -> Peer 3

   REGISTER sip:10.0.0.3:5060 SIP/2.0
   To: <sip:2@10.0.0.2:5060;user=peer>
   From: <sip:2@10.0.0.2:5060;user=peer>
   Contact: <sip:2@10.0.0.2:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:2@10.0.0.2:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   Require: dht
   Supported: dht


   Peer 3 -> Peer 2

   SIP/2.0 200 OK
   To: <sip:2@10.0.0.2:5060;user=peer>
   From: <sip:2@10.0.0.2:5060;user=peer>
   Contact: <sip:2@10.0.0.2:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   DHT-Link: <sip:10@0.0.0.10:5060;user=peer>;link=P1;expires=419
```

```
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=S1;expires=419
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

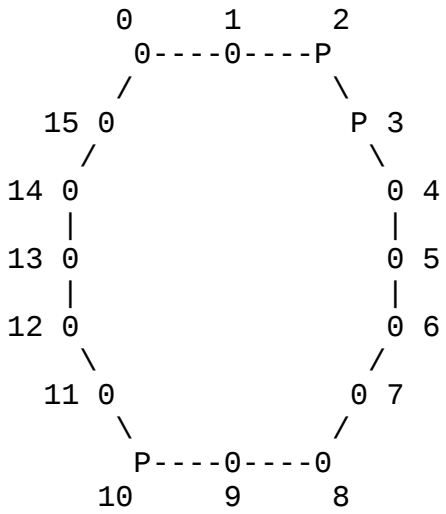
Peer 3 -> Peer 2

```
REGISTER sip:2.0.0.2:5060 SIP/2.0
To: <sip:bob@p2psip.org;resourceID=8>
From: <sip:3@10.0.0.3:5060;user=peer>
Contact: <sip:bob@10.0.0.10:5060>
Expires: 201
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 2 -> Peer 3

```
SIP/2.0 200 OK
To: <sip:bob@p2psip.org;resourceID=8>
From: <sip:3@10.0.0.3:5060;user=peer>
Contact: <sip:bob@10.0.0.3:5060>
Expires: 201
DHT-PeerID: <sip:2@10.0.0.2:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>link=P1;expires=800
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

After Peer 2's insertion, the system state is:



	Peer 2	Peer 3	Peer 10
Successor	3	10	3
Predecessor	10	2	3
2^0 Entry	[3,4) -> 3	[4,5) -> 10	[11,12) -> 3
2^1 Entry	[4,6) -> 10	[5,7) -> 10	[12,14) -> 3
2^2 Entry	[6,10) -> 10	[7,11) -> 10	[14, 2) -> 3
2^3 Entry	[10,2) -> 10	[11, 3) -> 3	[2, 10) -> 3
Resources	bob;resID=11 -> 10		alice;resID=8 -> 3

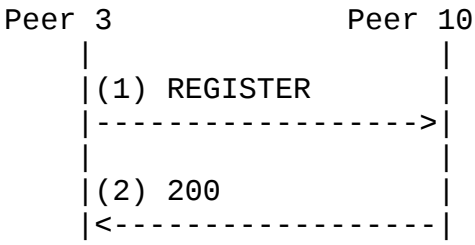
Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10
bob	11	10	2

Peer 3 now runs its periodic stabilization. It constructs a REGISTER as described in the Peer Query section, and sends it to its successor, Peer 10, querying for its successor's PeerID.

Peer 10 checks the query to determine if it is responsible for the region the search key lies within. Because Peer 10's PeerID directly matches the search key, it sends a 200 OK response message with its current successor and predecessor specified in the DHTLink headers.

Peer 3 examines the response from itself. Because the predecessor in the response form Peer 10 is the same as Peer3, the stabilizing peer, Peer 3 does not need to send any messages to the predecessor. At this point Peer 3 should send queries as discussed in Populating the Joining peer's Finger Table (Section 9.4) to ensure that the finger table is up to date. We do not show the SIP messages for this process, but do show the resulting changes in Peer 3's finger table.



