Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 19, 2008

E. Marocco
Telecom Italia
E. Ivov
L. Pasteur University/SIP Communicator

November 16, 2007

XPP Extensions for Implementing a Passive P2PSIP Overlay Network based on the CAN Distributed Hash Table
draft-marocco-p2psip-xpp-pcan-01

Status of this Memo

Copyright Notice

Abstract

This document defines a set of extensions for the Extensible Peer Protocol (XPP) required for creating a P2PSIP overlay network based on the CAN distributed hash table algorithm. It specifies how peers and clients must behave in order to maintain the overlay and use it for the establishment of multimedia communication sessions.

To limit the overhead due to maintenance operations and to allow the adoption of security policies for preventing malicious nodes to damage the overlay, joins are always initiated and controlled by existing peers (hence the passive in PCAN).

Table of Contents

1. Introduction

This document describes a possible solution for a P2PSIP Overlay Network [I-D.willis-p2psip-concepts] currently implemented in the SIPDHT [WWW.sipdht] open source project. The passive approach that the solution adopts is relatively uncommon in related work. Yet it seems to be well-suited for addressing issues arising from the wide spread deployment of NAT devices, high churn rate and possible participation of malicious peers. This solution is intended to:

- easily support deployments where many peers are located behind NAT boxes, adopting NAT traversal mechanisms such as STUN [I-D.ietf-behave-rfc3489bis] and ICE [I-D.ietf-mmusic-ice];
- provide mechanisms for keeping overlay size to an optimal minimum. Allowing peers located behind NAT devices to become members of the the overlay network implies frequent exchange of keep alive messages. XPP-PCAN addresses this issue by only allowing the best available clients (i.e. those offering most significant resources) and doing so only when their participation is really necessary.
- provide mechanisms for limiting the effects of malicious nodes attempting to degrade the service.

XPP-PCAN is based on the following key points:

1. the overlay is organized as a distributed database or hash table based on the CAN [WWW.icir.can] algorithm and it only offers functionalities for storing and retrieving data and for routing SIP [RFC3261] messages;
2. the P2P overlay is transparent to clients. A client can maintain SIP outbound flows [I-D.ietf-sip-outbound] and register its location with any peer (causing the insertion of a routing record on the peer authoritative for its address);

   It would also be possible for clients, not participating in the overlay, to allow others to maintain outbound flows with them, as this would significantly lighten the load of the overlay. At the moment such an option has been put aside for simplicity, but is listed as an open issue.

3. peers use XPP [I-D.marocco-p2psip-xpp] for performing maintenance operations;
4. peers use SIP (and possibly ICE) for establishing XPP sessions;

5. for every client using the overlay there is at least one registrar peer that the client uses as point of attachment to the overlay, and one authoritative peer (the one keeping location and routing information for the client (in some cases the two may coincide).

6. a peer can invite any client it is authoritative for to join the overlay.

Once again, one of the main characteristics of XPP-PCAN is the fact that it is not possible for a node to decide whether it would simply use the overlay network as a client or become a part of it as a peer. One could therefore imagine the network as a free service which, in some cases, could invite any of its clients to provide resources for use by others. Answering positively to an invitation is the only way for a client to become a peer.

It is worth noting that points 1 and 4 allow peers and clients to use the SIP routing functionality provided by the overlay for establishing XPP sessions, which, in turn, are needed for maintaining the overlay itself.

The remainder of this document, after giving a short overview of the CAN-based algorithm, specifies XPP extensions and defines operations each peer must perform in order to consistently maintain the overlay. However, it does not specify how a peer decides to invite a client to join the overlay; implementations must apply their own criteria for deciding both when and which node to invite.

1.1. The Passive Approach

Networks using XPP-PCAN would manage joins in a "passive" way, and only allow nodes to become members of the overlay once they have been invited by existing peers. Overlays, used by file-sharing applications, would generally adopt the opposite approach and let clients decide whether or not to become peers. A well known exception to this trend is the Skype network [WWW.columbia.skype], arguably one of the most popular overlay networks used for real-time communications today. Instances of the Skype application may operate as super-nodes or ordinary-nodes and the "promotions" are decided by the higher levels of the hierarchy.

The passive approach seems appropriate for P2PSIP because:

- all peer candidates are actually clients that have already registered with the service and are therefore known to the overlay.

- the overlay only needs to provide a limited amount of functionalities to clients and these could be handled by a relatively small subset of the nodes that actually use them.

- providing SIP connectivity to clients, storing their location, routing messages,in addition to maintaining tens or even hundreds of connections with neighbor peers could be very resource consuming. It is therefore important to confirm availability of such resources prior to inviting a client to join the overlay as a peer.

An additional advantage of passive joins is the fact that they allow the overlay to select peers among candidates based virtually any kind of performance and security characteristics.

It is possible, for example, to limit the effects of malicious peers by only inviting trusted nodes to join the overlay. Assessing the level of trust could be handled in many different ways like provider maintained white-lists or social networks.

1.2. Why CAN?

The choice of the distributed hash table algorithm has been heavily influenced by our goal to have robust overlay networks even in cases when many peers are behind a NAT. This requires maintaining persistent connections between peers and CAN fits this requirement quite well, because:

1. it is symmetric [I-D.baset-sipping-p2pcommon] (unlike Chord [WWW.mit.chord]). Without such a property, each connection is efficiently used by only one of the endpoints;

2. peers maintain a stable routing table with a limited number of entries (which is not the case with Pastry [WWW.microsoft.pastry] and Kademlia [WWW.rice.kademlia]).

A possible drawback when using the CAN algorithm is its relatively low performance. While other popular algorithms claim to always be logarithmic in complexity, overlays based on CAN cannot scale indefinitely. CAN based overlays need to be configured during deployment with parameters (e.g the number of dimensions for the hash space) depending on the size of the intended network.

However, in the case of P2PSIP, the service could be provided by a subset of interested nodes. This and the possibility to use delay measurements between peers when selecting best routes within the overlay, could help achieving acceptable performance even with a limited number of peers.

1.3. 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 RFC 2119 [RFC2119].

2. Algorithm Overview

The algorithm implemented by the overlay is a version of CAN [WWW.icir.can] slightly customized to fit the "passive" approach. This section briefly describes the overall design and the procedures for routing, joining the overlay and recovery after failures. Section 3 and Section 4 will then define how clients and peers establish and use XPP sessions for implementing these procedures.

2.1. General Design

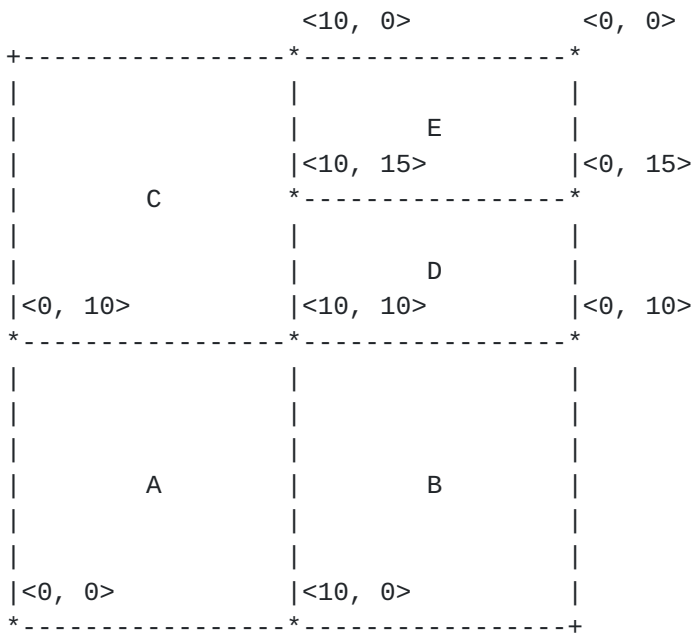
The functionality on which the overlay is based, like all other distributed hash table algorithms, is the mapping of keys over the peers. Such a functionality, unlike in algorithms based on the concept of consistent hashing [WWW.mit.chord] [WWW.microsoft.pastry] [WWW.rice.kademlia], is implemented in a virtual d-dimensional Cartesian coordinate space on a d-torus.

