

P2PSIP Working Group
Internet Draft
Intended status: Standards Track
Expires: August 20 2013

J. Peng
Q. Yu
China Mobile
Y. Li
Beijing University of Posts and
Telecommunications
February 16, 2013

**One Hop Lookups Algorithm Plugin for RELOAD
draft-peng-p2psip-one-hop-plugin-03**

Abstract

This document defines a specific Topology Plugin using a ramification of the basic One Hop Lookups based DHT algorithm which is called ONE-HOP-RELOAD. In the One Hop Lookups algorithm, each peer maintains a full routing table containing information about every node on the overlay in order to route RELOAD message in one hop. Compared with CHORD-RELOAD algorithm, ONE-HOP-RELOAD improves the routing efficiency, and can maintain complete membership information with reasonable bandwidth requirements. This algorithm is able to handle frequent membership changes by superimposing a well-defined hierarchy on the system that guarantees topology disturbance events notification reach every peer in the overlay within a specified amount of time. Currently some typical peer-to-peer storages systems have stringent latency requirements, such as Amazon's Dynamo which is built for latency sensitive applications uses One-Hop algorithm, so that each node maintains enough routing information locally to route a request to the appropriate node directly.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on August 20, 2013.

Peng, et al.

Expires August 20, 2013

[Page 1]

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	Terminology	4
3.	Hash function	5
4.	Network Architecture.....	5
5.	Peer data structure	6
5.1.	Routing Table	6
5.2.	Peer Type	6
5.2.1.	Peer Type Structure	7
5.3.	Region ID	8
5.4.	Special Identification Information	8
6.	Routing	9
6.1.	Next-Hop Selection Mechanism	9
6.2.	Fault Tolerance.....	10
6.2.1.	Resource-ID Routing Failure	11
6.2.1.1.	Next-Hop Peer Leaving	11
6.2.1.2.	New Peer Joining	12
6.2.2.	Node-ID Routing Failure	13
6.2.2.1.	Next-Hop Peer Leaving	13
6.2.2.2.	Next-Hop Client Leaving	14
7.	Replica Placement Policy	14
8.	Joining	15
8.1.	Joining Process.....	15
8.2.	Joinreq Message Structure	18
9.	Leaving	18
9.1.	Handling Neighbor Failures	18
9.2.	Leavereq Message Structure	20
10.	Updates	22
10.1.	Topology Anomaly Detection	22
10.2.	Topology Disturbance Propagation Procedure	22

10.3. Updatereq Message Structure	23
10.3.1. Update Types	23
10.3.2. Routing Information	24
10.3.3. Event notification	25
10.3.4. ONE-HOP-RELOAD Update Data	27
11. Security Considerations	28
12. IANA Considerations	28
13. References	28
13.1. Normative References	28
13.2. Informative References	28
14. Acknowledgments	29
Authors' Addresses	30

1. Introduction

RELOAD [[I-D.ietf-p2psip-base](#)] is a peer-to-peer (P2P) signaling protocol for use on the Internet. RELOAD is explicitly designed to work with a variety of overlay algorithms, each implementation of that is provided by a Topology Plugin so that each overlay can select an appropriate overlay algorithm that relies on the common RELOAD core protocols and code. In the RELOAD base protocol, the Topology Plugin is defined by a DHT based on Chord, and this specification defines a new DHT based on One Hop Lookups which provides a high routing efficiency and can handle frequent membership changes in a way that has reasonable bandwidth consumption.

Structured peer-to-peer overlay network like Chord, Pastry, CAN provide a substrate for building large-scale distributed applications. These overlays allow applications to locate objects stored in the system in a limited number of overlay hops. These peer-to-peer lookup algorithms strive to maintain a small amount of per-node routing state, typically $O(\log N)$. All these $O(\log N)$ algorithms incur less maintenance traffic on large or high-churn networks but need nearly $O(\log N)$ steps to find one peer or resource, if there are a large number of peers in the overlay, it costs a lot time to locate peer or resource which may have a bad influence on the application level of RELOAD.

The experiment results [[One-Hop-Lookups](#)] show that the peer-to-peer system which uses the One Hop Lookups algorithm can route very efficiently even though the system is large and membership is changing rapidly. They show analytic results to prove that the whole routing table maintenance bandwidth requirements including the topology anomaly detection and topology disturbance propagation are small enough to make most participants in a 10^5 system, and the load on the peer in the overlay increases linearly with the size of the system. For example, in a system with 10^5 nodes, the load on an

ordinary peer is 3.84 kbps and the load on a slice leader is 35 kbps

upstream. In the simulation experiments described in the paper, they assume dynamic membership behavior (i.e., node joining and leaving) of the peer-to-peer system as in Gnutella, which is representative of an open Internet environment. From the study of Gnutella [[Gnutella](#)], we can draw the conclusion that there are about 20 and 200 membership change events per second, in a system with 10^5 and 10^6 peers respectively.

The real time communication has a high demand on the routing efficiency, for example, the VoIP usage on the application level of RELOAD, it depends on the looking up efficiency of the Overlay Topology. So, in a relative small and stable network like a low-churn telecom core net with VoIP applications, the O (1) algorithms as a Topology Plugin of RELOAD is beneficial.

Currently some typical peer-to-peer storages systems have stringent latency requirements, such as Amazon's Dynamo which is built for latency sensitive applications uses One-Hop algorithm, so that each node on the overlay maintains enough routing information locally to route a request to the appropriate node directly.

The algorithm described in this document is assigned the name ONE-HOP-RELOAD to indicate it is an adaptation of the basic One Hop algorithm based DHT. This algorithm uses the core techniques of the original One Hop Lookups, and has been adapted to RELOAD protocol. In the One Hop Lookups algorithm, each peer maintains a complete description of system memberships, and it uses dissemination trees to propagate event information to update all the peers' routing table within several seconds so that peers can maintain their own membership information accurately with low communications costs.