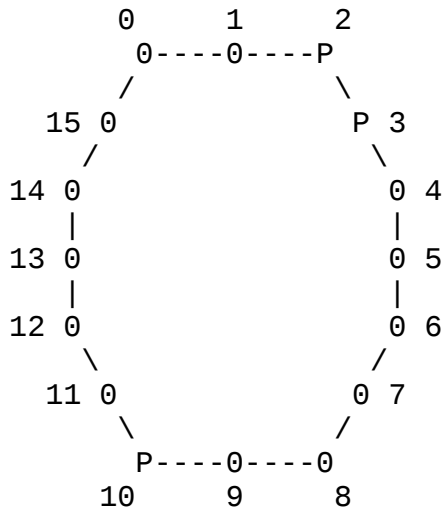
Peer 10 -> Peer 3

```
REGISTER sip:10.0.0.10:5060 SIP/2.0
To: <sip:10@0.0.0.0;user=peer>
From: <sip:3@10.0.0.3:5060;user=peer>
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 10 -> Peer 3

```
SIP/2.0 200 OK
To: <sip:10@0.0.0.0;user=peer>
From: <sip:3@10.0.0.3:5060;user=peer>
Contact: <sip:10@10.0.0.10:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=P1;expires=0
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

The state after Peer 3's periodic stabilization:



	Peer 2	Peer 3	Peer 10
Successor	3	10	3
Predecessor	10	2	3
2^0 Entry	[3,4) -> 3	[4,5) -> 10	[11,12) -> 3
2^1 Entry	[4,6) -> 10	[5,7) -> 10	[12,14) -> 3
2^2 Entry	[6,10) -> 10	[7,11) -> 10	[14, 2) -> 3
2^3 Entry	[10,2) -> 10	[11, 3) -> 2	[2, 10) -> 3
Resources	bob;resID=11->10		alice;resID=8 -> 3

Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10
bob	11	10	2

Peer 10 now runs its periodic stabilization. It constructs a REGISTER as described in the Peer Query section, and sends it to its successor, Peer 3, querying for its successor's PeerID.

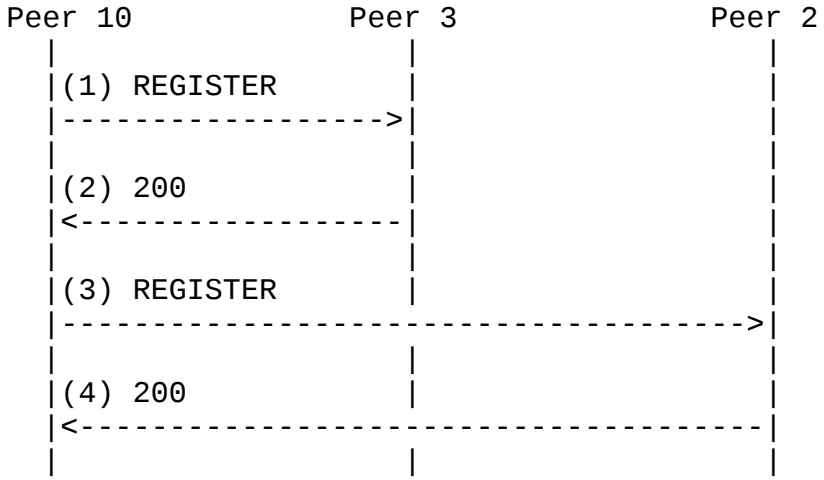
Peer 3 checks the query to determine if it is responsible for the region the search key lies within. Because Peer 3's PeerID directly matches the search key, it sends a 200 OK response with its current successor and predecessor in the response.

Because the predecessor in the response from Peer 3 is Peer 2, Peer 10 updates its successor to be Peer 2 and sends a REGISTER to Peer 2, structured as if Peer 10 had just entered the system.

When Peer 2 receives the message, it then sends a 200 OK response to Peer 10. Then, Peer 2 updates its predecessor to be Peer 10.

Then Peer 10 should perform searches to update its finger table.

These are simple peer queries and are omitted in this example, but we have updated the finger table.