> The d-torus is the generalization of the Chord ring in d dimensions. In fact, while in Chord keys are uni-dimensional and can be represented on a ring (i.e. a circular line), in CAN -- and so in the algorithm described here -- they are at least bi-dimensional and thus require a torus (i.e. a circular plane).
>
> A d-dimensional key can easily be obtained by splitting in d parts a uni-dimensional value returned by a common hash function.

Any peer is assigned a distinct zone of the overall space and is authoritative for all the keys falling in it. Zones are always defined by the coordinates of their lower left and top right corner, and contain the area locked between the borders including the bottom and left edges (thus excluding the right and top edges). At any point in time, the overlay is consistent if the whole space is completely covered. Figure 1 shows a bi-dimensional space partitioned among 5 peers.

A peer maintains XPP sessions with all its immediate neighbors, along with information about the zones they are authoritative for. When a peer receives an operation request for a key which falls out of its zone, it routes the request to the most appropriate neighbor.

> The most appropriate neighbor is generally the one whose center of mass is closer to the requested key; however, an implementation may also take into account link speed and so select not just according to geometric distance. In any case, in order to avoid loops, peers must not route messages to neighbors which are not geometrically closer to the targeted key.



A bi-dimensional space with 5 peers. For simplicity, the space is shown in two dimensions. In reality the figure is actually a torus.

Figure 1

2.2. Peer Join

The decision when to invite a client to join the overlay is always taken by existing peers. After choosing the best candidate to "promote", the inviting peer would either select a fragment recovered from a dead neighbor (see Section 2.3) or obtain a new one by splitting its zone through the longest edge into two equal parts. It would then assign the new fragment to the entering peer.

The selection process of the candidate to invite may be based on the evaluation of parameters like bandwidth, connectivity, uptime and trust; such a process strongly depends on the deployment and is outside the scope of this document.

It is worth noting that, in the case of P2PSIP, a peer may choose the most appropriate node to invite among the clients it stores a registration for, that is the client whose address fall under its authority.

After identifying the zone and the node to invite, the inviting peer sends a SIP INVITE request to the new peer and establishes an XPP session. Once the XPP session established, the join is completed in the following six steps:

1. the inviting peer transfers to the new peer all data bound to keys located in the new zone;

2. the inviting peer notifies all the neighbors of the zone being transferred that a peer is about to join, and that they should be ready to establish XPP sessions with their new neighbor;

3. if the zone of the inviting peer has been split, it notifies all its current neighbors of its new coordinates;

4. the inviting peer sends to the invitee the list of all neighbors it will have after joining the overlay;

5. the entering peer establishes XPP sessions with all its new neighbors using SIP;

6. the inviting peer updates its neighbor list and ends all XPP sessions with peers which are no longer its neighbors.

2.3. Failure Recovery

When a peer fails, one or more zones of the overlay need to be recovered through a distributed election. Elections are run by peers neighboring orphaned zones. The exact algorithm is describe in Section 4.2.2.

When a peer loses connectivity with one of is neighbors, it starts a timer waiting for other neighbors to also detect the failure. The exact value of the timer depends on XPP timers and must therefore be greater than the maximum allowed interval between two subsequent keep alive responses plus the time necessary for neighbors to complete retransmissions of their last keep alive message. When the timer fires, the peers would start the election algorithm.

At this point all neighbors of the failed peer start exchanging messages for electing the one to take over the orphaned zone. Every one of them would store the identity of the most appropriate candidate it sees and, after a stabilization interval Section 4.2.1, consider it authoritative on that zone.

When a peer is elected for taking over a zone, it is likely that it does not know all of its new neighbors; if this is the case, it must query the neighbors it knows in order to discover new ones and establish XPP connections with them.

2.4. Stabilization

Race conditions, temporary transport failures and misbehaving peers may cause inconsitencies in the topology of the network. Such inconsistencies most often result in neighbor peers have different representations of the geometry of a certain number of bordering zones.

A simple yet efficient stabilization procedure to realign the internal state of a peer with that of its neighbors is accomplished querying all peers in the neighborhood and checking that their geometrical view is consistent with that of the querying peer. Any time an incosistency is detected (i.e. the querying peer discovers a zone whose owner differs from the one it had recorded or whose coordinates overlap with some zones it is aware of), it is resolved on a popularity basis, aligning the internal state with the information reported by the highest number of peers.

2.5. Data Replication

Persistency of data against peer failures is achieved by making each peer replicate its local hash table on an arbitrary number of neighbors. Upon a peer failure, right after the recovery procedure described in Section 2.3 has completed, all peers which where neighbor of the failed peer check in their replica tables if they have records which are under the authority of the peer elected as the new owner of the recovered zone; if they find any, they immediately send records back to the new peer.

As long as peers consistently choose their replica holders (i.e. they send records bound to the same keys always to the same peers whenever this is possible), such a mechanism can recover the data stored by a failed peer unless all its neighbors storing a replica of that data fail at the same time.

3. Client Behavior

The overlay is intended to be transparent to conventional SIP user agents. Once such a client has discovered the location of one or more peers (how exactly it has done so is outside the scope of this document), it MAY set any of them as its outbound proxy. It SHOULD then register using the peer as a registrar and following the registration procedures described in [RFC3261] and. If needed, the SIP client MAY direct SIP outbound flows [I-D.ietf-sip-outbound] to this peer in order to allow NAT traversal for SIP messages.

If the client wants to be able to join the overlay, it MUST use a SIP contact which routes to itself from each point in the overlay (e.g. a global scope IP address); if it does not have such a contact, it MAY request a public GRUU [I-D.ietf-sip-gruu] from all peers it is sending SIP REGISTER requests to.

When additional resources are necessary for the maintenance of the overlay, a client MAY receive a SIP INVITE request asking to join the overlay, as described in Section 4.6. Since such a request MUST list the 'pcan' tag in the 'Require' header field, clients not implementing this specification, will answer with a 420 (Bad Extension) error message, as specified in [RFC3261].

4. Peer Behavior

4.1. Key-Point Mapping

At any point in time, every peer is authoritative for one or more geometric zones, which means that it is responsible for storing all data bound to keys that correspond to points in these zones.

In order to obtain the coordinates of a d-dimensional point, one has to split into 'd' equal components the 20 byte-long hash string returned by applying the SHA-1 [RFC3174] function on the key stripped of any URI parameters (i.e. all characters up to the first occurrence of a semi-colon - ';').

