P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 11, 2014

J. Maenpaa
G. Camarillo
Ericsson
May 10, 2014

Service Discovery Usage for REsource LOcation And Discovery (RELOAD)
draft-ietf-p2psip-service-discovery-11.txt

Abstract

REsource LOcation and Discovery (RELOAD) does not define a generic service discovery mechanism as a part of the base protocol. This document defines how the Recursive Distributed Rendezvous (ReDiR) service discovery mechanism used in OpenDHT can be applied to RELOAD overlays to provide a generic service discovery mechanism.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

REsource LOcation And Discovery (RELOAD) [RFC6940] is a peer-to-peer signaling protocol that can be used to maintain an overlay network, and to store data in and retrieve data from the overlay. Although RELOAD defines a Traversal Using Relays around Network Address Translation (TURN) specific service discovery mechanism, it does not define a generic service discovery mechanism as a part of the base protocol. This document defines how the Recursive Distributed Rendezvous (ReDiR) service discovery mechanism [Redir] used in OpenDHT can be applied to RELOAD overlays.

In a Peer-to-Peer (P2P) overlay network such as a RELOAD Overlay Instance, the peers forming the overlay share their resources in order to provide the service the system has been designed to provide. The peers in the overlay both provide services to other peers and request services from other peers. Examples of possible services peers in a RELOAD Overlay Instance can offer to each other include a

TURN relay service, a voice mail service, a gateway location service, and a transcoding service. Typically, only a small subset of the peers participating in the system are providers of a given service. A peer that wishes to use a particular service faces the problem of finding peers that are providing that service from the Overlay Instance.

A naive way to perform service discovery is to store the Node-IDs of all nodes providing a particular service under a well-known key k. The limitation of this approach is that it scales linearly with the number of nodes that provide the service. The problem is two-fold: the node n that is responsible for service s identified by key k may end up storing a large number of Node-IDs and most importantly, may also become overloaded since all service lookup requests for service s will need to be answered by node n. An efficient service discovery mechanism does not overload the nodes storing pointers to service providers. In addition, the mechanism must ensure that the load of providing a given service is distributed evenly among the nodes providing the service.

ReDiR implements service discovery by building a tree structure of the service providers that provide a particular service. The tree structure is stored into the RELOAD Overlay Instance using RELOAD Store and Fetch requests. Each service provided in the Overlay Instance has its own tree. The nodes in a ReDiR tree contain pointers to service providers. During service discovery, a peer wishing to use a given service fetches ReDiR tree nodes one-by-one from the RELOAD Overlay Instance until it finds a service provider responsible for its Node-ID. It has been proved that ReDiR can find a service provider using only a constant number of Fetch operations [Redir].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document uses the terminology and definitions from the Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] draft.

DHT: Distributed Hash Tables (DHTs) are a class of decentralized distributed systems that provide a lookup service similar to a regular hash table. Given a key, any peer participating in the system can retrieve the value associated with that key. The

responsibility for maintaining the mapping from keys to values is distributed among the peers.

H(x): Refers to a hash function (e.g., SHA-1) calculated over the value x.

I(lvl,k): An interval at level lvl in the ReDiR tree that encloses key k. As an example, I(5,10) refers to an interval at level 5 in the ReDiR tree within whose range key 10 falls.

n.id: Refers to the RELOAD Node-ID of node n.

Namespace: An arbitrary identifier that identifies a service provided in the RELOAD Overlay Instance. Examples of potential namespaces include "voice-mail" and "turn-relay". The namespace is an UTF-8 text string.

numBitsInNodeId: Refers to the number of bits in a RELOAD Node-ID. This value is used in the equations for calculating the ranges of intervals that ReDiR tree nodes are responsible for.

