Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin
July 11, 2011

A RELOAD Usage for Distributed Conference Control (DisCo)
draft-knauf-p2psip-disco-03

Abstract

This document defines a RELOAD Usage for Distributed Conference Control (DisCo) with SIP. DisCo preserves conference addressing through a single SIP URI by splitting its semantic of identifier and locator using a new Kind data structure. Conference members are enabled to select conference controllers based on proximity awareness and to recover from failures of indivihttp://www.ietf.org/id/draft-knauf-p2psip-share-01.txtdual resource instances. DisCo proposes call delegation to balance the load at focus peers.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 12, 2012.

Copyright Notice

Copyright (c) 2011 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 4
2. Terminology 5
3. Overview of DisCo 6
3.1. Reference Scenario 6
3.2. Initiating a Distributed Conference 7
3.3. Joining a Conference 8
3.4. Conference State Synchronization 9
3.5. Call delegation 10
3.6. Resilience 10
3.7. Topology Awareness 10
4. RELOAD Usage for Distributed Conference Control 11
4.1. Shared Resource DisCo-Registration 11
4.2. Kind Data Structure 11
4.3. Variable Conference Identifier 12
4.4. Conference Creation 13
4.5. Advertising Focus Ability 14
4.6. Determining Coordinates 14
4.7. Proximity-aware Conference Participation 15
4.8. Configuration Document Extension 17
5. Conference State Synchronization 18
5.1. Event Package Overview 18
5.2. <distributed-conference> 20
5.3. <version-vector>/<version> 20
5.4. <conference-description> 21
5.5. <focus> 22
5.5.1. <focus-state> 23
5.5.2. <users>/<user> 23
5.5.3. <relations>/<relation> 24
5.6. Distribution of Change Events 24
5.7. Translation to Conference-Info Event Package 25
5.7.1. <conference-info> 26
5.7.2. <conference-description> 26
5.7.3. <host-info> 26
5.7.4. <conference-state> 26
5.7.5. <users>/<user> 27
5.7.6. <sidebars-by-ref>/<sidebars-by-value> 27
6. Distributed Conference Control with SIP 28

6.1. Call delegation 28
6.2. Conference Access 29
6.3. Media Negotiation and Distribution 30
6.3.1. Offer/Answer 30
6.4. Restructuring a Conference 31
6.4.1. On Graceful Leave 31
6.4.2. On Unexpected Leave 32
7. DisCo Kind Definition 33
8. XML Schema 34
9. Relax NG Grammar 38
10. Security Considerations 39
10.1. Trust Aspects 39
11. IANA Considerations 40
12. Acknowledgments 41
13. References 42
13.1. Normative References 42
13.2. Informative References 43
Appendix A. Change Log 44
Authors' Addresses 46

1. Introduction

This document describes a RELOAD Usage for distributed conference control (DisCo) in a tightly coupled model with SIP [RFC3261]. The Usage provides self-organizing and scalable signaling that allows RELOAD peers, clients and plain SIP user agents to participate in a managed P2P conference. DisCo defines the following functions:

- A SIP protocol scheme for distributed conference control
- RELOAD Usage and definition of conferencing Kind
- Mechanisms for conference synchronization and call delegation
- Mechanism for proximity-aware routing within a conference
- An XML event package for distributed conferences

In this document, the term distributed conferencing refers to a multiparty conversation in a tightly coupled model in which the point of control (i.e., the focus) is identified by a unique URI, but the focus service is located at many independent entities. Multiple SIP [RFC3261] user agents uniformly control and manage a multiparty session. This document defines a new Usage for RELOAD, including an additional Kind code point with a corresponding data structure that complies with the demands for distributed conferences. The 'DisCo' data structure stores the mapping of a single conference URI to multiple conference controllers and thereby separates the conference identifier from focus instantiations.

Authorized controllers of a conference are permitted to register their mapping in the DisCo data structure autonomously. Thus, the DisCo data structure represents a co-managed Resource in RELOAD. To provide trusted and secure access to a co-managed Resource, this document uses the definitions for Shared Resources (ShaRe) [I-D.knauf-p2psip-share].

Delay and jitter are critical issues in multimedia communications. The proposed conferencing scheme supports mechanisms to build an optimized interconnecting graph between conference participants and their responsible conference controllers. Conference members will be enabled to select the closest focus with respect to delay or jitter.

DisCo extends conference control mechanisms to provide a consistent and reliable conferencing environment. Controlling peers maintain a consistent view of the entire conference state. The multiparty system can be re-structured based on call delegation operations.

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 [RFC2119].

We use the terminology and definitions from der RELOAD base draft[I-D.ietf-p2psip-base], the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts], the usage for shared resources draft [I-D.knauf-p2psip-share],and the terminology formed by the framework for conferencing with SIP [RFC4353]. Additionally the following terms are used:

Coordinate Value: An opaque string that describes a host's relative position in the network topology.

Focus peer: A RELOAD peer that provides SIP conferencing functions and implements the Usage for distributed conferencing. It is called 'active' if already involved in signaling relation to conference participants. Otherwise, if only registered in a distributed conference data structure, it is referred to as a 'potential' focus peer.

3. Overview of DisCo

3.1. Reference Scenario

The reference scenario for the Distributed Conference Control (DisCo) is shown in Figure 1. Peers are connected via a RELOAD [I-D.ietf-p2psip-base] instance, in which peers A and B are managing a single multiparty conference. The conference is identified by a unique conference URI, but located at peers A and B fulfilling the role of the focus. The mapping of the conference URI to one or more responsible focus peers is stored in a new RELOAD Resource for distributed conferencing within a data structure denoted as DisCo-Registration. The storing peer SP of the distributed conference resource holds this data.

The focus peers A and B maintain SIP signaling relations to conference participants, which may have different conference protocol capabilities. In this example, peer A is the focus for the RELOAD peer C and the plain SIP user agent E whereas peer B serves as a focus for RELOAD peer D and the RELOAD client F.

RELOAD peers and clients obtain the contact information for the conference from the storing peer. In contrast to this, the user agent E receives the conference URI not by RELOAD mechanisms, but resolves the ID and joins the conference by plain SIP negotiation.

Focus peers maintain a SIP signaling relation among each other used for notification messages that synchronize the conference focus peers' knowledge about the entire conference state. Additionally, focus peers can transfer calls to each other by a call delegation mechanism.

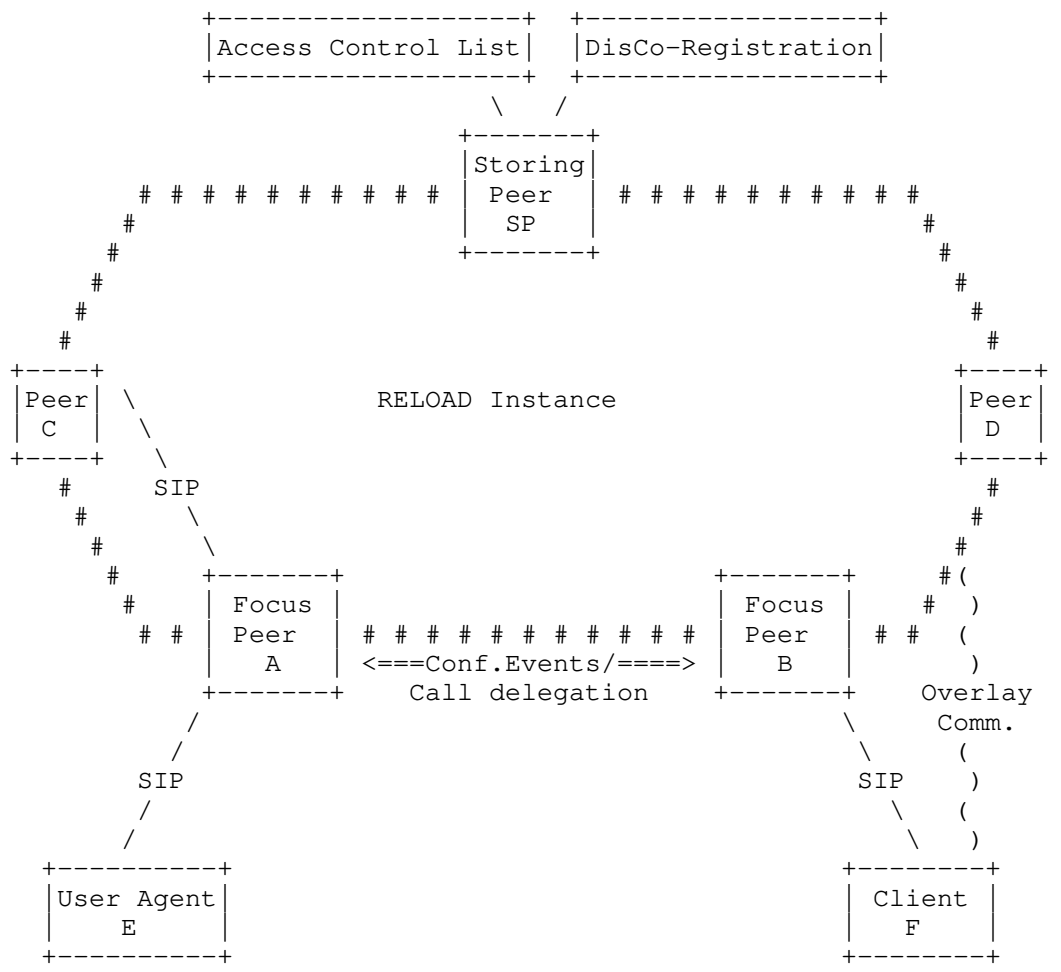


Figure 1: Reference Scenario: Focus peers A,B maintain a distributed conference

3.2. Initiating a Distributed Conference

To create a conference the initiating user agent announces itself as a focus for the conference. It stores its own contact information (Node-ID) as a DisCo-Registration Kind (cf. Figure 2) in the RELOAD overlay. The hashed conference URI is used as the Resource-ID. This Resource will later contain the contact IDs of all potential focus peers including optional topological descriptors.

3.3. Joining a Conference

A RELOAD-aware node (cf. Bob in Figure 2) intending to join an existing conference requests the list of potential focus peers stored in the DisCo-Registration under the conference's Resource-ID. The node selects any of the focus peers (e.g., Alice) and establishes a connection using AppAttach/ICE [RFC5245]. This transport is then used to send an INVITE to the conference applying the chosen focus as the contact. The selection of the focus peer can optionally be based on proximity information if available.

A conference member proposes itself as a focus for subsequent participants by adding its Node-ID to the DisCo-Registration stored under the conference URI in the RELOAD overlay. The decision whether a peer announces as a focus incorporates bandwidth, power, and other constraints, but details are beyond the scope of this document.

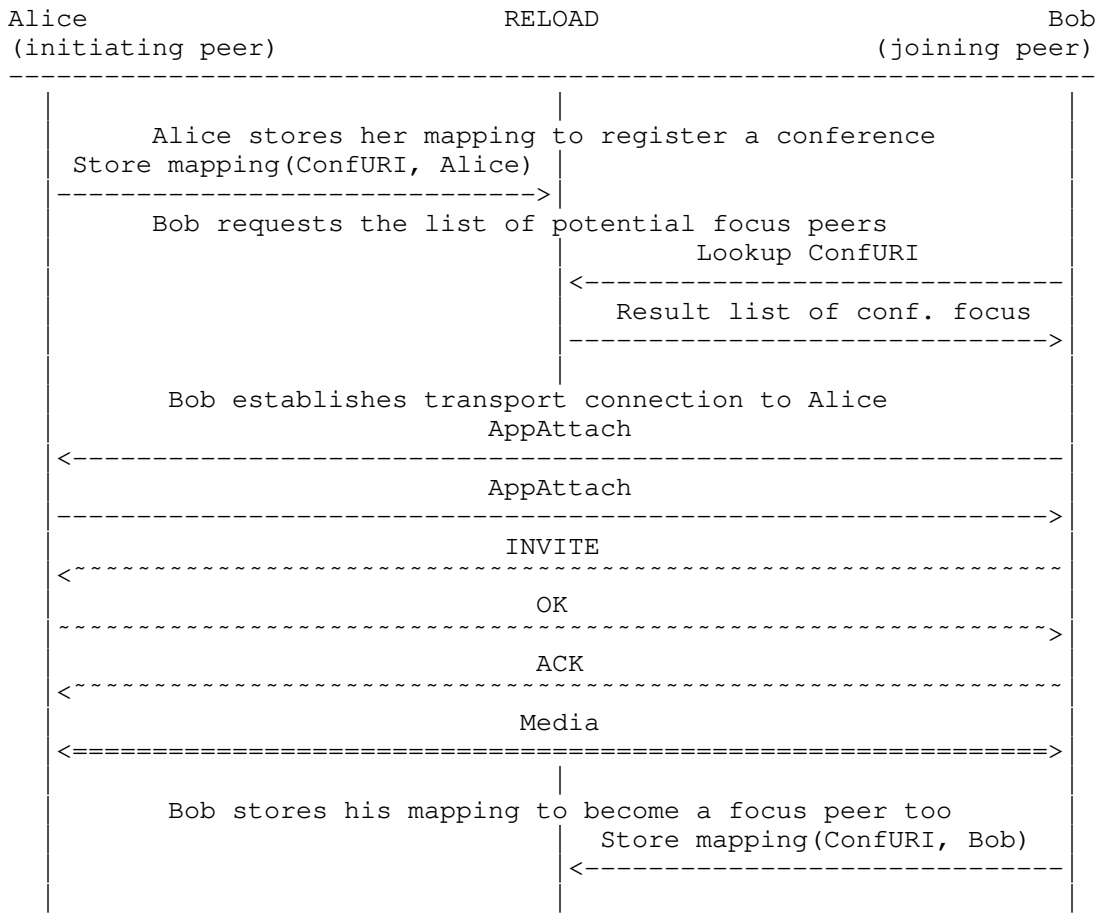


Figure 2: DisCo Usage generic Call Flow

3.4. Conference State Synchronization

Each focus of a conference maintains signaling connections to its related participants independently from other conference controllers. This distributed conference design effects that the entire SIP conference state is jointly held by all focus peers. In DisCo, state synchronization is based on a SIP specific event notifications mechanism [RFC3265].

Each focus peer maintains its view of the entire conference state by subscribing to the other focus peers' XML event package for distributed conferences. This is based on the event package for conference state (cf. [RFC4575]). Details are defined in this document in Section 5. Receivers of event notifications update their

local conference state document to represent a valid view of current total conference state.

The event notification package for distributed conferences enables focus peers to synchronize the entire conference state. The event package defines additional XML elements and complex types (see Section 8 for more details), which describe the responsibilities of any focus peer in the conference. By providing a global view each focus peer is enabled to perform additional load balancing operations and enhances the robustness against departures of focus peers.

3.5. Call delegation

Call delegation (see Section 6.1) is a feature used to transfer an incoming participation request to another focus peer. It can be applied to prevent overloading of focus peers. Call delegation is realized through SIP REFER requests, which carry signaling and session description information of the caller to be transferred. This feature is achieved transparently for the transferred user agent by using a source routing mechanism at SIP dialog establishment. Descriptions of overload detection are beyond the scope of this document.

3.6. Resilience

A focus peer can decide to leave the conference or may ungracefully fail. In a traditional conferencing scenario, loss of the conference controller or the media distributor would cause a complete failure of the multiparty conversation. Distributed conferencing uses the redundancy provided by multiple focus peers to reconfigure a current multiparty conversation. Participants that loose their entry point to the conference re-invite themselves via the remaining focus peers or will be re-invited by the latter. This option is based on the conference state and call delegation functions.

3.7. Topology Awareness

DisCo supports the optimization of the conference topology in respect of the underlying network using topological descriptors. An extension for the RELOAD XML configuration document is defined in Section 4.8 to support landmarking approaches. Each peer intending to create or participate in a distributed conference MAY determine a topological descriptor that describes its relative position in the network. Focus peers store these coordinate values in an additional data field in the DisCo-Registration data structure. This enables peers joining the conference to select the closest focus with respect to its coordinate values.

4. RELOAD Usage for Distributed Conference Control

4.1. Shared Resource DisCo-Registration

A distributed conference is a cooperative service of several individual devices that use a common identifier. To enable a mapping from one conference identifier to multiple focus peers, this document defines the new Kind data structure DisCo-Registration. The DisCo Kind uses the definitions for Shared Resources [I-D.knauf-p2psip-share] to allow a jointly maintenance by multiple focus peers. Hence, write permission to a DisCo-registration is shared by the conference creator with all authorized focus peers.

DisCo complies with the following requirements for Shared Resources:

Isolated Data Storage: DisCo uses the dictionary data model. Each dictionary key is the Node-ID of the certificate that will be used to sign the stored data

Resource_name field: Each DisCo-Registration data provides an opaque <0..2^16-1> initial field in the Kind data structure. It contains the conference URI of the registered DisCo-record.

User_name field: Each DisCo-Registration data provides an opaque <0..2^16-1> secondary field in the Kind data structure. It contains the username of the originator of a DisCo record.

4.2. Kind Data Structure

Each DisCo-Registration data structure stores a mapping of a conference identifier to one or multiple focus peers that cooperatively control the conference. Additionally, each DisCo-Registration provides the coordinate value, which indicates the relative network position of the focus peers.

The data structure uses the RELOAD dictionary type. The dictionary key MUST be the Node-ID of the focus peer that is associated with the dictionary entry. This allows a focus peer to update only its own mapping. The DisCo data structure of type DisCoRegistration is constructed as follows:

```
struct {
  opaque resource_name<0..2^16-1>;
  opaque user_name<0..2^16-1>;
  opaque coordinate<0..2^16-1>;
  NodeId node_id;
} DisCoRegistrationData;

struct {
  uint16 length;
  DisCoRegistrationData data;
} DisCoRegistration;
```

The DisCoRegistration structure is composed of the following values:

length: This field denotes the length of the DisCoRegistrationData.

data: This field denotes the distributed conference registration data.

The DisCoRegistrationData structure is composed of the following values:

resource_name: This field contains the conference URI and meets the requirement for the USER-CHAIN-ACL access policy defined in [I-D.knauf-p2psip-share]

user_name: This field contains the user name of the focus peer that stores the DisCoRegistration data. It meets the requirements for USER-CHAIN-ACL access policy defined in [I-D.knauf-p2psip-share].

coordinate: This field contains a topological descriptor that indicates the relative position of the peer in the network. To support different algorithms the coordinate field is represented as an opaque string.

node_id: This field contains the Node-ID of the peer storing the DisCoRegistration and is used to contact the peer when utilizing its service as a focus.

4.3. Variable Conference Identifier

DisCo-Registrations can be stored based on a variable Resource Name. However, a conference identifier set by a user MUST follow the requirements for Kinds using variable Resource Names as defined in the ShaRe Usage [I-D.knauf-p2psip-share].

4.4. Conference Creation

The registration of a distributed conference includes the storage of the following two Kinds (see Figure 3).

DisCo-Registration Kind: contains the conference identifier and the optional coordinate value. If used, the coordinate value MAY be determined previously to the conference registration. The Resource Name and hence the Resource-ID is defined by the hash over the desired conference identifier (using the hash algorithm of the overlay).

Access Control List Kind: contains the conference participants that are allowed to register as focus peers for a conference (see [I-D.knauf-p2psip-share]). Its Resource Name/ID is equal to those of the corresponding DisCo-Registration.

Preliminary to storage of DisCo-Registration and Access Control List (ACL) Kinds the conference creator SHOULD perform a RELOAD StatReq to verify that no former conference is present. If neither a DisCo-Registration nor an associated ACL exist, the conference creator stores a DisCo-Registration with a valid conference identifier (see Section 4.3) and an ACL referring to the DisCo-Registration Kind-ID.

If DisCo registrations and ACL Kinds from previous conferences are still existing there are two options. First, if conference creator is aware of the indexes from previous ACL Kinds, it refreshes the root item of this ACL and stores its registration as focus peer as DisCo-Registration Kind. Second, If the creator is unaware of indexes, it fetches all Access List Kinds to determine the index of the root item.

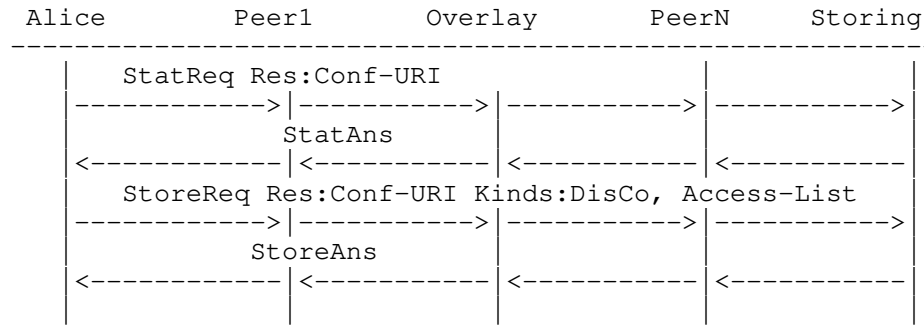


Figure 3: Initial creation of a Distributed Conference

Optionally the conference initiator (or any active focus) MAY store an additional RELOAD SIP-Registration in the overlay [I-D.ietf-p2psip-sip] or even at a standard SIP registrar [RFC3261] under a URI for which it has write permission. This allows DisCo-unaware or even legacy SIP user agents to participate in the conference. Those registrations SHOULD always point to a currently active focus, who is prepared to accept legacy user agents. The user agent who initially performed the registrations is responsible for updating them appropriately. How authorization has been obtained to perform those registration is out of scope of this document.

The lifetime of a distributed conference is not necessarily limited by the participation time of its creator. As long as the root item of an Access Control List to a DisCo-Registration is not expired, Authorized Peers are allowed to further provide a conferencing service and even store new focus registrations.

4.5. Advertising Focus Ability

All participants of a distributed conference MAY potentially become a focus peer for their conference. This depends on its capacities such as sufficient processing power (CPU, Memory) for the desired media type, the quality of the network connectivity, and the conference policy. Information about network connectivity with respect to NAT or restricted firewalls can be obtained via ICE [RFC5245] connectivity checks. If a peer is behind a NAT, it SHOULD allow for incoming connections via AppAttach/ICE. Peers that can only accept connections with the support of TURN should not act as a focus. Nevertheless, under special circumstances it might be reasonable to locate a focus peer behind such a NAT (e.g., within a companies network infrastructure).

If a participant is a candidate to become a focus for the conference, it stores its Node-ID and optional coordinate value into the DisCo data structure. An entry in the corresponding ACL [I-D.knauf-p2psip-share] must be present to allow this peer to write the DisCo-Registration resource. By storing the mapping into the data structure a participant becomes a potential focus.

4.6. Determining Coordinates

Each RELOAD peer within the context of a distributed conference MAY be aware of its relative position in the network topology. To constuct a topology-aware conference, the DisCo Usage provides the coordinate value within the DisCo-Registration data structure. Focus peers store their relative network position together with the announcement as conference focus. Joining peers that are able to determine their coordinates may then select a focus peer whose

relative position is in its vicinity (see Section 4.7).

Some algorithms determine topology information by measuring Round-Trip Times (RTT) towards a set of hosts serving as landmarks (e.g., [landmarks-infocomm02]). To support such algorithms this document describes an extension to the RELOAD XML configuration document that allows to configure the set of landmark hosts peer must use for position estimation (see Section 4.8). Once a focus peer has registered its mapping in the DisCo data structure, it also stores the according coordinates in the same mapping. These <Node-ID,coordinates> vectors are used by peers joining the conference to select the focus peer that is relatively closest to itself.

Because topology-awareness can be obtained by many different approaches a concrete algorithms is out of scope of this document.

4.7. Proximity-aware Conference Participation

The participation procedure to a distributed conference is composed of three operation.

1. Resolution of the conference identifier.

2. Establishment of of transport connection.

3. SIP signaling to join a conference.

Figure 4 and the following specifications give a more detailed view on the joining procedure.

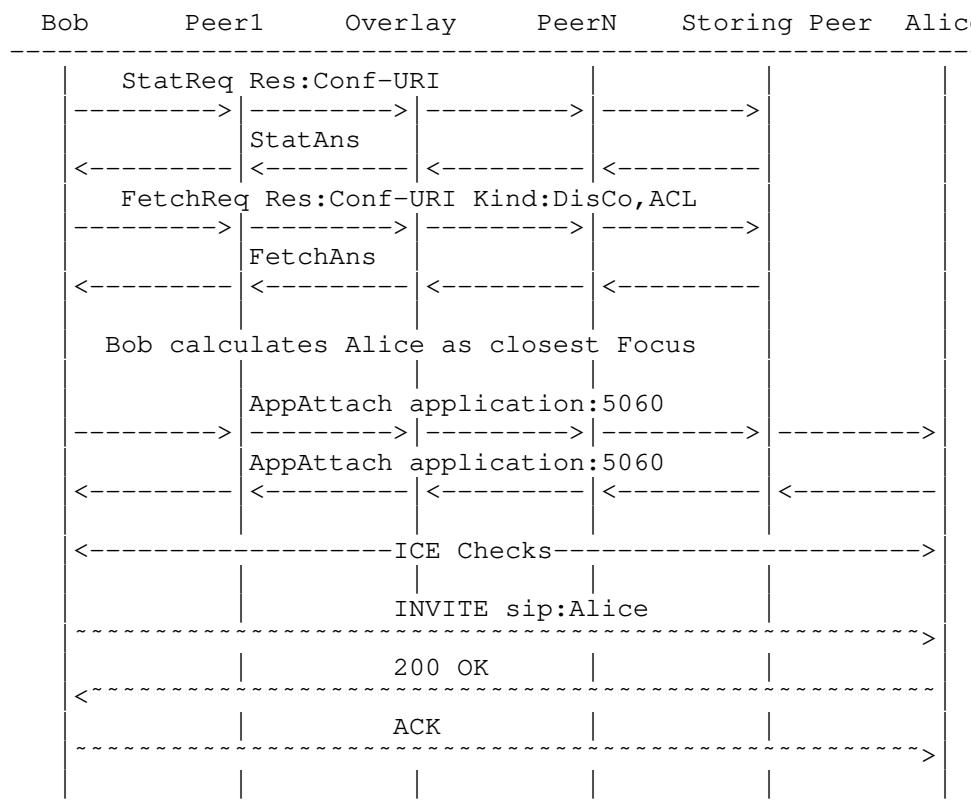


Figure 4: Participation of a Distributed Conference

1. The joining peer MAY determine its own coordinate value (if used).

2. The joining peer sends a StatReq message to obtain all indexes of the Access Control List (ACL) Kinds stored.

3. The joining peer sends a FetchReq message for the DisCo and ACL Kind to the Resource-ID of the conference URI. The FetchReq SHOULD NOT include any specific dictionary keys, but SHOULD fetch for those array ranges previously determined the StatReq. With the ACL items, the joining peer is able to verify whether DisCo-Registrations are stored by authorized focus peers (see [I-D.knauf-p2psip-share]).

4. Using the retrieved coordinates values of the DisCo-Registrations, the joining peer MAY calculate which of than opaque <0..2^16-1> initial field in the Kind data structure focus peers is the relatively closest to itself.

5. The joining peer then establishes a transport using RELOADs AppAttach, respectively, ICE procedures to create a transport to the chosen focus peer.

6. Finally, the established transport is used to create a SIP dialog from the joining peer to the focus peers.

The SIP INVITE request MAY contain the joining peer's topological descriptor as a URI-parameter called 'coord' in the contact-header in base64 encoded form [RFC4648]. An example contact URI is "sip:alice@example.com;coord=PEknbSBhIHRvcG9sb2dpY2FsIGRlc2NyaXB0b3I+". When the called focus is full and needs to delegate the call it uses the contents of the 'coord'-parameter. It determines the next available focus closest to the calling peer (Section 4.6) using the received descriptor and the other focuses' descriptors from the conference state synchronization document and delegates the call accordingly (see Section 6.1).

A conference focus MAY allow the joining peer to also become a focus (depending on the conference policy see Section 6.2). Therefore, it stores a new ACL Kind that delegates write permission for the DisCo-Registration to the new participant. It sets the 'user_name' field in the ACL Kind to its own user name and sets the 'to_user' field to the user name of the joining peer. If no other conference policy is defined, the focus peer MAY set the allow_delegation flag to true. For further details about the trust delegation using the ACL Kind see [I-D.knauf-p2psip-share].

4.8. Configuration Document Extension

This section defines an additional parameter for the <configuration> element that extends the RELOAD XML configuration document. The proposed <landmarks> element allows RELOAD provider to publish a set of accessible and reliable hosts that SHOULD be used if RELOAD peers use landmarking algorithms to determine relative position in the network topology.

The <landmarks> element serves as container for the <landmark-host> sub-elements, each representing a single host that serves as a landmark. The <landmark-host> uses the following attributes:

address: The IP address (IPv4 or IPv6) of the landmark host.

port: The port on which the landmark host responds for distance estimation.

More than one landmark hosts SHOULD be present in the configuration document.

5. Conference State Synchronization

The global knowledge about signaling and media relations among the conference participants and focus peers defines the global state of a distributed conference. It is composed of the union of every local state at the focus peers. To enable focus peers to provide conference control operations that modify and/or require the global state of a conference, this document defines a mechanism for inter-focus state synchronization. It is based on mutual subscriptions for an Event Package for Distributed Conferences and allows to preserve a coherent knowledge of the global conference state. This XML based event package named 'distributed-conference' MUST be supported by each RELOAD peer that is registered with a DisCo-Registration. Participants of a distributed conference MAY support it. To provide backward compatibility to conference members that do not support the distributed-conference event package, this document defines a translation to the Event Package for Conference State [RFC4575].

5.1. Event Package Overview

The 'distributed-conference' event package is designed to convey information about roles and relations of the conference participants. Conference controllers obtain the global state of the conference and use this information for load balancing or conference restructuring mechanisms in case of a focus failure. Figure Figure 5 gives a general overview of the document hierarchy.

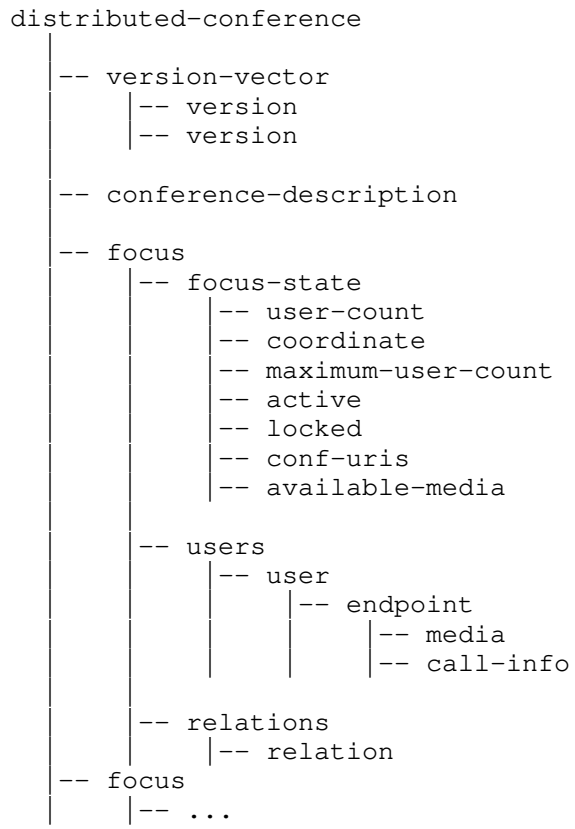


Figure 5: Overview of the event package for distributed conferences

The document structure is designed to allow concurrent change events at several focus peers. To prevent race conditions each focus peer has exclusive writing permission to the 'focus' sub element that describes itself. It is achieved by a unique mapping from a focus peer to its XML element using the 'Element Keys' mechanisms for partial notification [RFC4575](sections 4.4-5.). A focus peer is only allowed to update or change that <focus> sub element, whose 'entity' Element Key contains its RELOAD user name. This restriction also applies to the child elements of the 'version-vector' element. Each 'version' number belongs to a specific focus peer maintaining the version number.