The number of components MUST be equal to the number of dimensions indicated in the initial JOIN request; allowed values are 2, 4, 5 and 10.

4.2. Internal State

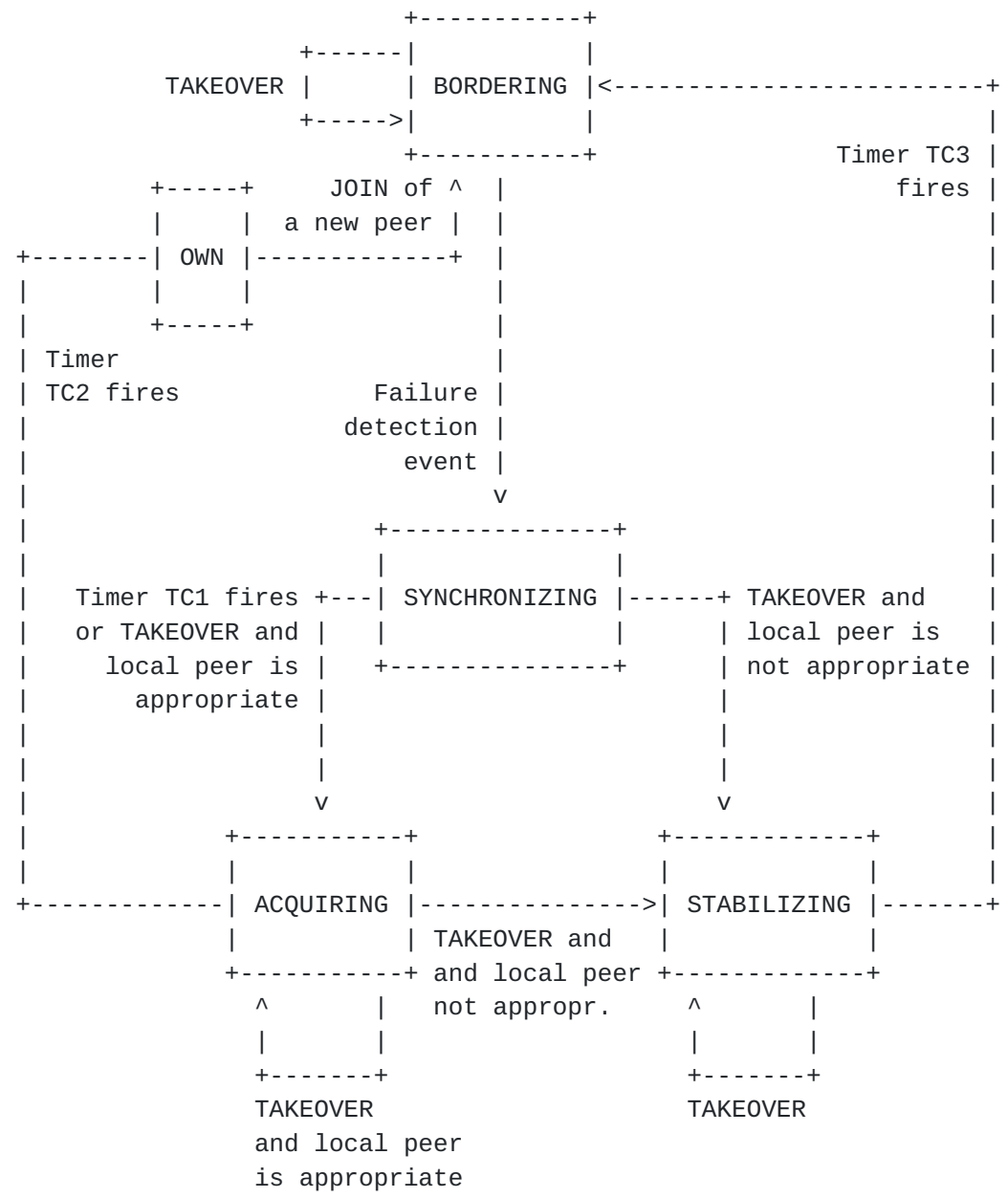
The state of a peer participating in an overlay can be represented as a list of zones the local peer is authoritative for or which border on zones the local peer is authoritative for. For each zone, a peer MUST store the SIP address-of-record and contact of the responsible peer, the coordinates of the geometric area and a state variable which can be set according to the state diagram in Figure 2. When keeping track of the states of all neighbor zones (as defined in Section 4.2.1), a peer MAY assign each one of them one of three timers -- timer TC1, timer TC2 or timer TC3.

Timers are named TC1, TC2 and TC3 for differentiating from XPP timers T1, T2 and T3.

At any point in time a peer SHOULD have active XPP sessions with all its neighbors. It MUST handle failures occurring in such sessions as explained in Section 4.2.1.

4.2.1. Zone States

Every peer maintains a list of all zones bordering on its own. Every such zone may be in one of the following states.



Zone state diagram.

Figure 2

The states have the following meaning:

- BORDERING: a neighbor peer is authoritative for the zone and the XPP session with this peer is active.

- SYNCHRONIZING: the neighbor authoritative for the zone is unreachable. The peer waits until all neighbors detect the failure.

- STABILIZING: a neighbor has been elected to take over the orphan zone. The local peer is waiting to make sure that no one else claims ownership of this zone.

- ACQUIRING: the local peer is about to take over a zone but it is still waiting in case a more appropriate neighbor would claim it.

- OWN: the local peer is authoritative for the zone.

In most cases, zones would be in either an OWN or a BORDERING state. States like SYNCHRONIZING, STABILIZING and ACQUIRING are only entered when a failure is detected and a zone needs to be taken over by another peer. The recovery algorithm uses three timers -- TC1, TC2, and TC3 -- and is completed through several exchanges of TAKEOVER messages. A peer MUST therefore be able to handle five different events for each of its neighbor zones, as defined in the remainder of this section. The transition from state OWN to BORDERING, which occurs only when a new peer joins the overlay, is separately described in Section 4.6.

4.2.1.1. Failure Detection Event

The event is fired when an XPP session between the local peer and one of its neighbors has failed. Upon According to the state that the neighbor zone had in the local zone list the local peer would do one of the following:

State: BORDERING

Set the zone state to SYNCHRONIZING; Start Timer TC1 for this zone and set it to 'tc1'. The 'tc1' value is proportional to the total volume of the zones under the authority of the local peer and always greater than the maximum time required for detecting a failure. Such a value MUST be calculated using the following formula (where r, t0, t1 and t2 are values used for handling retransmissions and keep alive in XPP [I-D.marocco-p2psip-xpp], v is the total volume under the authority authority of the local peer and V is the volume of the whole hash space):

r' = min(ceil(log2(t1/t0)), r)

tc0 = t0 * 2 ^ r' + (r - r') * t1 + t2

```
tc1 = tc0 * (1 + v / V)
```

State: SYNCHRONIZING, STABILIZING, ACQUIRING or OWN

Not possible.

4.2.1.2. Timer TC1 Event

State: SYNCHRONIZING

Set the zone state to ACQUIRING; start Timer TC2 with value tc2 (default: 6 seconds) and send a TAKEOVER operation request to all neighbors which are also neighbors of the zone to recover. The Takeover request MUST be generated as defined in Section 4.19.