In this draft, the network architecture and the routing information which peers maintain in the ONE-HOP-RELOAD are first described. Then the replication and fault tolerance strategies are stated. At last, the overlay specific data structures into the RELOAD frame are filled, and some procedures with topology messages defined in the RELOAD Topology Plugin are implemented. All of the definitions and implementations based on the requirements and methods provided by RELOAD.

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](#)].

We use the terminology and definitions from the RELOAD Base Protocol [[I-D.ietf-p2psip-base](#)] extensively in this document.

3. Hash function

In this One Hop Lookups based topology plugin, the size of the Node-ID and Resource-ID is 128 bits. The hash of a Resource-ID MUST be computed by SHA-1 [[RFC3174](#)].

4. Network Architecture

Like in Chord, ONE-HOP-RELOAD uses a ring topology, and the successor and predecessor of a peer with Node-ID i refer to the first peer clockwise and counterclockwise respectively from peer i in the ring. A peer, n , is responsible for a particular Resource-ID k if k is less than or equal to n and k is greater than p , where p is the Node-ID of the predecessor of peer n . Care must be taken when computing to note that all math is modulo 2^{128} .

On this basis, we construct a three layered DHT to form dissemination trees which are used to propagate topology disturbance event information, i.e., joins and leaves. It is suggested that the scale of peer-to-peer system is pre-configured manually, and we impose this hierarchy on the system with dynamic membership by dividing the 128-bit circular identifier space into k equal contiguous intervals called slices, the i th slice contains all nodes currently in the overlay whose node identifiers lie in the range $[i \cdot 2^{128}/k, (i+1) \cdot 2^{128}/k)$ [OneHopLookups].

Each slice has a slice leader, which is chosen dynamically as the node that is the successor of the mid-point of the slice identifier space in the One Hop Lookups Algorithm, i.e., the slice leader of the i th slice is the successor node of the key $(i+1/2) \cdot 2^{128} / k$. But frequent slice leader changing due to joining or leaving of peers will cause higher network traffic expenses, because all the peers in the slice and other slice leaders need to modify the configuration and to establish a new connection with the new slice leader. Therefore, it is better to choose the peer with high reliability and availability peer to become the slice leader, and assign the peer a fixed Node-ID which is the successor of the mid-point of the slice identifier space in order to reduce the dynamic slice leader replacement.

Similarly, each slice is divided into equal-sized intervals called units. Each unit has a unit leader, which is dynamically chosen as the successor of the mid-point of the unit identifier space.

The choice of the number of levels in the hierarchy involves a tradeoff. A large number of levels imply a larger delay in propagating the information, whereas a small number of levels generate a large load at the nodes in the upper levels [OneHopLookups]. Recommended to choose the three level DHT, because it leads to reasonable bandwidth consumption and message propagation delay.

5. Peer data structure

Each peer keeps track of the Whole Routing Table and a neighbor table. The Whole Routing Table of each node keeps all of nodes' routing information, so that after receiving the RELOAD request message the peer can find the destination peer by just one hop if the items of routing table are correct and the peer is working properly. The neighbor table contains at least the three peers before and after this peer in the DHT ring. It is used for topology maintenance and node anomaly detection. There may not be three entries in any cases, e.g. in the case of small rings or during changing of the ring topology.

5.1. Routing Table

The routing table is the union of the neighbor table and the whole routing table.

Fundamentally, the neighbor table entry contains the Node-IDs of the predecessors and successors. The Whole Routing Table (WRT) maintains the routing information of all the nodes in the overlay. The WRT entry should at least contain the Node-ID, the address information that may be IPv4 or IPv6 addresses and should include IP addresses, port numbers and Resource-ID range which the peer is responsible for.

5.2. Peer Type

In the One Hop Lookups algorithm, whenever one peer detects a change in membership (its successor failed or it has a new successor), it will notify its local slice leader by sending an event notification message as soon as possible. The local slice leader collects all event notifications it receives from its own slice and disseminates these notifications to other slice leaders. At the same time, the slice leaders aggregate messages they receive for a short time period and then dispatch these messages to all unit leaders of their respective slices. Unit leaders spread these messages to themselves, successor and predecessor in internal unit.

From above, we can conclude that there are four kinds of peer type in the One Hop Lookups as follows:

Unit Boundary:

The peer at unit boundaries do not send topology disturbance event notification message (defined in Update message) to their neighboring peers outside their unit. This ensures that there is no redundancy in the communications: a peer will get Update message only from its neighbor that is one step closer to its unit leader, so that the Update message just being transferred within unit. In a unit, message is always flowing from the unit leader to the both ends of the unit.

Unit Leader:

Unit Leader receives Update Messages from slice leader, and spread these messages to its successor and predecessor in internal unit.

Slice Leader:

Slice leader is a special peer which is responsible for topology maintenance and management. It can collect Update Messages from slice internal and other slices and do other management tasks. Any peer in the slice can send topology disturbance event notification message (defined in Update message) to its slice leader to inform events. Different from SPM mechanism in the SandStone [\[SandStone\]](#), the slice leader in ONE-HOP-RELOAD participate in the resource location and discovery procedures, and it is best to be pre-configured.

Ordinary Peer:

Ordinary peer do not belong to the above three roles, it receives Update message from its successor or predecessor, and spread forward the message along the direction. Ordinary peer in the overlay know its unit leader and slice leader, and when it detects a change in membership (its predecessor failed or it has a new predecessor), it will notify its local slice leader.

Each peer in the overlay all need to record their peer type. Note that unit leader may also be unit boundary while the ring topology is small.

5.2.1. Peer Type Structure

```
enum { reserved (0),
```



```
ordinary_node(1), unit_boundary(2), unit_leader(3),  
    slice_leader(4), (255)}  
OneHopPeerType;
```

The `OneHopPeerType` gives an enumeration of peer type which identifies the different peer role. When a peer is both a unit boundary and a unit leader, we can use array of the `OneHopPeerType` which contains `unit_boundary` and `unit_leader` two elements to identify its peer type.

