Using an External DHT as a SIP Location Service

Kundan Singh and Henning Schulzrinne
Department of Computer Science, Columbia University
{kns10,hgs}@cs.columbia.edu

Abstract

Peer-to-peer Internet telephony using the Session Initiation Protocol (P2P-SIP) can exhibit two different architectures: an existing P2P network can be used as a replacement for lookup and updates, or a P2P algorithm can be implemented using SIP messages. In this paper, we explore the first architecture using the OpenDHT service as an externally managed P2P network. We provide design details such as encryption and signing using pseudo-code and examples to provide P2P-SIP for various deployment components such as P2P client, proxy and adaptor, based on our implementation. The design can be used with other distributed hash tables (DHTs) also.

Keywords: P2P-SIP; OpenDHT; Security; Internet telephony

1 Introduction

Peer-to-peer Internet telephony using the Session Initiation Protocol (P2P-SIP) [1, 2, 3, 4, 5] has been proposed to avoid the maintenance and configuration cost of the server-based SIP architecture, and to prevent catastrophic failures of server-based systems. There are two approaches for combining SIP [6] and P2P: replace the SIP location service by a P2P protocol (*SIP-using-P2P*) [4], and additionally, implement the P2P protocol itself using SIP messaging (*P2P-over-SIP*). In the first case, P2P is used only for lookups and updates of SIP user's IP addresses, similar to LDAP or SQL databases used in existing SIP proxies. A scalable and global P2P location service automatically makes the SIP lookups scalable. In the second case, the P2P maintenance protocol can further exhibit two modes: (1) tunnel the P2P protocol messages in SIP, e.g., as a message body or headers, or (2) reuse the semantics of some of the SIP messages and headers to convey proximity and location information [2].

The P2P deployment architecture can be another dimension to classify P2P-SIP systems. Consider a simple server-based SIP call as shown in Fig. 1 (a). Bob's user agent sends a REGISTER request mapping his SIP identifier, sip:bob@example.net, and current host name or IP address, to the SIP server of domain example.net. When Alice wants to talk to Bob, her user agent sends the SIP INVITE request to her outbound proxy server. The outbound proxy finds the IP address of the SIP server of example.net via DNS, and sends it a SIP INVITE request. The server proxies or redirects the call to Bob's current location. Once the call is established, further communication can happen without going through the SIP server. Either the user agent or the proxy server can form the P2P network for lookup as shown in Fig. 1 (b) and (c), respectively. **P2P clients** are SIP user agents that do not require any server and directly perform P2P lookups and updates. **P2P proxies** are SIP proxy servers that perform P2P lookups and updates, transparent to the user agent, e.g., in a zero-configuration server farm of a VoIP provider. The tradeoffs are ease of deployment and integration with existing SIP clients or proxies, and reusability of other protocols and applications. These architectures and components should interoperate among each other.

We focus on SIP-using-P2P architecture in this paper. Since the user agents and proxies use an external P2P network, we need to define the precise data format for such operations for interoperability, i.e., contacts updated by one user agent are readable by another. For storing user contact locations, a distributed hash table (DHT) is enough instead of a full P2P database with various SQL-style search commands. In this paper, we provide an example data format for such a DHT-based SIP location service, and guidelines for implementing a SIP-using-P2P architecture with a managed external DHT based on our implementation experience. We describe various DHT keys, signing and encryption of data for P2P-SIP using pseudo-code and examples. We also describe the P2P presence and offline messaging. We do not propose any new algorithm but just apply existing algorithms to P2P-SIP clients and proxies. The assumption is that the DHT nodes are not malicious and correctly perform DHT operations. One example is OpenDHT [7, 8] run on PlanetLab.