The local state of a focus peer is described within a 'focus' element. It provides general information about a focus peer and its signaling and media relations to participants and focus peers. The Conference participants are aggregated within 'users' elements, respectively, 'user' sub elements.

General information about the conference as a whole, is provided within a 'conference-description' element. In contrast to the 'focus' and 'version-vector' elements, conference description is not meant for concurrent updating. Instead, it provides static conference descriptions that rarely change during a multiparty session.

More detailed descriptions about the event package and its elements are provided in the following sections. The complete XML schema definition is given in Section 8.

5.2. <distributed-conference>

The <distributed-conference> element is the root of a distributed conference XML document. It uses the following attributes:

entity: This attribute contains the conference URI that identifies the distributed conference. A SIP SUBSCRIBE request addressed to this URI initiates an subscribe/notify relation between participants and their related focus peer.

state: This attribute indicates whether the content of a distributed conference document is a 'full', 'partial' or 'deleted' information. It is in accordance with [RFC4575] to enable the partial notification mechanism.

The <distributed-conference> child elements are <vector-version>, <conference-description> and the <focus> elements. An event notification of state 'full' MUST include all these elements. An event notification of state 'partial' MUST contain at least <version-vector> and its sub elements.

5.3. <version-vector>/<version>

The Event Package for Distributed Conferences uses the <version-vector> and its <version> sub elements to enable a coherent versioning scheme. It is based on vector clocks as defined in [timestamps-acsc88]. Each <version> element contains a unsigned integer that describes the state of a specific focus peer and contains the following attributes:

entity: This attribute contains the user name of the focus peer whose local version number is described by this element.

node-id: This attribute contains the Node-ID of the focus peer.

Whenever the local status of a focus peer changes (e.g. a new participant arrived) the version number of the corresponding

<version> element MUST be incremented by one. Each change in the local state also triggers a new event notification containing the entire <version-vector> and the changes contained in a <focus> element.

The recipient of a change event needs to update it local XML document. If a received <version> number is higher that the local, it updates the <version> element and its associated <focus> element with the retrieved elements. All other elements remain constant.

If the length or contents of the retrieved <version-vector> is different to the local copy it indicates a incoherent knowledge about the entire conference state. If the retrieved <version-vector> contains any unknown focus peers and any local version numbers for the known focus peers is lower, the receiver SHOULD request a 'full' XML notification.

If any local <version> number is retarded more than one, the receiver SHOULD request a 'full' event notification from the sender. The full state notification updates all <focus> elements whose corresponding <version> element is out of date.

5.4. <conference-description>

The <conference-description> element provides general information and links to auxiliary services for the conference. The following sub elements provide human-readable text descriptions about the conference:

<display-text>: Contains a short textual description about the conference

<subject>: Contains the subject of the conference

<free-text>: Contains a longer textual description about the conference

<keywords>: Contains a list of keywords that match the conference topic. The XML schema definition and semantic is imported from the 'conference-info' event package [RFC4575].

The <service-uris> sub element enables focus peers to advertise auxiliary services for the conference. The XML schema definition and semantic is imported from the 'conference-info' event package [RFC4575].

The <conference-description> element uses the 'state' Element Key to enable the partial notification mechanism.

5.5. <focus>

Each <focus> element describes a focus peer actively controlling the conference. It provides general information about a focus peer (e.g., display-text, languages, etc.), contains conference specific information about the state of a focus peer (user-count, available media types, etc.) and announces signaling and media information about the maintained participants. Additionally, it describes signaling or media relations to further focus peers.

The <focus> element uses the following attributes:

entity: This attribute contains the user name of the RELOAD peer acting as focus peer. It uniquely identifies the focus peer that is allowed to update or change all sub elements of <focus>. All other focus peers SHOULD NOT update or change sub elements of this <focus> element. A SUBSCRIBE request addressed to the user name initiates a conference state synchronization with the focus peer.

Node-ID This attributed contains the Node-ID of the peer acting as conference focus

state: In accordance to [RFC4575], this attribute indicates whether the content of the <focus> element is a 'full', 'partial' or 'deleted' information. A 'partial' notification contains at maximum a single <focus> element.

The following sub elements of <focus> provide general information about a focus peer:

<display-text>: Contains a short text description of the user acting as focus peer.

<associated-aors>: This element contains additional URIs that are associated with this user.

<roles:> This element MAY contain human-readable text descriptions about the roles of the user in the conference.

<languages>: This element contains a list of tokens, each describing a language understood by the user.

The XML schema definition and semantic for <associated-aors>, <roles> and <languages> are imported from the 'conference-info' event package [RFC4575]

Following, a detailed description of the remaining sub elements.

5.5.1. <focus-state>

The <focus-state> element aggregates a set of conference specific information about the RELOAD user acting as focus peer. The following attribute is defined for the <focus-state> element:

status: This attribute indicates whether the content of the <focus-state> element is a 'full', 'partial' or 'deleted' information.

The <focus-state> element has the following sub elements:

<user-count>: This element contains the number of participants that are connected to the conference via this focus peer at a certain moment.

<coordinate> This element contains the coordinate value Section 4.2 of the focus peer

<maximum-user-count>: This number indicates a threshold of participants a focus peer is able to serve. This value might change during a conference, depending on the focus peers current load.

<conf-uris>: This element MAY contain other conference URIs in order to access the conference via different signaling means. The XML schema definition and semantic is imported from [RFC4575].

<available-media>: This element is imports the <conference-media-type> type XML scheme definitions from [RFC4575]. It allows a focus peer to list its available media streams.

<active>: This boolean element indicates whether a focus peer is currently active. Conference participation requests or a call delegation request SHOULD succeed.

<locked>: In contrast to <active>, this element indicates that a focus peer is not willing to accept anymore participation or call delegation request.

5.5.2. <users>/<user>

The <users>, respectively, each <user> element describes a single participant that is maintained by the focus peer described by the parent <focus> element. The <users> element XML schema definition and its semantic is imported from the 'conference-info' event package [RFC4575].

5.5.3. <relations>/<relation>

The <relations> element serves as container for <relation> elements, each describing a specific connection to another focus peer. The parent element <relations> uses the 'state' attribute to enable the partial notification mechanism. For the <relation> element the following attributes are defined:

entity: This attribute contains the user name of the remote focus.

node-id This attribute contains the Node-ID of the remote focus peer.

The content of each <relation> is a comma separated string that describes the tuple <CONNECTION-TYPE:IDENTIFIER>. The CONNECTION-TYPE is a string token describing the type of connection (e.g. media, signaling, etc.) and the IDENTIFIER contains a variable connection identifier. It is a generic method to announce any kind of connection to a remote focus. This specification defines following tuples:

media:<label>: This tuple identifies a single media stream. The 'label' variable contains the SDP "label" attribute. (see [RFC4574]).

sync:<call-id>: This tuple indicates a subscription for the Event Package for Distributed Conferences. The 'call-id' variable contains the call-id of the SIP subscription dialog.

5.6. Distribution of Change Events

Each focus peer in a distributed conference must be able to retrieve all change events from all other focus peers. A simple approach would be to inter-connect each focus with all other focus in a full meshed topology. To avoid a full mesh, this document describes a 'mutual' subscription scheme that constructs a spanning tree topology among the focus peers.

A conference participant that becomes a focus peer MUST send a SIP SUBSCRIBE to request the Event Package for Distributed Conferences to its own focus peer. The subscription request is addressed to user name of the active focus peer. The latter interprets this subscription as a request for conference state synchronization. The corresponding NOTIFY message contains a 'full' distributed-conference state XML document (see section Section Section 5.1).

The subscribed focus peer in turn subscribes the upcoming focus peer for the distributed conference event package. The corresponding

NOTIFY message carries a 'partial' conference state XML document. It contains the received <version-vector> including a new <version> element for itself and a new <focus> element that describes its local state (see Section 5.5).

Resulting by this subscription scheme, each focus peer has at least one subscription to obtain updates for the conference state and is a notifier for change events originated itself. In a incrementally increasing conference, the 1st and 2nd focus peer have a mutual subscription for conference change events. A 3rd focus could have a mutual subscription with the 1st focus, a 4th focus to the 2nd focus and so forth. The result is a spanning tree topology among the focus peers in which each focus peer is a possible root for distribution tree for conference change events.

However, the fact that event notifications need to traverse one or more intermediate focus peers until conference-wide delivery, demands a forwarding mechanism for change events. On receiving a change event, a notified focus validates based on the <version-vector> whether the incoming state event is not a duplicate to previews notifications. If its not a duplicate, the received change event triggers a new event notification at the receiver of the change event. It notifies all its subscribers with excepting the sender of the event notification. In this way, the change event will be 'flooded' among the focus peer of a conference.

5.7. Translation to Conference-Info Event Package

The Event Package for Distributed Conferences imports several XML element definitions of the Event Package for Conference State [RFC4575]. This is caused by two reasons. First, the semantic of these elements are fitting the demands to describe the global state of a distributed conference and, second, it facilitates a re-translation to [RFC4575] to enable a backward compatibility to DisCo-unaware clients. Therefore, each focus peer MAY provide a separate [RFC4575] conform event notification service to its connected participants.

The following sections describe the translation to the Event Package for Conference State [RFC4575] by defining translation rules for the root element and its direct sub elements. For a better understanding, the following sections use a notation ci.<ELEMENT> to refer to a sub element of the conference-info element, and disco.<ELEMENT> to refer to an element of the distributed-conference event package.

5.7.1. <conference-info>

The root element of Event Package for Conference State uses the attributes 'entity', 'state' and 'version' and is the counterpart of the <distributed-conference> root element in the DisCo Event Package. The former two attributes 'entity' and 'state' are equal in both root elements and can be seamlessly translated.

According to [RFC4575], the 'version' attribute SHOULD be incremented by one at any time a new notification being sent to a subscriber. Hence, in DisCo the 'version' attributed increments with each change event that originated by focus peer and each reception of a change events of remote focus peer.

5.7.2. <conference-description>

The <conference-description> element exists in both event packages, conference-info and distributed-conference. Thus, the following elements are seamlessly translatable: <keywords>, <display-text>, <subject>, <free-text> and <service-uris>.

The sub elements <conf-uris>, <maximum-user-count> and <available-media> in conference-info have there counterparts below the \focus\focus-state element of the distributed-conference event package. Each describes a local state of a focus peer in the conference. Hence, the intersection of every disco.<conf-uris>, disco.<available-media> and the sum over each disco.<maximum-user-count> element of each disco.<focus> element in distributed-conference, specifies the content of the corresponding conference-info elements.

5.7.3. <host-info>

According to [RFC4575] the ci.<host-info> element contains information about the entity hosting the conference. For participants in a distributed conference, the hosting entity is their focus peer. Thus, the ci.<host-info> element contains information about a focus peer.

5.7.4. <conference-state>

The ci.<conference-state> element allows subscribers obtain information about overall state of a conference. Its sub elements ci.<user-count>, ci.<active> and ci.<locked> are reused as sub elements of \focus\focus-state to describe the local state of a focus peer in a distributed conference. The translation rules from the distributed-conference to the conference-info event package are the following:

<user-count>: The sum over each value of the disco.<user-count> element defines the corresponding ci.<user-count>.

<active>: The boolean ci.<active> element is the logical concatenation over all disco.<active> elements by an OR-operator.

<locked> The boolean ci.<locked> element is the logical concatenation over all disco.<locked> elements by an AND-operator.

5.7.5. <users>/<user>

The distributed-conference event package imports the definitions of the ci.<users> and ci.<user> elements under a parent disco.<focus> element for each focus peer in a conference. Thus, the aggregation over all disco.<users> elements specifies the content of the corresponding ci.<users> element.

5.7.6. <sidebars-by-ref>/<sidebars-by-value>

In accordance to [RFC4575], if a participant is connected to a sidebar, its responsible focus peer creates a new <user> by referencing to the corresponding sidebar conference.

6. Distributed Conference Control with SIP

Distributed conference control with SIP defined in this document refers to multiparty conversation in a tightly coupled model that is controlled by several independent entities. It enables a resilient conferencing service for P2P scenarios and provides mechanisms for load-balancing among the focus peers. This section describes additional control operations for distributed conferences with SIP.

6.1. Call delegation

Distributed conference control enables load-balancing by a mechanism for call delegation. Call delegations are performed by focus peers that are running out of capacities to serve more participants. Incoming participation requests are then transferred to other established focus peer or conference participants that are registered as potential focus peers in the overlay. Call delegations use SIP REFER requests [RFC3515] that contain additional session information and are achieved transparently to the transferred party.

A focus peer initiates a call delegation by sending SIP REFER request containing the URI of the participant in the Refer-To header field. Additionally, the focus peer appends the following parameter to the URI of the participant:

call-id: Contains the call-ID of the initial SIP dialog between the referred participant and the referring focus peer.

sess-id: Contains the 'session identifier' value of the original SDP 'o=' field of the original offer/answer process between referred participant and referring focus peer.

If the recipient accepts the REFER request, it generates a re-INVITE towards the referred party and sets the SIP call-id header and the SDP 'session-identifier' field in the SDP offer, according to the URI parameter values of the initial REFER request. The From header field and contact header are set to the conference URI with setting the 'isfocus' tag to contact header. This identifies the peer as a focus to the conference and identifies this re-INVITE as a request of the SIP dialog between the party and the conference. To ensure that further signaling messages will be routed correctly, the new focus adds a Record-Route header field that contains its contact information (URI, IP-address,..).

An example call flow for call delegation is shown in Figure 6.

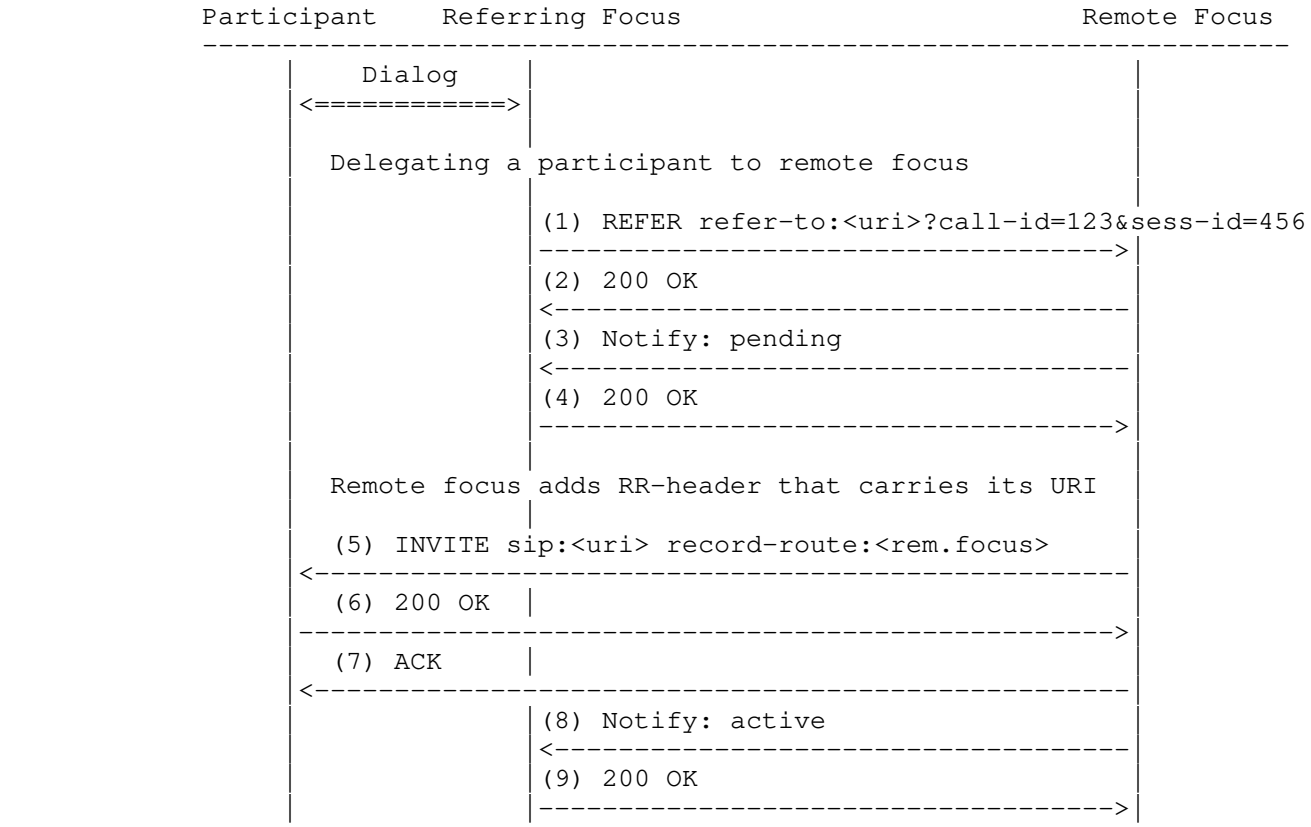


Figure 6: Delegating a participant with SIP REFER

Note, subscriptions for the event packages 'distributed-conference' and 'conference-info' are in scope of a specific focus peer and its connected participants. Hence, after a successful call delegation, the referring focus peer SHOULD terminate any subscription to the referred participant. The notifier SHOULD include a reason parameter "deactivated" to indicate a migration of the subscription as defined in [RFC3265]. The new SUBSCRIBE request by the party MUST be sent via the SIP dialog to the conference.

6.2. Conference Access

A conference policy defines who is allowed to participate in a multimedia conference. In many cases, a group conversation can be an open discussion free to participate, while in other occasions a closed privacy of a multiparty session is demanded. In distributed conferences, it is also an issue which of the conference participants is allowed to become a controller of the multiparty session.

Thus it must be decided whether:

- A peer is allowed to participate in a conference
- A peer is allowed to become a focus of the conference

Standard SIP authentication mechanisms can be used to authenticate and accordingly authorize joining participants.

6.3. Media Negotiation and Distribution

This section describes a basic scheme for media negotiation and distribution, which is done in an ad-hoc fashion. Each focus peer forwards all media streams it receives from the conference to all connected peers it is responsible for and similarly all streams from its peers to its responsible focus. This results in the media stream naturally following the SIP signalling paths. Implementations MAY choose to use a more sophisticated scheme, e.g. employing cross connections between different sub-trees of the conference, but MUST take measures to prevent loops in media routing.

When a new peer has been attached to a focus, new media steams may be available to the focus, which need to be forwarded to the conference. To accomplish this, the new media streams need to be signalled to the other participants. This is usually done by sending a SIP re-INVITE and modifying the media sessions, adding the new streams to the SDP. This renegotiation can be costly since it needs to be propagated through the whole conference. Also, distributing all media streams separately to all participants can be quite bandwidth intensive. Both problems can partially be mitigated by focus peers performing mixing of media streams, thus trading bandwidth and signalling overheads for computational load on focus peers.

6.3.1. Offer/Answer

A peer joining a conference negotiates media types and media parameters with its designated focus using the standard SDP offer/ answer protocol [RFC3264]. The focus SHOULD offer all existing media streams that it receives from the conference.

A new participant does not necessarily know about all media streams present in a conference beforehand, and thus some of the media streams might not be included in the initial SDP offer sent by the joining peer. An SDP answer sent by the corresponding focus MUST NOT contain any media streams not matching an offer (cf. [RFC3264] Section 6). A joining peer which is aware of the distributed nature of the conference, SHOULD NOT send an SDP offer in the initial INVITE message, but instead send an empty INVITE to which the focus replies

with an OK, containing the offer. This prevents the need for a second offer/answer-dialog to modify the session. But for compatibility the normal behavior with the INVITE containing the offer MUST be supported.

For new media streams (e.g. those sent by the new participant) the focus SHOULD only offer media types and codecs which are already used in the conference and which will probably be accepted when forwarded to neighboring peers, unless the focus is prepared to do transcoding and/or mixing of the received streams.

A focus has two options when distributing media streams from a new participant to the conference. The focus can either mix the new streams into his own, thus averting the need to modify the already established media sessions with neighboring peers or in case the focus is not willing or able to do mixing of the media streams, he SHOULD modify the media sessions with all attached peers by sending a re-INVITE, adding the new media streams coming from the newly joined participant to the SDP. This SHOULD subsequently be done by all other focus peers upon receiving the new stream, resulting in the stream being distributed across the conference.

6.4. Restructuring a Conference

Distributed conference control provides the possibility to delegate calls to remote focus peers. This feature is used to restructure a conference in case of departure of a focus peer. Following, this section presents restructuring schemes for graceful and unexpected leaves of conference focus peers.

6.4.1. On Graceful Leave

In a graceful case, the leaving focus peer (LP) accomplishes the following procedure:

- LP deletes its mapping in the DisCo-Registration by storing the "non-existing" value as described in the RELOAD base document [I-D.ietf-p2psip-base]. Afterwards, it fetches the lasted version of the DisCo-Registration to obtain all potential focus peers.

- LP calculates for all its participants the closest focus among all active and potential focus peer using the algorithm described in Section 4.6. LP then delegates all participants to those focus peers.

- LP then announces its leave by sending a NOTIFY to all its subscribers for the extended conference event package, setting its <focus> state to ’deleted’. Thereafter, it ends its own SIP

conference dialog by sending by to its related focus peer.

Since the state synchronization topology in a distributed conference is commonly arranged in a spanning tree, a leave of a focus peer effects a gab in the tree structure. Those focus peers which had the leaving focus peer as their parent, are supposed to reconnect to the synchronization graph by subscribing the parent focus of the leaving peer.

6.4.2. On Unexpected Leave

If an unexpected leave is detected by a participant (e.g. missing signaling and/or media packets) it MUST repeat the joining procedure as described in Section 4.7.

7. DisCo Kind Definition

This section formally defines the DisCo kind.

Name
DISCO-REGISTRATION

Kind IDs
The Resource name DISCO-REGISTRATION Kind-ID is the AoR of the conference. The data stored is the DisCoRegistrationData, that contains the Node-ID of a peer acting as a focus for the conference and optionally a coordinates value describing a peer's relative network position.

Data Model
The data model for the DISCO-REGISTRATION Kind-ID is dictionary. The dictionary key is the Node-ID of the peer action as focus.

Access Control
USER-CHAIN-ACL

The data stored for the Kind-ID DISCO-REGISTRATION is of type DisCoRegistration. It contains a "coordinates" value, that describes the peers relative network position and the Node-ID of the registered focus peer.

8. XML Schema

The XML schema for the event package for distributed conferences is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ci="urn:ietf:params:xml:ns:conference-info"
     xmlns="urn:ietf:params:xml:ns:distributed-conference"
     targetNamespace="urn:ietf:params:xml:ns:distributed-conference"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">
<!--
    This imports the definitions in conference-info
-->
<xs:import namespace="urn:ietf:params:xml:ns:conference-info"
        schemaLocation="http://www.iana.org/assignments/xml-registry/
                        schema/conference-info.xsd"/>
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
     schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
  <!--
   A DISTRIBUTED CONFERENCE ELEMENT
-->
<xs:element name="distributed-conference"
            type="distributed-conference-type"/>
  <!--
    DISTRIBUTED CONFERENCE TYPE
-->
  <xs:complexType name="distributed-conference-type">
    <xs:sequence>
      <xs:element name="version-vector"
            type="version-vector-type" minOccurs="1"/>
      <xs:element name="conference-description"
      type="conference-description-type"
      minOccurs="0" maxOccurs="1"/>
      <xs:element name="focus"
      type="focus-type"
      minOccurs="0"
      maxOccurs="unbounded"/>
       <xs:any namespace="##other" processContents="lax"/>
     </xs:sequence>
     <xs:attribute name="state" type="ci:state-type"/>
     <xs:attribute name="entity" type="xs:anyURI"/>
     <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
<!--
    VERSION VECTOR TYPE
-->
  <xs:complexType name="version-vector-type">
```

```
      <xs:sequence>
        <xs:element name="version"
                    type="version-type"
                    minOccurs="1"
                    maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
      CONFERENCE DESCRIPTION TYPE
  -->
    <xs:complexType name="conference-description-type">
      <xs:sequence>
        <xs:element name="display-text"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="subject" type="xs:string" minOccurs="0"/>
        <xs:element name="free" type="xs:string" minOccurs="0"/>
        <xs:element name="keywords"
                    type="ci:keywords-type" minOccurs="0"/>
        <xs:element name="service-uris"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
      FOCUS TYPE
  -->
    <xs:complexType name="focus-type">
      <xs:sequence>
        <xs:element name="display-text"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="associated-aors"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:element name="roles"
                    type="ci:user-roles-type" minOccurs="0"/>
        <xs:element name="languages"
                    type="ci:user-languages-type" minOccurs="0"/>
        <xs:element name="focus-state"
                    type="focus-state-type" minOccurs="0"/>
        <xs:element name="users"
                    type="ci:users-type" minOccurs="0"/>
        <xs:element name="relations"
                    type="relations-type" minOccurs="0"/>
        <xs:any namespace="#other" processContents="lax"/>
      </xs:sequence>
```

```
      <xs:attribute name="entity" type="xs:anyURI"/>
      <xs:attribute name="node-id" type="xs:string"/>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
     VERSION TYPE
  -->
    <xs:complexType name="version-type">
      <xs:simpleContent>
        <xs:extension base="xs:unsignedInt">
          <xs:attribute name="entity" type="xs:anyURI"/>
          <xs:attribute name="node-id" type="xs:string"/>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <!--
      FOCUS STATE TYPE
  -->
    <xs:complexType name="focus-state-type">
      <xs:sequence>
        <xs:element name="user-count"
                    type="xs:unsignedInt" minOccurs="0"/>
        <xs:element name="coordinate"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="maximal-user-count"
                    type="xs:unsignedInt" minOccurs="0"/>
        <xs:element name="conf-uris"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:element name="available-media"
                    type="ci:conference-media-type" minOccurs="0"/>
        <xs:element name="active" type="xs:boolean" minOccurs="0"/>
        <xs:element name="locked" type="xs:boolean" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
      RELATIONS TYPE
  -->
    <xs:complexType name="relations-type">
      <xs:sequence>
        <xs:element name="relation"
                    type="relation-type"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax"/>
```

```
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
      RELATION TYPE
  -->
    <xs:complexType name="relation-type">
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="entity" type="xs:anyURI"/>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>
```

9. Relax NG Grammar

The grammar for the Landmark configuration document extension is:

```
<!--
    LANDMARKS ELEMENT
-->
parameter &= element landmarks {
    attribute version { xsd:int }
    <!--
        LANDMARK-HOST ELEMENT
    -->
    element landmark-host {
        attribute address { xsd:string },
        attribute port { xsd:int }
    }*
}?
```

10. Security Considerations

10.1. Trust Aspects

TODO

11. IANA Considerations

TODO: register Kind-ID code point at the IANA

12. Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and SAM research group. We would like to thank all active members for constructive thoughts and feedback. In particular, the authors would like to thank (in alphabetical order) David Bryan, Toerless Eckert, Lothar Grimm, Cullen Jennings, Peter Musgrave, Joerg Ott, Peter Pogrzeba, Brian Rosen, and Jan Seedorf.

13. References

13.1. 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-17 (work in progress), July 2011.

[I-D.knauf-p2psip-share]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A Usage for Shared Resources in RELOAD (ShaRe)", draft-knauf-p2psip-share-00 (work in progress), March 2011.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

[RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

13.2. 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-03 (work in progress), October 2010.

[I-D.ietf-p2psip-sip]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-06 (work in progress), July 2011.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[landmarks-infocomm02]
Ratnasamy, Handley, Karp, and Shenker, "Topologically-Aware Overlay Construction and Server Selection", Proc. of 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02) pp. 1190-1199, 2002.

[timestamps-acsc88]
Fidge, C., "Timestamps in Message-Passing Systems that Preserve the Partial Ordering", Proceedings of 11th Australian Computer Science Conference, pp. 56-66, February 1988.

Appendix A. Change Log

The following changes have been made from version draft-knauf-p2psip-disco-02.

1. DisCo-Registration uses now only the USER-CHAIN-ACL access control policy.

2. Adapted mechanisms for storing DisCo-Registrations to new requirements of Shared Resources draft [I-D.knauf-p2psip-share]

The following changes have been made from version draft-knauf-p2psip-disco-01.

1. The conference registration is now based on the Shared Resources draft [I-D.knauf-p2psip-share]:

   * DisCo-Registration Kind now meets the requirements for ShaRe.

   * Conference creation procedure now uses the ShaRe Access List.

   * Replaced USER-CHAIN-MATCH access policy for DisCo-Registration. Now uses USER-CHAIN-ACL or USER-PATTERN-MATCH.

2. Allow focus peers behind NAT

3. Added a 'node-id' attribute to the event package XML <version> element.

4. Added a 'node-id' attribute to the event package XML <focus> element.

5. Added a 'coordinate' child element to the event package XML <focus> element.

6. Corrected typos/wording

The following changes have been made from version draft-knauf-p2psip-disco-00.

1. Updated references.

2. Corrected typos.

3. New Section: Conference State Synchronization

4. XML Event Package for Distributed Conferences

5. New mechanism for generating chained conference certificates

6. Allow shared writing of resources using Access Control Policy USER-CHAIN-MATCH

7. Media Negotiation mechanism in a distributed conference

8. New Section: Distributed Conference Control with SIP

Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: alexander.knauf@haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/knauf

Gabriel Hege
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: hege@fhtw-berlin.de
URI: http://inet.cpt.haw-hamburg.de/members/hege

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Email: schmidt@informatik.haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/schmidt

Matthias Waehlisch
link-lab & FU Berlin
Hoenower Str. 35
Berlin D-10318
Germany

Email: mw@link-lab.net
URI: http://www.inf.fu-berlin.de/~waehl

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 9, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin

May 8, 2012

A RELOAD Usage for Distributed Conference Control (DisCo)
draft-knauf-p2psip-disco-05

Abstract

This document defines a RELOAD Usage for Distributed Conference Control (DisCo) with SIP. DisCo preserves conference addressing through a single SIP URI by splitting its semantic of identifier and locator using a new Kind data structure. Conference members are enabled to select conference controllers based on proximity awareness and to recover from failures of individual resource instances. DisCo proposes call delegation to balance the load at focus peers.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on November 9, 2012.

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 4
2. Terminology 5
3. Overview of DisCo 6
3.1. Reference Scenario 6
3.2. Initiating a Distributed Conference 7
3.3. Joining a Conference 8
3.4. Conference State Synchronization 9
3.5. Call delegation 10
3.6. Resilience 10
3.7. Topology Awareness 10
4. RELOAD Usage for Distributed Conference Control 11
4.1. Shared Resource DisCo-Registration 11
4.2. Kind Data Structure 11
4.3. Variable Conference Identifier 12
4.4. Conference Creation 12
4.5. Advertising Focus Ability 13
4.6. Determining Coordinates 14
4.7. Proximity-aware Conference Participation 14
4.8. Configuration Document Extension 16
5. Conference State Synchronization 18
5.1. Event Package Overview 18
5.2. <distributed-conference> 20
5.3. <version-vector>/<version> 20
5.4. <conference-description> 21
5.5. <focus> 22
5.5.1. <focus-state> 23
5.5.2. <users>/<user> 23
5.5.3. <relations>/<relation> 24
5.6. Distribution of Change Events 24
5.7. Translation to Conference-Info Event Package 25
5.7.1. <conference-info> 26
5.7.2. <conference-description> 26
5.7.3. <host-info> 26
5.7.4. <conference-state> 26
5.7.5. <users>/<user> 27
5.7.6. <sidebars-by-ref>/<sidebars-by-value> 27
6. Distributed Conference Control with SIP 28
6.1. Call delegation 28
6.2. Conference Access 29

6.3. Media Negotiation and Distribution 30
6.3.1. Offer/Answer 30
6.3.2. New Peers Joining 31
6.4. Restructuring a Conference 31
6.4.1. On Graceful Leave 31
6.4.2. On Unexpected Leave 32
7. DisCo Kind Definition 33
8. XML Schema 34
9. Relax NG Grammar 38
10. Security Considerations 39
10.1. Trust Aspects 39
11. IANA Considerations 40
12. Acknowledgments 41
13. References 42
13.1. Normative References 42
13.2. Informative References 43
Appendix A. Change Log 44
Authors' Addresses 46

1. Introduction

This document describes a RELOAD Usage for distributed conference control (DisCo) in a tightly coupled model with SIP [RFC3261]. The Usage provides self-organizing and scalable signaling that allows RELOAD peers, clients and plain SIP user agents to participate in a managed P2P conference. DisCo defines the following functions:

- A SIP protocol scheme for distributed conference control
- RELOAD Usage and definition of conferencing Kind
- Mechanisms for conference synchronization and call delegation
- Mechanism for proximity-aware routing within a conference
- An XML event package for distributed conferences

In this document, the term distributed conferencing refers to a multiparty conversation in a tightly coupled model in which the point of control (i.e., the focus) is identified by a unique URI, but the focus service is located at many independent entities. Multiple SIP [RFC3261] user agents uniformly control and manage a multiparty session. This document defines a new Usage for RELOAD, including an additional Kind code point with a corresponding data structure that complies with the demands for distributed conferences. The 'DisCo' data structure stores the mapping of a single conference URI to multiple conference controllers and thereby separates the conference identifier from focus instantiations.

Authorized controllers of a conference are permitted to register their mapping in the DisCo data structure autonomously. Thus, the DisCo data structure represents a co-managed Resource in RELOAD. To provide trusted and secure access to a co-managed Resource, this document uses the definitions for Shared Resources (ShaRe) [I-D.knauf-p2psip-share].

Delay and jitter are critical issues in multimedia communications. The proposed conferencing scheme supports mechanisms to build an optimized interconnecting graph between conference participants and their responsible conference controllers. Conference members will be enabled to select the closest focus with respect to delay or jitter.

DisCo extends conference control mechanisms to provide a consistent and reliable conferencing environment. Controlling peers maintain a consistent view of the entire conference state. The multiparty system can be re-structured based on call delegation operations.

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 [RFC2119].

We use the terminology and definitions from der RELOAD base draft[I-D.ietf-p2psip-base], the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts], the usage for shared resources draft [I-D.knauf-p2psip-share],and the terminology formed by the framework for conferencing with SIP [RFC4353]. Additionally the following terms are used:

Coordinate Value: An opaque string that describes a host's relative position in the network topology.

Focus peer: A RELOAD peer that provides SIP conferencing functions and implements the Usage for distributed conferencing. It is called 'active' if already involved in signaling relation to conference participants. Otherwise, if only registered in a distributed conference data structure, it is referred to as a 'potential' focus peer.

3. Overview of DisCo

3.1. Reference Scenario

The reference scenario for the Distributed Conference Control (DisCo) is shown in Figure 1. Peers are connected via a RELOAD [I-D.ietf-p2psip-base] instance, in which peers A and B are managing a single multiparty conference. The conference is identified by a unique conference URI, but located at peers A and B fulfilling the role of the focus. The mapping of the conference URI to one or more responsible focus peers is stored in a new RELOAD Resource for distributed conferencing within a data structure denoted as DisCo-Registration. The storing peer SP of the distributed conference resource holds this data.

The focus peers A and B maintain SIP signaling relations to conference participants, which may have different conference protocol capabilities. In this example, peer A is the focus for the RELOAD peer C and the plain SIP user agent E whereas peer B serves as a focus for RELOAD peer D and the RELOAD client F.

RELOAD peers and clients obtain the contact information for the conference from the storing peer. In contrast to this, the user agent E receives the conference URI not by RELOAD mechanisms, but resolves the ID and joins the conference by plain SIP negotiation.

Focus peers maintain a SIP signaling relation among each other used for notification messages that synchronize the conference focus peers' knowledge about the entire conference state. Additionally, focus peers can transfer calls to each other by a call delegation mechanism.

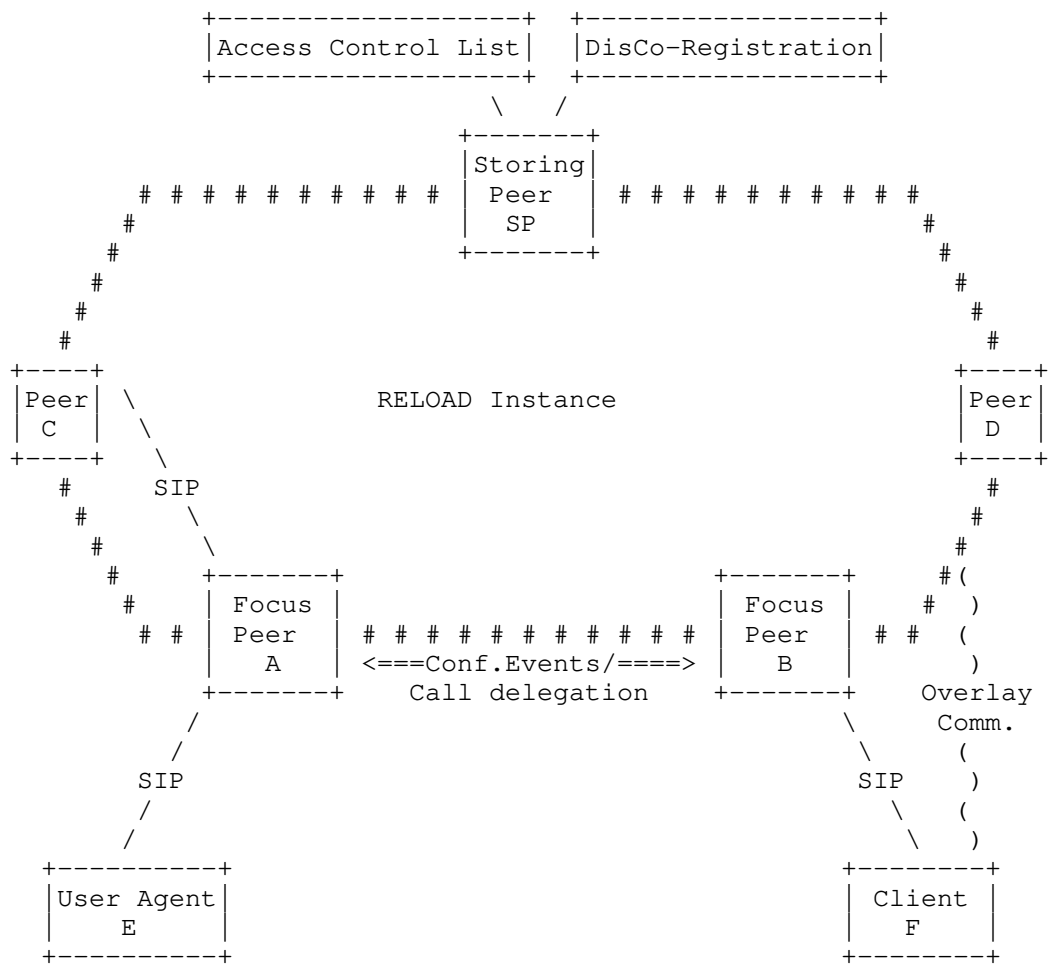


Figure 1: Reference Scenario: Focus peers A,B maintain a distributed conference

3.2. Initiating a Distributed Conference

To create a conference the initiating user agent announces itself as a focus for the conference. It stores its own contact information (Node-ID) as a DisCo-Registration Kind (cf. Figure 2) in the RELOAD overlay. The hashed conference URI is used as the Resource-ID. This Resource will later contain the contact IDs of all potential focus peers including optional topological descriptors.

3.3. Joining a Conference

A RELOAD-aware node (cf. Bob in Figure 2) intending to join an existing conference requests the list of potential focus peers stored in the DisCo-Registration under the conference's Resource-ID. The node selects any of the focus peers (e.g., Alice) and establishes a connection using AppAttach/ICE [RFC5245]. This transport is then used to send an INVITE to the conference applying the chosen focus as the contact. The selection of the focus peer can optionally be based on proximity information if available.

A conference member proposes itself as a focus for subsequent participants by adding its Node-ID to the DisCo-Registration stored under the conference URI in the RELOAD overlay. The decision whether a peer announces as a focus incorporates bandwidth, power, and other constraints, but details are beyond the scope of this document.

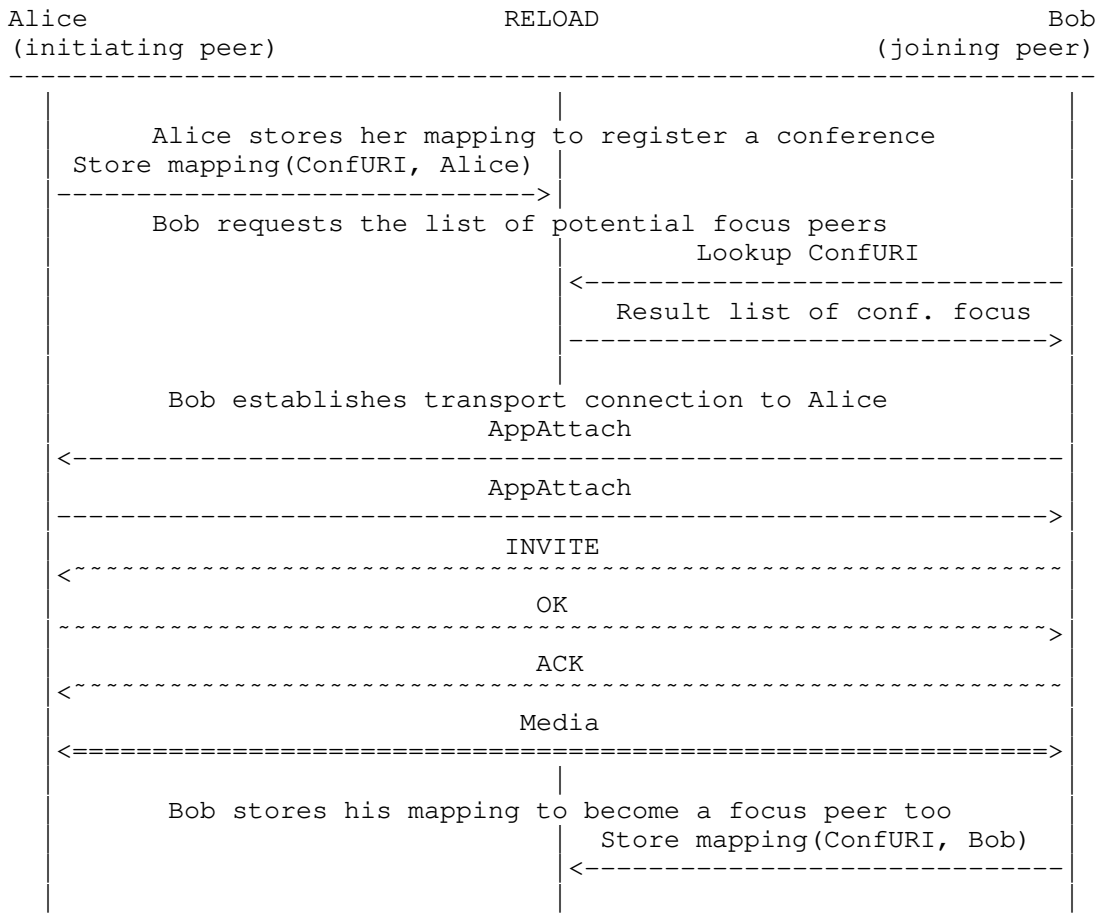


Figure 2: DisCo Usage generic Call Flow

3.4. Conference State Synchronization

Each focus of a conference maintains signaling connections to its related participants independently from other conference controllers. This distributed conference design effects that the entire SIP conference state is jointly held by all focus peers. In DisCo, state synchronization is based on a SIP specific event notifications mechanism [RFC3265].

Each focus peer maintains its view of the entire conference state by subscribing to the other focus peers' XML event package for distributed conferences. This is based on the event package for conference state (cf. [RFC4575]). Details are defined in this document in Section 5. Receivers of event notifications update their

local conference state document to represent a valid view of current total conference state.

The event notification package for distributed conferences enables focus peers to synchronize the entire conference state. The event package defines additional XML elements and complex types (see Section 8 for more details), which describe the responsibilities of any focus peer in the conference. By providing a global view each focus peer is enabled to perform additional load balancing operations and enhances the robustness against departures of focus peers.

3.5. Call delegation

Call delegation (see Section 6.1) is a feature used to transfer an incoming participation request to another focus peer. It can be applied to prevent overloading of focus peers. Call delegation is realized through SIP REFER requests, which carry signaling and session description information of the caller to be transferred. This feature is achieved transparently for the transferred user agent by using a source routing mechanism at SIP dialog establishment. Descriptions of overload detection are beyond the scope of this document.

3.6. Resilience

A focus peer can decide to leave the conference or may ungracefully fail. In a traditional conferencing scenario, loss of the conference controller or the media distributor would cause a complete failure of the multiparty conversation. Distributed conferencing uses the redundancy provided by multiple focus peers to reconfigure a current multiparty conversation. Participants that loose their entry point to the conference re-invite themselves via the remaining focus peers or will be re-invited by the latter. This option is based on the conference state and call delegation functions.

3.7. Topology Awareness

DisCo supports the optimization of the conference topology in respect of the underlying network using topological descriptors. An extension for the RELOAD XML configuration document is defined in Section 4.8 to support landmarking approaches. Each peer intending to create or participate in a distributed conference MAY determine a topological descriptor that describes its relative position in the network. Focus peers store these coordinate values in an additional data field in the DisCo-Registration data structure. This enables peers joining the conference to select the closest focus with respect to its coordinate values.

4. RELOAD Usage for Distributed Conference Control

4.1. Shared Resource DisCo-Registration

A distributed conference is a cooperative service of several individual devices that use a common identifier. To enable a mapping from one conference identifier to multiple focus peers, this document defines the new Kind data structure DisCo-Registration. The DisCo Kind uses the definitions for Shared Resources [I-D.knauf-p2psip-share] to allow a jointly maintenance by multiple focus peers. Hence, write permission to a DisCo-registration is shared by the conference creator with all authorized focus peers.

DisCo complies with the following requirements for Shared Resources:

Isolated Data Storage: DisCo uses the dictionary data model. Each dictionary key is the Node-ID of the certificate that will be used to sign the stored data

ResourceNameExtension field: A DisCo-Registration can contain the ResourceNameExtension structure an initial field in the Kind data structure. It contains the conference URI of the registered DisCo-record.

4.2. Kind Data Structure

Each DisCo-Registration data structure stores a mapping of a conference identifier to one or multiple focus peers that cooperatively control the conference. Additionally, each DisCo-Registration provides the coordinate value, which indicates the relative network position of the focus peers.

The data structure uses the RELOAD dictionary type. The dictionary key MUST be the Node-ID of the focus peer that is associated with the dictionary entry. This allows a focus peer to update only its own mapping. The DisCo data structure of type DisCoRegistration is constructed as follows:

```
struct {
  /* This field is optional, see documentation */
  ResourceNameExtension res_name_ext;
  opaque coordinate<0..2^16-1>;
  NodeId node_id;
} DisCoRegistration;
```

The DisCoRegistration structure is composed of the following values:

res_name_ext: This field can contain the conference URI. It meets the requirement for the USER-CHAIN-ACL access policy defined in [I-D.knauf-p2psip-share] to enable variable resource names.

coordinate: This field contains a topological descriptor that indicates the relative position of the peer in the network. To support different algorithms the coordinate field is represented as an opaque string.

node_id: This field contains the Node-ID of the peer storing the DisCoRegistration and is used to contact the peer when utilizing its service as a focus.

4.3. Variable Conference Identifier

DisCo-Registrations can be stored based on a variable Resource Name. However, a conference identifier set by a user MUST follow the requirements for Kinds using variable Resource Names as defined in the ShaRe Usage [I-D.knauf-p2psip-share].

4.4. Conference Creation

The registration of a distributed conference includes the storage of the following two Kinds (see Figure 3).

DisCo-Registration Kind: contains the conference identifier and the optional coordinate value. If used, the coordinate value MAY be determined previously to the conference registration. The Resource Name and hence the Resource-ID is defined by the hash over the desired conference identifier (using the hash algorithm of the overlay).

Access Control List Kind: contains the conference participants that are allowed to register as focus peers for a conference (see [I-D.knauf-p2psip-share]). Its Resource Name/ID is equal to those of the corresponding DisCo-Registration.

Preliminary to storage of DisCo-Registration and Access Control List (ACL) Kinds the conference creator SHOULD perform a RELOAD StatReq to verify that no former conference is present. If neither a DisCo-Registration nor an associated ACL exist, the conference creator stores a DisCo-Registration with a valid conference identifier (see Section 4.3) and an ACL referring to the DisCo-Registration Kind-ID.

If DisCo registrations and ACL Kinds from previous conferences are still existing there are two options. First, if conference creator is aware of the indexes from previous ACL Kinds, it refreshes the root item of this ACL and stores its registration as focus peer as

DisCo-Registration Kind. Second, If the creator is unaware of indexes, it fetches all Access List Kinds to determine the index of the root item.

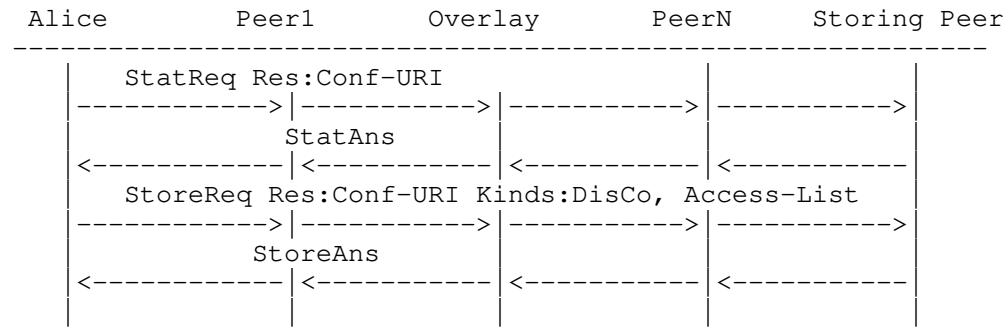


Figure 3: Initial creation of a Distributed Conference

Optionally the conference initiator (or any active focus) MAY store an additional RELOAD SIP-Registration in the overlay [I-D.ietf-p2psip-sip] or even at a standard SIP registrar [RFC3261] under a URI for which it has write permission. This allows DisCo-unaware or even legacy SIP user agents to participate in the conference. Those registrations SHOULD always point to a currently active focus, who is prepared to accept legacy user agents. The user agent who initially performed the registrations is responsible for updating them appropriately. How authorization has been obtained to perform those registration is out of scope of this document.

The lifetime of a distributed conference is not necessarily limited by the participation time of its creator. As long as the root item of an Access Control List to a DisCo-Registration is not expired, Authorized Peers are allowed to further provide a conferencing service and even store new focus registrations.

4.5. Advertising Focus Ability

All participants of a distributed conference MAY potentially become a focus peer for their conference. This depends on its capacities such as sufficient processing power (CPU, Memory) for the desired media type, the quality of the network connectivity, and the conference policy. Information about network connectivity with respect to NAT or restricted firewalls can be obtained via ICE [RFC5245] connectivity checks. If a peer is behind a NAT, it SHOULD allow for incoming connections via AppAttach/ICE. Peers that can only accept connections with the support of TURN should not act as a focus.

Nevertheless, under special circumstances it might be reasonable to locate a focus peer behind such a NAT (e.g., within a companies network infrastructure).

If a participant is a candidate to become a focus for the conference, it stores its Node-ID and optional coordinate value into the DisCo data structure. An entry in the corresponding ACL [I-D.knauf-p2psip-share] must be present to allow this peer to write the DisCo-Registration resource. By storing the mapping into the data structure a participant becomes a potential focus.

4.6. Determining Coordinates

Each RELOAD peer within the context of a distributed conference MAY be aware of its relative position in the network topology. To constuct a topology-aware conference, the DisCo Usage provides the coordinate value within the DisCo-Registration data structure. Focus peers store their relative network position together with the announcement as conference focus. Joining peers that are able to determine their coordinates may then select a focus peer whose relative position is in its vicinity (see Section 4.7).

Some algorithms determine topology information by measuring Round-Trip Times (RTT) towards a set of hosts serving as landmarks (e.g., [landmarks-infocomm02]). To support such algorithms this document describes an extension to the RELOAD XML configuration document that allows to configure the set of landmark hosts peer must use for position estimation (see Section 4.8). Once a focus peer has registered its mapping in the DisCo data structure, it also stores the according coordinates in the same mapping. These <Node-ID,coordinates> vectors are used by peers joining the conference to select the focus peer that is relatively closest to itself.

Because topology-awareness can be obtained by many different approaches a concrete algorithms is out of scope of this document.

4.7. Proximity-aware Conference Participation

The participation procedure to a distributed conference is composed of three operation.

1. Resolution of the conference identifier.

2. Establishment of of transport connection.

3. SIP signaling to join a conference.

Figure 4 and the following specifications give a more detailed view

on the joining procedure.

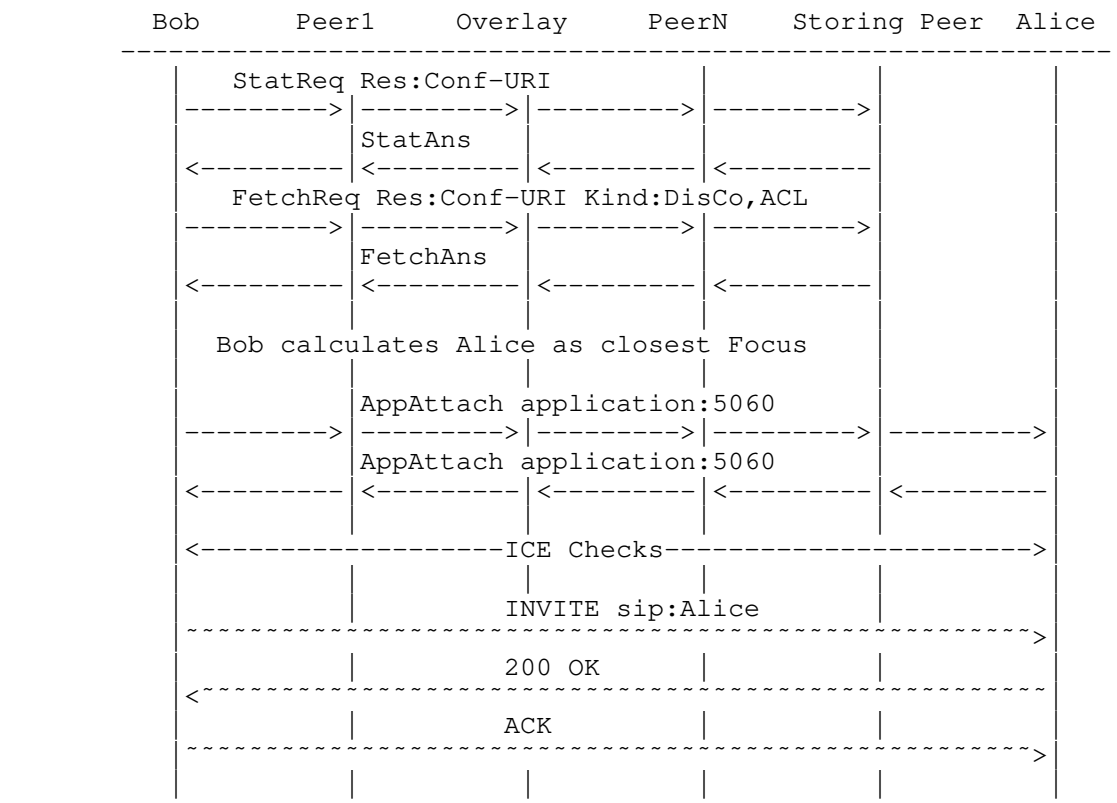


Figure 4: Participation of a Distributed Conference

1. The joining peer MAY determine its own coordinate value (if used).

2. The joining peer sends a StatReq message to obtain all indexes of the Access Control List (ACL) Kinds stored.

3. The joining peer sends a FetchReq message for the DisCo and ACL Kind to the Resource-ID of the conference URI. The FetchReq SHOULD NOT include any specific dictionary keys, but SHOULD fetch for those array ranges previously determined the StatReq. With the ACL items, the joining peer is able to verify whether DisCo-Registrations are stored by authorized focus peers (see [I-D.knauf-p2psip-share]).

4. Using the retrieved coordinates values of the DisCo-Registrations, the joining peer MAY calculate which of than

opaque <0..2^16-1> initial field in the Kind data structure focus peers is the relatively closest to itself.

5. The joining peer then establishes a transport using RELOADs AppAttach, respectively, ICE procedures to create a transport to the chosen focus peer.

6. Finally, the established transport is used to create a SIP dialog from the joining peer to the focus peers.

The SIP INVITE request MAY contain the joining peer's topological descriptor as a URI-parameter called 'coord' in the contact-header in base64 encoded form [RFC4648]. An example contact URI is "sip:alice@example.com;coord=PEknbSBhIHRvcG9sb2dpY2FsIGRlc2NyaXB0b3I+". When the called focus is full and needs to delegate the call it uses the contents of the 'coord'-parameter. It determines the next available focus closest to the calling peer (Section 4.6) using the received descriptor and the other focuses' descriptors from the conference state synchronization document and delegates the call accordingly (see Section 6.1).

A conference focus MAY allow the joining peer to also become a focus (depending on the conference policy see Section 6.2). Therefore, it stores a new ACL Kind that delegates write permission for the DisCo-Registration to the new participant. It sets the 'user_name' field in the ACL Kind to its own user name and sets the 'to_user' field to the user name of the joining peer. If no other conference policy is defined, the focus peer MAY set the allow_delegation flag to true. For further details about the trust delegation using the ACL Kind see [I-D.knauf-p2psip-share].

4.8. Configuration Document Extension

This section defines an additional parameter for the <configuration> element that extends the RELOAD XML configuration document. The proposed <landmarks> element allows RELOAD provider to publish a set of accessible and reliable hosts that SHOULD be used if RELOAD peers use landmarking algorithms to determine relative position in the network topology.

The <landmarks> element serves as container for the <landmark-host> sub-elements, each representing a single host that serves as a landmark. The <landmark-host> uses the following attributes:

address: The IP address (IPv4 or IPv6) of the landmark host.

port: The port on which the landmark host responds for distance estimation.

More than one landmark hosts SHOULD be present in the configuration document.

5. Conference State Synchronization

The global knowledge about signaling and media relations among the conference participants and focus peers defines the global state of a distributed conference. It is composed of the union of every local state at the focus peers. To enable focus peers to provide conference control operations that modify and/or require the global state of a conference, this document defines a mechanism for inter-focus state synchronization. It is based on mutual subscriptions for an Event Package for Distributed Conferences and allows to preserve a coherent knowledge of the global conference state. This XML based event package named 'distributed-conference' MUST be supported by each RELOAD peer that is registered with a DisCo-Registration. Participants of a distributed conference MAY support it. To provide backward compatibility to conference members that do not support the distributed-conference event package, this document defines a translation to the Event Package for Conference State [RFC4575].

5.1. Event Package Overview

The 'distributed-conference' event package is designed to convey information about roles and relations of the conference participants. Conference controllers obtain the global state of the conference and use this information for load balancing or conference restructuring mechanisms in case of a focus failure. Figure Figure 5 gives a general overview of the document hierarchy.

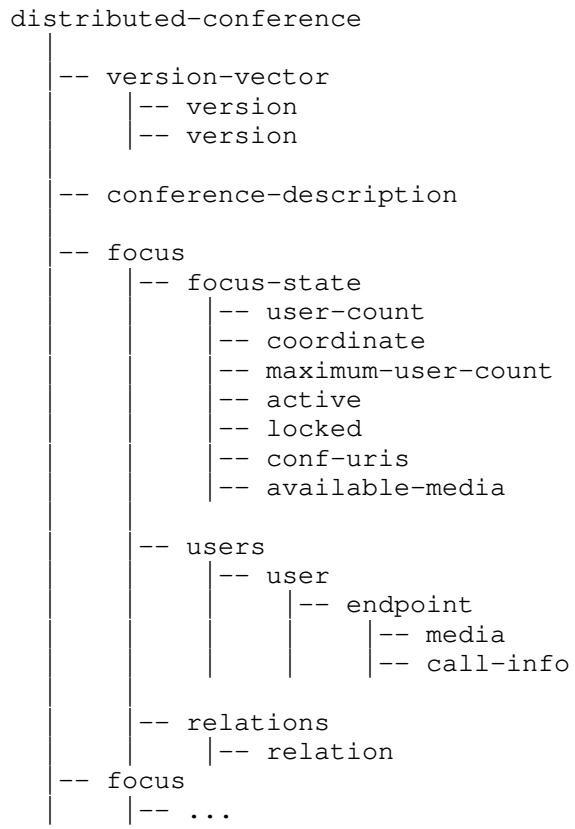


Figure 5: Overview of the event package for distributed conferences

The document structure is designed to allow concurrent change events at several focus peers. To prevent race conditions each focus peer has exclusive writing permission to the 'focus' sub element that describes itself. It is achieved by a unique mapping from a focus peer to its XML element using the 'Element Keys' mechanisms for partial notification [RFC4575](sections 4.4-5.). A focus peer is only allowed to update or change that <focus> sub element, whose 'entity' Element Key contains its RELOAD user name. This restriction also applies to the child elements of the 'version-vector' element. Each 'version' number belongs to a specific focus peer maintaining the version number.

The local state of a focus peer is described within a 'focus' element. It provides general information about a focus peer and its signaling and media relations to participants and focus peers. The Conference participants are aggregated within 'users' elements, respectively, 'user' sub elements.

General information about the conference as a whole, is provided within a 'conference-description' element. In contrast to the 'focus' and 'version-vector' elements, conference description is not meant for concurrent updating. Instead, it provides static conference descriptions that rarely change during a multiparty session.

More detailed descriptions about the event package and its elements are provided in the following sections. The complete XML schema definition is given in Section 8.

5.2. <distributed-conference>

The <distributed-conference> element is the root of a distributed conference XML document. It uses the following attributes:

entity: This attribute contains the conference URI that identifies the distributed conference. A SIP SUBSCRIBE request addressed to this URI initiates an subscribe/notify relation between participants and their related focus peer.

state: This attribute indicates whether the content of a distributed conference document is a 'full', 'partial' or 'deleted' information. It is in accordance with [RFC4575] to enable the partial notification mechanism.

The <distributed-conference> child elements are <vector-version>, <conference-description> and the <focus> elements. An event notification of state 'full' MUST include all these elements. An event notification of state 'partial' MUST contain at least <version-vector> and its sub elements.

5.3. <version-vector>/<version>

The Event Package for Distributed Conferences uses the <version-vector> and its <version> sub elements to enable a coherent versioning scheme. It is based on vector clocks as defined in [timestamps-acsc88]. Each <version> element contains a unsigned integer that describes the state of a specific focus peer and contains the following attributes:

entity: This attribute contains the user name of the focus peer whose local version number is described by this element.

node-id: This attribute contains the Node-ID of the focus peer.

Whenever the local status of a focus peer changes (e.g. a new participant arrived) the version number of the corresponding

<version> element MUST be incremented by one. Each change in the local state also triggers a new event notification containing the entire <version-vector> and the changes contained in a <focus> element.

The recipient of a change event needs to update it local XML document. If a received <version> number is higher that the local, it updates the <version> element and its associated <focus> element with the retrieved elements. All other elements remain constant.

If the length or contents of the retrieved <version-vector> is different to the local copy it indicates a incoherent knowledge about the entire conference state. If the retrieved <version-vector> contains any unknown focus peers and any local version numbers for the known focus peers is lower, the receiver SHOULD request a 'full' XML notification.

If any local <version> number is retarded more than one, the receiver SHOULD request a 'full' event notification from the sender. The full state notification updates all <focus> elements whose corresponding <version> element is out of date.

5.4. <conference-description>

The <conference-description> element provides general information and links to auxiliary services for the conference. The following sub elements provide human-readable text descriptions about the conference:

<display-text>: Contains a short textual description about the conference

<subject>: Contains the subject of the conference

<free-text>: Contains a longer textual description about the conference

<keywords>: Contains a list of keywords that match the conference topic. The XML schema definition and semantic is imported from the 'conference-info' event package [RFC4575].

The <service-uris> sub element enables focus peers to advertise auxiliary services for the conference. The XML schema definition and semantic is imported from the 'conference-info' event package [RFC4575].

The <conference-description> element uses the 'state' Element Key to enable the partial notification mechanism.

5.5. <focus>

Each <focus> element describes a focus peer actively controlling the conference. It provides general information about a focus peer (e.g., display-text, languages, etc.), contains conference specific information about the state of a focus peer (user-count, available media types, etc.) and announces signaling and media information about the maintained participants. Additionally, it describes signaling or media relations to further focus peers.

The <focus> element uses the following attributes:

entity: This attribute contains the user name of the RELOAD peer acting as focus peer. It uniquely identifies the focus peer that is allowed to update or change all sub elements of <focus>. All other focus peers SHOULD NOT update or change sub elements of this <focus> element. A SUBSCRIBE request addressed to the user name initiates a conference state synchronization with the focus peer.

Node-ID This attributed contains the Node-ID of the peer acting as conference focus

state: In accordance to [RFC4575], this attribute indicates whether the content of the <focus> element is a 'full', 'partial' or 'deleted' information. A 'partial' notification contains at maximum a single <focus> element.

The following sub elements of <focus> provide general information about a focus peer:

<display-text>: Contains a short text description of the user acting as focus peer.

<associated-aors>: This element contains additional URIs that are associated with this user.

<roles:> This element MAY contain human-readable text descriptions about the roles of the user in the conference.

<languages>: This element contains a list of tokens, each describing a language understood by the user.

The XML schema definition and semantic for <associated-aors>, <roles> and <languages> are imported from the 'conference-info' event package [RFC4575]

Following, a detailed description of the remaining sub elements.

5.5.1. <focus-state>

The <focus-state> element aggregates a set of conference specific information about the RELOAD user acting as focus peer. The following attribute is defined for the <focus-state> element:

status: This attribute indicates whether the content of the <focus-state> element is a 'full', 'partial' or 'deleted' information.

The <focus-state> element has the following sub elements:

<user-count>: This element contains the number of participants that are connected to the conference via this focus peer at a certain moment.

<coordinate> This element contains the coordinate value Section 4.2 of the focus peer

<maximum-user-count>: This number indicates a threshold of participants a focus peer is able to serve. This value might change during a conference, depending on the focus peers current load.

<conf-uris>: This element MAY contain other conference URIs in order to access the conference via different signaling means. The XML schema definition and semantic is imported from [RFC4575].

<available-media>: This element is imports the <conference-media-type> type XML scheme definitions from [RFC4575]. It allows a focus peer to list its available media streams.

<active>: This boolean element indicates whether a focus peer is currently active. Conference participation requests or a call delegation request SHOULD succeed.

<locked>: In contrast to <active>, this element indicates that a focus peer is not willing to accept anymore participation or call delegation request.

5.5.2. <users>/<user>

The <users>, respectively, each <user> element describes a single participant that is maintained by the focus peer described by the parent <focus> element. The <users> element XML schema definition and its semantic is imported from the 'conference-info' event package [RFC4575].

5.5.3. <relations>/<relation>

The <relations> element serves as container for <relation> elements, each describing a specific connection to another focus peer. The parent element <relations> uses the 'state' attribute to enable the partial notification mechanism. For the <relation> element the following attributes are defined:

entity: This attribute contains the user name of the remote focus.

node-id This attribute contains the Node-ID of the remote focus peer.

The content of each <relation> is a comma separated string that describes the tuple <CONNECTION-TYPE:IDENTIFIER>. The CONNECTION-TYPE is a string token describing the type of connection (e.g. media, signaling, etc.) and the IDENTIFIER contains a variable connection identifier. It is a generic method to announce any kind of connection to a remote focus. This specification defines following tuples:

media:<label>: This tuple identifies a single media stream. The 'label' variable contains the SDP "label" attribute. (see [RFC4574]).

sync:<call-id>: This tuple indicates a subscription for the Event Package for Distributed Conferences. The 'call-id' variable contains the call-id of the SIP subscription dialog.

5.6. Distribution of Change Events

Each focus peer in a distributed conference must be able to retrieve all change events from all other focus peers. A simple approach would be to inter-connect each focus with all other focus in a full meshed topology. To avoid a full mesh, this document describes a 'mutual' subscription scheme that constructs a spanning tree topology among the focus peers.

A conference participant that becomes a focus peer MUST send a SIP SUBSCRIBE to request the Event Package for Distributed Conferences to its own focus peer. The subscription request is addressed to user name of the active focus peer. The latter interprets this subscription as a request for conference state synchronization. The corresponding NOTIFY message contains a 'full' distributed-conference state XML document (see section Section Section 5.1).

The subscribed focus peer in turn subscribes the upcoming focus peer for the distributed conference event package. The corresponding

NOTIFY message carries a 'partial' conference state XML document. It contains the received <version-vector> including a new <version> element for itself and a new <focus> element that describes its local state (see Section 5.5).

Resulting by this subscription scheme, each focus peer has at least one subscription to obtain updates for the conference state and is a notifier for change events originated itself. In a incrementally increasing conference, the 1st and 2nd focus peer have a mutual subscription for conference change events. A 3rd focus could have a mutual subscription with the 1st focus, a 4th focus to the 2nd focus and so forth. The result is a spanning tree topology among the focus peers in which each focus peer is a possible root for distribution tree for conference change events.

However, the fact that event notifications need to traverse one or more intermediate focus peers until conference-wide delivery, demands a forwarding mechanism for change events. On receiving a change event, a notified focus validates based on the <version-vector> whether the incoming state event is not a duplicate to previews notifications. If its not a duplicate, the received change event triggers a new event notification at the receiver of the change event. It notifies all its subscribers with excepting the sender of the event notification. In this way, the change event will be 'flooded' among the focus peer of a conference.

5.7. Translation to Conference-Info Event Package

The Event Package for Distributed Conferences imports several XML element definitions of the Event Package for Conference State [RFC4575]. This is caused by two reasons. First, the semantic of these elements are fitting the demands to describe the global state of a distributed conference and, second, it facilitates a re-translation to [RFC4575] to enable a backward compatibility to DisCo-unaware clients. Therefore, each focus peer MAY provide a separate [RFC4575] conform event notification service to its connected participants.

The following sections describe the translation to the Event Package for Conference State [RFC4575] by defining translation rules for the root element and its direct sub elements. For a better understanding, the following sections use a notation ci.<ELEMENT> to refer to a sub element of the conference-info element, and disco.<ELEMENT> to refer to an element of the distributed-conference event package.

5.7.1. <conference-info>

The root element of Event Package for Conference State uses the attributes 'entity', 'state' and 'version' and is the counterpart of the <distributed-conference> root element in the DisCo Event Package. The former two attributes 'entity' and 'state' are equal in both root elements and can be seamlessly translated.

According to [RFC4575], the 'version' attribute SHOULD be incremented by one at any time a new notification being sent to a subscriber. Hence, in DisCo the 'version' attributed increments with each change event that originated by focus peer and each reception of a change events of remote focus peer.

5.7.2. <conference-description>

The <conference-description> element exists in both event packages, conference-info and distributed-conference. Thus, the following elements are seamlessly translatable: <keywords>, <display-text>, <subject>, <free-text> and <service-uris>.

The sub elements <conf-uris>, <maximum-user-count> and <available-media> in conference-info have there counterparts below the \focus\focus-state element of the distributed-conference event package. Each describes a local state of a focus peer in the conference. Hence, the intersection of every disco.<conf-uris>, disco.<available-media> and the sum over each disco.<maximum-user-count> element of each disco.<focus> element in distributed-conference, specifies the content of the corresponding conference-info elements.

5.7.3. <host-info>

According to [RFC4575] the ci.<host-info> element contains information about the entity hosting the conference. For participants in a distributed conference, the hosting entity is their focus peer. Thus, the ci.<host-info> element contains information about a focus peer.

5.7.4. <conference-state>

The ci.<conference-state> element allows subscribers obtain information about overall state of a conference. Its sub elements ci.<user-count>, ci.<active> and ci.<locked> are reused as sub elements of \focus\focus-state to describe the local state of a focus peer in a distributed conference. The translation rules from the distributed-conference to the conference-info event package are the following:

<user-count>: The sum over each value of the disco.<user-count> element defines the corresponding ci.<user-count>.

<active>: The boolean ci.<active> element is the logical concatenation over all disco.<active> elements by an OR-operator.

<locked> The boolean ci.<locked> element is the logical concatenation over all disco.<locked> elements by an AND-operator.

5.7.5. <users>/<user>

The distributed-conference event package imports the definitions of the ci.<users> and ci.<user> elements under a parent disco.<focus> element for each focus peer in a conference. Thus, the aggregation over all disco.<users> elements specifies the content of the corresponding ci.<users> element.

5.7.6. <sidebars-by-ref>/<sidebars-by-value>

In accordance to [RFC4575], if a participant is connected to a sidebar, its responsible focus peer creates a new <user> by referencing to the corresponding sidebar conference.

6. Distributed Conference Control with SIP

Distributed conference control with SIP defined in this document refers to multiparty conversation in a tightly coupled model that is controlled by several independent entities. It enables a resilient conferencing service for P2P scenarios and provides mechanisms for load-balancing among the focus peers. This section describes additional control operations for distributed conferences with SIP.

6.1. Call delegation

Distributed conference control enables load-balancing by a mechanism for call delegation. Call delegations are performed by focus peers that are running out of capacities to serve more participants. Incoming participation requests are then transferred to other established focus peer or conference participants that are registered as potential focus peers in the overlay. Call delegations use SIP REFER requests [RFC3515] that contain additional session information and are achieved transparently to the transferred party.

A focus peer initiates a call delegation by sending SIP REFER request containing the URI of the participant in the Refer-To header field. Additionally, the focus peer appends the following parameter to the URI of the participant:

call-id: Contains the call-ID of the initial SIP dialog between the referred participant and the referring focus peer.

sess-id: Contains the 'session identifier' value of the original SDP 'o=' field of the original offer/answer process between referred participant and referring focus peer.

If the recipient accepts the REFER request, it generates a re-INVITE towards the referred party and sets the SIP call-id header and the SDP 'session-identifier' field in the SDP offer, according to the URI parameter values of the initial REFER request. The From header field and contact header are set to the conference URI with setting the 'isfocus' tag to contact header. This identifies the peer as a focus to the conference and identifies this re-INVITE as a request of the SIP dialog between the party and the conference. To ensure that further signaling messages will be routed correctly, the new focus adds a Record-Route header field that contains its contact information (URI, IP-address,..).

An example call flow for call delegation is shown in Figure 6.

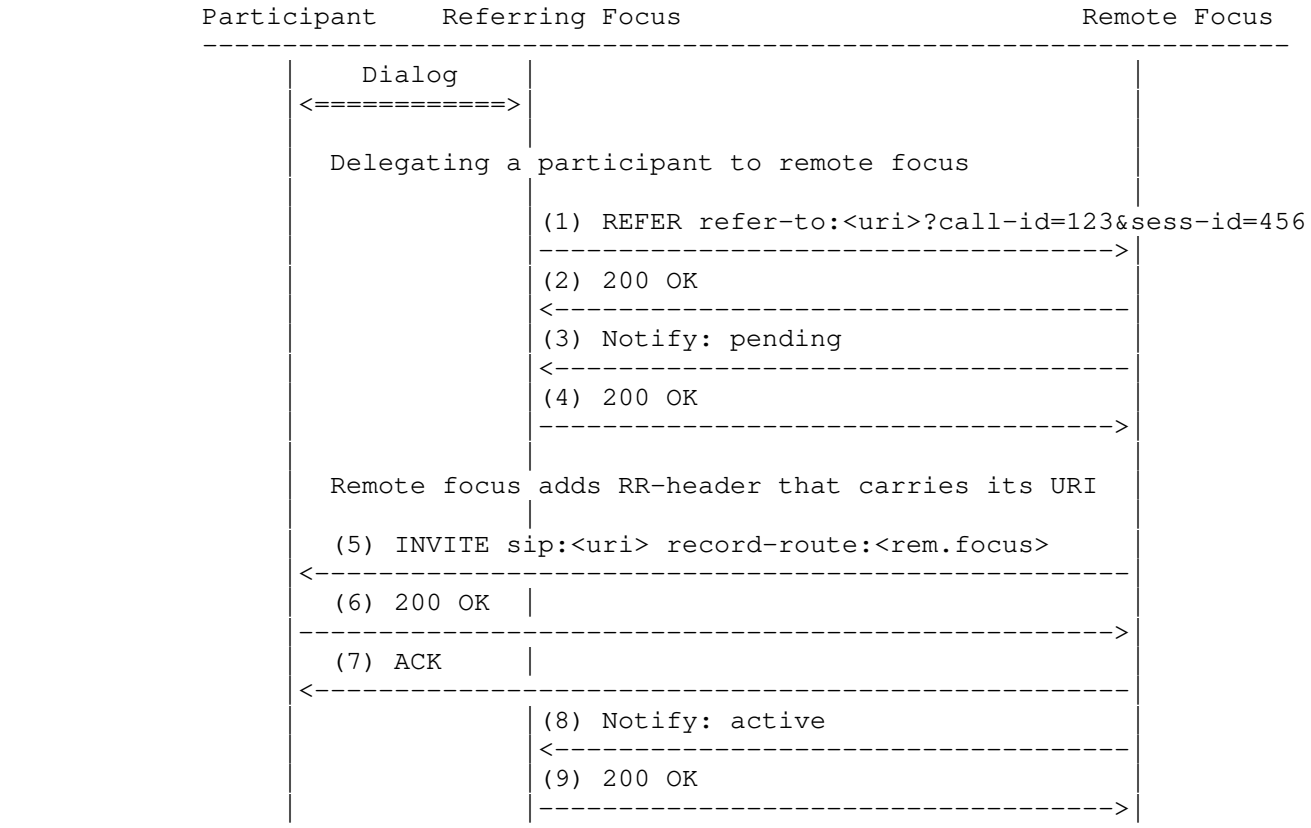


Figure 6: Delegating a participant with SIP REFER

Note, subscriptions for the event packages 'distributed-conference' and 'conference-info' are in scope of a specific focus peer and its connected participants. Hence, after a successful call delegation, the referring focus peer SHOULD terminate any subscription to the referred participant. The notifier SHOULD include a reason parameter "deactivated" to indicate a migration of the subscription as defined in [RFC3265]. The new SUBSCRIBE request by the party MUST be sent via the SIP dialog to the conference.

6.2. Conference Access

A conference policy defines who is allowed to participate in a multimedia conference. In many cases, a group conversation can be an open discussion free to participate, while in other occasions a closed privacy of a multiparty session is demanded. In distributed conferences, it is also an issue which of the conference participants is allowed to become a controller of the multiparty session.

Thus it must be decided whether:

- A peer is allowed to participate in a conference
- A peer is allowed to become a focus of the conference

Standard SIP authentication mechanisms can be used to authenticate and accordingly authorize joining participants.

6.3. Media Negotiation and Distribution

This section describes a basic scheme for media negotiation and distribution, which is done in an ad-hoc fashion.

In an established DisCo conference, each participant is attached to one focus (possibly itself), and all focus peers maintain mutual signaling relationships. Each focus peer receives media streams from two groups, its locally attached members, as well as the neighboring focus peers. It has two options when re-distributing media streams to the conference. It either mixes streams from each group and thus reduces the number of media sessions, or propagates the streams in individual media sessions.

The basic media distribution naturally follows the SIP signaling paths. Each focus peer forwards all media streams it receives from the conference (possibly mixed) to all connected peers it is responsible for, and similarly all streams from its peers to its responsible focus. Implementations can choose more sophisticated schemes for media distribution, e.g., some form of overlay multicast, but MUST take measures to prevent loops in media routing.

6.3.1. Offer/Answer

A peer joining a conference negotiates media types and media parameters with its designated focus using the standard SDP offer/answer protocol [RFC3264]. The focus SHOULD offer all existing media streams that it receives from the conference.

A new participant does not necessarily know about all media streams present in a conference beforehand, and thus some of the media streams might not be included in the initial SDP offer sent by the joining peer. An SDP answer sent by the corresponding focus though cannot offer additional media types that do not match an offer (cf. [RFC3264] Section 6). A joining peer, which is aware of a heterogeneous conference, can invert the offer/answer dialog by omitting an SDP offer in the initial INVITE message. Instead it MAY send an empty INVITE to which the focus replies with an OK, containing the SDP offer. This prevents the need for a second offer/

answer-dialog to modify the session. But for compatibility the normal behavior with the INVITE containing the offer MUST be supported.

For new media streams (e.g. those sent by the new participant), the focus SHOULD only offer media types and codecs which are already used in the conference and which will probably be accepted when forwarded to neighboring peers, unless the focus is prepared to do transcoding and/or mixing of the received streams.

6.3.2. New Peers Joining

When a new peer has been attached to a focus, new media streams may be available to the focus that need to be forwarded to the conference. To accomplish this, the new media streams need to be signalled to the other participants. This is commonly done by sending a SIP re-INVITE [RFC4353] for modifying the media sessions, adding the new streams to the SDP. This renegotiation can be costly since it needs to be propagated throughout the entire conference. In addition, distributing all media streams separately to all participants can be quite bandwidth intensive. Both problems can partially be mitigated by focus peers performing mixing of media streams, thus trading bandwidth and signalling overheads for computational load on focus peers.

6.4. Restructuring a Conference

Distributed conference control provides the possibility to delegate calls to remote focus peers. This feature is used to restructure a conference in case of departure of a focus peer. Following, this section presents restructuring schemes for graceful and unexpected leaves of conference focus peers.

6.4.1. On Graceful Leave

In a graceful case, the leaving focus peer (LP) accomplishes the following procedure:

- LP deletes its mapping in the DisCo-Registration by storing the "non-existing" value as described in the RELOAD base document [I-D.ietf-p2psip-base]. Afterwards, it fetches the lasted version of the DisCo-Registration to obtain all potential focus peers.

- LP calculates for all its participants the closest focus among all active and potential focus peer using the algorithm described in Section 4.6. LP then delegates all participants to those focus peers.

- LP then announces its leave by sending a NOTIFY to all its subscribers for the extended conference event package, setting its <focus> state to 'deleted'. Thereafter, it ends its own SIP conference dialog by sending by to its related focus peer.

Since the state synchronization topology in a distributed conference is commonly arranged in a spanning tree, a leave of a focus peer effects a gab in the tree structure. Those focus peers which had the leaving focus peer as their parent, are supposed to reconnect to the synchronization graph by subscribing the parent focus of the leaving peer.

6.4.2. On Unexpected Leave

If an unexpected leave is detected by a participant (e.g. missing signaling and/or media packets) it MUST repeat the joining procedure as described in Section 4.7.

7. DisCo Kind Definition

This section formally defines the DisCo kind.

Name
DISCO-REGISTRATION

Kind IDs
The Resource name DISCO-REGISTRATION Kind-ID is the AoR of the conference. The data stored is the DisCoRegistrationData, that contains the Node-ID of a peer acting as a focus for the conference and optionally a coordinates value describing a peer's relative network position.

Data Model
The data model for the DISCO-REGISTRATION Kind-ID is dictionary. The dictionary key is the Node-ID of the peer action as focus.

Access Control
USER-CHAIN-ACL

The data stored for the Kind-ID DISCO-REGISTRATION is of type DisCoRegistration. It contains a "coordinates" value, that describes the peers relative network position and the Node-ID of the registered focus peer.

8. XML Schema

The XML schema for the event package for distributed conferences is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ci="urn:ietf:params:xml:ns:conference-info"
     xmlns="urn:ietf:params:xml:ns:distributed-conference"
     targetNamespace="urn:ietf:params:xml:ns:distributed-conference"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">
<!--
    This imports the definitions in conference-info
-->
<xs:import namespace="urn:ietf:params:xml:ns:conference-info"
        schemaLocation="http://www.iana.org/assignments/xml-registry/
                        schema/conference-info.xsd"/>
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
     schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
  <!--
   A DISTRIBUTED CONFERENCE ELEMENT
-->
<xs:element name="distributed-conference"
            type="distributed-conference-type"/>
  <!--
    DISTRIBUTED CONFERENCE TYPE
-->
  <xs:complexType name="distributed-conference-type">
    <xs:sequence>
      <xs:element name="version-vector"
            type="version-vector-type" minOccurs="1"/>
      <xs:element name="conference-description"
      type="conference-description-type"
      minOccurs="0" maxOccurs="1"/>
      <xs:element name="focus"
      type="focus-type"
      minOccurs="0"
      maxOccurs="unbounded"/>
       <xs:any namespace="##other" processContents="lax"/>
     </xs:sequence>
     <xs:attribute name="state" type="ci:state-type"/>
     <xs:attribute name="entity" type="xs:anyURI"/>
     <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
<!--
    VERSION VECTOR TYPE
-->
  <xs:complexType name="version-vector-type">
```

```
      <xs:sequence>
        <xs:element name="version"
                    type="version-type"
                    minOccurs="1"
                    maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
      CONFERENCE DESCRIPTION TYPE
  -->
    <xs:complexType name="conference-description-type">
      <xs:sequence>
        <xs:element name="display-text"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="subject" type="xs:string" minOccurs="0"/>
        <xs:element name="free" type="xs:string" minOccurs="0"/>
        <xs:element name="keywords"
                    type="ci:keywords-type" minOccurs="0"/>
        <xs:element name="service-uris"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
      FOCUS TYPE
  -->
    <xs:complexType name="focus-type">
      <xs:sequence>
        <xs:element name="display-text"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="associated-aors"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:element name="roles"
                    type="ci:user-roles-type" minOccurs="0"/>
        <xs:element name="languages"
                    type="ci:user-languages-type" minOccurs="0"/>
        <xs:element name="focus-state"
                    type="focus-state-type" minOccurs="0"/>
        <xs:element name="users"
                    type="ci:users-type" minOccurs="0"/>
        <xs:element name="relations"
                    type="relations-type" minOccurs="0"/>
        <xs:any namespace="#other" processContents="lax"/>
      </xs:sequence>
```

```
      <xs:attribute name="entity" type="xs:anyURI"/>
      <xs:attribute name="node-id" type="xs:string"/>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  <!--
      VERSION TYPE
  -->
    <xs:complexType name="version-type">
      <xs:simpleContent>
        <xs:extension base="xs:unsignedInt">
          <xs:attribute name="entity" type="xs:anyURI"/>
          <xs:attribute name="node-id" type="xs:string"/>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <!--
      FOCUS STATE TYPE
  -->
    <xs:complexType name="focus-state-type">
      <xs:sequence>
        <xs:element name="user-count"
                    type="xs:unsignedInt" minOccurs="0"/>
        <xs:element name="coordinate"
                    type="xs:string" minOccurs="0"/>
        <xs:element name="maximal-user-count"
                    type="xs:unsignedInt" minOccurs="0"/>
        <xs:element name="conf-uris"
                    type="ci:uris-type" minOccurs="0"/>
        <xs:element name="available-media"
                    type="ci:conference-media-type" minOccurs="0"/>
        <xs:element name="active" type="xs:boolean" minOccurs="0"/>
        <xs:element name="locked" type="xs:boolean" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
      RELATIONS TYPE
  -->
    <xs:complexType name="relations-type">
      <xs:sequence>
        <xs:element name="relation"
                    type="relation-type"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax"/>
```

```
      </xs:sequence>
      <xs:attribute name="state" type="ci:state-type"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
      RELATION TYPE
  -->
    <xs:complexType name="relation-type">
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="entity" type="xs:anyURI"/>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>
```

9. Relax NG Grammar

The grammar for the Landmark configuration document extension is:

```
<!--
    LANDMARKS ELEMENT
-->
parameter &= element landmarks {
    attribute version { xsd:int }
    <!--
        LANDMARK-HOST ELEMENT
    -->
    element landmark-host {
        attribute address { xsd:string },
        attribute port { xsd:int }
    }*
}?
```

10. Security Considerations

10.1. Trust Aspects

TODO

11. IANA Considerations

TODO: register Kind-ID code point at the IANA

12. Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and SAM research group. We would like to thank all active members for constructive thoughts and feedback. In particular, the authors would like to thank (in alphabetical order) David Bryan, Toerless Eckert, Lothar Grimm, Cullen Jennings, Peter Musgrave, Joerg Ott, Peter Pogrzeba, Brian Rosen, and Jan Seedorf.

13. References

13.1. 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-21 (work in progress), March 2012.

[I-D.knauf-p2psip-share]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A Usage for Shared Resources in RELOAD (ShaRe)", draft-knauf-p2psip-share-03 (work in progress), April 2012.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

[RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

13.2. Informative References

[I-D.ietf-p2psip-concepts]
Bryan, D., Willis, D., Shim, E., Matthews, P., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-04 (work in progress), October 2011.

[I-D.ietf-p2psip-sip]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-07 (work in progress), January 2012.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[landmarks-infocomm02]
Ratnasamy, Handley, Karp, and Shenker, "Topologically-Aware Overlay Construction and Server Selection", Proc. of 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02) pp. 1190-1199, 2002.

[timestamps-acsc88]
Fidge, C., "Timestamps in Message-Passing Systems that Preserve the Partial Ordering", Proceedings of 11th Australian Computer Science Conference, pp. 56-66, February 1988.

Appendix A. Change Log

The following changes have been made from version draft-knauf-p2psip-disco-04.

1. Editorial improvements.

2. Updated references.

The following changes have been made from version draft-knauf-p2psip-disco-03.

1. Adapted mechanisms for storing DisCo-Registrations to new requirements of Shared Resources draft [I-D.knauf-p2psip-share]

The following changes have been made from version draft-knauf-p2psip-disco-02.

1. DisCo-Registration uses now only the USER-CHAIN-ACL access control policy.

2. Adapted mechanisms for storing DisCo-Registrations to new requirements of Shared Resources draft [I-D.knauf-p2psip-share]

The following changes have been made from version draft-knauf-p2psip-disco-01.

1. The conference registration is now based on the Shared Resources draft [I-D.knauf-p2psip-share]:

   * DisCo-Registration Kind now meets the requirements for ShaRe.

   * Conference creation procedure now uses the ShaRe Access List.

   * Replaced USER-CHAIN-MATCH access policy for DisCo-Registration. Now uses USER-CHAIN-ACL or USER-PATTERN-MATCH.

2. Allow focus peers behind NAT

3. Added a 'node-id' attribute to the event package XML <version> element.

4. Added a 'node-id' attribute to the event package XML <focus> element.

5. Added a 'coordinate' child element to the event package XML <focus> element.

6. Corrected typos/wording

The following changes have been made from version draft-knauf-p2psip-disco-00.

1. Updated references.

2. Corrected typos.

3. New Section: Conference State Synchronization

4. XML Event Package for Distributed Conferences

5. New mechanism for generating chained conference certificates

6. Allow shared writing of resources using Access Control Policy USER-CHAIN-MATCH

7. Media Negotiation mechanism in a distributed conference

8. New Section: Distributed Conference Control with SIP

Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: alexanderknauf@gmail.com
URI: http://inet.cpt.haw-hamburg.de/members/knauf

Gabriel Hege
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: hege@fhtw-berlin.de
URI: http://inet.cpt.haw-hamburg.de/members/hege

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Email: schmidt@informatik.haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/schmidt

Matthias Waehlisch
link-lab & FU Berlin
Hoenower Str. 35
Berlin D-10318
Germany

Email: mw@link-lab.net
URI: http://www.inf.fu-berlin.de/~waehl

P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin

July 11, 2011

A Usage for Shared Resources in RELOAD (ShaRe)
draft-knauf-p2psip-share-01

Abstract

This document defines a RELOAD Usage for managing shared write access to RELOAD Resources. Shared Resources in RELOAD (ShaRe) form a basic primitive for enabling various coordination and notification schemes among distributed peers. Access in ShaRe is controlled by a hierarchical trust delegation scheme maintained within an access list. A new USER-CHAIN-ACL access policy allows authorized peers to write a Shared Resource without owning its corresponding certificate. This specification also adds mechanisms to store Resources with a variable name which is useful whenever peer-independent rendezvous processes are required.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 12, 2012.

Copyright Notice

Copyright (c) 2011 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. Shared Resources in RELOAD 5
3.1. Mechanisms for Isolating Stored Data 6
4. Access Control List Definition 7
4.1. Overview 7
4.2. Data Structure 8
5. Extension for Variable Resource Names 11
5.1. Overview 11
5.2. Overlay Configuration Document Extension 11
6. Access Control to Shared Resources 13
6.1. Granting Write Access 13
6.2. Revoking Write Access 13
6.3. Storage and Validation 14
6.3.1. Operations of the Storing Peer 14
6.3.2. Operations of the Accessing Peer 15
6.4. USER-CHAIN-ACL Access Policy 16
7. Security Considerations 17
7.1. Resource Exhaustion 17
7.2. Malicious or Misbehaving Storing Peer 17
7.3. Privacy Issues 17
8. IANA Considerations 18
9. Acknowledgments 19
10. References 20
10.1. Normative References 20
10.2. Informative References 20
Appendix A. Change Log 21
Authors' Addresses 22

1. Introduction

This document defines a RELOAD Usage for managing shared write access to RELOAD Resources and a mechanism to store Resources with a variable name. The Usage for Shared Resources in RELOAD (ShaRe) enables overlay users to share their exclusive write access to specific Resource/Kind pairs with others. Shared Resources form a basic primitive for enabling various coordination and notification schemes among distributed peers. Write permission is controlled by an Access Control List (ACL) Kind that maintains a chain of Authorized Peers for a particular Shared Resource. A newly defined USER-CHAIN-ACL access control policy enables shared write access in RELOAD.

The Usage for Shared Resources in RELOAD is designed for jointly coordinated group applications among distributed peers (c.f. [I-D.knauf-p2psip-disco]). Of particular interest are rendezvous processes, where a single identifier is linked to multiple, dynamic instances of a distributed cooperative service. Shared write access is based on a trust delegation mechanism. It transfers the authorization to write a specific Kind data by storing logical Access Control Lists. An ACL contains the ID of the Kind to be shared and contains trust delegations from one authorized to another (previously unauthorized) user.

Shared write access extends the RELOAD security model, which is based on the restriction that peers are only allowed to write resources at a small set of well defined locations (Resource IDs) in the overlay. Using the standard access control rules in RELOAD, these locations are bound to the username or Node Id in the peer's certificate. This document extends the base policies to enable a controlled write access for multiple users to a common Resource Id.

Additionally, this specification defines an optional mechanism to store Resources with a variable Resource Name. It enables the storage of Resources whose name complies to a specific pattern. Definition of the pattern is arbitrary, but must contain the username of the Resource creator.

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 [RFC2119].

This document uses the terminology and definitions from the RELOAD base [I-D.ietf-p2psip-base]and the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts]. Additionally, the following terms are used:

Shared Resource: The term Shared Resource in this document defines a RELOAD Resource with its associated Kinds, that can be written or overwritten by multiple RELOAD users following the specifications in this document.

Access Control List: The term Access Control List in this document defines a logical list of RELOAD users allowed to write a specific RELOAD Resource/Kind pair by following the specifications in this document. The list items are stored as Access Control List Kinds that map trust delegations from user A to user B, where A is allowed to write a Shared Resource and the Access Control List, while B is a user that obtains write access to specified Kinds from A.

Resource Owner: The term Resource Owner in this document defines a RELOAD peer that initially stored a Resource to be shared. The Resource Owner possesses the RELOAD certificate that grants write access to a specific Resource/Kind pair using the RELOAD certificate based access control policies.

Authorized Peer: The term Authorized Peer in this document defines a RELOAD peer that was granted write access to a Shared Resource by permission of the Resource Owner or another Authorized Peer.

3. Shared Resources in RELOAD

A RELOAD user that owns a certificate for writing at a specific overlay location can maintain one or more RELOAD Kinds that are designated for a non-exclusive write access shared with other RELOAD users. The mechanism to share those Resource/Kind pairs with a group of users consists of two basic steps: Storage of the Resource/Kind pair to be shared and storage of an Access Control List (ACL) associated with those Kinds. ACLs are initiated by the Resource Owner and contain ACL items, each delegating the permission of writing the shared Kind to a specific user called Authorized Peer. This trust delegation to the Authorized Peer can include the right to further delegate the write permission. For each shared Kind data, the Resource owner stores a root item that starts an Access Control List. The result is a tree of trust delegations with the Resource Owner as trust anchor.