5.3. Region ID

The peer belonging to a specific unit of a specific slice should record its own location information. In the One Hop Lookups algorithm, we use `Slice-ID` and `Unit-ID` to mark the peer location. The scale of peer-to-peer system, i.e. the number of hierarchy levels, the number of slices and units, the `Node-ID` range of each slice and unit are all pre-configured manually. When a new peer prepares to join in the overlay, it can obtain configuration information from the configuration server which is responsible for assigning `Node-IDs` and providing `Node-ID` range forms for each slice and each unit. Then, the joining peer can compute its own `Region ID` by using the routing information obtained from its neighbors in the procedure of joining the overlay.

```
typedef opaque    SliceId[SliceIdLength];  
  
typedef opaque    UnitId[UnitIdLength];  
  
struct {  
SliceId    slice_id;  
UnitId    unit_id;  
} RegionId;
```

The struct of `RegionId` is used to identify the peer location. Both `SliceId` and `UnitId` is fixed-length structure represented as a series of bytes, with the most significant byte first. The length is set on a per-overlay basis within the range of 16-20 bytes (128 to 160 bits).

5.4. Special Identification Information

In the One Hop Lookups algorithm, each peer has its own peer type, such as ordinary node or unit leader. In order to maintain the

accuracy and stability of the overlay layer network architecture,

Peng, et al.

Expires August 20, 2013

[Page 8]

each peer also needs to maintain additional identification information which relates to the peer type closely, named as the special identification information.

Ordinary Node / Unit Boundary:

The ordinary node and unit boundary in the overlay should record its unit leader and slice leader identification. In addition, it can also maintain the Node-ID range of each slice and each unit obtained from the configuration server when joining in the overlay.

Unit Leader:

The identification information maintained by the unit leader and the ordinary node are basically the same, except that the unit leader does not need to maintain the Node-ID of unit leader.

Slice Leader:

Slice Leader collects Update messages from its own slice and other slices and spread these messages to the network according to the given rules. Thus, each slice leader should record the identification information about all the slice leaders and unit leaders within the slice it is responsible for in order to dispatch the Update messages.

Each peer needs to maintain appropriate identification information according to their peer type.

6. Routing

6.1. Next-Hop Selection Mechanism

Next-Hop Selection mechanism defines that a peer who receives a RELOAD message carrying the destination ID, i.e., Node-ID and Resource-ID, according to its own routing information, routes the message to the next hop peer on the overlay. Two scenarios, i.e. the Resource-ID routing and the Node-ID routing, are analyzed here.

Resource-ID routing:

If the peer is not responsible for the purpose Resource-ID k , but is directly connected to a node with Node-ID k , then it MUST route the message to that node. Otherwise, it finds the smallest Node-ID that is greater than k to indicate it should be responsible for the Resource-ID k ; and then route the message to that destination node.

Node-ID routing:

If the peer is not responsible for the purpose Node-ID k , but is directly connected to a node with Node-ID k , then it MUST route the message to that node. Otherwise, it finds the peer in the Whole Routing Table whose Node-ID equals to k to indicate it is the destination node k ; and then route the request to the peer.

If no such node is found, it finds the smallest Node-ID that is greater than k to indicate it should be the overlay access node of the client k ; and then route the message to that destination peer.

Note that in the CHORD-RELOAD algorithm [[I-D.ietf-p2psip-base](#)], each peer maintains a finger table which contains only a small fraction of routing information in the overlay, and peer has established TLS connections with all the peers in the routing table which is the union of the neighbor table and the finger table during the process of joining in the overlay. Thus, for the peer-to-peer system that uses the CHORD-RELOAD algorithm, the connection table that Link Management Module in the RELOAD protocol maintains is the set of nodes to which a node is directly connected, and the peers in the routing table will all be on the connection table but not vice versa.

The ONE-HOP-RELOAD algorithm differs from the CHORD-RELOAD algorithm. It gets each peer to maintain the Whole Routing Table, so when the system scales to million peers, the routing table will be very large. If the peer sets up TLS connections with all the other peers in the overlay, it will consume a large amount of network resources. Therefore, establishing full connections is not recommended. In the ONE-HOP-RELOAD algorithm, a peer only needs to select some related peers to establish TLS connections according to the needs of the network architecture, thus the connection table is a subset of the Whole Routing Table.

The consequence of changing the relationship between routing table and connection table is that the peer chosen by the Next-Hop Node Selection mechanism may not have an active TLS connection with the source node. Therefore, the peer may need to set up a connection before routing message to the destination node.

6.2. Fault Tolerance

Topology disturbance which is triggered by membership change events, i.e., joining and leaving, may lead to a one hop routing failure. If the source peer that prepares to send message to the next hop peer does not update its routing information in time when the topology

disturbance occurs, it may route messages to an incorrect peer or a peer which has withdrawn from the overlay.

Two scenarios, i.e. the Resource-ID routing and the Node-ID routing, are analyzed here. In Resource-ID routing scenario, the disturbances caused by peer joining and leaving are separately analyzed. In Resource-ID routing scenario, the disturbances caused by peer leaving and client leaving are separately analyzed. Note that Figure 1 is a peer-to-peer system schematic.