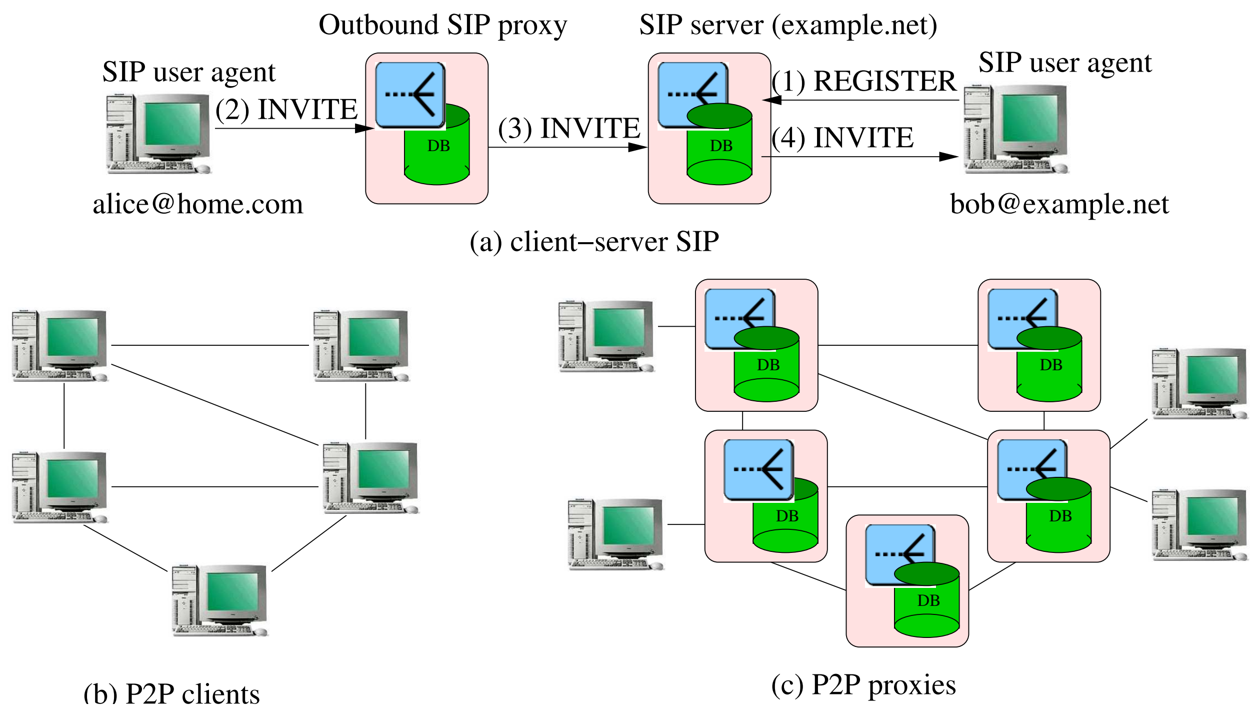

Figure 1: Deployment architectures

We give a background on DHT API in Section 2. Then, we describe the logical operations such as contact management and key storage in Section 3. Section 4 gives the motivation for the service model instead of the data model. We explain the P2P-SIP deployment scenarios such as client and proxy with pseudocode in Section 5. Section 6 presents some implementation issues. Security consideration, advanced services and evaluation are presented in Sections 7, 8 and 9 respectively. Finally, we present our conclusions in Section 10 and the proposed XML-based data format in Appendix A.

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

Table 1: Notations used in this paper

2 Background: DHT API

The current interface of OpenDHT is described in [7], and summarized here. The $\mathsf{put}(k, v, H(s), t)$ interface is used to store a value, v, associated with a key, k. The value expires after time-to-live (ttl), t, and can be removed before that using the secret, s. The value for the key, k, can be retrieved using $\mathsf{get}(k)$. It returns a list of tuples, $(v, H(s), t)$, where t is the remaining ttl. The value for the key, k, can be removed using $\mathsf{remove}(k, H(v), s, t)$, where $t >$ remaining ttl.

We use the existing interface as the basis to build P2P-SIP services. The interface allows putting multiple values under the same key, i.e., both (k_1, v_1) and (k_1, v_2) can be stored. For example, if Bob has many SIP phones, each phone can store its own contact IP address under Bob's key, and Alice's phone can retrieve all these contacts when making a call. The interface also allows putting the same value under the same key using different secrets. For example, both $(k_1, v_1, H(s_1))$ and $(k_1, v_1, H(s_2))$ can be stored. The secret controls who can remove the value associated with that key. Finally, a put with same key, value and secret, just updates the time-to-live (ttl). The ttl can be mapped to the Expires header in SIP REGISTER request for expiry of contact bindings. If a DHT does not provide an explicit remove, or no secret is specified in put, then the data is removed only on expiry.

Authenticated DHT interface [7] is required for protection against malicious users of the DHT and to filter get results at the DHT node.

3 Logical Operations

The P2P-SIP design consists of many logical operations such as key storage, contact management, NAT and firewall traversal, presence and offline message storage.

3.1 Contact management

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

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

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

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

3.2 Key storage

To avoid any central server, the certificates, keys, and any configuration are also stored on the DHT. For example, Bob can store his certificate and public key on the DHT with k_1=H(certificate:bob@example.net) and k_2=H(public:bob@example.net), respectively. Multiple certificates of Bob from different CAs can be put under the same DHT key. Since the information needs to be available to any potential caller, the value is unencrypted. There is a danger of other malicious users polluting the DHT values for this key. However, chained verification of the certificates can be used to retrieve the correct certificate.

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

3.3 Presence

The presence data is handled differently because, unlike the contact information, which needs to be available to all the potential callers, the watcher list should be visible only to the presentity (the entity being watched). For example, if Alice wants to subscribe for the presence status of Bob, she puts her signed identity in Bob's watcher list with k=H(subscribe:presence:bob@example.net). The value is encrypted using Bob's public key so that only Bob can decrypt watcher identity. This mechanism also works for events other than presence.

Additionally, Alice can store her encrypted friends list on the DHT similar to the private key storage described earlier.

3.4 Offline messages

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

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

3.5 NAT and firewall traversal

Inbound SIP messages to a client behind a NAT (Network Address Translator) requires connection reuse [9] and symmetric response routing [10]. Additionally, SIP phones use mechanisms such as STUN (Simple Traversal of UDP through NAT [11]), TURN (Traversal Using Relay NAT [12]) and ICE (Interactive Connectivity Establishment [13]), to traverse media through NATs and firewalls. This requires publically available STUN and TURN servers. A P2P-SIP node implements both STUN and TURN, and provides these services to other users.