ReDiR tree: A tree structure of the nodes that provide a particular service. The nodes embed the ReDiR tree into the RELOAD Overlay Instance using RELOAD Store and Fetch requests. Each tree node in the ReDiR tree belongs to some level in the tree. The root node of the ReDiR tree is located at level 0 of the ReDiR tree. The child nodes of the root node are located at level 1. The children of the tree nodes at level 1 are located at level 2, and so forth. The ReDiR tree has a branching factor b. At every level lvl in the ReDiR tree, there is room for a maximum of b^lvl tree nodes. Each tree node in the ReDiR tree is uniquely identified by a pair (lvl,j), where lvl is a level in the ReDiR tree and j is the position of the tree node (from the left) at that level.

Successor: The successor of identifier k in namespace ns is the node belonging to the namespace ns whose identifier most immediately follows the identifier k.

3. Introduction to ReDiR

Recursive Distributed Rendezvous (ReDiR) [Redir] does not require new functionality from the RELOAD base protocol. This is possible since ReDiR interacts with the RELOAD Overlay Instance by simply storing and fetching data, that is, using RELOAD Store and Fetch requests. ReDiR creates a tree structure of the service providers of a particular service and stores it into the RELOAD Overlay Instance using the Store and Fetch requests. ReDiR service lookups require a logarithmic number of Fetch operations. Further, if information from past service lookups is used to determine the optimal level in the ReDiR tree from which to start new service lookups, an average service lookup can typically finish after a constant number of Fetch operations assuming that Node-IDs are distributed uniformly at random.

In ReDiR, each service provided in the overlay is identified by an identifier, called the namespace. All service providers of a given service store their information under the namespace of that service. Peers wishing to use a service perform lookups within the namespace of the service. The result of a ReDiR lookup for an identifier k in namespace ns is a RedirServiceProvider structure (see Section 4.1) of a service provider that belongs to ns and whose Node-ID is the closest successor of identifier k in the namespace.

Each tree node in the ReDiR tree contains a dictionary of RedirServiceProvider entries of peers providing a particular service. Each tree node in the ReDiR tree also belongs to some level in the tree. The root node of the ReDiR tree is located at level 0. The child nodes of the root node are located at level 1 of the ReDiR tree. The children of the tree nodes at level 1 are located at level 2, and so forth. The ReDiR tree has a branching factor, whose value is determined by a new element in the RELOAD overlay configuration document, called branching-factor. At every level lvl in the ReDiR tree, there is room for a maximum of branching-factor^lvl tree nodes. As an example, in a tree whose branching-factor is 2, the second level can contain up to 4 tree nodes (note that a given level may contain less than the maximum number of tree nodes since empty tree nodes are not stored). Each tree node in the ReDiR tree is uniquely identified by a pair (lvl,j), where lvl is a level in the ReDiR tree and j is the position of the tree node (from the left) at that level.

As an example, the pair (2,3) identifies the 3rd tree node from the left at level 2.

The ReDiR tree is stored into the RELOAD Overlay Instance tree node by tree node, by storing the values of tree node (level,j) under a key created by taking a hash over the concatenation of the namespace, level, and j, that is, as H(namespace,level,j). As an example, the root of the tree for a voice mail service is stored at H("voice-mail",0,0). Each node (level,j) in the ReDiR tree contains b intervals of the DHT's identifier space as follows:

```
          [2^numBitsInNodeID*b^(-level)*(j+(b'/b)),
           2^numBitsInNodeID*b^(-level)*(j+((b'+1)/b))), for 0<=b'<b,
```

where b is the branching-factor.

Figure 1 shows an example of a ReDiR tree whose branching factor is 2. In the figure, the size of the identifier space of the overlay is 16. Each tree node in the ReDiR tree is shown as two horizontal lines separated by a vertical bar ('|') in the middle. The horizontal lines represent the two intervals each node is responsible for. At level 0, there is only one node, (0,0) responsible for two intervals that together cover the entire identifier space of the RELOAD Overlay Instance. At level 1, there are two nodes, (1,0) and (1,1), each of which is responsible for half of the identifier space of the RELOAD Overlay Instance. At level 2, there are four nodes. Each of them owns one fourth of the identifier space. At level 3, there are eight nodes each of which is responsible for one eight of the identifier space.

```
  Level 0  __________________|__________________
                   |                   |
  Level 1  ________|________   ________|________
              |         |         |         |
  Level 2  ___|___   ___|___   ___|___   ___|___
            |   |     |   |     |   |     |   |
  Level 3  _|_ _|_   _|_ _|_   _|_ _|_   _|_ _|_
```