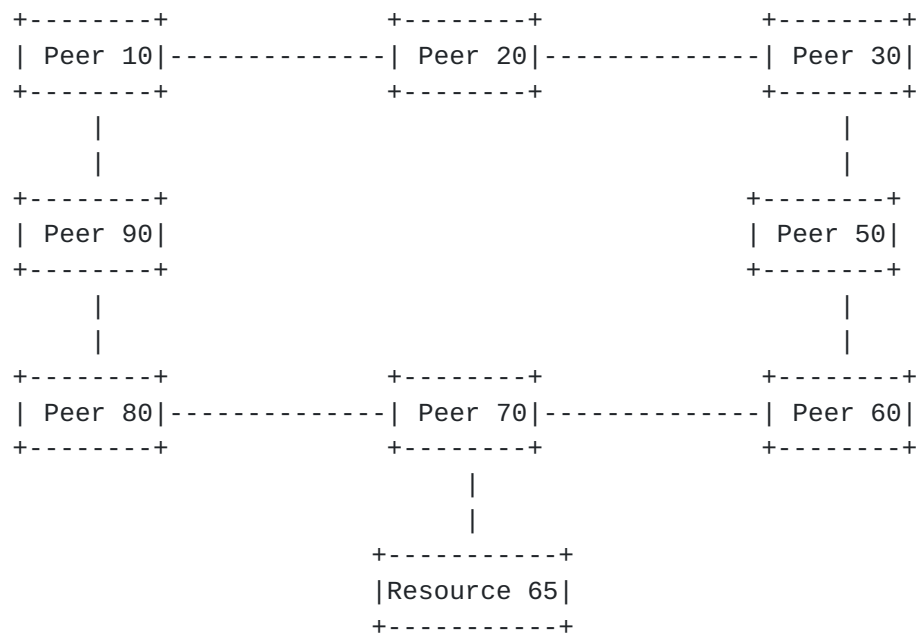


Figure 1 peer-to-peer system schematic

6.2.1. Resource-ID Routing Failure

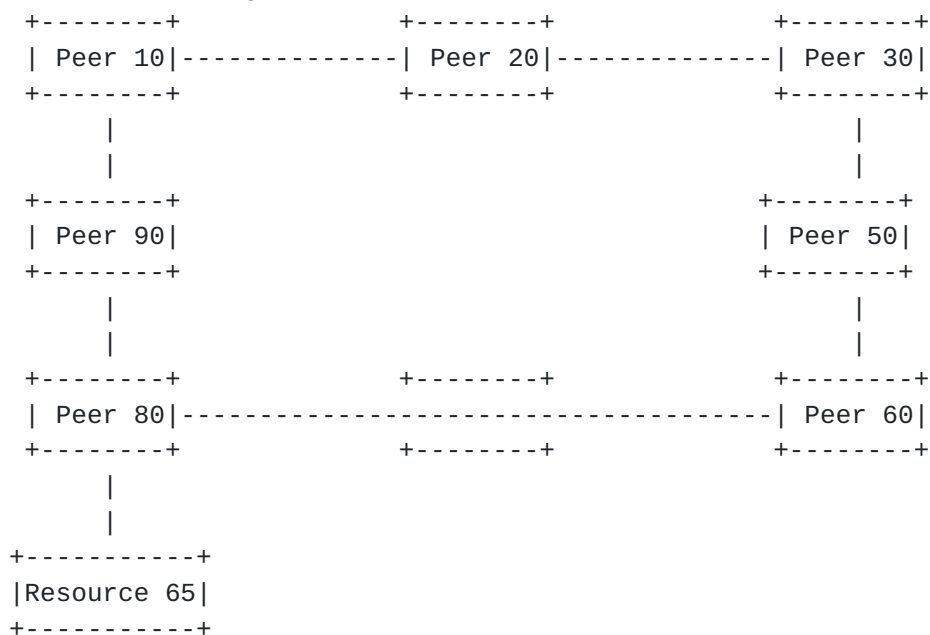
6.2.1.1. Next-Hop Peer Leaving

When a peer leaves the peer-to-peer system, the Resource-ID k which the peer is responsible for will be taken over by its successor. At the same time, a source peer in the overlay wants to route the RELOAD message to the peer who manages Resource-ID k and has just left the network. The message will then be routed to a non-existent peer so that the source peer will receive an error RELOAD response message, and the message Error Code name is Error_Request_Timeout which means that the request time is out or the target peer is unreachable.

For instance, Peer 70 has left the overlay, and its Resource 65 has been taken over by Peer 80. At this time, Peer 20 want to send a RELOAD message to the peer who is responsible for the Resource 65, but its routing table has not been updated due to this topology

changes, then Peer 20 would route the RELOAD message to the old Peer 70, and this one hop routing to the Peer 70 will fail due to Peer 70 leaving. After that, Peer 20 will receive an error RELOAD response message whose Error Code name is Error_Request_Timeout.

Peer 20 who receives this error message can take two response measures. If using reactive fault tolerance mechanism, it then sends an immediate RELOAD message to the successor Peer 80 of the failed Peer 70. Otherwise, it should wait for receiving a topology disturbance message to update its own routing information, and then resend the RELOAD message.



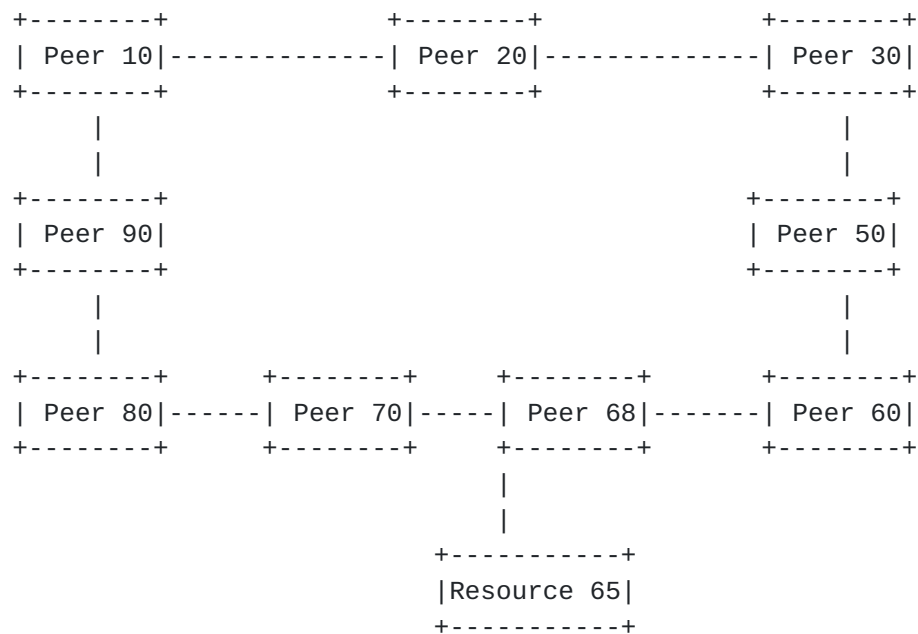
6.2.1.2. New Peer Joining

When a peer joins the peer-to-peer system, it will take over the Resource-ID k which its successor node is responsible for. At the same time, a source peer in the overlay wants to route the RELOAD message to the peer who manages Resource-ID k during the past and has just transferred Resource-ID k to the new online peer, then the message will be routed to an old and incorrect peer so that the source peer may receive an error RELOAD response message, and the message Error Code Name is Error_Not_Found which means that target resource is not part of the destination peer management. Another way to deal with the new peer joining is that the old peer forwards the RELOAD message to the new joining peer directly, and this mechanism will increase the numbers of routing hops.