The existing DHT interface of OpenDHT [7] is not sufficient for such service discovery. Consider the trivial approach of storing the STUN server's IP address with k=H(stun). This is not scalable because the DHT node storing this key will soon become overloaded with potentially millions of clients advertizing as STUN servers. There are two alternatives: DHT's service interface and hierarchical location-based key. OpenDHT provides additional API (ReDiR [7]) to join and lookup for a service. Thus, a P2P-SIP node joins OpenDHT for "stun" and "turn" services. Alternatively, if a node detects its location as "New York" and autonomous system (AS) number of his service provider as 1234, it can store its IP address with k_1=H(stun:geo:us.ny.newyork) and k_2=H(stun:as:1234). The use of AS number is useful because the users in the same AS are likely to have good connectivity.

4 Data and Service Models

There are two approaches to do the P2P-SIP operations of the previous section (Fig 2): any user directly updates the DHT (called as *data model*) or forwards the request to the service node responsible for that user key (*service model*).

4.1 Data model

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

This has several limitations to this approach. For example, presence composition [14] or programmable call routing [15] are *not* easy to implement. Moreover, we need service discovery for STUN and TURN servers anyway. An alternative service model solves this problem as described below.

4.2 Service model

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

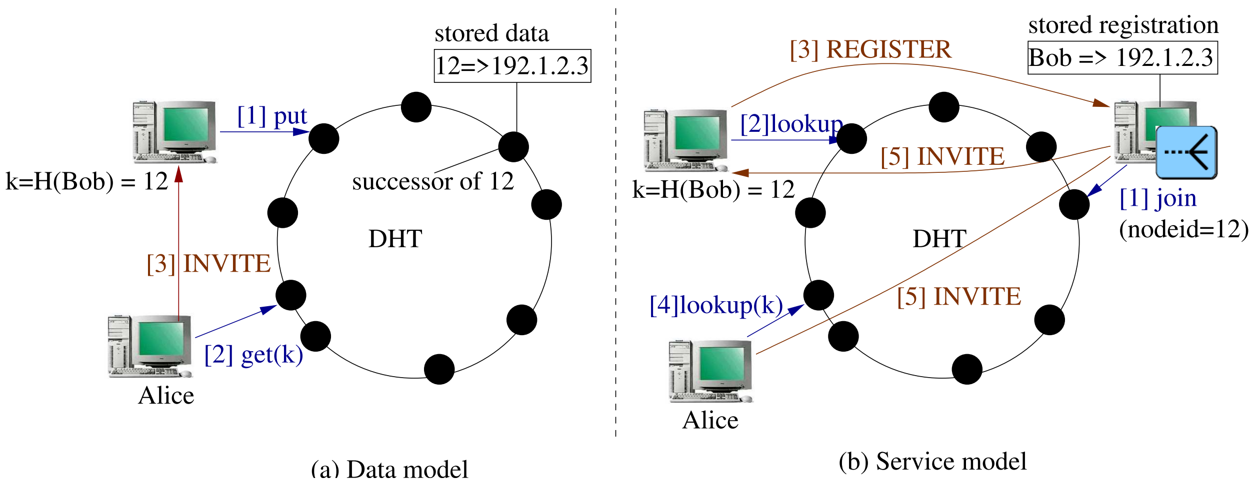

Figure 2: Data model vs service model

When a user, Alice, wants to send a SIP message to, say sip:bob@example.net, she looks up the DHT to find the service node responsible for this user identifier, and sends SIP request to that node. The service node acts as the proxy, registrar and presence server for all the users for which it is responsible. The service node also does any safe programmable call handling scripts [15] and presence composition [14].

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

The service model is more extensible than the data model. A P2P-SIP service node readily interworks with any non-P2P clients who just happen to know one or more service node addresses. The service mode readily extends to P2P-over-SIP architecture since only the service interface (join and lookup) is used in the DHT, instead of the data interface (get and put). Note however that the ReDiR interface of OpenDHT is in fact built on top of the data interface and resides purely on the client side. Thus, the service model is suitable for both P2P-over-SIP and SIP-using-P2P.

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

5 Deployment Scenarios

As mentioned earlier, a P2P-SIP node can run in different scenarios such as the P2P client, proxy or an adaptor for the existing SIP phones as shown in Fig. 3. In this section, we illustrate these scenarios using pseudo-code and examples.

5.1 P2P client

Consider a user Bob who picks his identifier, i=bob@example.net. For the first time use, he also picks a secret, s="mypass", and generates his RSA public and private keys, (K_P, K_S). The public key is put on the DHT under the key public:bob@example.net (procedure 5.1). Note that the DHT key is hashed using SHA-1, $H(key)$. Other users can get Bob's public key using his identifier. Bob, then encrypts his private key using mypass and puts it in DHT key private:bob@example.net:mypass.

Alternatively, instead of putting K_P, the user can store his X.509 certificate [16] issued by a trusted entity such as example.net in this case. The DHT key for the certificate is certificate:bob@example.net. If Bob knows that his issuer's identity may not be known to the prospective callers, he can also put his issuer's certificate on the DHT, say under the key certificate:example.net. Any caller should acquire the chain of certificates until she can trust the issuer.