The Resource/Kind pair to be shared can be any RELOAD Kind that complies to the following specifications:

Isolated Data Storage: To ensure that concurrent writing does not cause race conditions, each data item stored within a Shared Resource needs to be exclusively maintained by the RELOAD user who created it. Hence, Usages that allow the storage of Shared Resources MUST use either array or dictionary data model and require the additional mechanisms for isolating data as described in Section 3.1.

Access Control Policy: To ensure write access to Shared Resource by Authorized Peers, each Usage MUST use the USER-CHAIN-ACL access policy (see Section 6.4).

Resource_name field: Any Kind using the USER-CHAIN-MATCH policy MUST define an opaque <0..2^16-1> initial field within the Kind data structure definition. It contains the Resource Name of the Kind data to be stored. The Resource_name field allows any receicer of a shared data to validate if the Resource Name hashes to the Resource-ID of Shared Resource.

User_name field: Any a Kind using the USER-CHAIN-ACL policy MUST define an opaque <0..2^16-1> as second field within the Kind data structure definition. It contains the username from the certificate of the signer of the data to be stored. The User_name field allows any receiver of the data to request the public key certificate of the originator of the stored data and to verify his provenance and integrity.

3.1. Mechanisms for Isolating Stored Data

This Section defines mechanisms to avoid race conditions while concurrently writing an array or dictionary of a Shared Resource.

If a dictionary is used in the Shared Resource, the dictionary key MUST be the Node-ID of the certificate that will be used to sign the stored data.

If the data model of the Shared Resource is an array, the following algorithm will generate an array index that avoids collisions.

1. Obtain the Node-ID of the certificate that will be used to sign the stored data

2. Take the least significant 24 bits of that Node-ID

3. Concatenate an 8 bit long short individual index value to those 24 bit of the Node-ID

The resulting 32 bits long integer MUST be used as the index for storing an array entry in a Shared Resource. The 8 bit individual index can be incremented by one for each further array entry stored and allows for 256 distinct entries per Peer.

The mechanism to create the array index is related to the pseudo-random algorithm to generate an SSRC identifier in RTP, see Section 8.1 in [RFC3550] for calculating a collision probability.

4. Access Control List Definition

4.1. Overview

In this document, an Access Control List (ACL) contains a list of AccessControlListData structures defined in Section 4.2. Each entry delegates write access for a specific Kind data to a single RELOAD user. The information is stored at the same overlay location as the Shared Resource. An ACL allows the RELOAD user who is authorized to write a specific Resource-ID to delegate his exclusive write access for the specified Kinds to further users of a RELOAD instance. Each Access Control List data structure therefore carries the information about who delegates write access to whom, the Kind-ID of the Resource to be shared, and whether delegation includes write access to the ACL itself. The latter condition grants the right to delegate write access further for an Authorized Peer. Access Control Lists are stored within a RELOAD array data model. They are initially created by the Resource Owner.

Figure 1 shows an example of an Access Control List. We omit the resource_name field to simplify illustration. The array entry at index 0x123abc001 displays the initial entry of an ACL about a Shared Resource of Kind-ID 1234 at the same Resource-ID. It represents the root item of the trust delegation tree for this shared RELOAD Kind. The root entry MUST contain the mapping from Resource owner to Resource owner and MUST only be written by the owner of the public key certificate associated with this Resource-ID. The array index is generated by using the mechanism for isolating stored data as described in Section 3.1. Hence, the most significant 24 bits of the array index (0x123abc) are the least significant 24bits of the Node-ID of the Resource Owner.

The array item at index 0x123abc002 represents the first trust delegation to an Authorized Peer, which is thus permitted write access to the Shared Resource of Kind-ID 1234. Additionally, the Authorized peer Alice is also granted (limited) write access to the ACL as indicated by the allow_delegation flag (ad) set to 1. This configuration authorizes Alice to store further trust delegations to the Shared Resource i.e., add items to the ACL. In contrast to this, index 0x456def001 illustrates trust delegation for Kind-ID 1234, in which the Authorized Peer Bob is not allowed to grant access to further peers (ad = 0). Each Authorized Peer signs its ACL items with its own private key.

In order to maintain Shared Resource access for multiple Kinds at a single location, the Resource Owner can create new ACL entries that refer to another Kind-ID as shown in array entry index 0x123abc003. Note, that overwriting existing items in an Access Control List that

reference a different Kind-ID revokes all trust delegations in the tree (see Section 6.2). Hence, Authorized Peers are not enabled to overwrite any existing ACL item . The Resource Owner is allowed to overwrite existing ACL items, but should be aware of its consequences.

Access Control List		
#Index	Array Entries	signed by
123abc001	user:Owner -> to:Owner Kind:1234 ad:1	Owner
123abc002	user:Owner -> to:Alice Kind:1234 ad:1	Owner
123abc003	user:Owner -> to:Owner Kind:4321 ad:1	Owner
123abc004	user:Owner -> to:Carol Kind:4321 ad:0	Owner
...	...	...
456def001	user:Alice -> to:Bob Kind:1234 ad:0	Alice
...	...	...

Figure 1: Simplified example of an Access Control including entries for two different Kind-IDs and varying delegation (ad) configurations

Implementations of ShaRe should be aware that the trust delegation in an Access Control List need not be loop free. Self-contained circular trust delegation from A to B and B to A are possible, even though not very meaningful.

4.2. Data Structure

The Kind data structure for the Access Control List is defined as follows:

```
struct {
  opaque resource_name<0..2^16-1>;
  opaque user_name<0..2^16-1>;
  opaque to_user<0..2^16-1>;
  KindId kind;
  Boolean allow_delegation;
} AccessControlListData;

struct {
  uint16 length;
  AccessControlListData data;
} AccessControlListItem;
```

The AccessControlListItem structure is composed of:

length: This field contains the length of the Access Control List data structure

data: This field contains the data of an ACL item

The content of the AccessControlListData structure is defined as follows:

resource_name: This opaque string represents the Resource Name of the Shared Resource as an opaque string. It is used by the storing peer to validate whether a variable Resources Name matches the corresponding pattern defined in the configuration document.

user_name: This field contains the username of that RELOAD peer the grants write permission to the Shared Resource. The username is stored as an opaque string and contains the username value from the certificate that is associated with the private key that signed this Access Control List item.

to_user: This field contains the username of the RELOAD peer that obtains writing permission to the Shared Resource.

kind: This field contains the Kind-ID of the Shared Resource.

allow_delegation: If true, this Boolean flag indicates that the Authorized Peer in the 'to_user' field is allowed to add additional entries into the ACL for the specified Kind-ID.

The ACCESS-CONTROL-LIST kind is defined as follows:

Name: ACCESS-CONTROL-LIST

Data model: The Data model for the ACCESS-CONTROL-LIST data is array.

Access Control: USER-CHAIN-ACL access policy (see Section 6.4).

5. Extension for Variable Resource Names

5.1. Overview

In certain use cases such as conferencing (c.f. [I-D.knauf-p2psip-disco]) it is desirable to extend the set of Resource Names and thus Resource-IDs a peer is allowed to write beyond those defined through the username or Node-ID fields in its certificate. Therefore, this document presents the concept for variable Resources Names that enables providers of RELOAD instances to define a relaxed naming scheme for overlay Resources.

Each RELOAD node uses a certificate to identify itself using its user name (or Node-ID) while storing data under a specific Resource-ID. The specifications in this document scheme follows this paradigm by allowing to store values whose Resource Names are derived from the username in the certificate of a RELOAD peer. This is done by using a Resource Name which contains a variable substring but matches the username in the certificate using a pattern defined in the overlay configuration document. Thus despite being variable an allowed Resource Name is closely related to the Owner's certificate. A sample pattern might be formed as the following:

Example Pattern:
.*-conf-$USER@$DOMAIN

When defining the pattern care must be taken that no conflict arises for two usernames of witch one is a substring of the other. In this case the peer with the name which is the substring could choose the variable part of the Resource Name so that the resulting string contains the whole other username and thus he could write the other user's resources. This can easily be prevented by delimiting the variable part of the pattern from the username part by some fixed string, that is usually not part of a username (e.g. the "-conf-" in the above Example).

5.2. Overlay Configuration Document Extension

This document extends the overlay configuration document by defining new elements for patterns relating resource names to user names.

The <variable-resource-names> element serves as a container for one or multiple <pattern> sub-elements. It is an additional parameter within the kind block.

Each <pattern> element defines the pattern to be used for a single Kind. It is of type xsd:string, which is interpreted as a regular expression. In the regular expression $USER and $DOMAIN are used as

variables for the corresponding parts of the string in the certificate username field ($USER before and $DOMAIN after the '@'). Both variables MUST be present in any given pattern.

A <pattern> element MUST be present for every Kind using the USER-CHAIN-MATCH access policy Section 6.4.

The Relax NG Grammar for the Variable Resource Names Extension is:

```
<!-- VARIABLE RESOURCE URN SUB-NAMESPACE -->

namespace share = "urn:ietf:params:xml:ns:p2p:config-base:share"

<!-- VARIABLE RESOURCE NAMES ELEMENT -->

kind-parameter &= element share:variable-resource-names {

    <!-- PATTERN ELEMENT -->

    element pattern { xsd:string }*
}?
```

6. Access Control to Shared Resources

6.1. Granting Write Access

Write access to a Kind that is intended to be shared with other RELOAD users can solely be issued by the Resource Owner. A Resource Owner can share RELOAD Kinds by using the following procedure:

- The Resource Owner stores an ACL root item at the same Resource-ID as the Shared Resource. The root item contains the Resource Name and Kind-ID of the Shared Resource in the "resource_name" field and "Kind-ID" fields. It further contains the username of the Resource Owner in the "user_name" and "to_user" field. The "ad" flag SHOULD be set to 1. The array index of MUST be generated as described in Section 3.1

- Further ACL items stored by the Resource Owner will delegate write access to the Shared Resource. These ACL items contain the same Resource Name, Kind-ID and "user_name" values as the root item. Each "to_user" field is set to the username of an Authorized Peer. Optionally, the Resource Owner sets the "ad" to 1 (we define 0 as default). Each succeeding ACL item created by the Resource Owner MAY be stored in the numerical order of the array index starting with the index of the root item plus one.

An Authorized Peer (with "ad"=1) can further share an exiting Shared Resource as follows:

- An Authorized Peer stores one or more ACL items at the Resource-ID of the Shared Resource. These ACL items contain the shared Resource Name and Kind-Id in the "resource_name" field and "Kind-ID" fields. The "user_name" value is set to the username of the Authorized Peer and the "to_user" value is set to username of the added Authorized Peer. Optionally, the "ad" flag could be set to 1. The array index of MUST be generated as described in Section 3.1. Each succeeding ACL item MAY be stored in the numerical order of the array index.

6.2. Revoking Write Access

Write permissions MAY be revoked by soring a non-existent value [I-D.ietf-p2psip-base] to the corresponding item in the Access Control List. A revoked permission automatically invalidates all delegations performed by that user and also all subsequent delegations. This allows to invalidate entire subtrees of the delegations tree with only a single operation. Overwriting the root item with a non-existent value of an Access List invalidates the entire delegations tree.

An ACL item MUST only be written by the user who initially stored the corresponding entry. The only exception is by the Resource Owner that is allowed to overwrite every single ACL item for revoking write access.

6.3. Storage and Validation

6.3.1. Operations of the Storing Peer

The storing peer (the peer at which Shared Resource and Access List are physically stored) is responsible to control storing attempts on a Shared Resource and the corresponding Access Control List. To assert the USER-CHAIN-ACL (see Section 6.4) access policy a storing peer need to implement several operations to validate a store attempt on a Shared Resource. The following ACL validation mechanism has to be performed on an incoming store request to a Shared Resource by an Authorized Peer:

1. The storing peer sends a Stat request to Resource-ID of the Shared Resource to obtain all array indexes of stored ACL Kinds.

2. Using the retrieved meta information, the storing peer fetches for all ACL Kinds at this Resource-ID. Since the ACL is stored at the storing peer, both operations are local lookups.

3. The storing peer then validates if any of the retrieved ACL items contains a user_name field whose value is equal to the username in the certificate of the Authorized Peer and, if the Kind value is equal the Kind-ID of the Shared Resource.

4. If the requested Shared Resource is an Access Control List Kind, the storing peer MUST further validate if the "ad" flag is set to 1.

5. If true, the ACL validation procedure returns true.

A storing peer MAY validate the entire ACL to a Shared Resource on an inbound store request. Beginning at the ACL item from the above ACL validation procedure, the storing peer validates if any ACL item with the same Kind-ID has a "user_name" value equal to the "to_user" value of recently validated ACL item. If true, the validation continues with the next ACL item with any other ACL item with the same Kind-ID. This procedure continues until an ACL item is found with "user_name" is equal "to_user". Then, if this "user_name" value hashes to the Resource-ID or it matches the variable resource name pattern and the corresponding "resource_name" value hashes to the Resource-ID, the ACL root validation returns true.

If a store request by an Authorized Peer would overwrite and ACL item from another Authorized Peer (or Resource Owner), an Error_Forbidden response MUST be returned. The probability of a collision of two or more identical array index will be negligibly low using the mechanism for isolated stored data (see Section 3.1)

6.3.2. Operations of the Accessing Peer

The accessing peer (a RELOAD peer that fetches a Shared Resource) SHOULD validate whether its originator was allowed to store at this Resource-ID by processing the corresponding ACL. The accessing peer previously performs Stat request to retrieve the meta information at the Resource-ID of the Shared Resource. Then, it fetches all existing ACL Kinds at this location. After retrival, the accessing peer compares the "to_user" value of each ACL item with the mandatory "user_name" field of the Shared Resource for equality. If the comparison fails, the accessing peer MUST ignore the data of the retrieved Shared Resource.

Else, the accessing peer MAY validate whether the "user_name" value of recently observed ACL item exist in any other "to_user" field in the corresponding ACL. If true, this procedure continues with the next ACL item with any other ACL item of the list. The procedure contiues until both "user_name" and "to_user" values are equal, thus the root item to the ACL is found. The Shared Resource and corresponding ACL are permitted at this location if the hash over the "user_name" value equals to the Resource-ID. Or otherwise, if the "user_name" value matches the corresponding variable resource name pattern defined in the configuration document and the hash over the corresponding "resource_name" value matches the Resource-ID

The accessing peer can further verify provenance and integrity of the retrieved Shared Resource using the certificate corresponding to the mandatory user_name field of the Shared Resource entry. The certificate can be retrieved by applying the Certificate Usage [I-D.ietf-p2psip-base] or other means (e.g., caching from a previous request).

The accessing peer MAY cache previously fetched Access Control Lists to a maximum of the individual items' lifetimes. Since stored values could have been changed or invalidated prior to their expiration an accessing peer uses a Stat request to check for updates before using the cached data. If a change has been detected it fetches the latest Access Control List.

6.4. USER-CHAIN-ACL Access Policy

This document specifies an additional access control policy to the RELOAD base draft [I-D.ietf-p2psip-base]. The USER-CHAIN-ACL policy allows Authorized Peers to write a Shared Resource, even though they do not own the corresponding certificate. Additionally, the USER-CHAIN-MATCH allows the storage of Kinds with a variable resource name that are following the specified naming pattern. Hence, on an inbound store request on a Kind that uses the USER-CHAIN-ACL access policy, the following rules MUST applied:

In the USER-CHAIN-ACL policy, a given value MUST be written or overwritten if either one of USER-MATCH or USER-NODE-MATCH (mandatory if the data model is dictionary) access policies of the base document [I-D.ietf-p2psip-base] applies.

Otherwise, the value MUST be written if the certificate of the signer contains a username that matches to the variable resource name pattern (c.f. Section 5) specified in the configuration document and, additionally, the hashed Resource Name matches the Resource-ID. The Resource Name of the Kind been stored MUST be taken from the mandatory resource_name field in the corresponding Kind data structure.

Otherwise, the value MUST be written if the ACL Validation procedure as described in Section 6.3.1 returns true.

7. Security Considerations

In this section we discuss security issues that are relevant to the usage of shared resources in RELOAD.

7.1. Resource Exhaustion

Joining a RELOAD overlay inherently poses a certain resource load on a peer, because it has to store and forward data for other peers. In common RELOAD semantics, each Resource ID and thus position in the overlay may only be written by a limited set of peers - often even only a single peer, which limits this burden. In the case of Shared Resources, a single resource may be written by multiple peers, who may even write an arbitrary number of entries (e.g., delegations in the ACL). This leads to an enhanced use of resources at individual overlay nodes. The problem of resource exhaustion can easily be mitigated for Usages based on the ShaRe-Usage by imposing restrictions on size, i.e., <max-size> element for a certain Kind in the configuration document.

7.2. Malicious or Misbehaving Storing Peer

The RELOAD overlay is designed to operate despite the presence of a small set of misbehaving peers. This is not different for Shared Resources since a small set of malicious peers does not disrupt the functionality of the overlay in general, but may have implications for the peers needing to store or access information at the specific locations in the ID space controlled by a malicious peer. A storing peer could withhold stored data which results in a denial of service to the group using the specific resource. But it could not return forged data, since the validity of any stored data can be independently verified using the attached signatures.

7.3. Privacy Issues

All data stored in the Shared Resource is publicly readable, thus applications requiring privacy need to encrypt the data. The ACL needs to be stored unencrypted, thus the list members of a group using a Shared Resource will always be publicly visible.

8. IANA Considerations

TODO: register Kind-ID code point at the IANA

9. Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and SAM research group. We would like to thank all active members for constructive thoughts and feedback. In particular, the authors would like to thank (in alphabetical order) Lothar Grimm, Cullen Jennings, Peter Musgrave, Joerg Ott, Marc Petit-Huguenin, Peter Pogrzeba, and Jan Seedorf. This work was party funded by the German Federal Ministry of Education and Research, projects HAMcast and Mindstone.

10. References

10.1. 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-16 (work in progress), July 2011.

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

10.2. 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-03 (work in progress), October 2010.

[I-D.knauf-p2psip-disco]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A RELOAD Usage for Distributed Conference Control (DisCo)", draft-knauf-p2psip-disco-02 (work in progress), March 2011.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

Appendix A. Change Log

The following changes have been made from version draft-knauf-p2psip-share-00:

1. Integrated the USER-PATTERN-MATCH access policy into USER-CHAIN-MATCH

2. Access Control List Kind uses USER-CHAIN-ACL exclusively

3. Resources to be shared use USER-CHAIN-ACL exclusively

4. More precise specification of mandatory User_name and Resource_name fields for Shared Resources

5. Added mechanism for isolating stored data to prevent race conditions while concurrent storing

6. XML Extension for variable resource names uses its own namespace

7. Many editorial improvements

Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: alexander.knauf@haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/knauf

Gabriel Hege
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: hege@fhtw-berlin.de
URI: http://inet.cpt.haw-hamburg.de/members/hege

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Email: schmidt@informatik.haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/schmidt

Matthias Waehlisch
link-lab & FU Berlin
Hoenower Str. 35
Berlin D-10318
Germany

Email: mw@link-lab.net
URI: http://www.inf.fu-berlin.de/~waehl

P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 27, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin

April 25, 2012

A Usage for Shared Resources in RELOAD (ShaRe)
draft-knauf-p2psip-share-03

Abstract

This document defines a RELOAD Usage for managing shared write access to RELOAD Resources. Shared Resources in RELOAD (ShaRe) form a basic primitive for enabling various coordination and notification schemes among distributed peers. Access in ShaRe is controlled by a hierarchical trust delegation scheme maintained within an access list. A new USER-CHAIN-ACL access policy allows authorized peers to write a Shared Resource without owning its corresponding certificate. This specification also adds mechanisms to store Resources with a variable name which is useful whenever peer-independent rendezvous processes are required.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on October 27, 2012.

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. Shared Resources in RELOAD 5
3.1. Mechanisms for Isolating Stored Data 6
4. Access Control List Definition 7
4.1. Overview 7
4.2. Data Structure 8
5. Extension for Variable Resource Names 10
5.1. Overview 10
5.2. Data Structure 10
5.3. Overlay Configuration Document Extension 11
6. Access Control to Shared Resources 13
6.1. Granting Write Access 13
6.2. Revoking Write Access 14
6.3. Validating Write Access through an ACL 14
6.4. Operations of Storing Peers 15
6.5. Operations of Accessing Peers 15
6.6. USER-CHAIN-ACL Access Policy 15
7. ACL Kind Definition 17
8. Security Considerations 18
8.1. Resource Exhaustion 18
8.2. Malicious or Misbehaving Storing Peer 18
8.3. Privacy Issues 18
9. IANA Considerations 19
10. Acknowledgments 20
11. References 21
11.1. Normative References 21
11.2. Informative References 21
Appendix A. Change Log 22
Authors' Addresses 23

1. Introduction

This document defines a RELOAD Usage for managing shared write access to RELOAD Resources and a mechanism to store Resources with a variable name. The Usage for Shared Resources in RELOAD (ShaRe) enables overlay users to share their exclusive write access to specific Resource/Kind pairs with others. Shared Resources form a basic primitive for enabling various coordination and notification schemes among distributed peers. Write permission is controlled by an Access Control List (ACL) Kind that maintains a chain of Authorized Peers for a particular Shared Resource. A newly defined USER-CHAIN-ACL access control policy enables shared write access in RELOAD.

The Usage for Shared Resources in RELOAD is designed for jointly coordinated group applications among distributed peers (e.g., third party registration, or distributed conferencing, see[I-D.knauf-p2psip-disco]). Of particular interest are rendezvous processes, where a single identifier is linked to multiple, dynamic instances of a distributed cooperative service. Shared write access is based on a trust delegation mechanism. It transfers the authorization to write a specific Kind data by storing logical Access Control Lists. An ACL contains the ID of the Kind to be shared and contains trust delegations from one authorized to another (previously unauthorized) user.

Shared write access augments the RELOAD security model, which is based on the restriction that peers are only allowed to write resources at a small set of well defined locations (Resource-IDs) in the overlay. Using the standard access control rules in RELOAD, these locations are bound to the username or Node-ID in the peer's certificate. This document extends the base policies to enable a controlled write access for multiple users to a common Resource Id.

Additionally, this specification defines an optional mechanism to store Resources with a variable Resource Name. It enables the storage of Resources whose name complies to a specific pattern. Definition of the pattern is arbitrary, but must contain the username of the Resource creator.

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 [RFC2119].

This document uses the terminology and definitions from the RELOAD base [I-D.ietf-p2psip-base]and the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts]. Additionally, the following terms are used:

Shared Resource: The term Shared Resource in this document defines a RELOAD Resource with its associated Kinds, that can be written or overwritten by multiple RELOAD users following the specifications in this document.

Access Control List: The term Access Control List in this document defines a logical list of RELOAD users allowed to write a specific RELOAD Resource/Kind pair by following the specifications in this document. The list items are stored as Access Control List Kinds that map trust delegations from user A to user B, where A is allowed to write a Shared Resource and the Access Control List, while B is a user that obtains write access to specified Kinds from A.

Resource Owner: The term Resource Owner in this document defines a RELOAD peer that initially stored a Resource to be shared. The Resource Owner possesses the RELOAD certificate that grants write access to a specific Resource/Kind pair using the RELOAD certificate-based access control policies.

Authorized Peer: The term Authorized Peer in this document defines a RELOAD peer that was granted write access to a Shared Resource by permission of the Resource Owner or another Authorized Peer.

3. Shared Resources in RELOAD

A RELOAD user that owns a certificate for writing at a specific overlay location can maintain one or more RELOAD Kinds that are designated for a non-exclusive write access shared with other RELOAD users. The mechanism to share those Resource/Kind pairs with a group of users consists of two basic steps.

1. Storage of the Resource/Kind pairs to be shared.

2. Storage of an Access Control List (ACL) associated with those Kinds.

ACLs are created by the Resource Owner and contain ACL items, each delegating the permission of writing the shared Kind to a specific user called Authorized Peer. For each shared Kind data, its Resource owner stores a root item that initiates an Access Control List. Trust delegation to the Authorized Peer can include the right to further delegate the write permission, enabling a tree of trust delegations with the Resource Owner as trust anchor at its root.

The Resource/Kind pair to be shared can be any RELOAD Kind that complies to the following specifications:

Isolated Data Storage: To prevent concurrent writing from race conditions, each data item stored within a Shared Resource SHALL be exclusively maintained by the RELOAD user who created it. Hence, Usages that allow the storage of Shared Resources are REQUIRED to use either the array or dictionary data model and apply additional mechanisms for isolating data as described in Section 3.1.

Access Control Policy: To ensure write access to Shared Resource by Authorized Peers, each Usage MUST use the USER-CHAIN-ACL access policy as described in Section 6.6.

Resource Name Extension: To enable Shared Resources to be stored using a variable resource name, this document defines an optional ResourceNameExtension structure. It contains the Resource Name of the Kind data to be stored and allows any receiver of a shared data to validate whether the Resource Name hashes to the Resource-ID. The ResourceNameExtension is made optional by configuration. The ResourceNameExtension field is only present in the Kind data structure when configured in the corresponding kind-block of the overlay configuration document (for more details see Section 5.3). If the configuration allows variable resource names, a Kind using the USER-CHAIN-ACL policy MUST use the ResourceNameExtension as initial field within the Kind data

structure definition. Otherwise the Kind data structure does not contain the ResourceNameExtension structure.

3.1. Mechanisms for Isolating Stored Data

This section defines mechanisms to avoid race conditions while concurrently writing an array or dictionary of a Shared Resource.

If a dictionary is used in the Shared Resource, the dictionary key MUST be the Node-ID of the certificate that will be used to sign the stored data. Thus data access is bound to the unique ID-holder.

If the data model of the Shared Resource is an array, the following algorithm will generate an array index that avoids collisions.

1. Obtain the Node-ID of the certificate that will be used to sign the stored data

2. Take the least significant 24 bits of that Node-ID

3. Concatenate an 8 bit long short individual index value to those 24 bit of the Node-ID

The resulting 32 bits long integer MUST be used as the index for storing an array entry in a Shared Resource. The 8 bit individual index can be incremented individually for further array entries and allows for 256 distinct entries per Peer.

The mechanism to create the array index is related to the pseudo-random algorithm to generate an SSRC identifier in RTP, see Section 8.1 in [RFC3550] for calculating the probability of a collision.

4. Access Control List Definition

4.1. Overview

An Access Control List (ACL) is a (self-managed) shared resource that contains a list of AccessControlListItem structures as defined in Section 4.2. Each entry delegates write access for a specific Kind data to a single RELOAD user. An ACL enables the RELOAD user who is authorized to write a specific Resource-ID to delegate his exclusive write access to a specific Kind to further users of a RELOAD instance. Each Access Control List data structure therefore carries the information about who obtains write access, the Kind-ID of the Resource to be shared, and whether delegation includes write access to the ACL itself. The latter condition grants the right to delegate write access further for the Authorized Peer. Access Control Lists are stored at the same overlay location as the Shared Resource and use the RELOAD array data model. They are initially created by the Resource Owner.

Figure 1 shows an example of an Access Control List. We omit the res_name_ext field to simplify illustration. The array entry at index 0x123abc001 displays the initial creation of an ACL for a Shared Resource of Kind-ID 1234 at the same Resource-ID. It represents the root item of the trust delegation tree for this shared RELOAD Kind. The root entry MUST contain the username of the Resource owner in the "to_user" field and can only be written by the owner of the public key certificate associated with this Resource-ID. The allow_delegation (ad) flag for a root ACL item is set to 1 by default. The array index is generated by using the mechanism for isolating stored data as described in Section 3.1. Hence, the most significant 24 bits of the array index (0x123abc) are the least significant 24 bits of the Node-ID of the Resource Owner.

The array item at index 0x123abc002 represents the first trust delegation to an Authorized Peer that is thus permitted to write to the Shared Resource of Kind-ID 1234. Additionally, the Authorized peer Alice is also granted (limited) write access to the ACL as indicated by the allow_delegation flag (ad) set to 1. This configuration authorizes Alice to store further trust delegations to the Shared Resource, i.e., add items to the ACL. On the contrary, index 0x456def001 illustrates trust delegation for Kind-ID 1234, in which the Authorized Peer Bob is not allowed to grant access to further peers (ad = 0). Each Authorized Peer signs its ACL items with its own private key, which makes the item ownership transparent.

To manage Shared Resource access of multiple Kinds at a single location, the Resource Owner can create new ACL entries that refer to another Kind-ID as shown in array entry index 0x123abc003. Note that

overwriting existing items in an Access Control List that reference a different Kind-ID revokes all trust delegations in the corresponding subtree (see Section 6.2). Authorized Peers are only enabled to overwrite existing ACL item they own. The Resource Owner is allowed to overwrite any existing ACL item, but should be aware of its consequences.

Access Control List		
#Index	Array Entries	signed by
123abc001	to_user:Owner Kind:1234 ad:1	Owner
123abc002	to_user:Alice Kind:1234 ad:1	Owner
123abc003	to_user:Owner Kind:4321 ad:1	Owner
123abc004	to_user:Carol Kind:4321 ad:0	Owner
...	...	...
456def001	to_user:Bob Kind:1234 ad:0	Alice
...	...	...

Figure 1: Simplified example of an Access Control including entries for two different Kind-IDs and varying delegation (ad) configurations

Implementations of ShaRe should be aware that the trust delegation in an Access Control List need not be loop free. Self-contained circular trust delegation from A to B and B to A are syntactically possible, even though not very meaningful.

4.2. Data Structure

The Kind data structure for the Access Control List is defined as follows:

```
struct {
     /* res_name_ext is optional, see documentation */
     ResourceNameExtension  res_name_ext;
     opaque                 to_user<0..2^16-1>;
     KindId                 kind;
     Boolean                allow_delegation;
} AccessControlListItem;
```

The AccessControlListItem structure is composed of:

res_name_ext: This optional field contains the Resource Name of a ResourceNameExtension (see Section 5.2) to be used by a Shared Resource with variable resource name. This name serves the storing peer for validating, whether a variable resources name matches one of the predefined naming pattern from the configuration document for this Kind. The presence of this field is bound to a variable resource name element in the corresponding kind-block of the configuration document whose "enable" attribute is set to true (see Section 5.3). Otherwise, if the "enable" attribute is false, the res_name_ext field SHALL NOT be present in the Kind data structure.

to_user: This field contains the username of the RELOAD peer that obtains write permission to the Shared Resource.

kind: This field contains the Kind-ID of the Shared Resource.

allow_delegation: If true, this Boolean flag indicates that the Authorized Peer in the 'to_user' field is allowed to add additional entries to the ACL for the specified Kind-ID.

5. Extension for Variable Resource Names

5.1. Overview

In certain use cases such as conferencing (c.f. [I-D.knauf-p2psip-disco]) it is desirable to increase the flexibility of a peer in using Resource Names beyond those defined by the username or Node-ID fields in its certificate. For this purpose, this document presents the concept for variable Resources Names that enables providers of RELOAD instances to define relaxed naming schemes for overlay Resources.

Each RELOAD node uses a certificate to identify itself using its user name (or Node-ID) while storing data under a specific Resource-ID. The specifications in this document scheme adhere to this paradigm, but enable a RELOAD peer to store values of Resource Names that are derived from the username in its certificate. This is done by using a Resource Name with a variable substring that still matches the username in the certificate using a pattern defined in the overlay configuration document. Thus despite being variable, an allowable Resource Name remains tied to the Owner's certificate. A sample pattern might be formed as follows.

Example Pattern:
.*-conf-$USER@$DOMAIN

When defining the pattern, care must be taken to avoid conflicts arising from two usernames of witch one is a substring of the other. In such cases, the holder of the shorter name could threaten to block the resources of the longer-named peer by choosing the variable part of a Resource Name to contain the entire longer username. This problem can easily be mitigated by delimiting the variable part of the pattern from the username part by some fixed string, that by convention is not part of a username (e.g., the "-conf-" in the above Example).