State: BORDERING, STABILIZING, ACQUIRING or OWN

Not possible.

4.2.1.3. Timer TC2 Event

State: ACQUIRING

Set the zone state to OWN; the local peer is authoritative for the orphaned zone and the recovery algorithm is terminated. All neighbors are aware of the new owner, but the local peer may need to discover and establish XPP sessions with new neighbors acquired as a result of the recovery, and to notify its neighbors about possible changes in the geometry due to one or more merges between its previous zones and the recovered one.

New neighbors MUST be discovered sending QUERY operation requests to known neighbors, as defined in Section 4.13. The local peer MUST establish XPP sessions with all discovered neighbors using SIP.

> A peer MUST NOT establish XPP sessions with peers it does not know; however, after the recovery process, all neighbors of the dead zone will know the identity of the new peer and MUST accept incoming SIP requests from it.

If the recovered zone is mergeable with any of the zones previously owned by the local peer (i.e. they share an edge in 2-dimensional spaces, a plane in 3-dimensional ones and so on), all the neighbors MUST be notified of the merge through an UPDATE operation requests, as defined in Section 4.15.

State: BORDERING, SYNCHRONIZING, STABILIZING or OWN

Not possible.

4.2.1.4. TAKEOVER Operation Request Event

State: BORDERING

Leave the zone in a BORDERING state; send back a TAKEOVER operation request on behalf of the peer authoritative for the zone. The TAKEOVER request MUST be generated as defined in Section 4.19.

> This event would only occur when the failure is only detected by some of the neighbors, while others are still able to communicate with the peer.

State: SYNCHRONIZING

If, according to the rules defined in Section 4.2.2, the local peer is more appropriate than the one sending the TAKEOVER request, set the zone state to ACQUIRING; cancel Timer TC1, set Timer TC2 with value tc2 (default: 6 seconds) and send a TAKEOVER operation request to all neighbors that are also neighbors of the zone to recover. The TAKEOVER request MUST be generated as defined in Section 4.19.

Otherwise, if the local peer is less appropriate than the one sending the TAKEOVER request, set the zone state to STABILIZING; cancel Timer TC1, set Timer TC3 with value tc3 (default: 4 seconds), temporary store the identity of the peer candidate to take over the zone and send a TAKEOVER operation request to all neighbors which are also neighbors of the zone to recover on behalf of such peer. The TAKEOVER MUST be generated as defined in Section 4.19.

State: ACQUIRING

If, according to the rules defined in Section 4.2.2, the local peer is more appropriate than the one reported in the TAKEOVER request, keep the zone state as ACQUIRING; reset Timer TC2 to value tc2 (default: 6 seconds) and send a TAKEOVER operation request to all neighbors that are also neighbors of the zone to recover. The TAKEOVER request MUST be generated as defined in Section 4.19.

Otherwise, if the local peer is less appropriate than the one reported in the TAKEOVER request, set the zone state to STABILIZING; cancel Timer TC2, start Timer TC3 for a period of tc3 (default: 4 seconds), store the identity of the peer as the best candidate to take over the zone and send a TAKEOVER operation request to all neighbors which are also neighbors of the zone to recover on behalf of such peer. The TAKEOVER MUST be generated as defined in Section 4.19.

State: STABILIZING

If, according to rules defined in Section 4.2.2, the peer previously stored as the best candidate is more appropriate than the one reported in the TAKEOVER request, keep the zone in a STABILIZING state; reset Timer TC3 to tc3 (default: 4 seconds) and, on behalf of the old candidate, send TAKEOVER requests to all neighbors that are also neighbors of the zone to recover. The TAKEOVER request MUST be generated as defined in Section 4.19.

If the peer previously stored as the best candidate is less appropriate than the one reported in the TAKEOVER request, stay in STABILIZING; reset Timer TC3 with value tc3 (default: 4 seconds), store the identity of the latter as the best candidate and send a TAKEOVER operation request to all neighbors which are also neighbors of the zone to recover on behalf of such peer.

Otherwise, if the peer previously stored as the best candidate is the same as the one reported in the TAKEOVER request, keep the zone in a STABILIZING state and do nothing further.

State: OWN

Not possible.

4.2.1.5. Timer TC3 Event

State: BORDERING, SYNCHRONIZING, ACQUIRING or OWN

Not possible.

STABILIZING

Set the zone state to BORDERING; set the peer that has qualified as the best candidate as the new owner of the zone.

> The local peer may or may not have a connection with the new neighbor; in case it doesn't, it MUST be ready to accept a request for establishing one.

4.2.2. Neighbor Evaluation for Takeover

In order to determine whether one peer is more appropriate than another for taking over a zone, a list of conditions need to be taken into account in the following order:

1. if one of the peers is the current owner (e.g. the failure was temporary or just limited to some connections), it wins the

election;

2. if only one of the peers can merge the uncovered zone with its own, it wins the election. Note: two zones can be merged if they share a border component (an edge for 2d, a surface for 3d and so on);

3. if the sums of the zone volumes are different for the two peers, the one with the smallest value wins the election;

4. if none of the above produces a winner, the peer with the first identity in lexicographic order wins the election.

4.2.3. Stabilization Procedure

Since misbehaving peers, race conditions and temporary network failures may cause inconsistencies in their internal state, peers SHOULD periodically execute the following procedure in order to be sure it is coherent with that of their neighbors:

1. send a QUERY operation request targeted to each zone bordering on any of its own zone;

2. on reception of a QUERY operation response, update a temporary list for each of the reported zones:

   * if an equivalent zone with the same owner cannot be found in the list, insert it and assign a default initial score value;

   * if an equivalent zone with the same owner can be found in the list, increase its score value;

3. when QUERY operation responses have been received for all sent requests, resolve conflicting zones (i.e. zones with equal coordinates but different owners or zones with overlapping coordinates) keeping in the temporary list the one with the higher score value and dropping the other;

4. remove from the termorary list all zones conflicting with any zone in the effective list whose state is not BORDERING;

5. update the internal state replacing in the effective zone list all zones in BORDERING state with those in the temporary list.

4.3. Routing XPP Messages

When a peer receives a GET, a PUT or a QUERY operation request for a key or a target which is not contained in any of its zones, it MUST route it to the most appropriate neighbor.

> The most appropriate neighbor is generally the one whose center of mass is closest to the target; however, an implementation MAY also take in account link speed and so select not just according to geometric distance. In any case, in order to avoid loops, peers MUST NOT route messages to neighbors which are not geometrically closer than them to the destination zone.

XPP responses MUST be returned through the path that the originating request came through. When routing an XPP request, a peer MUST therefore locally store information for routing responses back on the same session it was received. Neither the XPP specification [I-D.marocco-p2psip-xpp] nor this document currently define for how long peers should keep state information about routed requests (this is for the moment an open issue, listed in (Section 8). However, it is still worth noting that as GET and PUT operations are supposed to be used for handling SIP transactions, it is RECOMMENDED that such an interval is not shorter than 32 seconds.

4.4. Handling SIP Registrations

All peers MUST be able to process SIP REGISTER requests sent directly by clients or routed by other peers in the overlay's domain, and update routing records in the distributed database with XPP PUT operations (see Section 4.7). Moreover, to overcome issues related to connectivity restrictions, such as NAT devices, peers MUST support the SIP outbound [I-D.ietf-sip-outbound] and GRUU [I-D.ietf-sip-gruu] extensions.