Peer 10 -> Peer 3

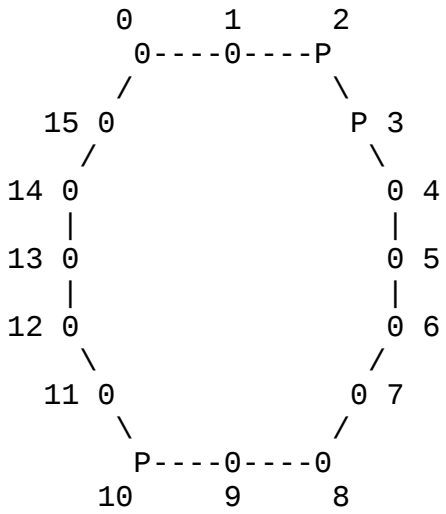
```
REGISTER sip:3.0.0.3:5060 SIP/2.0
To: <sip:3@0.0.0.0;user=peer>
From: <sip:10@10.0.0.10:5060;user=peer>
DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
Require: dht
Supported: dht
```

Peer 3 -> Peer 10

```
SIP/2.0 200 OK
To: <sip:3@0.0.0.0;user=peer>
From: <sip:10@10.0.0.10:5060;user=peer>
Contact: <sip:3@10.0.0.3:5060;user=peer>
Expires: 600
DHT-PeerID: <sip:3@10.0.0.3:5060;user=peer>;algorithm=sha1;
            overlay=chat;expires=600
DHT-Link: <sip:2@10.0.0.2:5060;user=peer>;link=P1;expires=0
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=S1;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F0;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F1;expires=919
DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F2;expires=125
DHT-Link: <sip:2@10.0.0.2:5060;user=peer>;link=F3;expires=600
Require: dht
Supported: dht
```

```
   Peer 10 -> Peer 2

   REGISTER sip:10.0.0.2:5060 SIP/2.0
   To: <sip:10@10.0.0.10:5060;user=peer>
   From: <sip:10@10.0.0.10:5060;user=peer>
   Contact: <sip:10@10.0.0.10:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:10@10.0.0.10:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   Require: dht
   Supported: dht


   Peer 2 -> Peer 10

   SIP/2.0 200 OK
   To: <sip:10@10.0.0.10:5060;user=peer>
   From: <sip:10@10.0.0.10:5060;user=peer>
   Contact: <sip:10@10.0.0.10:5060;user=peer>
   Expires: 600
   DHT-PeerID: <sip:2@10.0.0.2:5060;user=peer>;algorithm=sha1;
               overlay=chat;expires=600
   DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=P1;expires=419
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=S1;expires=419
   DHT-Link: <sip:3@10.0.0.3:5060;user=peer>;link=F0;expires=919
   DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F1;expires=919
   DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F2;expires=125
   DHT-Link: <sip:10@10.0.0.10:5060;user=peer>;link=F3;expires=600
   Require: dht
   Supported: dht
```

After Peer 10's stabilization, the system state is:



	Peer 2	Peer 3	Peer 10
Successor	3	10	2
Predecessor	10	2	3
2^0 Entry	[3,4) -> 3	[4,5) -> 10	[11,12) -> 3
2^1 Entry	[4,6) -> 10	[5,7) -> 10	[12,14) -> 3
2^2 Entry	[6,10) -> 10	[7,11) -> 10	[14, 2) -> 3
2^3 Entry	[10,2) -> 10	[11, 3) -> 2	[2, 10) -> 2
Resources	bob;resID=11 -> 10		alice;resID=8 -> 3

Resource Map

Resource name	ResID	ResLocation	ResStorage Location
alice	8	3	10
bob	11	10	2

This system now has 3 peers in a stable state, such that all predecessor and successor information is correct, and two user resources are stored in the overlay.

13.5. Example of a Peer Leaving the System

[To Do: Add an example here]

13.6. Example of a Successful User Search

[To Do: Add an example here]

13.7. Example of an Unsucessful User Search

[To Do: Add an example here]

14. Security Considerations

The goal of P2P SIP 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. Instead, P2P SIP offers a range of security models that should be selected according to the needs of the overlay.