5.2. Data Structure

This section defines the optional ResourceNameExtension structure for every Kind that uses the USER-CHAIN-ACL access control policy.

```
enum { pattern  (1),
(255)} ResourceNameType;

struct {
  ResourceNameType type;
  uint16           length;
  select(type) {
      case pattern:
        opaque resource_name<0..2^16-1>;

      /* Types can be extended */
  }
} ResourceNameExtension
```

The content of the ResourceNameExtension consist of

length: This field contains the length of the remaining data structure. It is only used to allow for further extensions to this data structure.

The content of the rest of the data structure depends of the ResourceNameType. Currently, the only defined type is "pattern".

If the type is "pattern", then the following data structure contains an opaque <0..2^16-1> field containing the Resource Name of the Kind being stored. The type "pattern" further indicates that the Resource Name MUST match to one of the variable resource name pattern defined for this Kind in the configuration document.

The ResourceNameType enum and the ResourceNameExtension structure can be extended by further Usages to define other naming schemes.

5.3. Overlay Configuration Document Extension

This section extends the overlay configuration document by defining new elements for patterns relating resource names to user names.

The <variable-resource-names> element serves as a container for one or multiple <pattern> sub-elements. It is an additional parameter within the kind block and has a boolean "enable" attribute that indicates, if true, that the overlay provider allows variable resource names for this Kind. The default value of the "enable" attribute is "false". In the absence of a <variable-resource-names> element for a Kind using the USER-CHAIN-ACL access policy (see Section 6.6), implementors SHOULD assume this default value.

A <pattern> element MUST be present if the "enabled" attribute of its parent element is set to true. Each <pattern> element defines a

pattern for constructing extended resource names for a single Kind. It is of type xsd:string and interpreted as a regular expression. In this regular expression, $USER and $DOMAIN are used as variables for the corresponding parts of the string in the certificate username field (with $USER preceding and $DOMAIN succeeding the '@'). Both variables MUST be present in any given pattern definition. If no pattern is defined for a Kind or the "enabled" attribute is false, allowable Resource Names are restricted to the username of the signer for Shared Resource.

The Relax NG Grammar for the Variable Resource Names Extension reads:

```
<!-- VARIABLE RESOURCE URN SUB-NAMESPACE -->

namespace share = "urn:ietf:params:xml:ns:p2p:config-base:share"

<!-- VARIABLE RESOURCE NAMES ELEMENT -->

kind-parameter &= element share:variable-resource-names {

    attribute enable { xsd:boolean }

    <!-- PATTERN ELEMENT -->

    element pattern { xsd:string }*
}?
```

6. Access Control to Shared Resources

6.1. Granting Write Access

Write access to a Kind that is intended to be shared with other RELOAD users can solely be issued by the Resource Owner. A Resource Owner can share RELOAD Kinds by using the following procedure.

- The Resource Owner stores an ACL root item at the Resource-ID of the Shared Resource. The root item contains the resource name extension field (see Section 5.2), the username of the Resource Owner and Kind-ID of the Shared Resource. The allow_delegation flag is set to 1. The index of array data structure MUST be generated as described in Section 3.1

- Further ACL items for this Kind-ID stored by the Resource Owner will delegate write access to Authorized Peers. These ACL items contain the same resource name extension field, the username of the Authorized Peer and the Kind-Id of the Shared Resource. Optionally, the Resource Owner sets the "ad" to 1 (the default equals 0) to enable the Authorized Peer to further delegate write access. Each succeeding ACL item created by the Resource Owner can be stored in the numerical order of the array index starting with the index of the root item incremented by one.

An Authorized Peer with delegation allowance ("ad"=1) can extend the access to an existing Shared Resource as follows.

- An Authorized Peer can store additional ACL items at the Resource-ID of the Shared Resource. These ACL items contain the resource name extension field, the username of the newly Authorized Peer, and the Kind-Id of the Shared Resource. Optionally, the "ad" flag is set to 1 for allowing the Authorized Peer to further delegate write access. The array index MUST be generated as described in Section 3.1. Each succeeding ACL item can be stored in the numerical order of the array index.

A store request by an Authorized Peer that attempts to overwrite any ACL item signed by another Peer is unauthorized and causes an Error_Forbidden response from the Storing Peer. Such access conflicts could be caused by an array index collision. However, the probability of a collision of two or more identical array indices will be negligibly low using the mechanism for isolating stored data (see Section 3.1)

6.2. Revoking Write Access

Write permissions are revoked by storing a non-existent value [I-D.ietf-p2psip-base] at the corresponding item of the Access Control List. Revoking a permission automatically invalidates all delegations performed by that user including all subsequent delegations. This allows to invalidate entire subtrees of the delegations tree with only a single operation. Overwriting the root item with a non-existent value of an Access List invalidates the entire delegations tree.

An existing ACL item MUST only be overwritten by the user who initially stored the corresponding entry, or by the Resource Owner that is allowed to overwrite all ACL items for revoking write access.

6.3. Validating Write Access through an ACL

Access Control Lists are used to transparently validate authorization of peers for writing a data value at a Shared Resource. Thereby it is assumed that the validating peer is in possession of the complete and most recent ACL for a specific Resource/Kind pair. The corresponding procedure consists of recursively traversing the trust delegation tree and proceeds as follows.

1. Obtain the username of the certificate used for signing the data stored at the Shared Resource.

2. Validate that an item of the corresponding ACL (i.e., for this Resource/Kind pair) contains a "to_user" field whose value equals the username obtained in step 1. If the Shared Resource under examination is an Access Control List Kind, further validate if the "ad" flag is set to 1.

3. Select the username of the certificate that was used to sign the ACL item obtained in step 2.

4. Validate that an item of the corresponding ACL contains a "to_user" field whose value equals the username obtained in step 3. Additionally validate that the "ad" flag is set to 1.

5. Repeat steps 3 and 4 until the "to_user" value is equal to the username of the signer of the previously selected ACL item. This final ACL item is expected to be the root item of this ACL which SHALL be further validated by verifying that the root item was signed by the owner of the ACL Resource.

The trust delegation chain is valid if and only if all verification steps succeed. In this case, the creator of the data value of the

Shared Resource is an Authorized Peer.

Note that the ACL validation procedure can be omitted whenever the creator of data at a Shared Resource is the Resource Owner itself. The latter can be verified by its public key certificate as defined in Section 6.6.

6.4. Operations of Storing Peers

Storing peers, i.e., peers at which Shared Resource and ACL are physically stored, are responsible for controlling storage attempts to a Shared Resource and its corresponding Access Control List. To assert the USER-CHAIN-ACL access policy (see Section 6.4), a storing peer MUST perform the access validation procedure described in Section 6.3 on any incoming store request using the most recent Access Control List for every Kind that uses the USER-CHAIN-ACL policy. It SHALL further ensure that only the Resource Owner stores new ACL root items for Shared Resources.

6.5. Operations of Accessing Peers

Accessing peers, i.e., peers that fetch a Shared Resource, MAY validate that the originator of a Shared Resource was authorized to store data at this Resource-ID by processing the corresponding ACL. To enable an accessing peer to perform the access validation procedure described in Section 6.3, it first needs to obtain the most recent Access Control List in the following way.

1. Send a Stat request to the Resource-ID of the Shared Resource to obtain all array indexes of stored ACL Kinds.

2. Fetch all indexes of existing ACL items at this Resource-ID by using the array ranges retrieved in the Stat request answer.

Peers can cache previously fetched Access Control Lists up to the maximum lifetime of an individual item. Since stored values could have been modified or invalidated prior to their expiration, an accessing peer SHOULD use a Stat request to check for updates prior to using the data cache.

6.6. USER-CHAIN-ACL Access Policy

This document specifies an additional access control policy to the RELOAD base draft [I-D.ietf-p2psip-base]. The USER-CHAIN-ACL policy allows Authorized Peers to write a Shared Resource, even though they do not own the corresponding certificate. Additionally, the USER-CHAIN-ACL allows the storage of Kinds with a variable resource name that are following one of the specified naming pattern. Hence, on an

inbound store request on a Kind that uses the USER-CHAIN-ACL access policy, the following rules MUST be applied:

In the USER-CHAIN-ACL policy, a given value MUST be written or overwritten, if either one of USER-MATCH or USER-NODE-MATCH (mandatory if the data model is dictionary) access policies of the base document [I-D.ietf-p2psip-base] applies.

Otherwise, the value MUST be written if the certificate of the signer contains a username that matches to one of the variable resource name pattern (c.f. Section 5) specified in the configuration document and, additionally, the hashed Resource Name matches the Resource-ID. The Resource Name of the Kind to be stored MUST be taken from the mandatory ResourceNameExtension field in the corresponding Kind data structure.

Otherwise, the value MUST be written if the ACL validation procedure described in Section 6.3 has been successfully applied.

7. ACL Kind Definition

This section defines the ACCESS-CONTROL-LIST Kind previously described in this document.

Name: ACCESS-CONTROL-LIST

Kind IDs: The Resource Name for ACCESS-CONTROL-LIST Kind-ID is the Resource Name of the Kind that will be shared by using the ACCESS-CONTROL-LIST Kind.

Data Model: The data model for the ACCESS-CONTROL-LIST Kind-ID is array. The array indexes are formed by using the mechanism for isolated stored data as described in Section 3.1

Access Control: USER-CHAIN-ACL (see Section 6.6)

8. Security Considerations

In this section we discuss security issues that are relevant to the usage of shared resources in RELOAD.

8.1. Resource Exhaustion

Joining a RELOAD overlay inherently poses a certain resource load on a peer, because it has to store and forward data for other peers. In common RELOAD semantics, each Resource ID and thus position in the overlay may only be written by a limited set of peers - often even only a single peer, which limits this burden. In the case of Shared Resources, a single resource may be written by multiple peers, who may even write an arbitrary number of entries (e.g., delegations in the ACL). This leads to an enhanced use of resources at individual overlay nodes. The problem of resource exhaustion can easily be mitigated for Usages based on the ShaRe-Usage by imposing restrictions on size, i.e., <max-size> element for a certain Kind in the configuration document.

8.2. Malicious or Misbehaving Storing Peer

The RELOAD overlay is designed to operate despite the presence of a small set of misbehaving peers. This is not different for Shared Resources since a small set of malicious peers does not disrupt the functionality of the overlay in general, but may have implications for the peers needing to store or access information at the specific locations in the ID space controlled by a malicious peer. A storing peer could withhold stored data which results in a denial of service to the group using the specific resource. But it could not return forged data, since the validity of any stored data can be independently verified using the attached signatures.

8.3. Privacy Issues

All data stored in the Shared Resource is publicly readable, thus applications requiring privacy need to encrypt the data. The ACL needs to be stored unencrypted, thus the list members of a group using a Shared Resource will always be publicly visible.

9. IANA Considerations

TODO: register Kind-ID code point at the IANA

10. Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and SAM research group. We would like to thank all active members for constructive thoughts and feedback. In particular, the authors would like to thank (in alphabetical order) Lothar Grimm, Cullen Jennings, Peter Musgrave, Joerg Ott, Marc Petit-Huguenin, Peter Pogrzeba, and Jan Seedorf. This work was partly funded by the German Federal Ministry of Education and Research, projects HAMcast and Mindstone.

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-21 (work in progress), March 2012.

[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., Willis, D., Shim, E., Matthews, P., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-04 (work in progress), October 2011.

[I-D.knauf-p2psip-disco]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A RELOAD Usage for Distributed Conference Control (DisCo)", draft-knauf-p2psip-disco-04 (work in progress), October 2011.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

Appendix A. Change Log

The following changes have been made from version draft-knauf-p2psip-share-02:

1. Editorial improvements

2. Updated References

The following changes have been made from version draft-knauf-p2psip-share-01:

1. Simplified the ACL data structure in response to WG feedback

2. Added ResourceNameExtension data structure to simplify the use of variable resource names

3. Restructured document

4. Many editorial improvements

The following changes have been made from version draft-knauf-p2psip-share-00:

1. Integrated the USER-PATTERN-MATCH access policy into USER-CHAIN-ACL

2. Access Control List Kind uses USER-CHAIN-ACL exclusively

3. Resources to be shared use USER-CHAIN-ACL exclusively

4. More precise specification of mandatory User_name and Resource_name fields for Shared Resources

5. Added mechanism for isolating stored data to prevent race conditions while concurrent storing

6. XML Extension for variable resource names uses its own namespace

7. Many editorial improvements

Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: alexanderknauf@gmail.com
URI: http://inet.cpt.haw-hamburg.de/members/knauf

Gabriel Hege
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Phone: +4940428758067
Email: hege@fhtw-berlin.de
URI: http://inet.cpt.haw-hamburg.de/members/hege

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg D-20099
Germany

Email: schmidt@informatik.haw-hamburg.de
URI: http://inet.cpt.haw-hamburg.de/members/schmidt

Matthias Waehlisch
link-lab & FU Berlin
Hoenower Str. 35
Berlin D-10318
Germany

Email: mw@link-lab.net
URI: http://www.inf.fu-berlin.de/~waehl

P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2012

J. Peng
China Mobile
K. Feng
Beijing University of Posts and Telecommunications
L. Le
China Mobile

July 2, 2011

One Hop Lookups Algorithm Plugin for RELOAD
draft-peng-p2psip-one-hop-plugin-00

Abstract

This document proposes an implementation of overlay plugin algorithm which is called one hop lookups to provide examples and references for the research of one hop based RELOAD. With the development of the real time communications, there are high demands for the improvement of routing efficiency. In the one hop algorithm, each peer maintains complete membership information which can guarantee one hop lookups to improve the routing efficiency. For the RELOAD, using the one hop lookups algorithm to construct Topology Plugin with the same RELOAD core can have a better support for VoIP applications, and the implementation of one hop lookups algorithm plugin is based on the methods provided by RELAOD.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 3, 2012.

Copyright Notice

Copyright (c) 2011 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 5
3. Hash functions 6
4. Peer data structure 7
5. Routing 10
6. Joining 11
7. Updates 12
7.1. Update messages definition 12
7.2. Primary event notification messages 17
7.3. Actions after receiving update messages 18
8. Stabilization 20
9. Leaving 21
10. Leader choosing strategies 22
11. Replication 23
12. Fault tolerance 24
12.1. First hop fail 24
12.2. Leaders fail 24
13. Security Considerations 26
14. IANA Considerations 27
15. Acknowledgements 28
16. References 29
16.1. Normative References 29
16.2. Informative References 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. The Architecture of RELOAD has a routing layer which is called the Topology Plugin. It is responsible for implementing the specific overlay algorithm being used. In the RELOAD base protocol, the Topology Plugin is described by Chord, and we define another DHT algorithm in order to provide a high rouging efficiency which can be used in real time communications for a better performance.

The real time communication has a high demand on the routing efficiency, for example, the VoIP usage on the application level of RELOAD, it is no doubt that the routing speed or efficiency must be very important, and it depends on the looking up efficiency of the Overlay Topology. There are many kinds of structured peer-to-peer overlay network algorithms like Chord, Pastry, CAN and so on. Most of them have one in common is that they have a routing table which can maintain a small amount of routing state. But all these algorithms 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 one hop lookups algorithm gives one way to maintain complete membership information at each node in order to increase the routing efficiency even though the system is large and membership is changing rapidly.

In the one hop algorithm, each peer maintains a complete description of system memberships, and it presents techniques for the peers to maintain this information accurately with lower costs of communications. Different one hop algorithm implementations have different ways to keep the accuracy of routing states. For example, the one hop lookups for peer-to-peer overlay [One-Hop-Lookups] uses event notification mechanism without broadcast to update all the peers' routing table during several seconds; the SandStone [SandStone] gives a way to collect update information by SPM (Super Peer Maintenance) which is called SPM assisted one hop enhancement. The SandStone overlay is typically deployed as a two-layered DHT, including a global DHT and several regional DHT. There is at least one SPM node in each region, and once one peer's state change is detected by its neighboring peer, the neighbor will notify its local SPM, and then the SPM will disseminate the event to other SPMs, finally each SPM will broadcast this notification to all peers under the charge of it. We use the one hop lookups algorithm to construct the RELOAD Topology Plugin because there is no SPM like peers and two layered topology in RELOAD base.

In this draft, we first give a simple overview of one hop lookups

algorithm with some definitions, then fill the overlay specific data structures into the RELOAD frame and implements this algorithm with topology messages defined in the RELOAD Topology Plugin; at last, the replication and fault tolerance strategies are stated. 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 [RFC2119].

3. Hash functions

In this one hop lookups algorithm topology plugin, the size of the node ID and resource ID is 128 bits. The hash of an ID can be computed by SHA-1 [RFC3174] and other hash functions.

4. Peer data structure

A peer, or a node, is responsible for a particular Resource-ID which is less than or equal to its Node-ID and is greater than the Node-ID of the previous peer in the neighborhood. In order to have a high routing accuracy and efficiency, each peer must keep and maintain a series of data structure.

In the one hop lookups algorithm, we divide the 128-bit circular identifier space into k equal contiguous intervals which are called slices. Each slice has a slice leader, which is chosen as the node that is the successor of the mid-point of the slice space (other choosing methods are list in Section X). Similarly, each slice also can be divided into u equal-sized intervals which are called units; each unit has a unit leader, which is chosen as the successor of the mid-point of the unit space. And of course every peer in the overlay should know its unit header and slice header. The slice leader and unit leader are both ordinary peers who are chosen dynamically, and they also can be configured at first. The leader choosing strategies will be stated in section 10.

In the SandStone, there is at least one SPM node in each region, so when one peer detects the state change of overlay in its neighborhood, it will notify its local SPM as soon as possible. The local SPM then disseminate this notification to other SPMs, at last, each SPM will broadcast this update to all peers under the charge of it. The SPM is a special peer which can collect update information and do some other management tasks. The SPM does not participate in the routing procedures and it is pre-configured.

On the other hand, the SPM uses the broadcast to spread the notifications. Only the bootstrap peer in RELOAD has the broadcast function. Using the broadcast is a simple and fast way to inform information, but it costs a lot of bandwidths and also may cause the notifications redundancy. Using the event based notification mechanism defined in one hop lookups can avoid redundancy in the communications: one peer will get information only from its neighbor that is one step closer to its unit leader. This implies that within a unit, information is always flowing from the unit leader to the ends of the unit. But its disadvantage is obvious: the notifications are spread too slowly which may cause the delay of overlay stabilization. In this draft, we choose the one hop lookups algorithm to be an example.

From above, we can conclude the information peer should maintain as follows.

Routing table:
The routing table of each node keeps all of nodes' address, and then the source node can find the destination by just one hop at most if the items of routing table are right. So it is very important to keep the correctness of each node's full routing table. In order to achieve the goal, the notification of membership change events must reach every node in the overlay within a specified amount of time.

Predecessor and successor information:
The information especially Node-ID and address about peer's predecessor and successor must be kept in order to maintain the overlay circle. A peer will communicate with its predecessor and successor with keep-alive message (defined in Update message) every several seconds. Of course, this structure can be added as mark-ups or identifiers in the routing table.

Unit leader information:
The information about the unit leader who is responsible for the unit the peer belongs to. A peer can communicate with its unit leader to receive the Update message. Of course, this structure can be added as mark-ups or identifiers in the routing table.

Slice leader information:
The information about the slice leader who is responsible for the slice the peer belongs to. A peer can send event notification message (defined in Update message) to its slice leader to inform events. Of course, this structure can be added as mark-ups or identifiers in the routing table.

The four kinds of information are maintained in all of peers, but there are still some kinds of other data structures should be kept in some special peers like unit leader and slice leader.

Unit boundary identifier:
This identifier does not keep in all the peers, it is stored on the peer who is the boundary of the unit in order to make sure the Update message just being transferred within unit.

Unit boundary peer list:
The unit leader should know the information about the unit's boundary peer in order to know its control area.

Slice leader list:
Every slice leader should know the information about all the slice leaders in the overlay in order to dispatch the event notification messages in time.

Each peer with these data structures can compose a well running overlay topology based on the one hop lookups algorithm.

5. Routing

The routing table of the peer has full information of all the nodes in the overlay. If the peer is not responsible for a resource whose ID is r, it will query the routing table in order to find the first peer whose id is greater than r, which means that the peer should be responsible for this resource; and then sends a request message to the destination node. If no such node is found, it finds the largest Node-ID in the interval between the peer and Resource-ID r.

6. Joining

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 for AP.

(3) JP MUST send a Join to AP, the AP sends the response to the Join.

(4) AP MUST do a series of Store requests to JP to store the data that JP will be responsible for.

(5) JP SHOULD send Attach request to its predecessor in order to let it knows the arriving of new peer; the predecessor should update its successor information at once.

(6) The predecessor detects a change in its routing table for example it has a new successor, it MUST send an event notification message which is a kind of Update messages to its slice leader directly. Of course, the Update message can also be sent by the AD.

(7) The slice leader MUST collect all notifications it receives from the peers in his slice and sends Update messages to other slice leaders every several seconds to inform these events.

(8) Other slice leaders MUST dispatch the Update messages they received to all the unit leaders in their respective slices.

(9) A unit leader SHOULD piggyback this information on its Update message to its successor and predecessor.

(10) Other nodes propagate this information from their successors; they SHOULD send it to their successors and vice versa.

(11) Peers at unit boundaries SHOULD NOT send information to their neighboring peers outside their unit.

The Update messages depending on the RELOAD message formats will be defined in the next section. The peer can find its successor by sending a Find request with the Resource-ID equals its Node-ID.

7. Updates

7.1. Update messages definition

The update messages in one hop lookups algorithm is based on the event notification. So the definition of update message can be started with the type definition.

```
enum { eventUpdate (0), dataStructureUpdate (1), (255) }
  UpdateType;
```

The UpdateType gives an enumeration of Update message which includes eventUpdate and dataStrucruteUpdate. The eventUpdate message will take a list of event notifications and the dataStructureUpdate message will take a list of data structure information which may include the routing table items. This structure allows for unknown options types.

```
enum { keepAlive (0), ordinaryPeerChange (1),
  unitLeaderChange(2), sliceLeaderChange (3), (255) } EventType;
```

The EventType gives an enumeration of event kinds, it is composed of keepAlive, ordinaryPeerChange, unitLeaderChange and sliceLeaderChange. The keepAlive event is like the Ping message in Chord. The ordinaryPeerChange, unitLeaderChange and sliceLeaderChange represent the state change of different peers in the overlay. This structure allows for unknown options types.

```
enum { peerJoin (0), peerLeave (1), peerChange (2), (255) }
  ChangeType;
```

The ChangeType gives an enumeration of peer state change type which includes the PeerJoin, PeerLeave and peerChange. The meaning of join and leave are easy to understand, and the peerChange means the unit leader change event or slice leader change event, it also can be constructed by peer leave and peer join. This structure allows for unknown options types.

```
enum { ordinaryPeer (0), unitLeaderPeer (1),
  sliceLeaderPeer (2), (255) } PeerType;
```

The PeerType gives an enumeration of peer type which is used to cooperate with the ChangeType. This structure allows for unknown options types.

```
struct {
  Node-ID               peer_id
  IpAddressPort         peer_addr_port
} PeerContactItem;
```

The structure PeerContactItem is used to represent a basic item in the routing table or other data structures that need peers' information. It is composed by a Node-ID and IpAddressPort. The IpAddressPort structure is defined in the RELOAD base which includes one peer's IP address and listening port. If one peer get the information of another peer's PeerContactItem, then it can contact this peer with the IP address.

```
struct {
  uint32                uptime;
  EventType             event_type;
  PeerContactItem       detect_peer;
  uint32                length;

  select (event_type) {
  case keepAlive:
    ;

  case ordinaryPeerChange:
    ChangeType          ordinary_change_type;
    select (ordinary_change_type) {
      case peerJoin:
        PeerContactItem       joining_peer;
      case peerLeave:
        PeerContactItem       leaving_peer;
    }

  case unitLeaderChange:
    ChangeType          unit_leader_change_type;
    select (unit_leader_change_type) {
      case peerJoin:
        PeerContactItem       new_unit_leader;
        PeerContactItem       old_unit_leader;
      case peerLeave:
        PeerContactItem       leaving_unit_leade;
    }

  case sliceLeaderChange:
    ChangeType          slice_leader_change_type;
    select (slice_leader_change_type) {
      case peerJoin:
        PeerContactItem       new_slice_leader;
        PeerContactItem       old_slice_leader;

      case peerLeave:
        PeerContactItem       leaving_slice_leader;
    }

  }
} EventNotification;
```

The structure EventNotification is very important because it is the base of event notification. The length of the structure changes in different event type. This structure has enough information to describe some important events. The contact information of detect

peer (detect_peer) is useless and can be ignored. In the event type, the three different event types can still be divided by the ChangeType. The notification must have enough information for all the peers in the overlay to update their data structures like routing table.

```
struct {
  PeerType             peer_type;
  PeerContactItem      predecessor_peer;
  PeerContactItem      successor_peer;
  PeerContactItem      routing_table_list <0...n>;
  uint32               length;

  select (peer_type) {
  case ordinaryPeer:
    PeerContactItem      unit_leader;
    PeerContactItem      slice_leader;

  case unitLeaderPeer:
    PeerContactItem      boundary_peers <2>;

  case sliceLeaderPeer:
    PeerContactItem      slice_leaders_list <0...n>;
  }
} DataStructureContent
```

The structure DataStructureContent mainly used in the transferring of data structure during the peer joining procedure. The length of the structure changes in different peer type because different peer has different data structure. The ordinary peer only has predecessor, successor, unit leader, slice leader and routing table information. And the unit leader and slice leader should also know some other information.

```
typedef EventNotification<0...n>        EventNotificationList;
typedef DataStructureContent<0...n>     DataStructureContentList;

struct {
  UpdateType           update_type;
  uint32               length;

  select (update_type) {
  case eventUpdate:
    EventNotificationList       event_notification_list;

  case dataStructureUpdate:
    DataStructureContentList    data_structure_list;
  }
} OneHopUpdateReq;
```

This structure is the Update message used in the one hop lookups. The Update message is composed by two kinds of list. Using list can take more information. One of them is the EventNotificationList which includes a list of EventNotification structures, and the other is the DataStructureContentList which includes a list of DataStructureContent structures. All the peers in the overlay can use this Update request to inform event notification or transfer routing information.

```
struct {
  uint32          update_response;
} OneHopUpdateAns;
```

The structure OneHopUpdateAns is used to be a response to the OneHopUpdateReq message. It is only has a response number to represent the receiver's attitude which may include success, fail and error. This number also can be defined as a type.

The PeerContactItem structure contains parameters as follows.

peer_id
: The peer's Node-ID.

peer_addr_port
: The contact information of peer's especially the IP address.

The EventNotification structure contains parameters as follows.

uptime
The time this peer has been up in seconds.

event_type
The type of event the message will take, this structure allows for unknown event types.

length
The length of the remainder of this message.

ordinary_change_type, unit_leader_change_type, slice_leader_change_type
The type of peer changing event, this structure allows for unknown peer changing types

The DataStructureContent structure contains parameters as follows.

peer_type
The type of peer who wants to get these information, this structure allows for unknown peer types.

routing_table_list
The peer's routing table which has all of the peers' information in the overlay.

boundary_peers
The information of peers' who are the boundary of unit.

slice_leaders_list
All of the slice leaders' information will be stored in this list, and it can be transferred from one slice leader to another or from the old slice leader to the new one.

The OneHopUpdateAns message contains parameters as follows.

update_response
The response of the Update message, different number has different meaning includes success, error and so on.

7.2. Primary event notification messages

There are four main messages for the event notification among the one-hop lookups algorithm, the keep-alive messages, the event notification to slice leader messages, the messages exchanged between slice leaders and the messages from slice leaders to unit leaders.

The keep-alive message is like the Ping defined in the Link Management of Reload, and the messages are used to form the base

level communication between a peer and its predecessor and successor. The event notification message to slice leaders is used to send event notification to the peer's slice leader directly which can use the template of Update message. The messages exchanged between slice leaders are the most important. Each message sent from one slice leader to another batches together events that occurred during several seconds. We can still use the Update message to play this role. The messages from slice leaders to unit leaders can dispatch event notification. Messages received by a slice leader are batched for one second and the forwarded to unit leaders. As above, this message still can use Update message to express.

Above all, these important event notification messages can be expressed by Update messages. In some scenarios like the peer joining, the Update message should be extended to support the transfer of the routing table information.

7.3. Actions after receiving update messages

When peers in the overlay receive update messages which based on the event mechanism, the peer will check the type of event first.

(1) If the message received is a keep-alive message, then the peer can make sure the relationship with the sender and return an answer as soon as possible.

(2) If the message contains an ordinary peer join event, the actions taken should depend on the peer type of receiver, the slice leader will send node join notification to other slice leaders and its unit leader; the unit leader will send this notification to its peers; the peers will use the Update message to inform their processor or successor.

(3) If the message contains an ordinary peer leave event, the actions taken should depend on the peer type of receiver, the slice leader will send node leave notification to other slice leaders and its unit leader; the unit leader will send this notification to its peers; the peers will use the Update message to inform their processor or successor.

(4) If the message's event type is unit leader change, the actions taken should depend on the peer type of receiver, the slice leader will inform the sender to become the new unit leader, and then the new unit leader will spread this information to its unit members via the Update message. And the unit leader change message may be constructed by unit leader join and unit leader leave messages, but the procedure is same.

(5) If the message's event type is slice leader change, the other slice leaders who receive this will record the event caused peer's information and return the acknowledge information, the new slice leader will inform its unit leaders to tell them the change of slice leader, and then the unit leaders will spread this information to its unit members via the Update message.

8. Stabilization

The stabilization of one hop lookups pay much more attention on the refreshing of routing table because if one peer has a right routing table, then it can contact any peers in the overlay directly. The accuracy of routing table depends on the speed of Update message spreading. So the notification message frequency is a very important case which can have effects on the stabilization. The event notification to slice leader, the messages exchanged between slice leaders and the messages from slice leaders to unit leaders have different sending frequency, and all of them are not fixed numbers. The frequency is calculated by the acceptable fraction of queries that fail in the first routing attempt, the peers in the overlay and the expected rate of membership changes. The details of frequency and time calculating are described in [One-Hop-Lookups]. Sometimes we can use the broadcast to inform notification in order to save time.

On the other hand, keeping the accuracy of neighborhood peers' information is also very important. For example, in the one hop lookups, the keep-alive messages are sent every second, and the keep-alive messages construct the base level communications between a peer and its predecessor and successor.

9. Leaving

The leaving action is also a kind of event, so it can be expressed by the Update message. When a peer leaves the overlay peacefully, it needs to send an Update message as an event notification message, so the Leaving message may not be defined and just using the Update message. And the data transfer can also use the Update messages, but it sometimes depends on the replication strategies (Section 11).

If a peer leaves the overlay without any notification, the neighbor peer (predecessor or successor) must inform this event to the leader by the Update messages, and then the backup peer of the leaving one should transfer the backup data to the new one who is charging the leaving peer's namespace. When an unit leader or slice leader leaves this overlay, the overlay should choose a new one to replace the old by the leader choosing strategies described in section 10.

10. Leader choosing strategies

The default slice and unit leader strategy is dynamic choosing the successor of the mid-point peer of the slice or unit space. It is convenient to manage and choose, but it is so simple that does not consider some other important cases like handling ability and so on. On the other hand, the slice leader can be treated as a "Super Node", and we can give some other super node choosing strategies as follows.

On the other hand, a SandStone like schema can be used to choose leaders. In some scenarios, we can identify well connected and well provisioned peers as "Super Node" which may have the same meaning with SPM. The chosen super nodes can form a parallel ring. And of course the super slice will act as a slice leader; it does not participate in the routing procedure because its focus of working is on the collecting of event notifications and spreading them in time. The unit leader can be chosen by the default schema because the working burden of unit leader is much more relaxed than the slice leader, and if the chosen one has a poorly provisioned node with a low bandwidth connection to the Internet, the predecessor or successor of the chosen one can replace it.