For instance, Peer 68 has just joined the overlay network and taken over the Resource 65 from successor Peer 70. At this time, Peer 20 want to send a RELOAD message to the peer who is responsible for the

Resource 65, but its routing table has not been updated due to this topology changes, then Peer 20 would route the RELOAD message to the old Peer 70.

Peer 70 who receives this error message can take two response measures. If using routing forwarding schema, Peer 70 would directly forward the message to the new joining Peer 68 who is responsible for the Resource 65. Otherwise, Peer 70 would return an error response to Peer 20 whose Error Code name is Error_Not_Found.

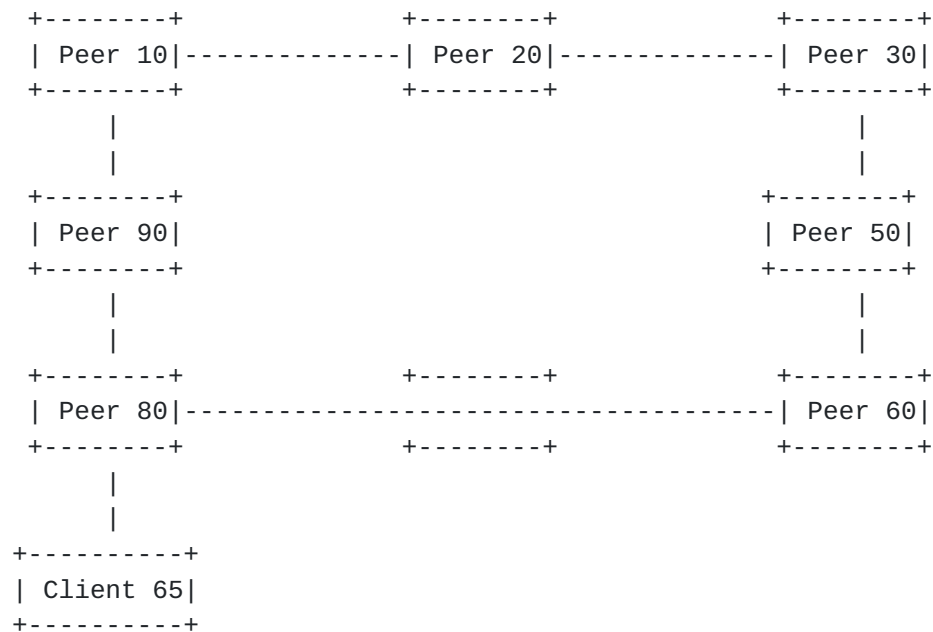


6.2.2. Node-ID Routing Failure

6.2.2.1. Next-Hop Peer Leaving

The source peer in the overlay route the RELOAD message to the Node-ID m of the peer who has left the network, then message will be routed to a non-existent peer so that the source peer will receive an error RELOAD response message, and the message Error Code name is Error_Request_Timeout which means that the request time is out or the target peer is unreachable.

For instance, Peer 70 has left the overlay network. At this time, Peer 20 want to send a RELOAD message to the Peer 70, but its routing table has not been updated due to this topology changes, then Peer 20 would route the RELOAD message to the old Peer 70, and this one hop routing to the Peer 70 will fail due to Peer 70 leaving. After that, Peer 20 will receive an error RELOAD response message whose Error Code name is Error_Request_Timeout.



6.2.2.2. Next-Hop Client Leaving

The source peer in the overlay route the RELOAD message to the Node-ID *m* of the client who has left the network, then message will be routed to the Overlay Access Peer that is responsible for the leaving client *m*.

This Overlay Access Peer who receives the RELOAD message can take two response measures. If using reactive fault tolerance mechanism, it then response an error RELOAD response message, and the message Error Code name is `Error_Not_Found` which means that target resource is not part of the destination peer management. Otherwise, the source peer will receive an error RELOAD response message, and the message Error Code name is `Error_Request_Timeout` which means that the request time is out or the target peer is unreachable.

7. Replica Placement Policy

To achieve high availability and reliability, ONE-HOP-RELOAD replicates data in multiple peers, three on default. The replica placement policy has two kinds of design scheme.

1. Two backup data stored in two successors.

2. The way defined in SandStone [[SandStone](#)] can also be used. The first data is stored in the primary peer; the second replica is saved in a peer of different unit but in the same slice; and the third replica is saved in a peer in a different slice. The most important issue of this scheme is the choosing of the replica peer. Recommended that two levels of strip segmentation based ID assignment mechanism can be used to allocate Node-ID and choose backup nodes.

To guarantee the consistency among multiple replicas is a difficult but inevitable task. When a peer receives a Store request for Resource-ID k, and it is responsible for Resource-ID k, it MUST store the data and returns a success response and then send a Store request to its backup nodes to maintain replicas consistency. This part is beyond the scope of the present document, specific details can be found in reference papers.

The topology disturbance always leads to the changes of data backup relationships between several the data storage peers. If using replica placement policy similar to the SandStone, the distance between the master data storage peer and backup peer may be far, therefore the procedure of data migration caused by the topology disturbance may not be triggered immediately. There are two kinds of trigger the procedure of data migration strategy:

1. Reactive Notification: The joining or leaving peer or the peer who detected topology disturbances should take the initiative to inform the affected backup node, and then trigger the data migration.
2. Passively Waiting: When the peer received topology change event notification message, it should test whether its own data backup relationship is affected or not. If affected, it will trigger data migration process.