If the Contact header field in the incoming REGISTER contains the 'reg-id' parameter, the connection from which it was received MUST be kept alive as described in [I-D.ietf-sip-outbound] and the routing record MUST consist of a path where the first element is the URI identifying the peer (a SIP contact or a GRUU), the last element is the value in the Contact header field, and the elements in between are copied from those read in the Path header field [RFC3327], if one is set. Otherwise, if the registration does not require outbound support, the record MUST only contain the first value in the Contact header field.

If the REGISTER request has a Supported or Require header field containing the 'gruu' tag, the peer MUST generate a GRUU appending a 'gr' parameter with a value equal to the instance ID (reported in the

'+sip.instance' parameter in the Contact header field) to the address-of-record of the registering peer. In such cases, the peer MUST insert records for both the address-of-record and the GRUU into the distributed database, and return the GRUU in the SIP response as specified in [I-D.ietf-sip-gruu].

> It is worth mentioning that, according to mapping rules in Section 4.1, the same peer will be authoritative on both the GRUU and the address-of-record.

In general, a peer handling a REGISTER request is not necessarily the one storing the corresponding routing record; A registrar peer MUST send an XPP PUT operation request as described in Section 4.7 and wait for the XPP response to generate the SIP response. Since the XPP request could silently fail, if a response is not received after a "reasonable" time, it SHOULD close the SIP transaction with a 504 "Server Time-out" error code.

> The "reasonable" time period depends on several parameters, such as the overlay size and the transport protocol used for the SIP registration. It is RECOMMENDED that, if the REGISTER request was sent over UDP, such period is shorter than SIP's Timer F value (default: 32 seconds) [RFC3261]. The matter is still considered an open issue (Section 8).

4.5. Routing SIP Requests

When a peer receives a SIP request not addressed to itself and with no Route header field set, it MUST first determine if the target (the request URI) belongs to the overlay domain. If not, it SHOULD route it according to rules defined in [RFC3263]. Otherwise, if the target of the request is a SIP URI in the overlay domain, it MUST retrieve the routing record bound to the URI in the request line, generating an XPP GET operation request as defined in Section 4.9.

The corresponding GET operation response, returned by the peer authoritative for the destination URI, MUST contain the path SIP requests need to follow to reach the target. After receiving the GET operation response, the peer MUST recursively resolve all path components that are represented as URIs belonging to the overlay domain. Next, it MUST add Route header fields reflecting the resolved path and use it to route the request.

If the peer does not receive a GET response after a "reasonable" amount of time, it SHOULD close the SIP transaction with a 504 "Server Time-out" error code. As already mentioned , this "reasonable" amount of time is currently considered an open issue (Section 8).

As noted in Section 4.4, the "reasonable" time interval depends on several parameters, such as the overlay size and the transport protocol used for sending the SIP message. It is RECOMMENDED that, if the SIP request was sent over UDP, such period is shorter than 32 seconds.

4.6. Inviting a Client to Join

When a peer needs to invite a new client to join the overlay it first needs to select the "best" available among those it stores an active SIP registration for. This document does not specify how it could be done; implementations will apply their own criteria.

After identifying the client to invite and a zone to transfer (either one recovered from a dead peer or one obtained by splitting its own), the inviting peer MUST send a SIP INVITE request to the joining client in order to establish an XPP session with it. The INVITE request MUST have a Require header field containing the 'pcan' extension tag and will be routed as specified in Section 4.5.

In network environments where it is expected that peers might be located behind NAT devices, the session negotiation SHOULD be completed using the ICE [I-D.ietf-mmusic-ice] mechanism.

If the client answers positively and after the XPP session has been correctly established as described in [I-D.marocco-p2psip-xpp], the join procedure would continue through the following steps:

1. the inviting peer sends a PUT request generated as described in Section 4.7 to the entering peer for transferring all the data bound to keys located in the new zone;

2. the inviting peer sends an UPDATE request to all its neighbors. Generated UPDATE requests is described in Section 4.15. This request would report all zones of authority of the inviting peer as well as the one transferred to the entering peer. Peers receiving an UPDATE request MUST take into account any changes of the overlay topology and, in case they are also going to be neighbors of the new peer, prepare to establish XPP sessions with it.

3. the inviting peer then sends a JOIN request generated as described in Section 4.17 to the entering peer. The request contains coordinates of the zone being transfered and the list of its neighbors;

4. the entering peer establishes XPP sessions with all its new neighbors using SIP;

5. the inviting peer updates its zone list and closes sessions with peers that, as a result of the transfer, are no longer its neighbors.

4.7. Generating PUT Operation Requests

PUT operations insert key-value pairs in the distributed database. In general, such requests would be used when storing SIP registrations. Every such request is composed of one or more tuples containing:

- KEY: a SIP URI, usually an address-of-record or a GRUU [I-D.ietf-sip-gruu];
- VALUE: the route-path SIP messages addressed to the user SHOULD follow;
- BINDING-ID: the token identifying the key-value pair instance. Two subsequent PUT operations with same KEY and same BINDING-ID overwrite the same record; two PUT operations with same KEY and a different BINDING-ID would cause the insertion of two different records. For PUT requests generated upon SIP registrations, the BINDING-ID SHOULD contain the value of the Call-ID header field in the REGISTER request;
- EXPIRES: the amount of time (in seconds) that the record needs to be kept.

It is RECOMMENDED that requests containing more than one tuple are generated only if all tuples fall under the authority of the same peer; this could be the case, for example, during a join (Section 4.6) or when handling a registration requesting a GRUU (Section 4.4).

4.8. Handling PUT Operation Requests

Upon reception of a PUT operation request, a peer MUST first check if it is responsible for the KEY parameter contained in the first tuple. If it is not, it MUST route the request as defined in Section 4.3; otherwise, for each tuple in the request it is responsible for, it MUST update or insert a record in its local table, respectively if one with same KEY and BINDING-ID previously existed or not.

Once stored records it is responsible for, the peer MUST return in a PUT operation response, all records in its local table bound to keys matching one of those contained in the initial request. PUT operation responses are thus syntactically identical to PUT requests (see Section 4.7).

In general, according to replication policies, after storing data a peer peer SHOULD replicate it on some of its neighbor peers generating a REPLICA operation request as described in Section 4.11.

Even if the selection of the neighbor peers to send REPLICA requests to is not important per se, it needs to be consistent in a way that, whenever possible, REPLICA operation requests for records bound to the same keys are sent to the same peers.

4.9. Generating GET Operation Requests

GET operation requests retrieve data in the distributed database. Most often, GET requests would be used when querying the location of clients and peers for routing SIP messages. Such requests contain one or more KEY parameters, typically a SIP address-of-record or a GRUU [I-D.ietf-sip-gruu].

It is RECOMMENDED that requests containing more than one KEY parameter be generated only when all keys fall under the authority of the same peer.

4.10. Handling GET Operation Requests

Upon reception of a GET operation request, a peer MUST first check if it is responsible for the first KEY parameter. If it is not, it MUST route the request as defined in Section 4.3; otherwise, it MUST return in a GET operation response, all records in its local table bound to keys matching one of those contained in the initial request.

GET operation responses are syntactically identical to PUT responses (see Section 4.8).