11. Replication

The replica placement policy of the one hop algorithm can use the way defined in SandStone. It is designed to maximize data reliability and availability, and to maximize network bandwidth utilization. In the SandStone, the subscriber data can be replicated on multiple peers, three on default. We can use a SandStone like replica replacement policy with three replications. The first replica is stored in the primary peer's successor; the second is saved in a peer of different unit but in same slice; and the third is put in a peer from different slice.

Above all, one of the most important issues is the choosing of the replica peer. Maybe we can use a special function whose input is peer's Node-ID to calculate the three replica peers, and of course the root peer should know about the replication method in order to be able to search the data.

12. Fault tolerance

12.1. First hop fail

In the one hop algorithm, if a query fails on its first attempt it does not return an error to an application. Instead, queries can be rerouted with the RELOAD message RouteQueryAns. We can define the RouteQueryAns as follows.

```
struct {
   uint32               response_state;
   PeerContactItem      redirect_peer;
} RouteQueryAns;
```

The structure RouteQueryAns is composed by a state number and a Node-ID. The state number tells the query state and the PeerContactItem gives a peer's information that is closer to the destination. And in most of cases, two hops are enough to locate one peer or resource. If the two hops are still wrong, then we can know the lookup procedure is fail.

response_state
: The response mark of the RouteQueryReq message's; and it may include success, failure and errors.

redirect_peer
: If the query is failed, the application can send RouteQueryReq message to this peer again for finding resources.

If a lookup query from peer A to peer B because the routing table is outdate or the peer B has left, peer A can retry the query by sending the RouteQueryReq to the peer C who is the successor of peer B. And at that time joining a peer D to be peer B's new successor and peer C's predecessor, then peer C can reply RouteQueryAns message with the redirect_peer of peer D's, and peer A can contact it in this routing step.

12.2. Leaders fail

The most important issues are how to keep the correct functioning of unit leaders and slice leaders; we need to recover from their failure. When a slice or unit leader fails, its successor soon detects the failure and becomes the new leader; the successor of a failed slice leader will communicate with its unit leaders and other slice leaders to recover information about the missed events. And of course, the leaser choosing strategies described in section 10 also can be used to choose a new leader. All of these events can be constructed by the Update message with the EventType (e.g. the

UnitLeaderChange and SliceLeaderChange).

13. Security Considerations

14. IANA Considerations

There are no IANA considerations associated to this memo.

15. Acknowledgements

16. References

16.1. Normative References

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

16.2. Informative References

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

[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.

Authors' Addresses

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

Email: Penjin@chinamobile.com

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

Email: fengkai_sunny@139.com

Lifeng Le
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Lelifeng@chinamobile.com

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

Feburary 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.

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 ith slice contains all nodes currently in the overlay whose node identifiers lie in the range [i*2^128/k, (i+1)*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 ith slice is the successor node of the key (i+1/2)*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,

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.

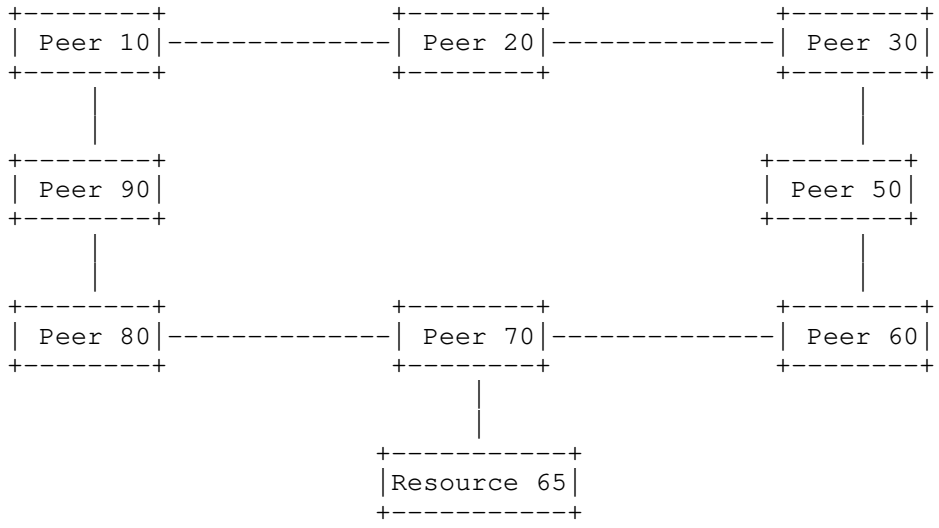


Figure 1 peer-to-peer system schematic

6.2.1. Resource-ID Routing Failure

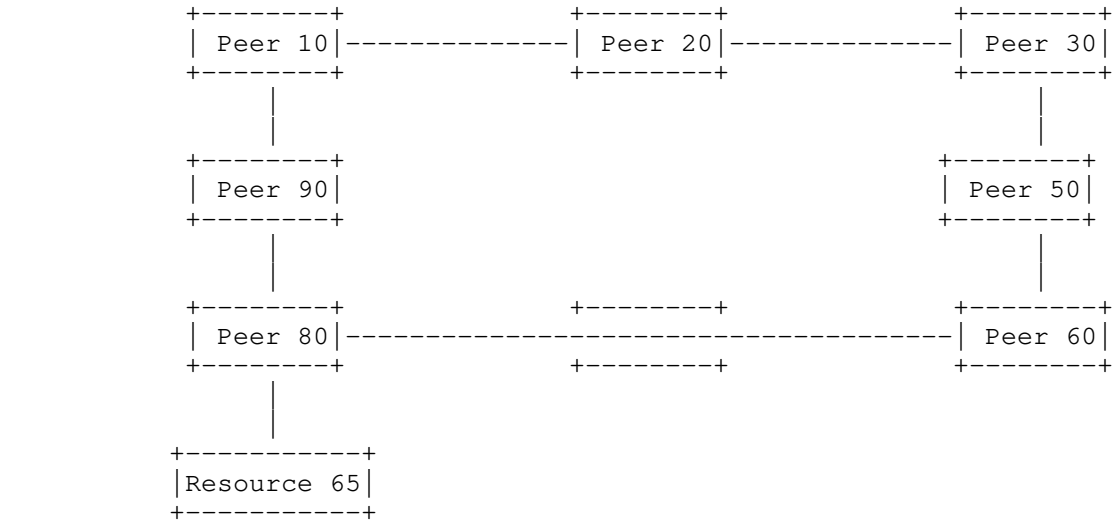
6.2.1.1. Next-Hop Peer Leaving

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.



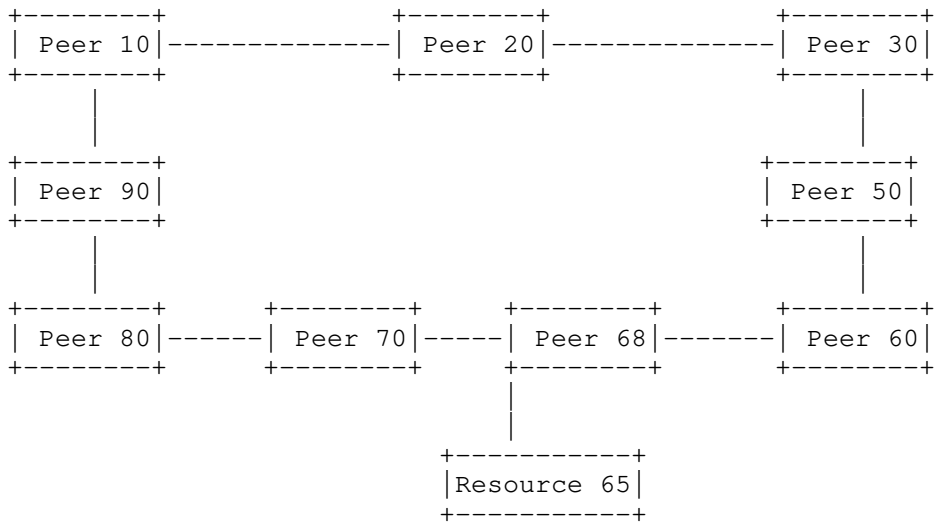
6.2.1.2. New Peer Joining

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.

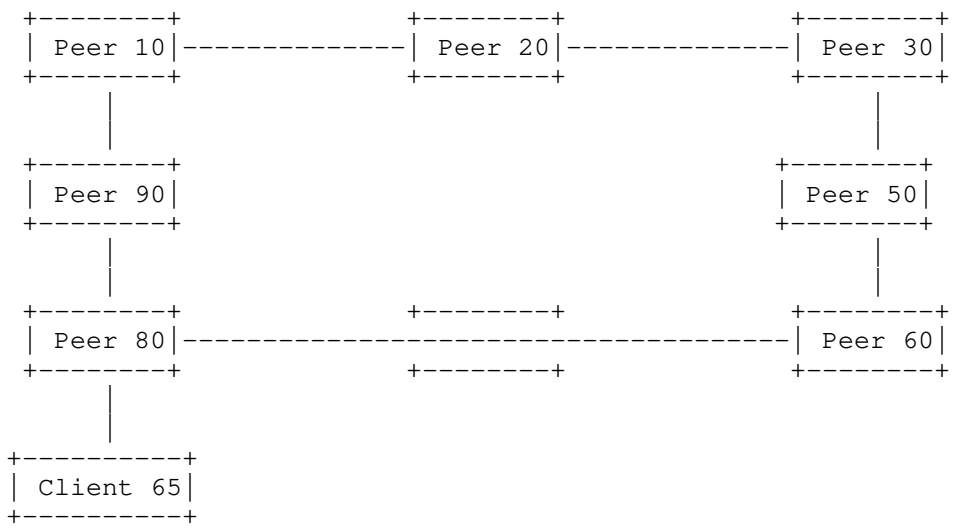


6.2.2. Node-ID Routing Failure

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 locate s.

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.

[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

P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2012

J. Peng
L. Deng
L. Le
G. Li
China Mobile

July 2, 2011

Proposals for RELOAD to support Promotion and Demotion for User-owned Nodes
draft-peng-p2psip-promotion-00

Abstract

This document proposes extensions to RELOAD to support flexible client promotion and demotion modes. RELOAD aims at providing a uniform protocol for both overlay clients and peers, where promotion of a client to peer is triggered and completed at the client's pleasure. It is proposed that RELOAD provide a more restrictive framework to enable passive promotion and demotion, where decisions are made by the network rather than individual user-owned nodes.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 3, 2012.

Copyright Notice

Copyright (c) 2011 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. Problem Statement 5
3.1. Passive promotion instead of active promotion 5
3.2. Passive demotion as well as active demotion 5
4. Security Considerations 7
5. IANA Considerations 8
6. Acknowledgements 9
7. References 10
7.1. Normative References 10
7.2. Informative References 10
Authors' Addresses 11

1. Introduction

RELOAD[I-D.ietf-p2psip-base], a peer-to-peer (P2P) signaling protocol for use on the Internet, provides a generic, self-organizing overlay network service, allowing nodes to efficiently route messages to other nodes and to efficiently store and retrieve data in the overlay.

There are two types of roles in the RELOAD architecture: peer and client. Clients are merely users of the basic messaging and storage function provided by the overlay, while peers (i.e. non-client nodes) are active participants of the serving overlay as they collaborate in a P2P manner to serve one another.

For Internet services on the basis of a RELOAD-enabled P2P overlay, it is appealing to exploit the collectively abundant resources of UNs (i.e. user-owned nodes) to further reduce service operator's CAPEX (i.e. capital expenses on dedicated serving equipment), thus indicating an application scenario for on-demand passive client promotion. On the other hand, due to the distributed nature of P2P overlays, undesirable implications often arises after imprudent peer exits, demanding for regulated passive peer demotion to client in reverse.

However, unlike normal clients in RELOAD overlay, individual UNs are featured with more restrictive resource limits, considerable capability heterogeneity, and diverse interest groups, whose promotion/demotion demands for more restrictive regulations than the simple active client promotion facility defined in RELOAD.

A SIP usage could be used to illustrate the UN-oriented promotion/ demotion problem, where a UN-oriented client refers to a user-owned node attached originally for SIP UAs, while a peer refers to a dedicated SIP servers or UN-promoted SIP servants of the RELOAD overlay.

In this draft, the basic problems for promoting/demoting User-owned clients to/from serving peers using the currently available mechanism are outlined from the overlay operator's point of view, followed by a detailed analysis of specific functionality requirements. Potential extensions to RELOAD are summarized in Section 5. Relevant security considerations are stated in Section 6.

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 [RFC2119].

3. Problem Statement

3.1. Passive promotion instead of active promotion

Since a UN generally lacks the incentive to serve others proactively as a result of its relatively bounded resources in combination with the autonomous and self-centered nature of its individual owner, the UN-oriented promotion procedures are expected to be largely triggered from the overlay network side for the benefit of the servicing system as a whole. Restrictive regulations are needed in the passive procedure for UN-oriented client promotion. Firstly, a UN may be incapable of or malfunctioning in serving others. Secondly, a UN may be preferred not to become a peer for the sake of network considerations, e.g. a moderate network size may be preferred to remain efficient overlay routing.

The simple join-update method allows for active promotion only, where an successfully attached client (either UN-oriented or not), spontaneously initiates the promotion procedure by sending JoinReq messages to expected overlay neighboring peers and indicates its ready-to-go status as a serving peer by sending UpdateReq messages with revised routing information.

It is therefore proposed that the promoter (e.g. an overloading peer) rather than the UN in question triggers the correspondent passive promotion procedure, and the promoted UN is selected from a potentially large client group of candidates and certified by the promoter to other peers in the overlay to be recognized as an authorized peer.

Potential extensions to RELOAD include extensions to certain kinds of messages (e.g. Probe and Join) in order to support the passive client promotion procedure.

3.2. Passive demotion as well as active demotion

For a successfully promoted UN peer, one of the following two demotion scenarios would ultimately terminates its current peering service in the RELOAD overlay.

- Active demotion decision from the UN side, that the user/UN decides to cease being a peer, for instance:
  - Case 1: if the user perceives a considerable decrease in user experience; or
  - Case 2: if the UN terminal runs out of battery.

- Passive demotion decision from the overlay side to get rid of unnecessary UN-promoted peers, for instance:
  - Case 3: Overlay decides to shrinks its peer group to maintain a tolerable routing delay;
  - Case 4: Overlay decides to exclude the peer(s) for unsatisfactory service provision.

The current leave-without-response method fits well in the active demotion scenarios, where a peer (either promoted earlier from UN-oriented client or not), spontaneously initiates the demotion procedure by sending LeaveReq messages to its overlay neighboring peers and be free to go without any further confirmation. This simple but kind of abrupt mode is ungracefull in comparison with another mode of peer demotion mode (i.e. graceful mode, where the demoting peer takes an active part in the data migration and routing update for the resultant overlay adaptation before its physical exit.

It is therefore proposed to add support to allow for both active and passive peer demotion procedures for UN-oriented peers. Moreover, graceful demotion mode is highly preferable for passive demotion scenarios.

Potential extensions to RELOAD include extensions to current method (e.g. extension to LeaveReq message or may be introduction of new types of messages such as explicit LeaveRes messages) to support peers graceful demotion mode.

4. Security Considerations

As stated above, the group of UNs manifests diversity in both physical capabilities and public morals in terms of serving as an overlay peer. Hence, it is reasonable to conduct explicit authorization to distinguish a promotion candidate's potential to serve as a peer from normal UN clients on one hand, and guarantee timely revocation to limit the impact of a misbehaving promoted UN-oriented peer on the other hand.

It is therefore proposed that:

- a qualified UN acquires a separate peer certificate to attest its capabilities and willingness to serve as a peer; and
- a UN-promoted peer's certificate is revoked if it fails to deliver expected performance while its client certificate remains intact.

Potential extensions to RELOAD include separate peer certification and proactive certificate revocation.

5. IANA Considerations

There are no IANA considerations associated to this memo.

6. Acknowledgements

7. References

7.1. Normative References

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

7.2. 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-03 (work in progress), October 2010.

Authors' Addresses

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

Email: Penjin@chinamobile.com

Lingli Deng
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Denglingli@chinamobile.com

Lifeng Le
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Lelifeng@chinamobile.com

Gang Li
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Ligangyf@chinamobile.com

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

J. Peng
L. Deng
L. Le
G. Li
China Mobile
X. Ma
Beijing University of Posts and Telecommunications

Feburary 16, 2013

Proposals for RELOAD to support Promotion and Demotion for User-owned Nodes
draft-peng-p2psip-promotion-02

Abstract

This document proposes extensions to RELOAD to support flexible client promotion and demotion modes. RELOAD aims at providing a uniform protocol for both overlay clients and peers, where promotion of a client to peer is triggered and completed at the client's pleasure. It is proposed that RELOAD provide a more restrictive framework to enable passive promotion and demotion, where decisions are made by the network rather than individual user-owned nodes.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

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

Copyright Notice

Copyright (c) 2013 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. Problem Statement 5
3.1. Passive promotion instead of active promotion 5
3.2. Passive demotion as well as active demotion 5
4. Extensions to RELOAD 7
4.1. Configuration file 7
4.2. Probe 7
4.2.1. ProbeInformationType 7
4.2.2. ProbeReq message 8
4.2.3. ProbeAns message 9
5. Update 11
5.1. Update_type 11
5.2. SecutrityBlock 11
5.3. UpdateInformation structure 12
5.4. UpdateReq structure 13
5.5. UpdateAns structure 13
6. Leave 14
6.1. LeaveType 14
6.2. LeaveReq 14
6.3. LeaveAns 15
7. Message Flow 16
7.1. Passive promotion 16
7.2. Active demotion 17
7.3. Passive demotion 18
8. Security Considerations 21
9. IANA Considerations 22
10. Acknowledgements 23
11. References 24
11.1. Normative References 24
11.2. Informative References 24
Authors' Addresses 25

1. Introduction

RELOAD[I-D.ietf-p2psip-base], a peer-to-peer (P2P) signaling protocol for use on the Internet, provides a generic, self-organizing overlay network service, allowing nodes to efficiently route messages to other nodes and to efficiently store and retrieve data in the overlay.

There are two types of roles in the RELOAD architecture: peer and client. Clients are merely users of the basic messaging and storage function provided by the overlay, while peers (i.e. non-client nodes) are active participants of the serving overlay as they collaborate in a P2P manner to serve one another.

For Internet services on the basis of a RELOAD-enabled P2P overlay, it is appealing to exploit the collectively abundant resources of UNs (i.e. user-owned nodes) to further reduce service operator's CAPEX (i.e. capital expenses on dedicated serving equipment), thus indicating an application scenario for on-demand passive client promotion. On the other hand, due to the distributed nature of P2P overlays, undesirable implications often arises after imprudent peer exits, demanding for regulated passive peer demotion to client in reverse.

However, unlike normal clients in RELOAD overlay, individual UNs are featured with more restrictive resource limits, considerable capability heterogeneity, and diverse interest groups, whose promotion/demotion demands for more restrictive regulations than the simple active client promotion facility defined in RELOAD.

A SIP usage could be used to illustrate the UN-oriented promotion/ demotion problem, where a UN-oriented client refers to a user-owned node attached originally for SIP UAs, while a peer refers to a dedicated SIP servers or UN-promoted SIP servants of the RELOAD overlay.

In this draft, the basic problems for promoting/demoting User-owned clients to/from serving peers using the currently available mechanism are outlined from the overlay operator's point of view, followed by a detailed analysis of specific functionality requirements. Potential extensions to RELOAD are summarized in Section 5. Relevant security considerations are stated in Section 6.

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 [RFC2119].

3. Problem Statement

3.1. Passive promotion instead of active promotion

Since a UN generally lacks the incentive to serve others proactively as a result of its relatively bounded resources in combination with the autonomous and self-centered nature of its individual owner, the UN-oriented promotion procedures are expected to be largely triggered from the overlay network side for the benefit of the servicing system as a whole. Restrictive regulations are needed in the passive procedure for UN-oriented client promotion. Firstly, a UN may be incapable of or malfunctioning in serving others. Secondly, a UN may be preferred not to become a peer for the sake of network considerations, e.g. a moderate network size may be preferred to remain efficient overlay routing.

The simple join-update method allows for active promotion only, where an successfully attached client (either UN-oriented or not), spontaneously initiates the promotion procedure by sending JoinReq messages to expected overlay neighboring peers and indicates its ready-to-go status as a serving peer by sending UpdateReq messages with revised routing information.

It is therefore proposed that the promoter (e.g. an overloading peer) rather than the UN in question triggers the correspondent passive promotion procedure, and the promoted UN is selected from a potentially large client group of candidates and certified by the promoter to other peers in the overlay to be recognized as an authorized peer.

Potential extensions to RELOAD include extensions to certain kinds of messages (e.g. Probe and Join) in order to support the passive client promotion procedure. To see more details in the next section.

3.2. Passive demotion as well as active demotion

For a successfully promoted UN peer, one of the following two demotion scenarios would ultimately terminates its current peering service in the RELOAD overlay.

- Active demotion decision from the UN side, that the user/UN decides to cease being a peer, for instance:
  - Case 1: if the user perceives a considerable decrease in user experience; or
  - Case 2: if the UN terminal runs out of battery.

- Passive demotion decision from the overlay side to get rid of unnecessary UN-promoted peers, for instance:
  - Case 3: Overlay decides to shrinks its peer group to maintain a tolerable routing delay;
  - Case 4: Overlay decides to exclude the peer(s) for unsatisfactory service provision.

The current leave-without-response method fits well in the active demotion scenarios, where a peer (either promoted earlier from UN-oriented client or not), spontaneously initiates the demotion procedure by sending LeaveReq messages to its overlay neighboring peers and be free to go without any further confirmation. This simple but kind of abrupt mode is ungracefull in comparison with another mode of peer demotion mode (i.e. graceful mode, where the demoting peer takes an active part in the data migration and routing update for the resultant overlay adaptation before its physical exit.

It is therefore proposed to add support to allow for both active and passive peer demotion procedures for UN-oriented peers. Moreover, graceful demotion mode is highly preferable for passive demotion scenarios.

Potential extensions to RELOAD include extensions to current method (e.g. extension to LeaveReq message or may be introduction of new types of messages such as explicit LeaveRes messages) to support peers graceful demotion mode. To see more details in the next section.

4. Extensions to RELOAD

In this section, we introduce extensions to RELOAD to enable restrictive promotion and demotion.

4.1. Configuration file

It would be better to have an announcement about promotion-related extensions to the configuration file.

A new label should be defined like below.

<clientpromotion-permitted> true </clientpromotion-permitted >

This element represents whether clients in the overlay can be promoted, and be defaulted as "true" when absent.

It is desirable to make the "right" client to be promoted based on the observation of its expected serving capability. In other words, one should get some local capability statistics about promoting candidate during the procedure of promotion. It is preferred that they refer to the same measuring tools in order to get the statistics (e.g. client CPU, Memory or Disk capabilities) for the same standard. While this is easy to do with memory and disk (in MB for instance), it is not the case with CPU. Hence another new label should be defined here which offer the URL at which the common measuring tool for clients' CPU capability can be downloaded like below:

<benchmark-location> http://example_for_cpu-benchmark.com:82/download.rar </benchmark-location>

4.2. Probe

4.2.1. ProbeInformationType

```
enum{reservedProbeInformation(0),
  responsible_set(1), num_resources(2), uptime(3),
  client_capability(4), peer_demotion (5)  (255)} ProbeInformationType;
```

The ProbeInformationType gives an enumeration of information type which the requester would like the responder to provide. The first four parameters have already been defined in [draft-ietf-p2psip-base], and the last one is a new parameter defined here which is important in promotion procedure.

responsible_set
: It indicates that the peer should respond with the fraction of the overlay for which the responding peer is responsible.

num_resources
: It indicates that the peer should respond with the number of resources currently being stored by the peer.

uptime
: It indicates that the peer should respond with how long the peer has been up in seconds.

client_capability
: It indicates that the client should respond with a list of values which stands for its capability. For it is the main item to be considered in promotion. The values include the capability of CPU, Memory, Disk and so on. These values form an array, different integer index stands for different client's capability. For example integer 1 equals CPU while integer 2 stands for Memory. While the available memory and disk may be termed as quantitative values easily, CPU capabilities may not. However, an overlay may specify a benchmark in the configuration file for its participants to be used for measuring its CPU's capability as quantitative values.

peer_demotion
: This structure indicates that a peer may want another peer to be demoted after considering the situation of the overlay, which means a kind of passive demotion of peers. The response to this structure includes an array of Boolean values collected by the peer which is to be demoted. The array of values shows different responses to the peer's demotion, and these responses are from the peers which are direct successors to the peer to be demoted.

4.2.2. ProbeReq message

```
struct {
  probeInformationType     requested_info<0..2^8-1>;
} ProbeReq
```

The structure of ProbeReq gives a list of the piece of status information that the requester want to get.

4.2.3. ProbeAns message

```
typedf uint32<0...n>      client_capability_list;
typedf Boolean<0...n>     demotion_response_list;

struct {
  select (type) {
    case responsible_set:
       uint32           responsible_set;
    case num_resources:
       uint32           num_resources;
    case uptime:
       uint32           uptime;
    case client_capability:
       client_capability_list     capability_list;
    case peer_demotion:
       demotion_response_list     response_list;
  };
} ProbeInformationData;
```

The structure of ProbeInformationData is an item of the ProbeInformation message. It gives the value of related type.

```
struct {
  ProbeInformationType    type;
  uint8                   length;
  ProbeInformationData    value;
} ProbeInformation;
```

The structure of ProbeInformation gives the type and value of a probe item. What's more, it also contains the length of this message.

```
struct {
  ProbeInformation        probe_info<0..2^16-1>;
} ProbeAns;
```

This message is the response to ProbeReq, and the variable ProbeInformation is just related to variable ProbeInformationType in the structure ProbeReq.

For example, if a ProbeReq message contains a type of client_capability, its response ProbeAns must have a value of the client_capability. OverloadedPeer will use this pair of messages to collect information about the clients connecting to it, so that it can make a decision on which client to promote.

Moreover, if a ProbeReq message contains a type of peer_demotion, its response ProbeAns must have a value of the demotion_response_list. In this value there are responses from the neighbor of the peer which may be demoted. These responses have an influence on whether the demotion will continue.

5. Update

5.1. Update_type

```
enum { reservedevent(0),ue_promotion(1), peer_demotion(2), routing_table(3), (255
)}
Update_Type;
```

Update_Type defines three kinds of update event here. Parameter Ue_promotion means the promotion of client and peer_demotion indicates the demotion of peers while parameter routing_table stands for updating routingtables. The parameter reservedevent(0) means other extensions that can be made to this structure if needed. And it is useful for other Topology Plugins to make extension with this parameter. Even more important, different Topology Plugins can connect to RELOAD protocol stack through this extension mechanism, and they can also decide whether it is needed to deal with the received Update message through the Update_Type in the message. In this way, different Topology Plugins can work together to some degree.

5.2. SecutrityBlock

```
Struct{
  CertificateType      type;
  Opaque            certificate<0..2&#65342;16-1>;
}GenericCertificate;


Struct{
  GenericCertificat   certificates<0..2&#65342;16-1>;
  Signature         signature;
}SecurityBlock;
```

The two above structures have already been defined in [draft-ietf-p2psip-base]. Maybe here it will be desired to define a new CertificateType, because it is generated by Overloaded Peer which is different from that generated by ES (enrollment server). What's more, they are the items of message UpdateInformation.

5.3. UpdateInformation structure

```
struct{
  select (update_type) {
    case  ue_promotion:
      NodeId            Overloaded_Peer_id;
      NodeId            Promotion_Client_id;
      NodeId            expect-peer-id-top;
      NodeId            expect-peer-id-floor;
      SecurityBlock     securityblock;

    case  peer_demotion:
      NodeId            Administrating_Peer_id;
      NodeId            Demoting_Peer_id;
      SecurityBlock     securityblock;

    case  routing_table:
      RoutingTable_List     routing_table_list <0...n>;
  }
} UpdateInformation
```

Here is the value of arranged UpdateType in the message UpdateReq.

For the UpdateType ue_promotion, Overloaded_Peer_id offers the peer_id which is overloaded. And Promotion_Client_id gives the Client_id which is going to be promoted. When the client is in the procedure of promotion, it may have to change to another peer_id. If the client needs to do so, Expected_Peer_id can help to achieve it.

For the UpdateType peer_demotion, Administrating_Peer_id indicates the peer_id which initiates the command of demotion in the procedure of passive demotion. And Demoting_Peer_id tells the peer_id which is to be demoted.

The securityblock in the above two types can prevent malicious promotion or demotion to some degree.

For the UpdateType routing_table, it contains information about routing_table which has already been defined in [draft-ietf-p2psip-base], and exchanging routing_table is the main function of Update message. And more details can be referred to that draft.

5.4. UpdateReq structure

```
struct {
  UpdateType            update_type;
  uint32                length;
  UpdateInformation     Value;
} UpdateReq;
```

The structure of UpdateReq gives what kind of update it will do, and the information which is needed to achieve that. And the length of this message is also added.

5.5. UpdateAns structure

```
struct {
  uint32       update_response;
}UpdateAns;
```

The structure of UpdateAns is response to the UpdateReq message. The variable update_response may stand for success, fail, error and so on.

6. Leave

6.1. LeaveType

```
enum {reserved(0), Node_Leave(1), Peer_Demotion(2), (255)}
LeaveType
```

The LeaveType gives two kinds of coiditions. Firstly, one node is leaving the overlay; secondly, one peer is in the procedure of demotion no matter it is active or passive.

6.2. LeaveReq

```
struct {
  leaveType             type;
  select (type){
    case  Node_Leave:
      NodeId            leaving_node_id;
    case  Peer_Demotion:
      NodeId            demoting_peer_id;
  };
} LeaveReq;
```

The sturucture of LeaveReq contains the kind of leaving event and related node_id. When the type is Node_Leave, the variable leaveing_node_id may be a client_id or a peer_id, which means the node is leaving. But when the type is Peer_Demotion, the variable demoting_peer_id must be a peer_id. But in the procedure of demotion, there are two options. The client continues to use its peer_id before demotion or changes to the original client_id which is generated by ES (enrollment server). The former option is preferred, for the consideration that an earlier overloaded peer is more likely to get overloaded again in the near future than other peers. Hence it would be desirable to keep some highly capable clients available around these "vulnerable" peers, and be at hand for potential promotion operations. Moreover, one may expect this client migration to be an effective case for the population evolution for the clients as well as peers in the network, leading to a more load-balanced manner.

6.3. LeaveAns

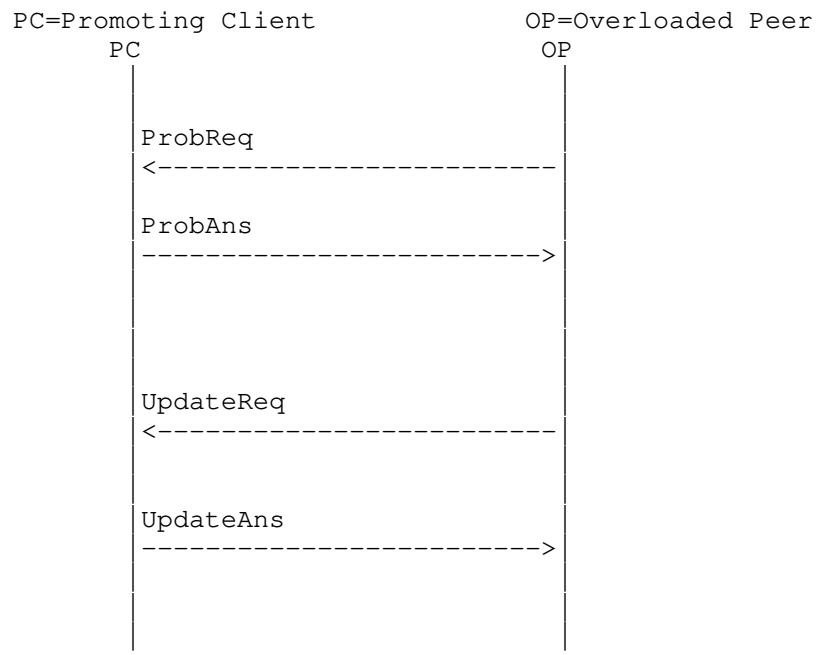
```
struct{
  LeaveType             type;
  select (type){
    case  Node_Leave:
      uint32            leave_response;
    case  Peer_Demotion:
      uint32            demotion_response;
  };
} LeaveAns;
```

The structure of LeaveAns is response to the LeaveReq message. For both of the LeaveType, the variable leave_response and demotion_response may stand for success, fail, error and so on.