8. Joining

8.1. Joining Process

The joining process for a joining peer (JP) with Node-ID n is as follows:

1. JP MUST connect to its chosen or preconfigured bootstrap node.
2. JP SHOULD send an Attach request to its admitting peer (AP) for Node-ID n. The "send_update" flag should be used to acquire the routing table and other information from AP.

3. JP determines its peer type and Region-ID according to the routing information of AP. If JP is an ordinary peer, it should send Attach requests to initiate connections to each of the peers in the neighbor table. After establishing connections with all the peers in the neighbor table, JP MUST record all the peers it has contacted into the neighbor table. If JP is other type of peer, it will perform some additional processes for a special peer type described as follows:

Unit Boundary:

If JP is the Unit Boundary where AP locates, AP is the old unit boundary and JP will replace its role. Then JP piggybacks its own peer types on the Update message to AP in order to inform AP to convert peer role.

If JP and AP are not in the same unit or even slice, and JP and PP are in the same unit, JP will replace the role of PP which is the old Unit Boundary where JP locates.

If JP is the new Unit Leader and there is no peer within the unit, JP also is the new Unit Boundary of this unit.

Unit Leader:

If JP is the new Unit Leader and AP is in the same unit with JP, AP is the old leader who is about to be replaced. JP informs AP to convert the peer type of AP and add Node-ID of the new Unit Leader into the routing information of AP. Then JP should notify its slice leader to modify the unit leader identification information and trigger the topology disturbance procedure.

If there is no peer within the unit and JP is both the new Unit Leader and the new Unit Boundary whose Node-ID is pre-configured manually, JP should notify its slice leader to modify the unit leader identification information and trigger the topology disturbance procedure.

Slice Leader:

If JP is the new Slice Leader where AP locates, AP is the old leader who is about to be replaced. JP informs AP to convert its peer type and modify its special identification information. If using reactive scheme, JP should send Attach message to establish TLS connection with all the other slice leaders and the whole peers in the slice.

If there is no peer within the slice and JP is the new Slice Leader whose Node-ID is pre-configured manually, JP needs to obtain the special identification information of Slice Leader from the configuration server, and wait for peer joining its slice.

4. If JP is not new slice leader, it MUST send an Attach request to the slice leader where JP locations in order to timely report topology changing information to its slice leader.
5. JP MUST send a Join to AP, the AP sends the response to the Join.
6. AP MUST do a series of Store requests to JP to store the data that JP will be responsible for.
7. AP MUST send JP an Update explicitly labeling JP as its predecessor. At this point, JP is part of the ring and responsible for a section of the overlay. AP can now forget any data which is assigned to JP and not AP.
8. JP piggybacks its routing information on the Update message to AP. AP should start topology change event propagation process. The process is described in [section 10.2](#).

If JP sends an Attach to other peers in the overlay with send_update, it will receive the Update messages from the other peers which carry the routing information of other peers, including the type of the peer, its region, and its own neighbor table. JP can construct its own routing information from these information, and perform related procedures to join the overlay and play its own role.

Note that in the CHORD-RELOAD algorithm [[I-D.ietf-p2psip-base](#)], each peer keeps track of a finger table and a neighbor table, and peer has established TLS connections with all the peers in the routing table which is the union of the neighbor table and the finger table in the process of joining in the overlay. But the ONE-HOP-RELOAD algorithm differs from the CHORD-RELOAD. It defines that each peer maintains the Whole Routing Table so that the routing information is too large that the peer can't establish connections with all the peers in the Whole Routing Table. Therefore, in the joining process described above, we select some related peers in the overlay to establish connections with the joining peer according to the different peer type.

8.2. Joinreq Message Structure

Before sending Joinreq message to AP, JP needs to construct its own routing information and send Attach messages to related peers. If its peer type is special, it also should trigger some relevant peers to perform peer type conversion procedure. After completing these tasks, JP sends Joinreq message to AP, informing AP that it has been completed the preparatory work to join the overlay, and can start to take over the overlay data which it is responsible for.

The `overlay_specific_data` field in Joinreq message MUST contain the `OneHopJoinData` structure defined below:

```
struct {  
  
    OneHopPeerType    joining_peer_type;  
  
    RegionId  
    region_id;  
  
    IPAddressPort     joining_peer_address;  
} OneHopJoinData;
```

The contents of this structure are as follows:

`joining_peer_type`:

The peer type of JP.

`region_id`:

The identification of the region where JP locates.

`joining_peer_address`:

The address information of JP may be IPv4 or IPv6 addresses. If AP receives Joinreq message carried the address information of JP, it should add the information into its own routing table.

AP which receives a Joinreq message from JP should check whether the message is correct and return a success response if correct.

9. Leaving

9.1. Handling Neighbor Failures

Every time the peer in the overlay finds that its neighboring peer

has already left the network or is ready to leave (maybe as

determined by connectivity pings or the Leave message from the leaving peer), it MUST perform the following tasks:

- o Obtaining the routing information of leaving node. When the peer finds that its neighbor has left the network abnormally, it needs to calculate the routing information of the leaving peer according to its local routing information.
- o Updating its routing information, including removing the entry from its neighbor table and replacing it with the best match peer in its own Whole Routing Table.
- o If the peer is the successor of the leaving peer, it also should start the topology disturbance propagation procedure to report and propagate the topology disturbance event. The process is described in [section 10.2](#). It is also recommended that multiple relevant peers should simultaneously report the events to its slice leader, in order to avoid the loss of topology disturbance information.
- o If the data management or backup relationship changes, triggering the corresponding data migration procedure.

If the type of the leaving peer is special, it will perform some additional processes for a special peer type described as follows:

Unit Boundary:

LP is the Unit Boundary where its predecessor or successor locations, then one of the two neighbors will become the new unit boundary and replace the role of LP.