4.11. Generating REPLICA Operation Requests

REPLICA operations insert key-value pairs in the local replica table of a neighbor peer. In general, such requests would be generated as a result of handling PUT operation requests. Every such request is composed of one or more tuples containing:

- KEY: a SIP URI, usually an address-of-record or a GRUU [I-D.ietf-sip-gruu];
- VALUE: the route-path SIP messages addressed to the user SHOULD follow;
- BINDING-ID: the token identifying the key-value pair instance. Two subsequent REPLICA operations with same KEY and same BINDING-ID overwrite the same record; two REPLICA operations with

same KEY and a different BINDING-ID would cause the insertion of two different records.

- EXPIRES: the amount of time (in seconds) that the record needs to be kept.

4.12. Handling REPLICA Operation Requests

Upon reception of a REPLICA operation request, a peer MUST update or insert a record in its local replica table, respectively if one with same KEY and BINDING-ID previously existed or not.

An entry in the local replica table is stored until either it expires or a new neighbor peer becomes responsible for it. In the latter case, other than removing from its replica table all entries the new neighbor is responsible for, the local peer MUST send a PUT operation request for all the purged entries, as described in Section 4.7.

4.13. Generating QUERY Operation Requests

QUERY operations are used by peers to gather information about the overlay topology, for example, for discovering new neighbors after a recovery. QUERY requests consist of one TARGET parameter used for routing the operation.

4.14. Handling QUERY Operation Requests

Upon reception of a QUERY operation request, a peer MUST first check if the TARGET parameter belongs to one of the zones it is authoritative for. If not, it MUST route the request as explained in Section 4.3; otherwise, it MUST return a representation of the zones it is aware of in a QUERY response containing:

- a list describing zones the answering peer is authoritative for, each one containing:
  - OWN-CONTACT: contact of the answering peer;
  - OWN-AOR: address-of-record the answering peer is registered for;
  - OWN-ZONE: coordinates of the zone;
- [TODO: The OWN-CONTACT and OWN-AOR entries in the above list are currently redundantly copied in every record. We should fix this in the next draft version.]

- a list describing zones neighboring those of the answering peer. Every list entry contains:
  - PEER-CONTACT: contact of the neighbor peer;
  - PEER-AOR: address-of-record the neighbor peer is registered for;
  - PEER-ZONE: coordinates of the zone.

4.15. Generating UPDATE Operation Requests

UPDATE operations are used for advertising changes in the overlay topology, such as joins, recoveries or zone merges. UPDATE requests MUST report information related to zones the sending peer is authoritative for or is currently transferring. They contain the following parameters:

- a list describing zones the sending peer is currently authoritative for, each one containing:
  - OWN-CONTACT: contact of the sending peer;
  - OWN-AOR: address-of-record the sending peer is registered with;
  - OWN-ZONE: coordinates of the zone;
- [TODO: The OWN-CONTACT and OWN-AOR entries in the above list are currently redundantly copied in every record. We should fix this in the next draft version.]
- a list containing all zones neighboring those of the sending peer. The list may also contain zones that used to be under the authority of the sending peer but have just been transfered to a joining node. List tuples MUST contain the following parameters:
  - PEER-CONTACT: contact of the neighbor peer;
  - PEER-AOR: address-of-record the neighbor peer is registered for;
  - PEER-ZONE: coordinates of the zone.

Contrary to QUERY responses, UPDATE requests MUST only report details that the sending peer is authoritative for, or zones that it is in the process of transfer to a joining peer after inviting it. (see Section 4.6).

4.16. Handling UPDATE Operation Requests

Upon reception of an UPDATE operation request, a peer MUST first remove from its zone list all zones overlapping with any of those reported in the request. It MUST then insert all zones from the UPDATE request that border zones it is authoritative for. UPDATE is a responseless request and MUST not be routed.

4.17. Generating JOIN Operation Requests

JOIN operations are used to pass the data that a client needs in order to join the overlay and become a peer. JOIN requests are generated by the inviting peer as described in Section 4.6 and contain the following parameters:

- SPACE: coordinates of the whole space;
- YOUR-ZONE: coordinates of the zone the entering peer will be authoritative for;
- a list describing zones the inviting peer is authoritative for, each one containing:
  - OWN-CONTACT: contact of the inviting peer;
  - OWN-AOR: address-of-record the inviting peer is registered for;
  - OWN-ZONE: coordinates of the zone;
- [TODO: The OWN-CONTACT and OWN-AOR entries in the above list are currently redundantly copied in every record. We should fix this in the next draft version.]
- a list describing zones bordering on the one the entering peer will be authoritative for, each one containing:
  - PEER-CONTACT: contact of the neighbor peer;
  - PEER-AOR: address-of-record the neighbor peer is registered for;
  - PEER-ZONE: coordinates of the zone.

4.18. Handling JOIN Operation Requests

JOIN operation requests can only be received during the join phase, as described in Section 4.6. JOIN is a responseless end-to-end operation and MUST not be routed; a peer receiving an unexpected JOIN

request MUST ignore it.

4.19. Generating TAKEOVER Operation Requests

TAKEOVER operations are used for electing the most appropriate peer to take over a zone left by a peer that has become unreachable. TAKEOVER requests are generated and exchanged by neighbors of the dead peer and contain the following parameters:

- DEAD-CONTACT: contact of the dead peer;
- DEAD-AOR: the address-of-record the dead peer was registered with;
- DEAD-ZONE: coordinates of the dead zone;
- PEER-CONTACT: contact of the peer candidate to take over the dead zone;
- PEER-AOR: address-of-record the peer candidate to take over the zone is registered for;
- PEER-ZONE: coordinates of a zone the peer candidate to take over the zone is authoritative for. If one or more of the zones the candidate peer is authoritative for is mergeable with the dead one, it SHOULD be reported in this parameter as it would increase the probability for the peer to be elected (see Section 4.2.2);
- PEER-VOLUME: the total volume of the zones the candidate peer is authoritative for.

4.20. Handling TAKEOVER Operation Requests

TAKEOVER operation requests are received only when recovering from failures and are handled according to the state diagram defined in Section 4.2.1.

Unexpected TAKEOVER requests, for example referring to non-bordering zones, MUST be ignored.

5. XPP Extensions

This section extends the XPP specification [I-D.marocco-p2psip-xpp] and defines codes and formats for operations and parameters used in XPP sessions between peers.

5.1. Parameter Formats

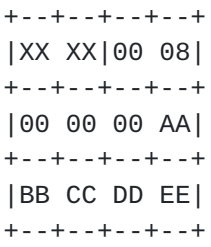
For convenience purposes we define a non-exclusive set of formats that we later use when defining PCAN related XPP parameters.

5.1.1. Integer

Length: the total number of bytes transported in the value. The length MUST always be divisible by 4.

Value: numeric, encoded in network byte order.

Example:



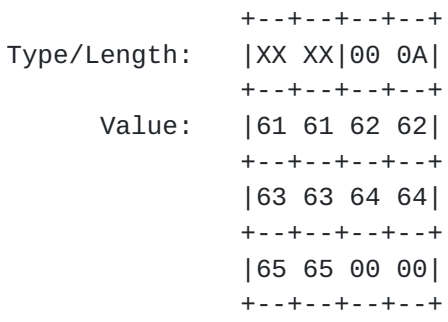
Encoding of an integer parameter with value 0xAABBCCDDEE.