Figure 1: ReDiR tree

Figure 2 illustrates how tree nodes are numbered in the ReDiR tree at levels 0-2.

```
     Level 0  ________________(0,0)________________
                      |                   |
     Level 1  ______(1,0)______   ______(1,1)______
                 |         |         |         |
     Level 2  _(2,0)_   _(2,1)_   _(2,2)_   _(2,3)_
               |   |     |   |     |   |     |   |
     Level 3  _|_ _|_   _|_ _|_   _|_ _|_   _|_ _|_
```

Figure 2: ReDiR tree nodes

Figure 3 illustrates how intervals are assigned to tree nodes in the ReDiR tree at levels 0 and 1. As an example, the single tree node (0,0) at level 0 is divided into two intervals, each of which covers half of the identifier space of the overlay. These two intervals are [0,7] and [8,15].

```
     Level 0  ______[0,7]_______|_______[8,15]_____
                      |                   |
     Level 1  _[0,3]__|__[4,7]_   _[8,11]_|_[12,15]
                 |         |         |         |
     Level 2  ___|___   ___|___   ___|___   ___|___
               |   |     |   |     |   |     |   |
     Level 3  _|_ _|_   _|_ _|_   _|_ _|_   _|_ _|_
```

Figure 3: Intervals in ReDiR tree

Note that all of the examples above are simplified. In a real ReDiR tree, the default ReDiR branching factor is 10, meaning that each tree node is split into 10 intervals and that each tree node has 10 children. In such a tree, level 1 contains 10 nodes and 100 intervals. Level 2 contains 100 nodes and 1000 intervals, level 3 1000 nodes and 10000 intervals, etc. Further, the size of the identifier space of a real RELOAD Overlay Instance is at the minimum 2^128.

4. Using ReDiR in a RELOAD Overlay Instance

4.1. Data Structure

ReDiR tree nodes are stored using the dictionary data model defined in RELOAD base [RFC6940]. The data stored is a RedirServiceProvider Resource Record:

```
         enum { none(0), (255) }
           RedirServiceProviderExtType;

         struct {
           RedirServiceProviderExtType   type;
           Destination                   destination_list<0..2^16-1>;
           opaque                        namespace<0..2^16-1>;
           uint16                        level;
           uint16                        node;
           uint16                        length;

           select (type) {
               /* This type may be extended */
           } extension;

         } RedirServiceProvider;
```

The contents of the RedirServiceProvider Resource Record are as follows:

type

The type of an extension to the RedirServiceProvider Resource Record. Unknown types are allowed.

destination_list

A list of IDs through which a message is to be routed to reach the service provider. The destination list consists of a sequence of Destination values. The contents of the Destination structure are as defined in RELOAD base [RFC6940].

namespace

An opaque UTF-8 encoded string containing the namespace.

level

The level in the ReDiR tree.

node

The position of the node storing this RedirServiceProvider record at the current level in the ReDiR tree.

length

The length of the rest of the Resource Record.

extension

An extension value. The RedirServiceProvider Resource Record can be extended to include for instance service or service provider specific information.

4.2. Selecting the Starting Level

Before registering as a service provider or performing a service lookup, a peer needs to determine the starting level Lstart for the registration or lookup operation in the ReDiR tree. It is RECOMMENDED that Lstart is set to 2. In subsequent registrations, Lstart MAY, as an optimization, be set to the lowest level at which a registration operation has last completed.

In the case of subsequent service lookups, nodes MAY, as an optimization, record the levels at which the last 16 service lookups completed and take Lstart to be the mode of those depths.