Unit Leader:

If JP is the Unit Leader where its successor NP locates, NP will become the new Unit Leader who will replace the role of LP. NP piggybacks Unit Leader replacement information on the Update message to the slice leader. The slice leader received this Update message would modify the unit leader identification information.

Slice Leader:

If JP is the Slice Leader where its successor NP locates, NP will become the new Slice Leader who will replace the role of LP. If using reactive scheme, the new slice leader NP should send Attach messages to establish TLS connections with all the other slice leaders and the whole peers in its own slice; and then send the topology disturbance event information to the peers in its own

slice and all the other slice leaders. Note that the successor NP MUST maintain the replica of routing information of LP which contains the identifications of all the slice leaders and unit leaders in the slice.

9.2. Leavereq Message Structure

Peers SHOULD send a Leave request to all members of their neighbor table prior to exiting the Overlay Instance. The `overlay_specific_data` field MUST contain the `OneHopLeaveData` structure defined below:

```

struct {
    NodeId      predecessors<0...2^16-1>;

    NodeId      successors<0...2^16-1>;
} Neighbors;

struct {
    OneHopNodeType
predecessors<0...2^16-1>;

    NodeId      successors<0...2^16-1>;
} OneHopLeaveData;

struct {
    OneHopPeerType
leaving_peer_type;

    RegionId
region_id;

    Neighbors    neighbor_table;

    select (leaving_peer_type) {
        case ordinary_peer:

NodeId      unit_leader;

NodeId      slice_leader;

```

```
case unit_leader:
```

```
    NodeId      slice_leader;
```

```
case slice_leader:
```

```
    NodeId      unit_leader_list<0...N>;
```

```
    NodeId      slice_leader_list<0...N>;
```

```
        }  
    } OneHopLeaveData;
```

The contents of this structure are as follows:

region_id:

The identification of the region where JP locates.

neighbor_table:

The neighbor_table contains all the Node-IDs of the current entries in the neighbor table except for the leaving one.

The 'leaving_peer_type' field indicates the type of the leaving peer:

If the type of the leaving peer is 'ordinary_peer' the contents will be:

unit_leader

The Node-ID of the Unit Leader of the unit where leaving peer locates.

slice_leader

The Node-ID of the Slice Leader of the slice where leaving peer locates.

If the type of the leaving peer is 'unit_leader' the contents will be:

slice_leader

The Node-ID of the Slice Leader of the slice where leaving peer locates.

If the type of the leaving peer is 'slice_leader' the contents will be:

unit_leader_list

The Node-ID list of each Unit Leader in the slice which the the leaving Slice Leader manages.

slice_leader_list

The Node-ID list of all the other Slice Leader.

When a peer is ready to leave the overlay, it takes the initiative to send a Leave message to all peers in its own neighbor table. Any peer which receives a Leave for a peer in its neighbor set follows procedures as if it had detected a peer failure as described in [Section 9.1](#).

10. Updates

10.1. Topology Anomaly Detection

A peer MUST maintain connections with all the peers in its neighbor table. A peer MUST try to maintain at least three predecessors and successors at least. In the CHORD-RELOAD algorithm [I-D.ietf-p2psip-base], it is RECOMMENDED that $O(\log(N))$ predecessors and successors be maintained in the neighbor table.

Every peer in the overlay including Slice Leader MUST periodically send a keep-alive message (defined by Update message) to all the peer in its neighbor table. The purpose of this is to keep the predecessor and successor lists up to date and to detect failed peers. The default time is about every ten minutes. A peer SHOULD randomly offset these Update requests so they do not occur all at once.

When detecting the topology anomaly, the peer follows procedures as described in [Section 9.1](#).

10.2. Topology Disturbance Propagation Procedure

The successor of the joining peer or leaving peer will trigger the topology disturbance propagation procedure that realizes topology change event spread in the network in accordance with the given direction. The purpose of this is to keep the Whole Routing Table up to date and to adjust the network topology.

The topology disturbance propagation procedure for the successor with Node-ID n who detects a topology change, i.e., new predecessor joining and old predecessor leaving, as follows:

1. The successor MUST send an Update message which piggybacks the topology change event information to its slice leader directly.
2. The slice leader MUST collect all topology change events it receives from the peers in its own slice and aggregates them during several seconds (default time is about 20 seconds); and then sends Update messages to the other slice leaders to spread these events. Best not to send Update messages to other slice leaders at the same time.
3. The slice leader waits for a short minutes (default time is about 10 seconds), and aggregates all Update messages received during this period; and then dispatch the Update message to all the unit leaders in its own slice.

4. A unit leader SHOULD forward the Update message to its successor and predecessor.
5. Other peers propagate this Update message in one direction: if they receive from their successors, they SHOULD send it to their successors and vice versa. Note that the Unit Boundaries SHOULD NOT send Update messages to their neighbor peers outside their unit.

10.3. Updatereq Message Structure

The purpose of using Update message is to piggyback the routing information of the peer or to spread the topology disturbance events information on the overlay. Thus, Update message can carry two kinds of content: routing information of the peer on the overlay and topology disturbance event information, i.e., peer joining and peer leaving.

The Update message for the ONE-HOP-RELOAD is defined as follows.

10.3.1. Update Types

```
enum { reserved (0),  
        routing_info (1), event_notification(2), (255)}
```

UpdateType;

The 'UpdateType' gives an enumeration of Update message including 'routing_info' and 'event_notification'

routing_info

The routing_info message piggybacks the routing information of the message sending peer, including the routing table which may be the union of the Whole Routing Table and neighbor table or only the neighbor table, peer type, region ID and special identification information.

event_notification

The event_notification message will take a list of topology disturbance events information which indicates topology changing in the overlay and may trigger peers to modify their routing information.