14.1. Threat Model

Without other security, the attacker is able to generate an identity 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 P2P SIP are also threats to regular C/S SIP. As such, P2P SIP imports much of its security from C/S. However, because C/S SIP generally relies on secure servers to maintain the integrity of the system, modifications to the C/S techniques are required to maintain the same level of security.

14.2. 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 ring where the user is registered, and on the overlay itself, by means of a Sybil [13] 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.

14.2.1. 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. While this is not a guarantee of security, it is a reasonable assurance that security needs will be met.

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.

14.2.2. Cryptographic Protection

Stronger security guarantees are possible through the extensions to sip-identity discussed in Section 12. For use in protecting the namespace, the hmac-sha1 identity prevents attackers from entering the ring unless they know the secret key. hmac-sha1 is most useful in small networks.

Larger networks, or those with more stringent security requirements, cannot rely on a single shared secret and must, instead, rely on a PKI to protect the network. Although this introduces some amount of centralization into the protocol, contacting a CA for a certificate is only required at the time of initial enrollment and not on subsequently connecting to the network.

Two options are available for protecting the ID namespace using user-

certificates. In the first, any peer that can authenticate with a user-certificate (issued to a user) is allowed to join and selects its Peer-ID in the usual manner. This authenticates that the CA has issued an identity for this user, but provides no protection to the overlay if an attacker either steals a user's certificate or convinces the CA to issue a certificate to them.

In the second, the user-certificate itself contains a SubjectAltName that specifies the Peer-ID to be used. The second approach has the advantage that it eliminates the difficulty of registering for an overlay from a node that is visible from multiple IP addresses, either from multiple interfaces or because it is behind a NAT that does not hairpin. It also prevents an attacker who acquires a certificate from using the certificate to launch attacks against the system by creating multiple IDs from different IP addresses. As such, overlays wishing to secure themselves from attackers SHOULD use user-certificates that specify the Peer-ID the user is authorized to use. Users who wish to join the overlay from multiple devices simultaneously MUST acquire a separate certificate for each device. The CA SHOULD issue these devices consecutive Peer-IDs.

14.3. Protecting the resource namespace

For secured networks, the same sip-identity techniques that protect the ID namespace can be used to protect the resource (username) namespace. However, there are two remaining concerns. First, addressing unsecured networks, and second, addressing the random DoS attacks possible when an attacker gains a certificate and simply DoSes any resource for which they are responsible.

In an unsecured network, multiple peers can register the same resource (username) in the overlay. However, self-signed certificates [9] can be used to authenticate a user as the same user previously contacted with that certificate, thus establishing identity upon initial contact is the remaining problem. If a C/S credential server is available, this problem is solved, otherwise it is up to the user or another external service to address initial authentication. Furthermore, an overlay that is expected to persist over long time-frames can be configured to store the credentials of previous users for verification of a new registration. These techniques are beyond the scope of this document.

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. 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 and can misdirect any message which

is routed through it. The primary means of defense of such attacks is relying on the replication described in Section Section 8.2.2. 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.

14.4. 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. However, appropriate use of user-certificates can alleviate the ability of an attacker to compromise multiple peers, thus rendering this attack more difficult.

Further protection is possible by performing multiple searches in the same region of the namespace with different starting peers, assuming that those other peers were not learned from a single subverted peer. We currently anticipate that replicas are sufficient to address these issues, but a UA MAY implement this to validate the routing it receives.

14.5. Protecting the Signaling

The goal here is to stop the 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 location randomization discussed above.

14.6. Protecting the Media

All the media SHOULD be S/MIME encrypted. Doing so reduces the value of intercepting others' communications, because the media cannot be seen in the message. This is critical.

14.7. Replay Attacks

Replay attacks are defended by using any of the sip-identity techniques described above.

15. Open Issues

There are certainly many open issues. Here are a few.

Still to be worked out are details of how P2PSIP names are disambiguated from traditional names that use DNS based routing.

Should it be possible to trigger a node to recheck a finger table entry after it 302s to a node that appears to be down? Presumably this can be integrated together with the loose routing NAT traversal.