5.1.2. String

Length: the total number of characters. If the length is not divisible by 4; the value field MUST be padded as defined in [I-D.marocco-p2psip-xpp].

Value: a sequence of ASCII characters.

Example:



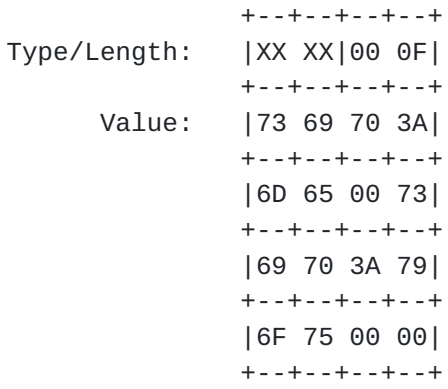
Encoding of a string parameter with value "AABBCCDDEE".

5.1.3. String List

Value: a list of ASCII strings terminated by a byte with value zero.

Length: the total number of characters, including the terminator bytes. If the length is not divisible by 4 it MUST be padded as defined in [I-D.marocco-p2psip-xpp].

Example:



Encoding of a URI list parameter with value {"sip:me", "sip:you"}.

5.1.4. Point

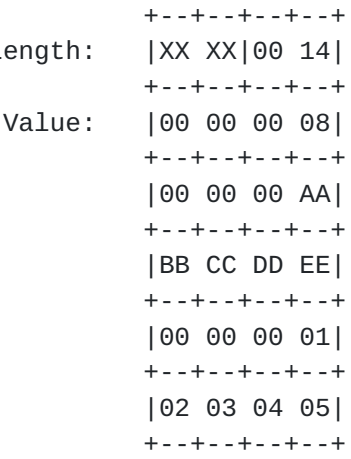
Value:

1. one 4 byte-long numeric value encoded in network byte order, representing the number of bytes used for encoding each one of the following components. The value of this field MUST always be divisible by 4.

2. a sequence of numeric values representing the components of a multidimensional point, each one encoded in network byte order and taking the number of bytes reported as the the first value of the parameter.

Length: the total number of bytes occupied by the length value and by the components. Such a value MUST be divisible by 4.

Example:



Encoding of a point parameter with components 0xAABBCCDDEE and 0x0102030405.

5.1.5. Zone

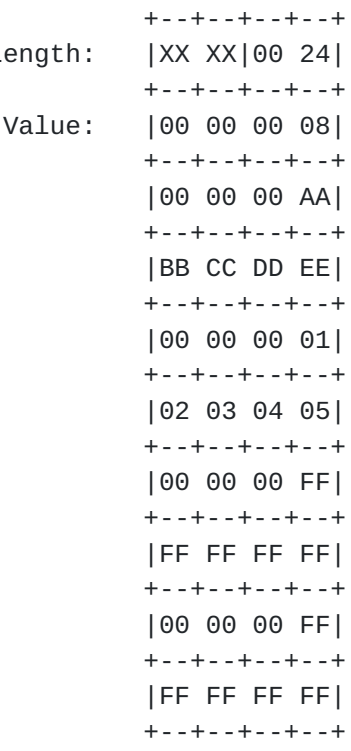
Value:

1. one 4 byte-long numeric value encoded in network byte order, representing the number of bytes used for encoding each one of the following components. The value of this field MUST always be divisible by 4;

2. a sequence of numeric values representing the components of two multidimensional points, each one encoded in network byte order and taking the number of bytes reported as the the first value of the parameter. The point defined by the first half of values represents the start vertex (the bottom-left corner of a rectangular zone in a bi-dimensional space) and the other defines the end vertex (the top-right corner, in a bi-dimensional zone).

Length: the total length of the Value field. Values of the length field must always be divisible by 4.

Example:



Encoding of a zone parameter represented by points (0xAABBCCDDEE, 0x0102030405) and (0xFFFFFFFFFF, 0xFFFFFFFFFF).

5.2. Parameters

This section describes all TLV parameters used by XPP-PCAN.

5.2.1. KEY

Code: 0x8001.

Format: String (Section 5.1.2).

Semantic: the key identifying a record to insert or to retrieve.

5.2.2. VALUE

Code: 0x8002.

Format: String list (Section 5.1.3).

Semantic: the set of peer URIs to traverse for reaching a client or a peer, as described in RFC 3327 [RFC3327]. Every URI MUST be encoding according to the rules defined in [RFC3986].

5.2.3. BINDING-ID

Code: 0x8003.

Format: string (Section 5.1.2).

Semantic: the token identifying a key-value pair.

5.2.4. EXPIRES

Code: 0x8004.

Format: integer (Section 5.1.1).

Semantic: the expiration time of a key-value pair, in seconds.

5.2.5. TARGET

Code: 0x8005.

Format: point (Section 5.1.4).

Semantic: the point in the overlay that a request is addressed to.

5.2.6. SPACE

Code: 0x8006.

Format: zone (Section 5.1.5).

Semantic: the bounds of the space used in the crrent overlay.

5.2.7. OWN-AOR

Code: 0x8007.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying the user who owns the sending peer, as defined in RFC 3261 [RFC3261] and RFC 3986 [RFC3986].

5.2.8. OWN-CONTACT

Code: 0x8008.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying the sending peer.

> It is worth noting that, when the peer has restricted connectivity (e.g. it is located in a NAT-ted network), the value of this parameter SHOULD be a GRUU [I-D.ietf-sip-gruu].

5.2.9. OWN-ZONE

Code: 0x8009.

Format: zone (Section 5.1.5).

Semantic: a zone the sending peer is authoritative for.

5.2.10. YOUR-ZONE

Code: 0x800A.

Format: zone (Section 5.1.5).

Semantic: a zone the receiving peer is going to be authoritative for (usually sent to joining peers).

5.2.11. PEER-AOR

Code: 0x800B.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying a user whose peer is known by the sending peer, as defined in RFC 3261 [RFC3261] and RFC3986 [RFC3261].

5.2.12. PEER-CONTACT

Code: 0x800C.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying a peer known by the sending peer.

It is worth noting that, when the peer has restricted connectivity, the value will be a GRUU [I-D.ietf-sip-gruu]. URI values MUST be encoded as specified in RFC3986 [RFC3986]

5.2.13. PEER-ZONE

Code: 0x800D.

Format: zone (Section 5.1.5).

Semantic: a zone a peer known by the sending peer is authoritative for.

5.2.14. PEER-VOLUME

Code: 0x800E.

Format: integer (Section 5.1.1).

Semantic: the total volume owned by a peer known by the sending peer.

5.2.15. DEAD-AOR

Code: 0x800F.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying a user whose peer is known by the sending peer, as defined in RFC 3261 [RFC3261]. URI values MUST be encoded as specified in RFC3986 [RFC3986]

5.2.16. DEAD-CONTACT

Code: 0x8010.

Format: String (Section 5.1.2).

Semantic: the SIP URI identifying a peer known by the sending peer.

It is worth noting that, when the peer has restricted connectivity, the value will be a GRUU [I-D.ietf-sip-gruu]. URI values MUST be encoded as specified in RFC3986 [RFC3986]

5.2.17. DEAD-ZONE

Code: 0x8011.