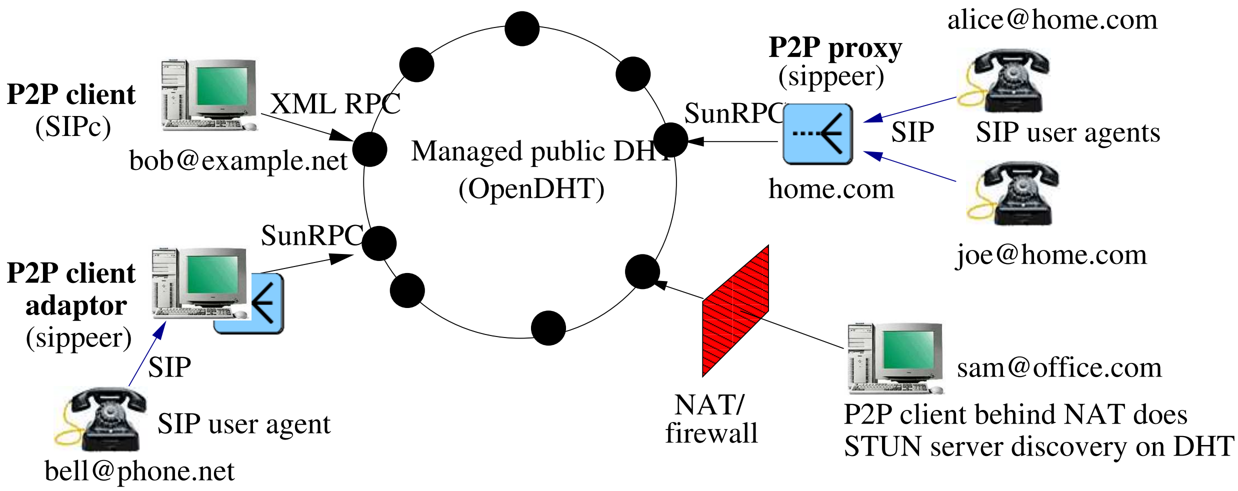

Figure 3: P2P-SIP: SIP-using-P2P architecture

global: id=n, keys=(K_P, K_S)
$n \leftarrow i$
if $k \leftarrow get(H($private:i:$s)$ failed **then**
  $(K_P, K_S) \leftarrow generate\ RSA\ keys$
  $put(H($public:$i), K_P)$ /* *no* $H(s) \Rightarrow$ *never remove* */
  $put(H($private:i:$s), [K_S]_s)$
  /* $[a]_b$ *means encrypt* a *using secret* b */
else
  $K_S \leftarrow [k]^s$ /* $[a]^b \Rightarrow$ *decrypt* a *using* b */
  $K_P \leftarrow get(H($public:$i)$ verified with K_S

Procedure 5.1: on-startup(identifier: i, password: s)

Now, when Bob wants to register his contact location, say sip:bob@192.1.2.3:5060, he creates an RSA digital signature of this contact. He then creates a value containing his contact, signer's name (which is his own identifier in this case), and the signature. This value is put on the DHT under the key sip:bob@example.net. One problem is that a malicious user can fetch the contacts and signature of Bob, and when Bob's registration expires, registers him again with the old signed contact. Alternatively, he can use this signed contact to register for some other user, thus messing up with other user's call routing.

To prevent this problem, one can use the authenticated interface of [7]. We use the similar signing procedure on top of the existing interface, until the DHT implements the authenticated interface. The signed data includes an absolute expiry time of the registration, the user's identifier and the signer's identifier in addition to the contacts. This will guarantee that the signature can not be used for another user or after it expires. The pseudo-code to add and remove a SIP contact is shown as procedures 5.2 and 5.3, respectively.

When the registration is refreshed, the planned authenticated interface [7] just updates the TTL of the existing contact record. However, with the existing DHT interface, a registration refresh creates a new record under the key instead of replacing, since the expiration and hence the value is changed. Unless the old record is expiring soon, it is

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

Procedure 5.2: put-contact(id: i, contact: c, ttl: t, password: s)

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

Procedure 5.3: remove-contact(id: i, contact: c, password: s)

recommended that the old record be explicitly removed to prevent storing dangling contact information in the DHT.

When Alice wants to call Bob, she looks up sip:bob@example.net in the DHT. If Alice knows Bob's public key, from earlier communication, she can use that to verify Bob's signature. Otherwise, she does another DHT lookup for the signer's certificate with DHT key, certificate:bob@example.net. If the certificate is found and issuer is trusted, the signature is verified. Otherwise, the issuer's certificate is looked up and the process repeats. If a certificate is not found, the corresponding public key is looked up. If the public key is found and there is only one record under that DHT key, then Alice may trust the identifier since nobody else has claimed that identifier. Any unverified contact is discarded (procedure 5.4). The existing DHT interface may return the same contact multiple times with different expiration, if the old contacts were not removed by the user on registration refresh. After removing such duplicate entries, Alice can call one or more contact location in sequence or parallel. After successfully talking to the right person, Bob, Alice remembers his public key, or at least H(public-key), for future communication. This is like the known_hosts file in OpenSSL [16].