4.3. Service Provider Registration

A node MUST use the following procedure to register as a service provider in the RELOAD Overlay Instance:

1. A node n with Node-ID n.id wishing to register as a service provider starts from a starting level Lstart (see Section 4.2 for the details on selecting the starting level). Therefore, node n sets the current level to level=Lstart.

2. Node n MUST send a RELOAD Fetch request to fetch the contents of the tree node responsible for I(level,n.id). An interval I(l,k) is the interval at level l in the ReDiR tree that includes key k. The fetch MUST be a wildcard fetch.

3. Node n MUST send a RELOAD Store request to add its RedirServiceProvider entry to the dictionary stored in the tree node responsible for I(level,n.id). Note that node n always stores its RedirServiceProvider entry, regardless of the contents of the dictionary.

4. If node n's Node-ID (n.id) is the lowest or highest Node-ID stored in the tree node responsible for I(Lstart,n.id), node n MUST reduce the current level by one (i.e., set level=level-1) and continue up the ReDiR tree towards the root level (level 0), repeating the steps 2 and 3 above. Node n MUST continue in this way until it reaches either the root of the tree or a level at which n.id is not the lowest or highest Node-ID in the interval I(level,n.id).

5. Node n MUST also perform a downward walk in the ReDiR tree, during which it goes through the tree nodes responsible for intervals I(Lstart,n.id), I(Lstart+1,n.id), I(Lstart+2,n.id), etc. At each step, node n MUST fetch the responsible tree node, and store its RedirServiceProvider record in that tree node if n.id is the lowest or highest Node-ID in its interval. Node n MUST end this downward walk as soon as it reaches a level l at which it is the only service provider in its interval I(l,n.id).

Note that above, when we refer to 'the tree node responsible for I(l,k)', we mean the entire tree node (that is, all the intervals within the tree node) responsible for interval I(l,k). In contrast, I(l,k) refers to a specific interval within a tree node.

4.4. Refreshing Registrations

All state in the ReDiR tree is soft. Therefore, a service provider needs to periodically repeat the registration process to refresh its RedirServiceProvider Resource Record. If a record expires, it MUST be dropped from the dictionary by the peer storing the tree node. Deciding an appropriate lifetime for the RedirServiceProvider Resource Records is up to each service provider. Every service provider MUST repeat the entire registration process periodically until it leaves the RELOAD Overlay Instance.

Note that no new mechanisms are needed to keep track of the remaining lifetime of RedirServiceProvider records. The 'storage_time' and 'lifetime' fields of RELOAD's StoredData structure can be used for this purpose in the usual way.

4.5. Service Lookups

The purpose of a service lookup for identifier k in namespace ns is to find the node that is a part of ns and whose identifier most immediately follows (i.e., is the closest successor of) the identifier k.

A service lookup operation resembles the service registration operation described in Section 4.3. Service lookups start from a given starting level level=Lstart in the ReDiR tree (see Section 4.2 for the details on selecting the starting level). At each step, a node n wishing to discover a service provider MUST fetch the tree node responsible for the interval I(level,n.id) that encloses the search key n.id at the current level using a RELOAD Fetch request. Having fetched the tree node, node n MUST determine the next action to carry out as follows:

1. If there is no successor of node n present in the just fetched ReDiR tree node (note: within the entire tree and not only within the current interval) responsible for I(level,n.id), then the successor of node n must be present in a larger segment of the identifier space (i.e., further up in the ReDiR tree where each interval and tree node covers a larger range of the identifier space). Therefore, node n MUST reduce the current level by one to level=level-1 and carry out a new Fetch operation for the tree node responsible for n.id at that level. The fetched tree node is then analyzed and the next action determined by checking conditions 1-3.

2. If n.id is neither the lowest nor the highest Node-ID within the interval (note: within the interval, not within the entire tree node) I(level,n.id), n MUST next check the tree node responsible

for n.id at the next level down the tree. Thus, node n MUST increase the level by one to level=level+1 and carry out a new Fetch operation at that level. The fetched tree node is then analyzed and the next action determined by checking conditions 1-3.

