P2PSIP Working Group J. Maenpaa
Internet-Draft G. Camarillo
Intended status: Standards Track Ericsson
Expires: January 13, 2011 July 12, 2010

Service Discovery Usage for REsource LOcation And Discovery (RELOAD)
draft-ietf-p2psip-service-discovery-01.txt

Abstract

REsource LOcation and Discovery (RELOAD) does not define a generic service discovery mechanism as 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) [I-D.ietf-p2psip-base] 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 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 in 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 nodes that provide a particular service and embedding it 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 hash table. Given a key, any participating peer 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): Hash calculated over x.

I(l,k): The unique interval at level l in the ReDiR tree that encloses key k.

n.id: Node-ID of node n.

Namespace: An arbitrary identifier that identifies a service provided in the RELOAD Overlay Instance. An example of a namespace is "voice-mail".

numBitsInNodeId: Number of bits in a Node-ID.

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.

Successor: The successor of identifier k in namespace ns is the node that has joined ns whose identifier most immediately follows 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 through a put/get API using RELOAD Store and Fetch requests. ReDiR builds a tree structure of the nodes that provide a particular service and embeds it into the RELOAD Overlay Instance using the Store and Fetch requests. ReDiR

performs lookup in a logarithmic number of Fetch operations with high probability. Further, if the tree's height is estimated based on past lookups, the average lookup can be reduced to 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 arbitrary identifier, called its namespace. All service providers join a namespace and peers wishing to use a service perform lookups within the namespace of the service. A ReDiR lookup for identifier k in namespace ns returns a node that has joined ns whose identifier is the closest successor of k.

Each tree node in the ReDiR tree contains a list of Node-IDs of peers providing a particular service. Each node in the tree has a level. The root is at level 0, the immediate children of the root are at level 1, 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 redirBranchingFactor. At every level i in the tree, there are at most redirBranchingFactor^i nodes. The nodes at any level are labeled from left to right, such that a pair (i,j) uniquely identifies the jth node from the left at level i. This tree is embedded into the RELOAD Overlay Instance node by node, by storing the values of node (i,j) at key H(namespace,i,j). As an example, the root of the tree for a voice mail service is stored at H("voice-mail",0,0). Each node (i,j) in the tree is associated with redirBranchingFactor intervals of the DHT keyspace as follows:

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

where b is the redirBranchingFactor.

Figure 1 shows an example of a ReDiR tree with a branching factor of 2. Each node is shown as two horizontal lines separated by a vertical bar. The 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. 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

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 [I-D.ietf-p2psip-base]. The data stored is a RedirServiceProvider structure:

```
            struct {
              NodeId                   serviceProvider;
              opaque                   namespace<0..2^16-1>;
              uint16                   level;
              uint16                   node;

              /* This type can be extended */

            } RedirServiceProviderData;

            struct {
              uint16                   length;
              RedirServiceProviderData data;
            } RedirServiceProvider;
```

The contents of the RedirServiceProvider structure are as follows:

length
   The length of the rest of the structure.

data
   The service provider data.

The contents of the RedirServiceProviderData structure are as follows:

serviceProvider
   The Node-ID of a service provider.

namespace
   An opaque 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.

The RedirServiceProviderData structure 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 initially set to 2. In subsequent registrations, Lstart MUST be set to the lowest level at which registration last completed.

In the case of service lookups, nodes MUST 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 level Lstart (see Section 4.2 for the details on selecting the starting level). Therefore, node n sets level=Lstart.
2. Node n MUST send a RELOAD Fetch request to fetch the contents of the tree node associated with I(level,n.id). An interval I(l,k) is defined as the unique interval at level l in the ReDiR tree that encloses 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 associated with I(level,n.id)

4. If node n's Node-ID is the numerically lowest or highest among the Node-IDs stored in the tree node associated with I(Lstart,n.id), node n MUST continue up the tree towards the root (level 0), repeating steps 2 and 3. Node n MUST continue this until it reaches either the root 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 walk down the tree through tree nodes associated with the intervals I(Lstart,n.id),I(Lstart+1,n.id),..., at each step fetching the current contents of the tree node, and storing its redirServiceProvider record if n.id is the lowest or highest Node-ID in the interval. Node n MUST end this downward walk when it reaches a level at which it is the only service provider in the interval.

Note that above, when we refer to 'the tree node associated with 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. If an entry (Node-ID) is not refreshed for redirRefreshPeriod seconds, it MUST be dropped from the dictionary by the peer storing the tree node. redirRefreshPeriod is a new element in the RELOAD overlay configuration document (see Section 8). 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 on identifier k in namespace ns is to find the node that has joined ns whose identifier most immediately follows identifier k.

A service lookup is similar to the registration operation. The service lookup starts from some level level=Lstart (see Section 4.2 for the details on selecting the starting level). At each step, the node n wishing to discover a service provider MUST fetch the tree node associated with the current interval I(level,n.id) using a RELOAD Fetch request and MUST determine where to look next as follows:

1. If the successor of node n is not present in the tree node associated with I(level,n.id), then its successor must occur in a larger range of the keyspace. Node n MUST set level=level-1 and try again.
2. If n.id is sandwiched between two Node-IDs in I(level,n.id), the successor must lie somewhere in I(level,n.id). Node n MUST set level=level+1 and repeat.
3. Otherwise, there is a Node-ID s stored in the node associated with I(level,n.id) whose identifier succeeds n.id, and there are no Node-IDs between n.id and s. Thus, s must be the closest successor of n.id, and the lookup is done.

Note that above, when we refer to 'the tree node associated with 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.

4.6. Removing Registrations

Before leaving the RELOAD Overlay Instance, a service provider MUST remove the redirServiceProvider records it has stored by storing "nonexistent" values in their place, as described in [I-D.ietf-p2psip-base].

5. Access Control Rules

As specified in RELOAD base [I-D.ietf-p2psip-base], 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 [I-D.ietf-p2psip-base] 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, the Node-ID MUST belong to one of the two intervals associated with the tree node. Finally, 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: service name, level, and node number. Service name is a string identifying a service, such as "voice-mail". 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. Example

Figure 2 shows an example of a ReDiR tree containing information about four different service providers whose Node-IDs are 2, 3, 4, and 7. Initially, the ReDiR tree is empty.

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

Figure 2: 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 overlay. 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 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 record in the same interval I(2,3) that already contains the Node-ID of peer 2. 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).

8. Overlay Configuration Document Extension

This document extends the RELOAD overlay configuration document by adding two new elements inside each "configuration" element.

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

redirRefreshPeriod: The interval at which a service provider needs to repeat the ReDiR registration process.

9. Security Considerations

There are no new security considerations introduced in this document. The security considerations of RELOAD [I-D.ietf-p2psip-base] 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. 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.

11. References

11.1. Normative References

[I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-08 (work in progress), March 2010.

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

11.2. 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-02 (work in progress), July 2008.

[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".

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