$V[..] \leftarrow get(H(\text{sip:}i))$ */* get all contacts */*
$ret \leftarrow ()$
for all u in V **do**
  $(v, H(s), t) \leftarrow u$ **and** $(c, e, S, \sigma) \leftarrow v$
  if $e > now$ **and** S is i or domain of i **then**
    $K_P \leftarrow$ get-public-key(S) */* procedure 5.5 */*
    if $\sigma = \{H(i|c|e)\}_{K_P}$ **then**
      append v to ret
return ret

Procedure 5.4: get-contacts(id: i)

if $C \leftarrow$ get-certificate(i) **then**
  return public key from C
else if $(v, ..) \leftarrow get(H(\text{public:}i))$ gives 1 value **then**
  return v

Procedure 5.5: get-public-key(id: i)

Bob may store certificates from multiple issuers in the DHT, in the hope that the caller will recognize at least one of the issuer, and minimize the number of get operations on the DHT. This leads to a friend-to-friend trust model, where after successfully communicating with Bob, Alice may herself issue a certificate to Bob. The certificate indicates that Bob is the owner of the private key corresponding to the signed public key of the certificate. Thus, other users who know Alice, can verify Bob's certificate.

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

5.2 P2P proxy

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

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

Procedure 5.6: get-certificate(X.509 subject's common name: i)

(procedure 5.1), so that other proxies in the farm can retrieve them. When signing a user contact, the signer's identity is set to home.com.

The proxy also stores the appropriate authentication credentials for the users in the domain for authenticating SIP REGISTER requests. For example, it may store Alice's credentials in DHT key digest:alice@home.com:mypass as shown in procedure 5.7. Since most digest authentication **??** implementations use MD5, procedure 5.7 stores the MD5 hash of the user credentials, which is sufficient for digest authentication by the proxy.

global: domain=n, secret=s
$h \leftarrow$MD5(i:n:p) **and** $put(\text{digest:}i\text{:}s, [h]_s)$

Procedure 5.7: signup-user(identifier: i, password: p)

When Alice registers with the proxy, the proxy authenticates her using the stored credentials. If the authentication succeeds, it updates the contacts using procedure 5.2 like a P2P client (procedure 5.8). Similarly, procedure 5.3 is used to unregister. Since the contacts are signed over the absolute expiration time, a SIP REGISTER refresh causes one more contact to be added in DHT. The proxy should then removes the old contact with previous expiration value.

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

Procedure 5.8: on-register(SIP message: R)

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

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

When the proxy receives a SIP INVITE or other request, it looks up the existing contacts for the destination user and proxies or redirects the call. The lookup is same as that done by P2P clients (procedure 5.4).

5.3 P2P client adaptor

A SIP proxy may also be used as a **P2P client adaptor** for existing SIP phones that do not support P2P-SIP. In that case, the P2P proxy (adaptor) runs along with the SIP phone in P2P client scenario on the same host or within the same trusted network. The adaptor is logically part of the user's phone except that the functionality is split between the phone and adaptor. It should not be necessary to keep two different passwords, one for digest authentication by the phone to the adaptor and the other by the adaptor to sign the contacts in DHT.

To solve this, the adaptor uses basic authentication instead of digest and learns the user's password on the fly on REGISTER. The adaptor behaves like a P2P client instead of a P2P proxy, but implements SIP registrar and proxy. There are two common authentication modes: basic and digest. While digest authentication never transfers the password, basic send password as base64 encoded text. Although, basic authentication is not supported in SIP, basic over TLS is considered safe and in some case better than digest if the server stores the hashed user credential without encryption. In procedure 5.8, the secret s is obtained from the SIP REGISTER request's Authorization header, n is obtained from the From header, and keys (K_P, K_S) are obtained using procedure 5.1.

6 Implementation issues

We have implemented the OpenDHT-based SIP contact management and key storage for P2P client and adaptor modes in our P2P-SIP implementation, SIPPEER [17]. Additionally, we have implemented the SIP contact management, key storage, service advertisement and discovery of STUN servers for NAT/firewall traversal, presence, and offline instance message (IM) storage for the P2P client mode in Columbia SIP user agent, SIPc [18]. The module that connects to OpenDHT, is called connector, and can be replaced by other similar DHT connectors in future.

The connector connects to the DHT nodes and uses the get, put and remove interface to perform P2P-SIP operations described in this paper. In this section we describe some of the implementation highlights of SIPPEER [17] and SIPc. The SIPPEER implementation is done in C++, using Sun RPC as the OpenDHT interface [8], whereas SIPc is in Tool Command Language (Tcl), using XML-RPC as the OpenDHT interface. Both use OpenSSL [16] for cryptographic routines. SIPPEER runs on Linux, but can be easily ported to other Unix platforms and Win32, using our portability libraries. SIPc runs on both Unix and Win32 platforms.

6.1 Redundant connections

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

6.2 Data format

In SIPPEER, RSA keys are generated using 1024-bit modulus and exponent as 65537. All certificates and RSA keys are stored in ASCII PEM (privacy enhanced mail) format. Alternatively, use of ASN.1 binary format can save some space.

In SIPPEER, when a tuple or a list needs to be evaluated in a scalar context, e.g., in procedure 5.2 for the tuples being put or hashed, the elements in the tuple or list are concatenated together and delimited by nul character. To prevent the ambiguity if the actual data has nul character, data may be base64 encoded before concatenating.