Format: zone (Section 5.1.5).

Semantic: a zone a peer known by the sending peer is authoritative for.

5.3. Operations

5.3.1. PUT

Code: 0x80000001.

Request parameters:

+[KEY VALUE BINDING-ID EXPIRES

Response parameters:

+[KEY VALUE BINDING-ID EXPIRES

Semantic: insert one or more records in the distributed database.

Routing: requests MUST be routed to the peer responsible for the first key; responses MUST follow the reverse path of requests.

5.3.2. GET

Code: 0x80000002.

Request parameters:

+[KEY

Response parameters:

*[KEY VALUE BINDING-ID EXPIRES

Semantic: retrieve one or more records in the distributed database.

Routing: requests MUST be routed to the peer responsible for the first key; responses MUST follow the reverse path of requests.

5.3.3. REPLICA

Code: 0x80000003.

Request parameters:

+[KEY VALUE BINDING-ID EXPIRES

Response parameters:

+[KEY VALUE BINDING-ID EXPIRES

Semantic: insert one or more records in the local replica table.

Routing: end-to-end, MUST NOT be routed.

5.3.4. QUERY

Code: 0x80000004.

Request parameters:

[TARGET

Response parameters:

+[OWN-CONTACT OWN-AOR OWN-ZONE

*[PEER-CONTACT PEER-AOR PEER-ZONE

Semantic: query the status of the peer authoritative for the zone a given point falls in.

Routing: requests MUST be routed to the peer authoritative on the zone which include the point encoded in TARGET parameter; responses MUST follow the reverse path of requests.

5.3.5. UPDATE

Code: 0x80000005.

Request parameters:

+[OWN-CONTACT OWN-AOR OWN-ZONE

*[PEER-CONTACT PEER-AOR PEER-ZONE

Semantic: status update upon a change.

Routing: end-to-end, MUST NOT be routed.

5.3.6. JOIN

Code: 0x80000006.

Request parameters:

[SPACE YOUR-ZONE

+[OWN-CONTACT OWN-AOR OWN-ZONE

*[PEER-CONTACT PEER-AOR PEER-ZONE

Semantic: join the overlay becoming authoritative on a given zone.

Routing: end-to-end, MUST NOT be routed.

5.3.7. TAKEOVER

Code: 0x80000007.

Request parameters:

[DEAD-CONTACT DEAD-AOR DEAD-ZONE

[PEER-CONTACT PEER-AOR PEER-ZONE PEER-VOLUME

Semantic: candidate a peer for taking over a zone.

Routing: end-to-end, MUST NOT be routed.

6. Security Considerations

It is possible to identify (at least) four different categories of security issues:

1. XPP transport issues, that can be addressed by securing the transport channels using a mechanism like DTLS [RFC4347];

2. eavesdropping and message forgery of SIP and media traffic by seemingly benign peers. While media can be encrypted using SRTP [RFC3711], SIP end-to-end security is still an open issue;

3. unsolicited XPP session establishments. Procedures for join and recovery are defined so that any peer needs to allow session establishments to peers whose identity have already been presented by one of its neighbors. Such a mechanism builds a sort of "overlay trust", which requires further investigation, because, while it seems one of the most powerful techniques for managing authorization in peer-to-peer environments, it could be exploited by malicious peers that have entered the overlay through an error or deceit;

4. overlay damages caused by malicious peers. This kind of issue is characteristic for peer-to-peer systems and, in general, is the hardest to deal with. However, with the "passive" approach, a malicious node cannot determine when it will become peer, which zone of the overlay it will be assigned and can only cause damages proportional to the area under its authority; moreover, implementations can apply their own methods, possibly based on both performance and social information, to filter malicious nodes and select the best ones to upgrade.

7. IANA Considerations

This document makes use following values which need to be registered with IANA:

1. 'pcan' SIP option tag.

8. Open Issues

1. Other than peer identities, credentials for authentication should be exchanged too. How to do that?

2. Can clients allow other clients to establish SIP outbound flows with them? In case, they need to implement RFC3327, don't they?

3. For the time being this document does not define the amount of time that peers should keep state information about routed requests, and whether this should be indicated in the requests. Shouldn't this be the case?

9. Acknowledgments

This document is a mere description of what has been implemented initially in SIPDHT [WWW.sipdht] and then in SIP Communicator [WWW.sip-communicator] opensource projects. Thanks to all the smart developers contributing there; special thanks to Stefano Marengo, who wrote almost all the code in the second version of SIPDHT.

Thanks also to many people working in IETF for providing input in various discussions; thanks in particular to Vijay Gurbani, Henning Schulzrinne and Salman Abdul Baset for giving useful feedback in the very initial phase of this work.

10. References

10.1. Normative References

[I-D.ietf-sip-gruu]
Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-12 (work in progress), March 2007.

[I-D.ietf-sip-outbound]
Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-07 (work in progress), January 2007.

[I-D.marocco-p2psip-xpp]
Marocco, E. and E. Ivov, "Extensible Peer Protocol (XPP)", draft-marocco-p2psip-xpp-01 (work in progress), June 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3261] 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.

[RFC3327] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

10.2. Informative References

[I-D.baset-sipping-p2pcommon]
Baset, S., "A Protocol for Implementing Various DHT Algorithms", draft-baset-sipping-p2pcommon-00 (work in progress), October 2006.

[I-D.ietf-behave-rfc3489bis]
Rosenberg, J., "Simple Traversal Underneath Network

Address Translators (NAT) (STUN)",
draft-ietf-behave-rfc3489bis-05 (work in progress),
October 2006.

[I-D.ietf-mmusic-ice]
Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols",
draft-ietf-mmusic-ice-13 (work in progress), January 2007.

[I-D.willis-p2psip-concepts]
Willis, D., Bryan, D., Matthews, P., and E. Shim, "Concepts and Terminology for Peer to Peer SIP",
draft-willis-p2psip-concepts-00 (work in progress), June 2006.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[WWW.columbia.skype]
Baset, S. and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol",
<http://www1.cs.columbia.edu/~salman/publications/skype1_4.pdf>.

[WWW.icir.can]
Ratnasamy, S., Francis, P., Handley, M., Karp, R., and S. Shenker, "A Scalable Content-Addressable Network",
<http://www.icir.org/sylvia/cans.ps>.

[WWW.microsoft.pastry]
Rowstron, A. and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems",
<http://research.microsoft.com/~antr/PAST/pastry.pdf>.

[WWW.mit.chord]
Stoica, I., Morris, R., Karger, D., and Frans Kaashoek, M., "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications",

<http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf>.

[WWW.rice.kademlia]
Maymounkov, P. and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", <http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf>.

[WWW.sip-communicator]
"SIP Communicator - the Java VoIP and Instant Messaging client", <https://sip-communicator.dev.java.net/>.

[WWW.sipdht]
"SIPDHT: A SIP-based Distributed Hash-table", <http://sipdht.sourceforge.net>.

Authors' Addresses

Enrico Marocco
Telecom Italia
Via G. Reiss Romoli, 274
Turin 10148
Italy

Email: enrico.marocco@telecomitalia.it

Emil Ivov
Louis Pasteur University and SIP Communicator
4 rue Blaise Pascal
Strasbourg Cedex F-67070
France

Email: emcho@sip-communicator.org

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).