3. If neither of the conditions above holds, meaning that there is a successor s of n.id present in the just fetched ReDiR tree node and n.id is the highest or lowest Node-ID in its interval, the service lookup has finished successfully and s must be the closest successor of n.id in the ReDiR tree.

Note that above, when we refer to 'the tree node responsible for interval I(l,k)', we mean the entire tree node (that is, all the intervals within the tree node) responsible for interval I(l,k). In contrast, I(l,k) refers to a specific interval within a tree node.

Note also that there may be some cases in which no successor can be found from the ReDiR tree. An example is a situation in which all of the service providers stored in the ReDiR tree have a Node-ID smaller than identifier k. In this case, the upward walk of the service lookup will reach the root of the tree without encountering a successor. An appropriate strategy in this case is to pick one of the RedirServiceProvider entries stored in the dictionary of the root node at random.

Since RedirServiceProvider records are expiring and registrations are being refreshed periodically, there can be certain rare situations in which a service lookup may fail even if there is a valid successor present in the ReDiR tree. An example is a case in which a ReDiR tree node is fetched just after a RedirServiceProvider entry of the only successor of k present in the tree node has expired and just before a Store request that has been sent to refresh the entry reaches the peer storing the tree node. In this rather unlikely scenario, the successor that should have been present in the tree node is temporarily missing. Thus, the service lookup will fail and needs to be carried out again.

To recover from the kinds of situations described above, a ReDiR implementation MAY choose to use the optimization described next. The ReDiR implementation MAY implement a local temporary cache that is maintained for the duration of a service lookup operation in a RELOAD node. The temporary cache is used to store all RedirServiceProvider entries that have been fetched during the upward and downward walks of a service lookup operation. Should it happen that a service lookup operation fails due to the downward walk reaching a level that does not contain a successor, the cache is searched for successors of the search key. If there are successors

present in the cache, the closest one of them is selected as the service provider.

4.6. Removing Registrations

Before leaving the RELOAD Overlay Instance, a service provider MUST remove the RedirServiceProvider records it has stored by storing exists=False values in their place, as described in [RFC6940].

5. Access Control Rules

As specified in RELOAD base [RFC6940], every kind which is storable in an overlay must be associated with an access control policy. This policy defines whether a request from a given node to operate on a given value should succeed or fail. Usages can define any access control rules they choose, including publicly writable values.

ReDiR requires an access control policy that allows multiple nodes in the overlay read and write access to the ReDiR tree nodes stored in the overlay. Therefore, none of the access control policies specified in RELOAD base [RFC6940] is sufficient.

This document defines a new access control policy, called NODE-ID-MATCH. In this policy, a given value MUST be written and overwritten only if the the request is signed with a key associated with a certificate whose Node-ID is equal to the dictionary key. In addition, provided that exists=TRUE, the Node-ID MUST belong to one of the intervals associated with the tree node (the number of intervals each tree node has is determined by the branching-factor parameter). Finally, provided that exists=TRUE, H(namespace,level,node), where namespace, level, and node are taken from the RedirServiceProvider structure being stored, MUST be equal to the Resource-ID for the resource. The NODE-ID-MATCH policy may only be used with dictionary types.

6. REDIR Kind Definition

This section defines the REDIR kind.

Name

REDIR

Kind IDs

The Resource Name for the REDIR Kind-ID is created by concatenating three pieces of information: namespace, level, and node number. Namespace is an opaque UTF-8 encoded string identifying a service, such as "turn-server". Level is an integer specifying a level in the ReDiR tree. Node number is an integer identifying a ReDiR tree node at a specific level. The data stored is a RedirServiceProvider structure that was defined in Section 4.1.

Data Model

The data model for the REDIR Kind-ID is dictionary. The dictionary key is the Node-ID of the service provider.

Access Control

The access control policy for the REDIR kind is the NODE-ID-MATCH policy that was defined in Section 5.