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

6.3 Data size

One of the restrictions of OpenDHT is that the data size for every put is limited to 1024 bytes. The X.509 certificates sometimes exceed the limit. We wrote another interface layer to put larger data, by splitting it into chunks of maximum 1024 bytes. The original DHT key stores the index containing DHT keys to the individual chunks. Assuming, a 20-byte index (SHA1), a one level indirection can store index of other 50 blocks of 1 kB, thus giving a total of 50 kB of data under a key. This is more that sufficient for storing user keys, contacts or presence data in P2P-SIP.

We use some ugly hack as shown in procedure 6.1 and 6.2. If the first byte of the data is the nul character, then the data is assumed to be index of other chunks separated by nul characters.

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

Procedure 6.1: put-large(k,v,H(s),t)

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

Procedure 6.2: get-large(k)

6.4 Data expiration

OpenDHT has a maximum TTL of one week for any data. Although most user contacts have much lower TTL, semi-permanent data such as certificates and RSA keys are limited to a maximum of one week. To continue using the system the P2P clients and proxies should periodically refresh the certificates and keys on the DHT. Alternatively, there can be service nodes that walk the DHT and slowly refresh all the data.

6.5 Storing time

All expirations and absolute times should be stored in GMT format, because the data may be read by a user in a different timezone, e.g., contact information.

6.6 Fairness

OpenDHT allocates space quota fairly to different clients, identified by IP addresses. This means a single proxy handling a lot of users and storing lot of data, may fails if the quota exceeds. Thus, the current OpenDHT fairness policy favors the P2P client and adaptor modes.

6.7 Privacy

Another scenario for the SIP proxy is to use the DHT just as a replacement for back-end database. Lookup is still done via DNS and SIP [19]. In this case, the proxy encrypts the data stored on the public DHT, so that others can not directly use the data. Unlike a P2P proxy, in this mode the proxy works in the server-based architecture. Our SIPPEER, in this mode, encrypts all user contacts on the DHT using triple-DES EDE cipher. This mode does not require signing and verification of the user contacts, since the data is encrypted and not visible to others in the DHT.

6.8 Authenticated interface

Once the authenticated interface is implemented in OpenDHT, some of the procedures of P2P-SIP can be simplified. In particular, the two step put-remove process of register refresh (procedure 5.8), will be done using a single put. Also, a get request will return only the desired data if the public key of the creator is specified. Similarly, certificate and key verification can specify the public-key in get to avoid getting unnecessary data and becomes a single step process.

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

7 Security and Trust

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

Since anyone can pick any user identifier and store the contacts and keys for that identifier on the free public DHT such as OpenDHT, there is some risk of talking to the wrong person. On Unix systems, the known_hosts file contains an encoded ssh fingerprint for each host that this machine has contacted through ssh. Similarly, the P2P-SIP node can store the fingerprint of the user after initial communication. The fingerprint contains the user's identity and public key. The encrypted fingerprint can be put on the DHT for future verifications. If storing the public keys of all the contacted users is not space efficient, SHA-1 is used (procedure 7.1). When making a call, the user gets the public key from the DHT and verifies it with the hash stored in his mapping (procedure 7.2). The fingerprints can be used as a "friends" list similar to those maintained in popular IM clients such as Yahoo and MSN.

global: private key=K_S of signer
$put(H(i), \{H(i|P)\}_{K_S})$

Procedure 7.1: sign(identifier: i, public-key: P)

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

Procedure 7.2: verify(identifier: i, public-key: P)

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

Alternatively, the DHT may provide a service model in which every user first signs up with the DHT providing the mapping between the identifier and his public key. The DHT guarantees that there will be only one user with the given identifier at any time, and can verify his public key when requested. This can be implemented using the existing OpenDHT interface as shown in procedure 7.1 and 7.2, but requires a signer to sign every new user identifier. We assume for scalability that the new identifiers are not created very often. Also, the signer verifies the identifier of the form user@domain to some extent, e.g., by requesting confirmation from that email address as mentioned earlier.

One important difference between our approach and Skype [20] is in the use of central servers. Skype uses centralized login server(s) to authenticate the user every time the client is started. On the other hand, centralized certifying authority (CA) in our architecture are contacted only for issuing the initial user or domain certificate. Subsequent user logins just use the DHT without contacting the CA. Thus, this is more scalable than the central login server architecture.

8 Presence and offline messages

In addition to the user contact locations and keys, configuration such as "friends" list and media such as voicemails may be stored on the DHT. Any configuration needs to be accessed only by the owner, hance can be encrypted. On the other hand, subscription requests and offline messages are stored and retrieved by two different users, but not accessible by any other users. Thus, the P2P client or proxy encrypts the signed subscription request or offline message using the recipient's public key so that only the recipient may read the request or message. The details for offline messages are shown in procedure 8.1 and 8.2. It allows the caller to store a message and the recipient to read and delete the message. The message, M, is in email format and may have voice attachments. One must carefully store large values in the DHT, since the data size may exceed 50 kB now.