Are certificate chains being handled properly, particularly when used with multiple-CA overlays or for use by UAs outside the overlay?

16. Acknowledgments

Thanks to all who have been actively participating in the P2PSIP efforts. In particular, thanks to Marcia Zangrilli Bryan, who helped to correct, expand and check much of the text on the DHT algorithms presented here, provided overall editorial feedback, and wrote much of the example section. Thanks also to Spencer Dawkins, Enrico Marocco, and Jean-Francois Wauthy for providing editorial feedback.

17. IANA Considerations

This document would require registering the following:

- Option tag "DHT"
- "DHT-Link" as a Header Field
- "DHT-PeerID" as a Header Field
- "peer" as a valid value for parameter user (?)
- "Resource-ID" as a valid URI parameter (?)
- "hmac-sha1" as an Identity-Info 'alg' parameter

[ToDo: This section needs to be revamped to include all the new BNF introduced]

18. Changes to this Version

1. We have attempted to use the new terminology defined in [2] wherever possible, and have attempted not to replicate definitions here. In particular, we have substituted the use of the term "peer" for "node"
2. As a consequence of the above, NodeID has been replaced with PeerID, both in text and in the actual defined messages sent over the wire.
3. We have made many changes to include details essential to using this in real deployed systems or clarifying difficult concepts; lessons learned from building a commercial application based on this draft.
4. Large parts of the description of how an initial overlay is formed were quite confusing as our description did not explicitly embrace the NULL predecessor concept of Chord. We have corrected this in the sections describing the algorithms.

5. A full and detailed example showing the startup of a 3 node system has been inserted into the examples section.
6. A new section has been added detailing early work on incorporating SIP identity into a P2P environment. This work is then used in the security section.
7. The security section has been thoroughly rewritten to reflect changes both in our thoughts and the thoughts of the P2PSIP working group as a whole.
8. We corrected a number of outright errors and typos pointed by a number of individuals, as mentioned in the acknowledgments.

19. References

19.1. Normative References

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

[2] Willis, D., "Concepts and Terminology for Peer to Peer SIP", draft-willis-p2psip-concepts-02 (work in progress), October 2006.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[4] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[5] Rosenberg, J., "Simple Traversal Underneath Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-04 (work in progress), July 2006.

[6] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

[7] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[8] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-11 (work in progress), October 2006.

[9] Jennings, C., "Certificate Management Service for The Session Initiation Protocol (SIP)", draft-ietf-sip-certs-01 (work in

progress), June 2006.

19.2. Informative References

[10] Bryan, D., Shim, E., and B. Lowekamp, "Use Cases for Peer-to-Peer Session Initiation Protocol (P2PSIP)", Internet Draft draft-bryan-sipping-p2p-usecases-00, November 2005.

[11] Bryan, D., Jennings, C., and B. Lowekamp, "SOSIMPLE: A Serverless, Standards-based, P2P SIP Communication System", Proceedings of the 2005 International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA) '05, June 2005.

[12] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", IEEE/ACM Transactions on Networking Volume 11, Issue 1, 17-32, Feb 2003.

[13] Douceur, J., "The Sybil Attack", IPTPS '02, March 2002.

[14] Cao, F., Bryan, D., and B. Lowekamp, "Providing Secure Services in Peer-to-Peer Communications Networks with Central Security Server", Internation Conference on Internet and Web Applications and Services (ICIW) '06, February 2006.

Authors' Addresses

David A. Bryan
SIPeerior; William & Mary
3000 Easter Circle
Williamsburg, VA 23188
USA

Phone: +1 757 784 5601
Email: bryan@ethernot.org

Bruce B. Lowekamp
William & Mary; SIPeerior
Department of Computer Science
P.O. Box 8795
Williamsburg, VA 23187
USA

Phone:
Email: lowekamp@cs.wm.edu

Cullen Jennings
Cisco Systems
170 West Tasman Drive
MS: SJC-21/3
San Jose, CA 95134
USA

Phone: +1 408 421 9990
Email: fluffy@cisco.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.