7. Examples

7.1. Service Registration

Figure 4 shows an example of a ReDiR tree containing information about four different service providers whose Node-IDs are 2, 3, 4, and 7. In the example, numBitsInNodeID=4. Initially, the ReDiR tree is empty; Figure 4 shows the state of the tree at the point when all the service providers have registered.

```
     Level 0  ____2_3___4_____7_|__________________
                      |                   |
     Level 1  ____2_3_|_4_____7   ________|________
                 |         |         |         |
     Level 2  ___|2_3   4__|__7   ___|___   ___|___
               |   |     |   |     |   |     |   |
     Level 3  _|_ _|3   _|_ _|_   _|_ _|_   _|_ _|_
```

Figure 4: Example of a ReDiR tree

First, peer 2 whose Node-ID is 2 joins the namespace. Since this is the first registration peer 2 performs, peer 2 sets the starting

level Lstart to 2, as was described in Section 4.2. Also all other peers in this example will start from level 2. First, peer 2 fetches the contents of the tree node associated with interval I(2,2) from the RELOAD Overlay Instance. This tree node is the first tree node from the left at Level 2 since key 2 is associated with the second interval of the first tree node. Peer 2 also stores its RedirServiceProvider record in that tree node. Since peer 2's Node-ID is the only Node-ID stored in the tree node (i.e., peer 2's Node-ID fulfills the condition in Section 4.3 that it is the numerically lowest or highest among the keys stored in the node), peer 2 continues up the tree. In fact, peer 2 continues up in the tree all the way to the root inserting its own Node-ID in all levels since the tree is empty (which means that peer 2's Node-ID always fulfills the condition that it is the numerically lowest or highest Node-ID in the interval I(level, 2) during the upward walk). As described in Section 4.3, peer 2 also walks down the tree. The downward walk peer 2 does ends at level 2 since peer 2 is the only node in its interval at that level.

The next peer to join the namespace is peer 3, whose Node-ID is 3. Peer 3 starts from level 2. At that level, peer 3 stores its RedirServiceProvider entry in the same interval I(2,3) that already contains the RedirServiceProvider entry of peer 2. Interval I(2,3), that is, the interval at Level 2 enclosing key 3, is associated with the right hand side interval of the first tree node. Since peer 3 has the numerically highest Node-ID in the tree node associated with I(2,3), peer 3 continues up the tree. Peer 3 stores its RedirServiceProvider record also at levels 1 and 0 since its Node-ID is numerically highest among the Node-IDs stored in the intervals to which its own Node-ID belongs. Peer 3 also does a downward walk which starts from level 2 (i.e., the starting level). Since peer 3 is not the only node in interval I(2,3), it continues down the tree to level 3. The downward walk ends at this level since peer 3 is the only service provider in the interval I(3,3).

The third peer to join the namespace is peer 7, whose Node-ID is 7. Like the two earlier peers, also peer 7 starts from level 2 because this is the first registration it performs. Peer 7 stores its RedirServiceProvider record at level 2. At level 1, peer 7 has the numerically highest (and lowest) Node-ID in its interval I(1,7) (because it is the only node in interval I(1,7); peers 2 and 3 are stored in the same tree node but in a different interval) and therefore it stores its Node-ID in the tree node associated with that interval. Peer 7 also has the numerically highest Node-ID in the interval I(0,7) associated with its Node-ID at level 0. Finally, peer 7 performs a downward walk, which ends at level 2 because peer 7 is the only node in its interval at that level.

The final peer to join the ReDiR tree is peer 4, whose Node-ID is 4. Peer 4 starts by storing its RedirServiceProvider record at level 2. Since it has the numerically lowest Node-ID in the tree node associated with interval I(2,4), it continues up in the tree to level 1. At level 1, peer 4 stores its record in the tree node associated with interval I(1,4) because it has the numerically lowest Node-ID in that interval. Next, peer 4 continues to the root level, at which it stores its RedirServiceProvider record and finishes the upward walk since the root level was reached. Peer 4 also does a downward walk starting from level 2. The downward walk stops at level 2 because peer 4 is the only peer in the interval I(2,4).