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

Procedure 8.1: put-offline(caller:(a, A_S), callee:(b, B_P), M)

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

Procedure 8.2: get-offline(user:(b, B_S)

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

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

9 Evaluation

9.1 Comparison of deployment architectures

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

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

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

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

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

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

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

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

9.2 Performance evaluation

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

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

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

9.3 Reliability

OpenDHT does data replication for reliability. This means the P2P-SIP node itself does not have to do any replication. The redundant connection (Section 6) takes care of fail-over to the next DHT node if the closest DHT node dies.

The service discovery module for locating STUN servers also fails over to the next serving node if the first looked up server does not respond.

10 Conclusions

We have presented an example design of P2P-SIP using OpenDHT as an externally managed peer-to-peer network. We explained various P2P deployment components such as clients, proxies and adaptors using pseudo-code and examples. We also presented some of the design issues based on our implementation, SIPPEER. The architecture can be used for other DHTs with similar interfaces.

Based on our analysis, we recommend using P2P clients instead of the P2P proxies or adaptors as much as possible, and the planned authenticated interfaces [7] when implemented in OpenDHT. This reduces the number of lookup and updates in the P2P network and, hence, is more scalable.

The design and data format presented in this paper can be used by other P2P-SIP implementations to build an interoperable network of P2P-SIP nodes for contact management, key storage, NAT and firewall traversal, presence and offline message storage.

This is a continuation of our work on peer-to-peer Internet telephony using SIP [1, 2, 17].

Acknowledgment

Xiaotao Wu helped in integration with Columbia SIP user agent, SIPc. The work is supported by grant from SIPquest Inc., and the National Science Foundation under Grant No. 0202063.

A Proposed Data Format

In this section we propose a XML-based data format for storing SIP related information on the DHT for interoperability among different P2P-SIP implementations. The data format applies to both existing and planned authenticated DHT interfaces [7].

An example user contact of user bob@example.net stored in the DHT at key H(sip:bob@example.net) is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts xmlns="urn:ietf:params:xml:ns:p2p-sip">
 <contact>sip:bob@192.1.2.3:5060</contact>
</contacts>
```

For existing DHT interface, we need the expires and user attributes as part of the contact information, so that the signature can not be misused as described earlier. These are not needed for the authenticated DHT interface, since they can be securely derived using other means such as ttl returned by get interface and DHT key, respectively. An example signed contact is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts xmlns="urn:ietf:params:xml:ns:p2p-sip" Id="One" user="sip:bob@example.net">
 <contact display-name="Bob Wilson" expires="2006-01-31T18:22:38Z">
  sip:bob@192.1.2.3:5060
 </contact>
</contacts>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
  <Reference URI="#One">
   <Transforms>
    <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
  </Reference>
 </SignedInfo>
 <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
 <KeyInfo><KeyName>bob@example.net</KeyName></KeyInfo>
</Signature>
```

Any signature is formatted using W3C"s Signature element [21]. The URI in Reference tag points to the data this signature is for. The KeyName refers to the user identifier of the signer.

The user"s public key or certificate is stored using the KeyInfo element [21] in the DHT at key H(public:bob@example.net) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
 <X509Data>
  <X509SubjectName>CN=bob@example.net,O=P2P Inc.,ST=New York,C=US</X509SubjectName>
  <X509Certificate>MIID5jCCA0+gA...lVN</X509Certificate>
 </X509Data>
</KeyInfo>
```

A user Bob can subscribe for presence status of alice@home.com, by storing the following information in the DHT at key H(subscribe:alice@home.com).

```
<?xml version="1.0" encoding="UTF-8"?>
<watchers xmlns="urn:ietf:params:xml:ns:p2p-sip">
 <watcher event="presence" entity="alice@home.com" expires="2006-01-31T18:22:38Z">
   sip:bob@example.net
 </watcher>
 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  ...
 </Signature>
</watchers>
```

Since this information needs to be encrypted, it gets stored as follows, using the W3C"s EncryptedData element [22]:

```
<?xml version="1.0" encoding="UTF-8"?>
<EncryptedData Type="urn:ietf:params:xml:ns:p2p-sip#watchers
 xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
 <ds:KeyInfo xmlns:ds="http://www.w3.org/2001/04/xmldsig#">
  <EncryptedKey CarriedKeyName="TempKey" xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa1_5"/>
   <CipherData><CipherValue>xyza21212sdfdsfs7989fsdbc</CipherValue></CipherData>
  </EncryptedKey>
 <ds:KeyInfo>
 <CipherData><CipherValue>A23B45C564587</CipherValue></CipherData>
</EncryptedData>
```

An offline message is also stored as an EncryptedData element. The Type attribute refers to text or audio format for offline text or voice message, respectively.

The complete schema definition for urn:ietf:params:xml:ns:p2p-sip is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:p2p-sip"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:p="urn:ietf:params:xml:ns:p2p-sip"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

 <import namespace="http://www.w3.org/XML/1998/namespace"
   schemaLocation="http://www.w3.org/2001/xml.xsd"/>

 <element name="contacts" type="p:contactsType"/>

 <complexType name="contactsType">
  <sequence>
   <element name="contact" type="p:contactType" maxOccurs="unbounded"/>
  </sequence>
```

```
  <attribute name="Id" type="ID" use="optional"/>
 </complexType>

 <complexType name="contactType">
  <simpleContent>
   <extension base="anyURI">
    <attribute name="Id" type="ID" use="optional"/>
    <attribute name="user" type="anyURI" use="optional"/>
    <attribute name="display-name" type="string" use="optional"/>
    <attribute name="expires" type="dateTime" use="optional"/>
    <attribute name="priority" type="p:priority" use="optional"/>
   </extension>
  </simpleContent>
 </complexType>

 <simpleType name="priority">
  <restriction base="decimal">
   <pattern value="0(.[0-9]{0,3})?"/>
   <pattern value="1(.0{0,3})?"/>
  </restriction>
 </simpleType>

 <element name="watchers" type="p:watchersType"/>

 <complexType name="watchersType">
  <sequence>
   <element name="watcher" type="p:watcherType" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
 </complexType>

 <complexType name="watcherType">
  <simpleContent>
   <extension base="anyURI">
    <attribute name="Id" type="ID" use="optional"/>
    <attribute name="entity" type="anyURI" use="optional"/>
    <attribute name="expires" type="dateTime" use="optional"/>
   </extension>
  </simpleContent>
 </complexType>

</schema>
```