10.3.2. Routing Information

```
enum { reserved (0),  
  
      full (1), peer_info(2), (255)}  
  
RoutingInfoType;
```

The 'RoutingInfoType' gives an enumeration of routing_info including 'full' and 'peer_info' it identifies the different types of routing_info message:

full

The kind of routing_info message piggybacks all the routing table of the message sending peer, including the Whole Routing Table and neighbor table.

peer_info

The kind of routing_info message piggybacks only the neighbor table of the message sending peer and some other identification information.

```
struct {  
  
    NodeId      peer_id;  
  
    IPAddressPort  address;  
  
} OneHopRoutingInfo;
```

The struct of 'OneHopRoutingInfo' is used to construct the Whole Routing Table entry. The WRT entry should at least contain the Node-ID and the address information that may be IPv4 or IPv6 addresses:

peer_id

The Node-ID of the peer in the overlay.

address

The address information that may be IPv4 or IPv6 addresses.

```
struct {  
    OneHopPeerType  
    peer_type;
```

```
        RegionId      region_id;

        Neighbors      neighbor_table;

        select (peer_type) {

            case ordinary_peer:

                NodeId      unit_leader;

                NodeId      slice_leader;

        case unit_leader:

            NodeId      slice_leader;

        case slice_leader:

            NodeId      unit_leader_list<0...N>;

            NodeId      slice_leader_list<0...N>;

        }

        RoutingInfoType  routing_info_type;

        select (routing_info_type) {

            case full:

                OneHopRoutingInfo  whole_routing_info<0...N>;

        case peer_info:

            }
        } RoutingInfo;
```

The parameters of this structure are similar to the 'OneHopLeaveData'

struct described in [section 9.2](#), except that the 'RoutingInfo' could piggyback the Whole Routing Table.

The 'Routing_info_type' field indicates whether the message includes the WRT or not. If the type of the field is 'Full' the contents will be:

whole_routing_info

The variable represents the information of peers' routing table which has all of the peers' information in the overlay.

If the type of the field is 'peer_info' the message does not carry WRT.

10.3.3. Event notification

```
enum { reserved (0),  
peer_joining (1), peer_leaving(2), (255)}
```

EventNotificationType;

The 'EventNotificationType' gives an enumeration of topology disturbance event kinds, including peer joining and peer leaving.

```
struct {  
    EventNotificationType    event_notification_type;  
    select (event_notification_type) {  
        case peer_joining:  
  
            OneHopRoutingInfo    joining_peer_info;  
  
            OneHopPeerType        joining_peer_type;  
  
            RegionId              region_id;  
  
            select (joining_peer_type) {  
  
                case unit_leader:  
  
                    RegionId        change_region_id;  
  
                    NodeId           old_unit_leader;  
  
                case slice_leader:  
  
                    RegionId        change_region_id;  
  
                    NodeId           old_slice_leader;  
  
            }  
  
        case peer_leaving:  
  
            OneHopRoutingInfo    leaving_peer_info;  
  
            OneHopPeerType        leaving_peer_type;
```

```

RegionId      region_id;

select (leaving_peer_type) {

    case unit_leader:

        RegionId      change_region_id;

        NodeId        new_unit_leader;

    case slice_leader:

        RegionId      change_region_id;

        NodeId        new_slice_leader;

    }

}

} EventNotificationItem;

```

The structure of 'EventNotificationItem' is an item of the EventNotification message. The peer who receives this item should refresh its routing information and do some other topology management tasks.

The contents of this structure are as follows:

The 'Event_notification_type' field indicates the type of the topology disturbance event, including 'peer_joining' and 'peer_leaving'

If the type of the event is 'peer_joining' the contents will be:

```

joining_peer_info

```

The entry corresponds to the Node-ID of joining peer in the Whole Routing Table

region_id

The identification of the region where joining peer locates.

The 'joining_peer_type' field indicates the peer type of the joining peer whose type is special, including 'slice_leader' and 'unit_leader'

If the type of the joining peer is 'unit_leader' the contents will be:

change_region_id

The identification of the region where joining peer locates.
The joining peer will take the place of the old Unit Leader.

old_unit_leader

The Node-ID of the old Unit Leader who has been replaced by the joining peer.

If the type of the joining peer is 'slice_leader' the contents will be:

change_region_id

The identification of the region where joining peer locates.
The joining peer will take the place of the old Slice Leader.

old_slice_leader

The Node-ID of the old Slice Leader who has been replaced by joining peer.

If the type of the event is 'peer_leaving' the contents are similar to the parameters of 'peer_joining' event, except that when the leaving peer is the old Unit Leader or Slice Leader, the corresponding parameters specify the Node-ID of the new Unit Leader or Slice Leader.

10.3.4. ONE-HOP-RELOAD Update Data

```
struct {  
    UpdateType      update_type;
```



```
        Select (update_type) {

case routing_info:

    RoutingInfo    routing_info;

case event_notification:

    EventNotificationItem  event_notification_list<0...N>;

    }
    } OneHopUpdateData;
```

This structure is the Update message used in the ONE-HOP-RELOAD. The Update message is composed by two kinds of data. One of them is the RoutingInfo structures, and the other is the EventNotification structures. All the peers in the overlay can use this Update message to carry routing information or spread topology disturbance event information.

11. Security Considerations

There is no specific security consideration associated with this draft.

12. IANA Considerations

There are no IANA considerations associated to this memo.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", [RFC 2234](#), Internet Mail Consortium and Demon Internet Ltd., November 1997.
- [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-15](#) (work in progress),

May 2011.

13.2. Informative References

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

Peng, et al.

Expires August 20, 2013

[Page 28]

[One-Hop-Lookups]

Gupta, A., Liskov, B., and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays", June 2003.

[SandStone]

Shi, G., Chen, J., Gong, H., Fan, L., Xue, H., Lu, Q., and L. Liang, "SandStone: A DHT based Carrier Grade Distributed Storage System", September 2009.

[Gnutella]

S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of per-to-peer file sharing systems", Jan 2002.

14. Acknowledgments

Authors?Addresses

Jin Peng
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: pengjin@chinamobile.com

Qing Yu
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: yuqing@chinamobile.com

Yuan Li
Beijing University of Posts and Telecommunications
10 Xi Tu Cheng Rd.
Haidian District
Beijing 100876
P.R.China

Email: liyuan8903@139.com