7.2. Service Lookup

This subsection gives an example of peer 5 whose Node-ID is 5 performing a service lookup operation in the ReDiR tree shown in Figure 4. This is the first service lookup peer 5 carries out and thus the service lookup starts from the default starting level 2. As the first action, peer 5 fetches the tree node corresponding to the interval I(2,5) from the starting level. This interval maps to the second tree node from the left at level 2 since that tree node is responsible for the interval (third interval from left) to which Node-ID 5 falls at level 2. Having fetched the tree node, peer 5 checks its contents. First, there is a successor, peer 7, present in the tree node. Therefore, condition 1 of Section 4.5 is false and there is no need to perform an upward walk. Second, Node-ID 5 is the highest Node-ID in its interval, so condition 2 of Section 4.5 is also false and there is no need to perform a downward walk. Thus, the service lookup finishes at level 2 since Node-ID 7 is the closest successor of peer 5.

Note that the service lookup procedure would be slightly different if peer 5 used level 3 as the starting level. Peer 5 might use this starting level for instance if it has already carried out service lookups in the past and follows the heuristic in Section 4.2 to select the starting level. In this case, peer 5's first action is to fetch the tree node at level 3 that is responsible for I(3,5). Thus, peer 5 fetches the third tree node from the left. Since this tree node is empty, peer 5 decreases the current level by one to 2 and thus continues up in the tree. The next action peer 5 performs is identical to the single action in the previous example of fetching the node associated with I(2,5) from level 2. Thus, the service lookup finishes at level 2.

8. Overlay Configuration Document Extension

This document extends the RELOAD overlay configuration document by adding a new element "branching-factor" inside the new "REDIR" kind element:

redir:branching-factor: The branching factor of the ReDir tree. The default value is 10.

The Relax NG Grammar for this element is:

```
namespace redir = "urn:ietf:params:xml:ns:p2p:redir"

parameter &= element redir:branching-factor { xsd:unsignedInt }?
```

The 'redir' namespace is added into the <mandatory-extension> element in the overlay configuration file.

9. Security Considerations

There are no new security considerations introduced in this document. The security considerations of RELOAD [RFC6940] apply.

10. IANA Considerations

10.1. Access Control Policies

This document introduces one additional access control policy to the "RELOAD Access Control Policy" Registry:

NODE-ID-MATCH

This access control policy was described in Section 5.

10.2. A New IETF XML Registry

This document registers one new URI for the redir namespace in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:p2p:redir

Registrant Contact: The IESG

XML: N/A, the requested URI is an XML namespace

10.3. Data Kind-ID

This document introduces one additional data Kind-ID to the "RELOAD Data Kind-ID" Registry:

Kind	Kind-ID	RFC
REDIR	104	RFC-AAAA

This Kind-ID was defined in Section 6.

Note to RFC Editor: please replace AAAA with the RFC number for this specification.

10.4. ReDiR Namespaces

IANA SHALL create a "ReDiR Namespaces" Registry. Entries in this registry are strings denoting ReDiR namespace values. The initial contents of this registry are:

Namespace	RFC
turn-server	RFC-AAAA

The namespace 'turn-server' is used by nodes that wish to register as providers of a TURN relay service in the RELOAD overlay and by nodes that wish to discover providers of a TURN relay service from the RELOAD overlay.

Note to RFC Editor: please replace AAAA with the RFC number for this specification.

11. Acknowledgments

The authors would like to thank Marc Petit-Huguenin, Joscha Schneider, and Carlos Bernardos for their comments on the document.

12. References

12.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC6940] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", RFC 6940, January 2014.

12.2. Informative References

[I-D.ietf-p2psip-concepts]
Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-05 (work in progress), July 2013.

[Redir] Rhea, S., Godfrey, P., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and H. Yu, "Open DHT: A Public DHT Service and Its Uses", October 2005.

Authors' Addresses

Jouni Maenpaa
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Jouni.Maenpaa@ericsson.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com