References

[1] Kundan Singh and Henning Schulzrinne, "Peer-to-peer Internet telephony using SIP," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Skamania, Washington, June 2005.

[2] Kundan Singh and Henning Schulzrinne, "Peer-to-peer Internet telephony using SIP," Tech. Rep. CUCS-044-04, Department of Computer Science, Columbia University, New York, NY, Oct. 2004.

[3] Salman Baset, Henning Schulzrinne, Eunsoo Shim, and Krishna Dhara, "Requirements for SIP-based Peer-to-Peer Internet Telephony," Internet Draft draft-baset-sipping-p2preq-00, Internet Engineering Task Force, Oct 2005, work in progress.

[4] Alan Johnston, "SIP, P2P, and Internet Communications," Internet Draft draft-johnston-sipping-p2p-ipcom-01, Internet Engineering Task Force, Mar 2005, work in progress.

[5] David Bryan and Cullen Jennings, "A P2P Approach to SIP Registration," Internet Draft draft-bryan-sipping-p2p-01, Internet Engineering Task Force, Jul 2005, work in progress.

[6] J. Rosenberg, Henning Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[7] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu, "Opendht: a public dht service and its uses," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 73–84, 2005.

[8] "OpenDHT: a public distributed hash table service," http://www.opendht.org.

[9] Rohan Mahy, "Connection Reuse in the Session Initiation Protocol (SIP)," Internet Draft draft-ietf-sip-connect-reuse-04, Internet Engineering Task Force, Jul 2005, work in progress.

[10] Jonathan Rosenberg and Henning Schulzrinne, "An extension to the session initiation protocol (SIP) for symmetric response routing," RFC 3581, Internet Engineering Task Force, Aug. 2003.

[11] Jonathan Rosenberg, J. Weinberger, Christian Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (nats)," RFC 3489, Internet Engineering Task Force, Mar. 2003.

[12] Jonathan Rosenberg, Rohan Mahy, and Christian Huitema, "Traversal Using Relay NAT (TURN)," Internet Draft draft-rosenberg-midcom-turn-08, Internet Engineering Task Force, Sep 2005, work in progress.

[13] Jonathan Rosenberg, "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Draft draft-ietf-mmusic-ice-06, Internet Engineering Task Force, Oct 2005, work in progress.

[14] Henning Schulzrinne, "Composing Presence Information," Internet Draft draft-schulzrinne-simple-composition-00, Internet Engineering Task Force, Jul 2005, work in progress.

[15] Jonathan Lennox, Xiaotao Wu, and Henning Schulzrinne, "Call processing language (CPL): a language for user control of Internet telephony services," RFC 3880, Internet Engineering Task Force, Oct. 2004.

[16] "The OpenSSL project," http://www.openssl.org.

[17] Kundan Singh and Henning Schulzrinne, "SIPpeer: a session initiation protocol (SIP)-based peer-to-peer Internet telephony client adaptor," White paper, Computer Science Department, Columbia University, New York, NY, Jan 2005, http://www.cs.columbia.edu/IRT/p2p-sip/papers/sip-p2p-design.pdf.

[18] Xiaotao Wu and Henning Schulzrinne, "sipc, a multi-function SIP user agent," in *7th IFIP/IEEE International Conference, Management of Multimedia Networks and Services (MMNS)*. IFIP/IEEE, Oct. 2004, pp. 269–281, Springer.

[19] J. Rosenberg and Henning Schulzrinne, "Session initiation protocol (SIP): locating SIP servers," RFC 3263, Internet Engineering Task Force, June 2002.

[20] "Skype: Free internet telephony that just works," http://www.skype.com.

[21] Donald Eastlake, Joseph Reagle, and David Solo, "XML-Signature Syntax and Processing," W3c recommendation, World Wide Web Consortium, Feb 2002, http://www.w3.org/TR/xmldsig-core.

[22] Donald Eastlake and Joseph Reagle, "XML Encryption Syntax and Processing," W3c recommendation, World Wide Web Consortium, Dec 2002, http://www.w3.org/TR/xmlenc-core.