7. Message Flow

In this section, three message flows are given respectively for passive promotion, active demotion and passive demotion of a UN.

7.1. Passive promotion



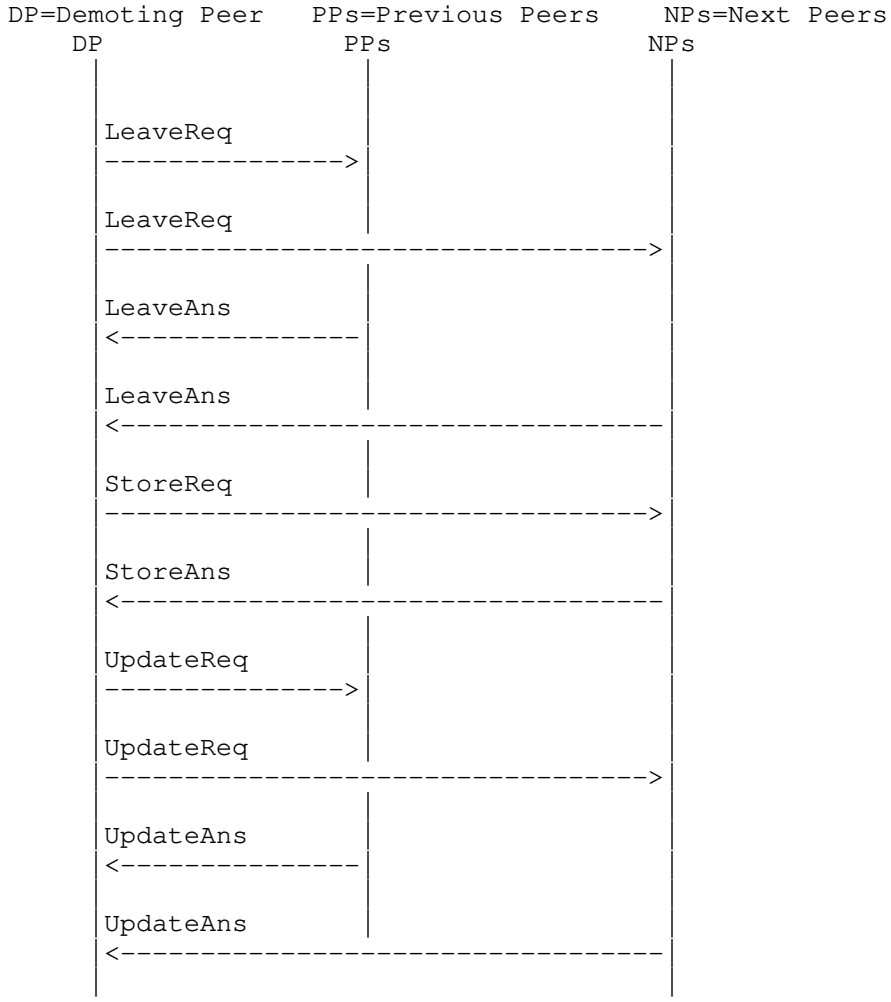
(1) After finding that it is overloaded, a peer sends ProbReq messages of type "client_capability" to the client directly connected to it in order to collect some information about these clients' capability.

(2) Receiving the ProbReq messages, a willing client returns a ProbAns message which reports the current available local resources (e.g. CPU, Memory or Disk capabilities) for serving as a peer if promoted.

(3) On the basis of the collected information, the overloaded peer selects the most appropriate client and sends an UpdateReq message of type "peer_promotion"to it informing the promotion decision to the selected client, along with correspondent delegation information (e.g. delegation certificate, UpdateType, UpdateInformation).

(4) The selected client returns UpdateAns message to acknowledge the command reception and initializes the procedure of acquiring the expected node_id from ES and joining the network again as a peer.

7.2. Active demotion



(1) A peer, decided to be demoted for some reason, sends LeaveReq messages to the nodes directly connected to it including its successors, predecessor and clients.

(2) The peers which receives these messages return LeaveAns, which stand for their attitude to the demotion. Actually the demoting peer don't wait for the response.

(3) Then the peer will continue the operation of data migration. It sends StoreReq messages to its successors in order to store the data for which it is responsible before.

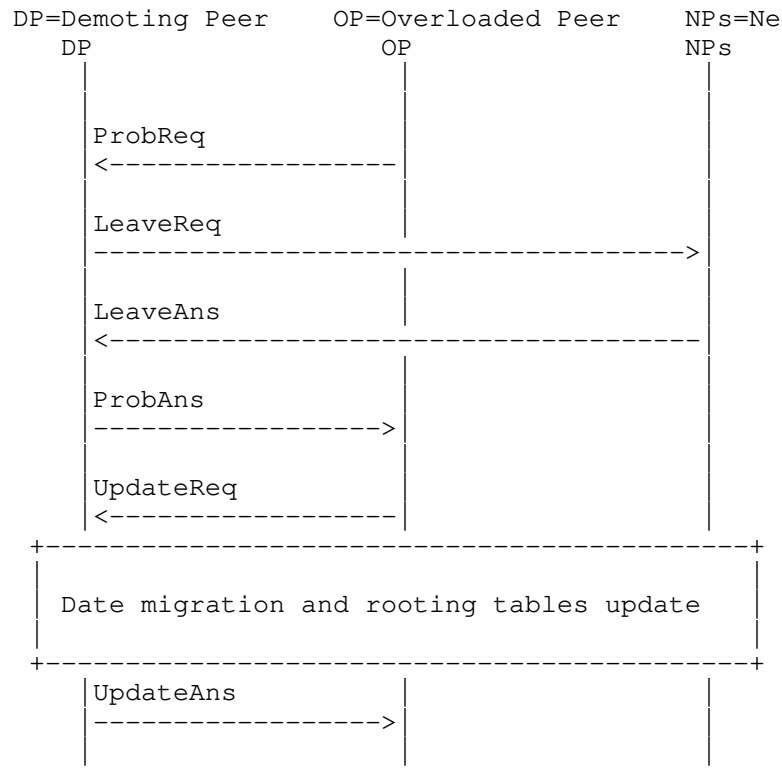
(4) The data recipients return StoreAns messages to acknowledge the data migration.

(5) When data migration is over, the peer sends UpdateReq messages to other peers in its routing table in order to update their routing tables.

(6) Informed peers delete the demoting peer from their routing table and return UpdateAns messages to acknowledge the update operation.

7.3. Passive demotion

In order to maintain a tolerable routing delay or some reason else, some peer may be demoted. And if they are not to do that actively, they will be made to do that. In other words, some peers which can be as a role of administrator will send orders to other peers for the sake of passive demotion.



(1) Firstly the commander sends ProbeReq messages to some peer in order find out whether it is acceptable to demote this peer.

(2) The to-be-demoted peer which receives ProbeReq messages sends LeaveReq messages to its successors, querying for their consents.

(3) The successors receiving LeaveReq messages returns LeaveAns messages whether they agree or disagree with the peer's demotion after considering their own situation taking account of the load to be migrated from the demoting peer to them after demotion.

(4) The to-be-demoted peer initiates a ProbeAns message which contains the responses it has just collected. And it returns it to the commander which sended ProbeReq messages.

(5) After receiving the ProbeAns messages, the commander makes a decision whether or not to enforce the demotion. If the demotion is to be carried out, it sends UpdateReq message to the peer in question informing it to be demoted.

(6) The informed peer continues to initialize a procedure of active demotion like the one described in the above subsection, and returns a UpdateAns message to the commander to report if the operation is successful or not.

8. Security Considerations

As stated above, the group of UNs manifests diversity in both physical capabilities and public morals in terms of serving as an overlay peer. Hence, it is reasonable to conduct explicit authorization to distinguish a promotion candidate's potential to serve as a peer from normal UN clients on one hand, and guarantee timely revocation to limit the impact of a misbehaving promoted UN-oriented peer on the other hand.

It is therefore proposed that:

- a qualified UN acquires a separate peer certificate to attest its capabilities and willingness to serve as a peer; and
- a UN-promoted peer's certificate is revoked if it fails to deliver expected performance while its client certificate remains intact.

Potential extensions to RELOAD include separate peer certification and proactive certificate revocation.

9. IANA Considerations

There are no IANA considerations associated to this memo.

10. Acknowledgements

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-15 (work in progress), May 2011.

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-03 (work in progress), October 2010.

Authors' Addresses

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

Email: Penjin@chinamobile.com

Lingli Deng
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Denglingli@chinamobile.com

Lifeng Le
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Lelifeng@chinamobile.com

Gang Li
China Mobile
Unit 2, 28 Xuanwumenxi Ave,
Xuanwu District
Beijing 100053
P.R.China

Email: Ligangyf@chinamobile.com

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

Email: maxiao_bupt@139.com

P2PSIP
Internet-Draft
Intended status: Informational
Expires: January 12, 2012

Y. Peng
W. Wang
Z. Hao
Y. Meng
ZTE Corporation

July 11, 2011

An SNMP Usage for RELOAD
draft-peng-p2psip-snmp-02

Abstract

This document defines a SNMP Usage for REsource LOcation And Discovery(RELOAD), The SNMP Usage provides the functionality of managing the RELOAD network. The SNMP Usage provides lookup service for the network manager's address stored in the overlay. The SNMP Usage also defines the method that allow the registrations to map a network manager'name to a specific node reachable through the overlay. The AppAttach method is used to establish a direct connection between nodes through which SNMP messages are exchanged.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 12, 2012.

Copyright Notice

Copyright (c) 2011 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 3
3. Network Management Requirements 3
4. Basic Operations and SNMP 4
5. Overview of SNMP Usage 4
6. Network Manager Registering 5
7. O-Node Looks up Network Manger and Forms a Direct Connection with the Manager 6
8. O-Node information Collection 8
9. Manager Forms a Direct Connection with O-Node 8
10. Network Manger Looks up the O-Node for a Resource 9
11. SNMP-REGISTRATION Kind Definition 10
12. Security Considerations 10
12.1. Overview 10
12.2. Generate Keys 11
12.3. Distribute Keys 11
12.4. SNMP Message Transmission 13
12.5. Timeliness 13
12.6. Update Keys 13
13. IANA Considerations 14
14. Acknowledgments 14
15. References 14
15.1. Normative References 14
15.2. Informative References 14
Appendix A. Additional Stuff 15
Authors' Addresses 15

1. Introduction

This document defines an SNMP Usage for REsource LOcation And Discovery (RELOAD),which can be used to manage the RELOAD network. It can provide important network management functions, such as adjusting the network configuration, monitoring the performance of the network, collecting real-time failure information, etc. These network management functions are essential for stable operation and high-quality services of the network. As traditional network management protocols (e.g., SNMP) cannot be directly applied to RELOAD network management, it is necessary to introduce new RELOAD usage of SNMP.

As defined in [I-D.ietf-p2psip-base], there are two kinds of nodes in RELOAD network: one is centralized servers, such as the Enrollment Server; the other is distributed nodes, such as Peer and Client. The management function of centralized servers can be carried out by traditional management methods, so we don't discuss the management of the centralized server in this document, and only focus on the management of the distributed nodes.

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 Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] and the RELOAD Base Protocol [I-D.ietf-p2psip-base] extensively in this document.

SNMP: Simple Network Management Protocol.

Manager: Network Manager.

Node: RELOAD Node, including both Peer and Client.

O-Node: Objective Node, which is managed by a network manager.

R-Node: Responsibility Node, which is responsible for storing the data according to P2P algorithm.

3. Network Management Requirements

RELOAD network SHOULD provide management functions for RELOAD Nodes. At the same time, as the RELOAD network is a provider of resource

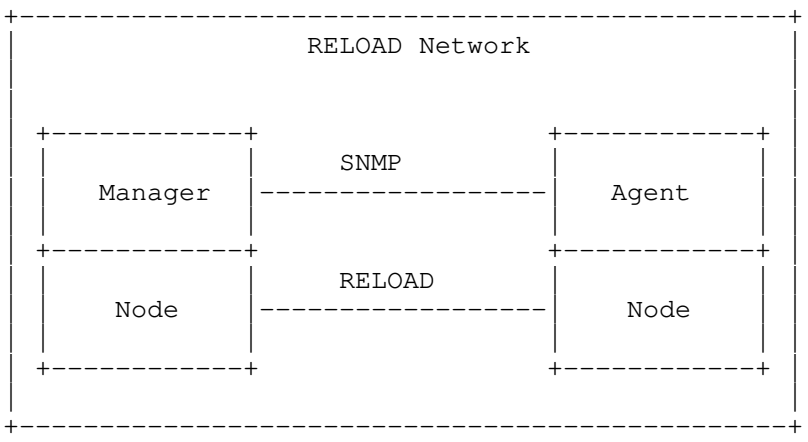
location and discovery services, it SHOULD provide monitoring of the resources and its associated operation in the network as an important part of the network management. Therefore, there are two kinds of the management targets in the RELOAD network: nodes and resources. The management functions of the nodes MAY include setting node name, software version or other configuration information, monitoring the number of the messages initiated, forwarded or processed by nodes, reporting program failure , message forwarding failure or other error on nodes. The management of resources MAY include tracing forwarded the RELOAD messages or processing flows of resources.

4. Basic Operations and SNMP

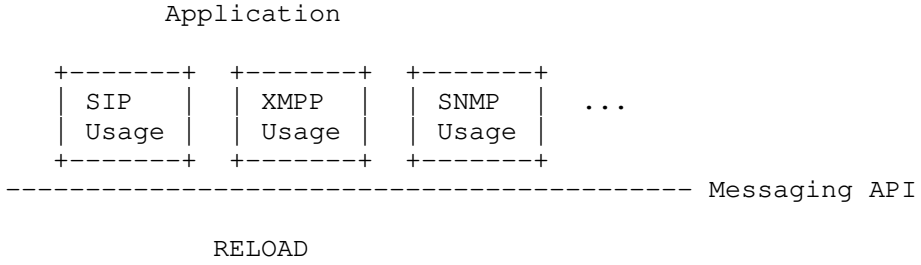
The management interactions between nodes can be abstracted into a few basic operations: 1). the network manager requests data of nodes and resources; 2). the network manager sets data of nodes and resources; 3). nodes initiate data reports to the network manager. A variety of management functions can be carried out by these basic operations or their combinations. This document adopts SNMP as a RELOAD Usage to achieve the management of the RELOAD network. The basic operations described above can be implemented by messages defined in SNMP, such as GetRequest, GetNextRequest, GetBulkRequest, Response, SetRequest, Trap, InformRequest.

5. Overview of SNMP Usage

The network manager needs to join the RELOAD network as a peer or a client before performing management operations. In other words, the manager function is deployed on the managing node, and agent function is deployed on managed node. The manager executes management operations of the RELOAD network through the agent deployed on the managed nodes. The protocol is RELOAD between nodes, and the protocol is SNMP between manager and agent. The diagram of system composition and protocol is as follow:



SNMP Usage's position in the RELOAD Architecture diagram is as follow:



6. Network Manager Registering

The Node ID of the network manager which acts as a provider of management service should be able to be found by other RELOAD nodes, thus managed nodes can send messages to the manager. The Node ID of network manager may not be fixed or predefined in advance. So a recognizable name is necessary and the managed nodes should find the Node ID of the manager through this fixed name. Therefore, it is necessary for the manager to register itself in the network after joining the network. In other words, the manager needs to store the mapping between its name and its Node ID in the RELOAD network. When a managed node wants to contact the manager, it needs to first look up the manager's Node ID corresponding to the predefined management service name. This registration is achieved by storing the name of the network manager and the structure of SnmpRegistration into the RELOAD network. The corresponding SNMP-REGISTRATION Kind-ID will be formally defined in the following chapter. It is proposed to store

the mapping of the manager's name to a destination list in this document. Therefore, a single Node ID as a special case for a destination list. The contents of a SnmpRegistration structure are as follows:

```
struct {

   opaque        contact_prefs<0..2^16-1>;

   Destination        destination_list<0..2^16-1>;

} SnmpRegistrationData;

struct {

   uint16        length;

   SnmpRegistrationData        data;

} SnmpRegistration;
```

The contents of the SnmpRegistration PDU are:

length

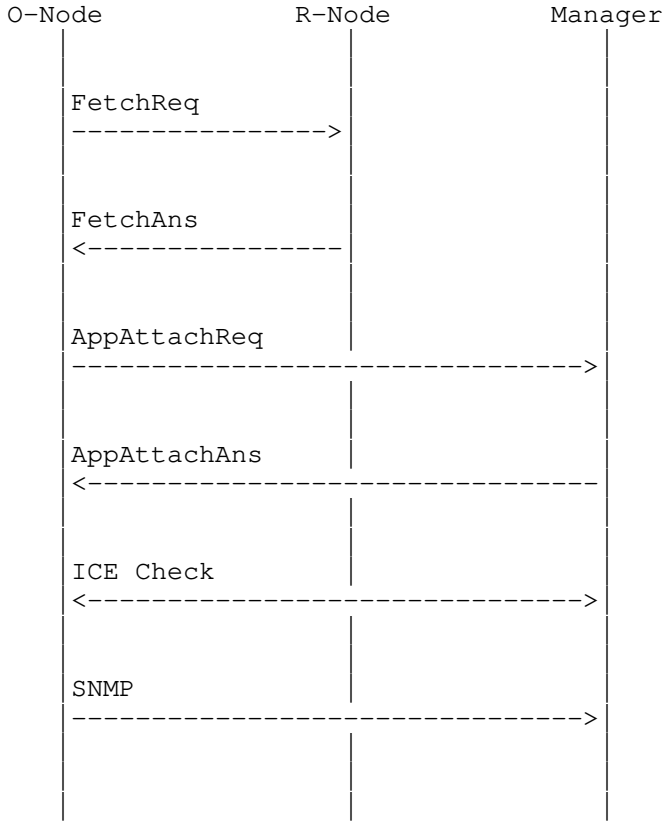
the length of the rest of the PDU

data

the contents of the registration data is an opaque string containing the network manager's contact preferences and a destination list for the peer.

7. O-Node Looks up Network Manger and Forms a Direct Connection with the Manager

When a target node needs to send messages to a network manager, it must check whether a connection has been established between itself and the network manager first. If the connection has been established, the target node will directly send messages to the manager. Otherwise, the target node must perform a query of the manager's Node ID, then exchanges ICE addresses with the network manager and checks connectivity and establishes connection, as shown below:



1. O-Node performs a Fetch for kind SNMP-REGISTRATION at the Resource-ID corresponding to the manager's name through the overlay to get the manager's Node ID.

2. The overlay transmits the FetchReq to R-Node that is responsible for the Resource-ID corresponding to the manager's name.

3. R-Node return FetchAns with the manager's Node ID through the overlay.

4. O-Node send an AppAttachReq with the manager's ID to the network manager through the overlay. The AppAttachReq contains O-Node's candidate addresses.

5. The network manager returns an AppAttachAns to O-Node. The AppAttachAns contains the manager's candidate addresses.

6. O-Node and Network manager perform ICE check to select a pair of appropriate addresses from candidate addresses to communicate with

each other.

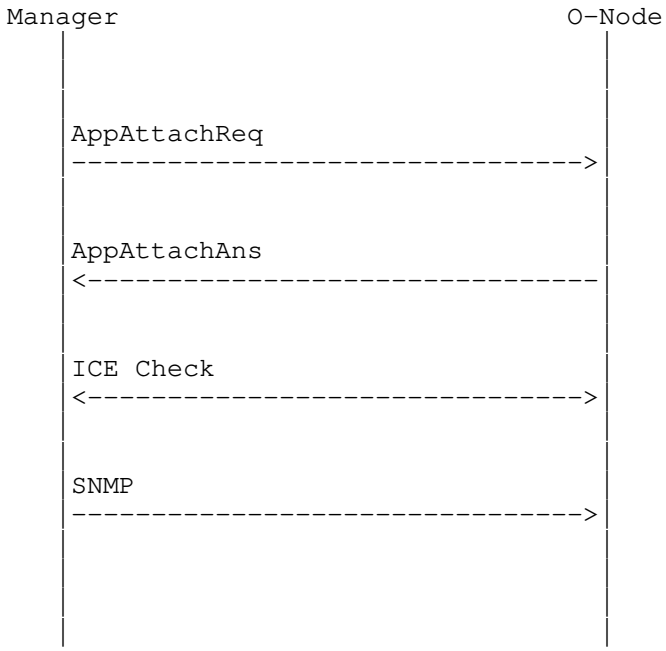
7. O-Node sends SNMP messages to the network manager through the selected connection.

8. O-Node information Collection

Before a network manager performs management tasks for nodes, it must first collect the Node ID and its status information of managed nodes. The way of a manager collecting the information of RELOAD nodes (including Peer and Client) is as follows: when a managed node joins the RELOAD network, it needs to get the name of a network manager from its configuration or an Enrollment Server; then this node connects to the network manager and registers its own information, such as node name, Node ID, status, etc., to the manager. The procedure of finding the manager and connecting to it has been introduced in the previous section. There are many other ways to collect information of managed nodes, which could be studied in future documents.

9. Manager Forms a Direct Connection with O-Node

When a network manager needs to send a management task message to a target node, it must check whether a connection has been established between itself and the target node. If the connection has been established, the network manager will directly send the message to the target node. Otherwise the network manager must exchange ICE addresses with the target node, then checks connectivity and establishes connection, as shown below:



1. Network manager sends an AppAttachReq to the O-Node through the overlay. The AppAttachReq contains the manager's candidate addresses.

2. O-Node returns an AppAttachAns to the network manager through the overlay. The AppAttachAns contains the O-Node's candidate addresses.

3. Network manager and O-Node perform ICE check to select a pair of appropriate addresses from candidate addresses to communicate with each other.

4. Network manager send SNMP messages to O-Node through the selected connection.

10. Network Manger Looks up the O-Node for a Resource

When a network manager needs to send a management task for resource, it is necessary that the network manager first gets the Node ID of the O-Node responsible for the resource so as to judge whether there is a connection with the O-Node. One way for the manager to get the Node ID of the O-Node responsible for the resource is to acquire the Node ID of the O-NODE responsible for the target resource through via_list of Forwarding Header in FindAns. The process is as follows:

Firstly, the network manager sends an FindReq to the RELOAD network, the target Reource ID is put into the destination_list of the FindReq. Then the RELOAD network routes FindReq to the node responsible for the target Resource ID according to its routing algorithm.

Secondly, the O-Node returns FindAns to the network manager through the RELOAD network. The first Node ID in the via_list of the Forwarding Header of the FindAns is the Node ID of the O-Node responsible for the target resource.

After the network manager gets the Node ID of O-Node, it will be able to judge whether there is a connection between itself and the O-Node. If the connection exists, the network manager may directly send SNMP message to the O-Node, otherwise it is necessary to establish a new connection to the O-Node.

11. SNMP-REGISTRATION Kind Definition

This section defines the SNMP-REGISTRATION kind.

Name SNMP-REGISTRATION

Kind ID The Resource Name for the SNMP-REGISTRATION Kind-ID is the Name of the network manager. The data stored is a SnmpRegistrationData, which can contain a destination list and contact preferences to the peer which is acting for the network manager.

Data Model The data model for the SNMP-REGISTRATION Kind-ID is single value.

Access Control USER-NODE-MATCH.

Data stored under the SNMP-REGISTRATION kind is of type SnmpRegistration. A destination list can be used to reach the network manager.

12. Security Considerations

12.1. Overview

The goals of SNMP Usage forRELOAD security are same as SNMP security [RFC3414].

1. Provide for verification that each received SNMP message has not

been modified during its transmission through the network.

2. Provide for verification of the identity of the user on whose behalf a received SNMP message claims to have been generated.

3. Provide for detection of received SNMP messages, which request or contain management information, whose time of generation was not recent.

4. Provide, when necessary, that the contents of each received SNMP message are protected from disclosure.

There are three solutions to the security problem in SNMP Usage for RELOAD. The first option is shared key based solution, which is SNMPv3 security solution (USM). The second option is PKI based security solution, which is to use the certificate of RELOAD to authenticate and encrypt the SNMP messages. The third option is DTLS based security solution, which uses the secure DTLS links to transfer the SNMP message.

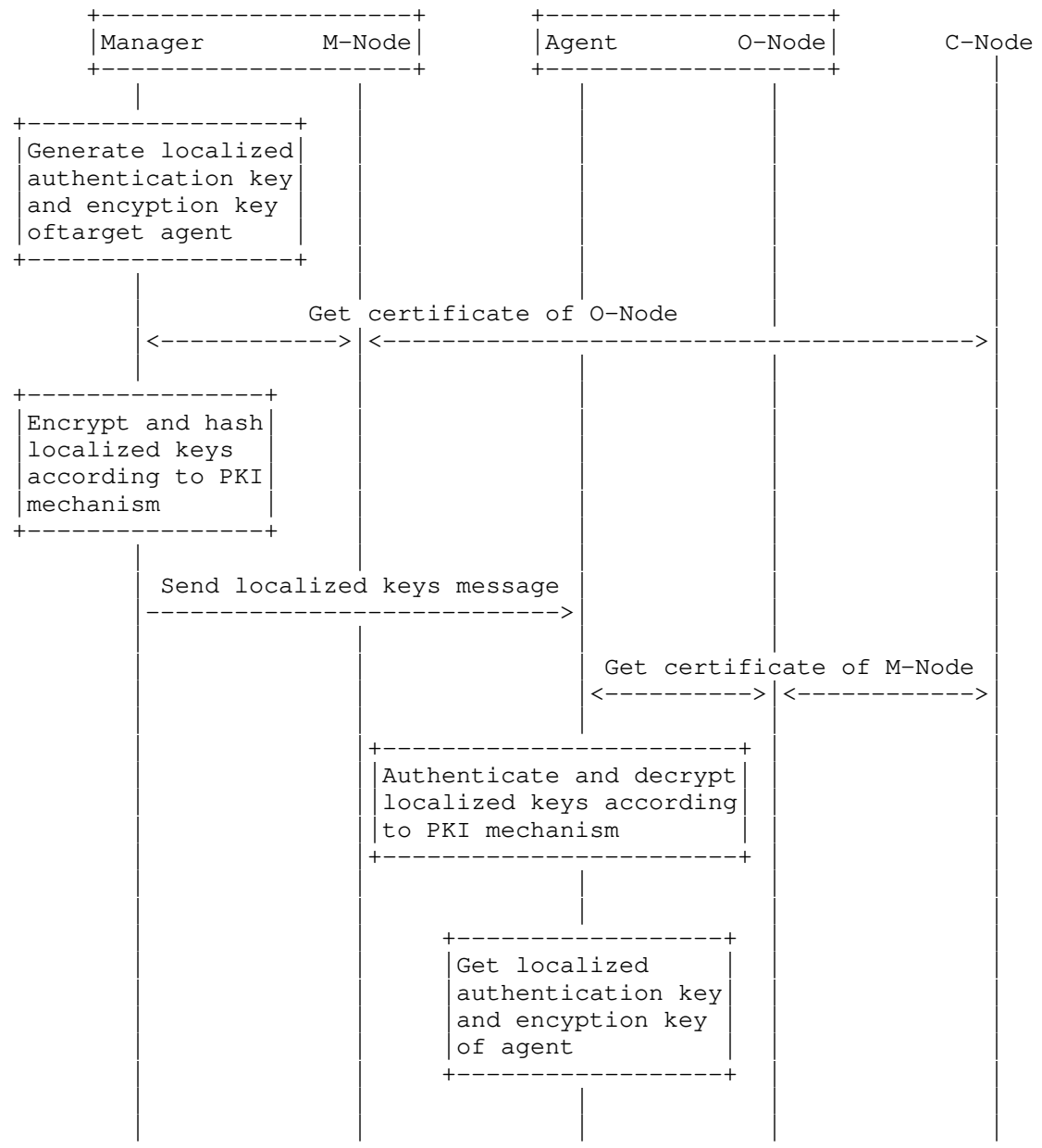
Because the first option has been supported widely by the SNMPv3 devices, we recommend the first option. But it’s difficult to distribute securely the keys to large numbers of agents in the SNMP security solution. In this document, we distribute the key to each agent by the certificate mechanism of RELOAD. This method will be introduced below.

12.2. Generate Keys

Every user has an authentication password and an encryption password which is stored in SNMP manager. The user!_s authentication key and encryption key are generated from these passwords. Due to reduce the risk of keys leakage, SNMP security model requires to generate different localized key for each agent. The localized key is generated from concatenating user key and engine ID of the agent. The engine ID of agent can be got by the discovery process defined in SNMP USM.

12.3. Distribute Keys

In order to ensure distributed keys not to be intercepted and tampered in transmission path, we use certificate provided by RELOAD to distribute keys. The process is as follows.



1. The manager generates the localized authentication key and encryption key for the SNMP agent and the user.

2. The manager needs to get the certificate of O-Node, on which the target agent resides, to encrypt the key message. So it directs

M-Node, on which the manager resides, to get the arbitrary user!_s certificate of O-Node from the RELOAD network.

3. According to PKI mechanism, the manager encrypts the local authentication key and encryption key, and computes digest of these keys, with the certificate of O-Node and the private key of M-Node.

4. The manager sends this encrypted keys message and its digest to the target agent. The distribution message is extended SNMP message, so RELOAD certificate(4) need to be added to options of the msgSecurityModel in SNMP USM message.

5. The agent needs to get the certificate of M-Node to authenticate the key message. So it directs O-Node get the user's certificate of M-Node from the RELOAD network.

6. According to PKI mechanism, the agent authenticates and decrypts the key message with the certificate of M-Node and the private key of O-Node.

7. The agent gets its own localized authentication key and encryption key from the key message, then may do related process.

12.4. SNMP Message Transmission

After the manager and the agent know their shared keys, they can protect the SNMP messages transmitted between them by using the keys. The method detail of protecting SNMP messages is defined in SNMP USM, so we won!_t describe it in this document.

12.5. Timeliness

In order to protect the management messages from being delayed and replayed by malicious users, timeliness check is demanded in SNMPv3 security model. It is a kind of loose clock synchronization mechanism. We still use this mechanism to against being malicious delayed and replayed in SNMP Usage for RELOAD.

12.6. Update Keys

In order to protect the keys from being disclosing, it is recommended to update local keys timely in SNMP security model, and the update mechanism is also defined. In this document, we still use the same mechanism to update keys.

13. IANA Considerations

This document has no IANA Considerations.

14. Acknowledgments

This draft is based on "REsource LOcation And Discovery (RELOAD) Base Protocol" draft by C. Jennings, B. Lowekamp, E. Rescorla, S. Baset and H. Schulzrinne.

This draft make a reference to "A SIP Usage for RELOAD" draft by C. Jennings, B. Lowekamp, Ed., E. Rescorla, S. Baset, H. Schulzrinne.

Thanks to the many people of the IETF P2PSIP WG whose many drafts we have learned.

15. References

15.1. 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", August 2010.

[I-D.ietf-p2psip-sip]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", July 2010.

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

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

15.2. 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", July 2008.

[I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs",

draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

Appendix A. Additional Stuff

Authors' Addresses

YongLin Peng
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13776637274
Email: peng.yonglin@zte.com.cn

Wei Wang
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13851658076
Email: wang.wei108@zte.com.cn

ZhenWu Hao
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13382087596
Email: hao.zhenwu@zte.com.cn

Yu Meng
ZTE Corporation
Nanjing, 210012
China

Phone: +86 18651806839
Email: meng.yu@zte.com.cn

P2PSIP
Internet-Draft
Intended status: Informational
Expires: April 21, 2013

Y. Peng
W. Wang
Z. Hao
Y. Meng
ZTE Corporation

October 18, 2012

An SNMP Usage for RELOAD
draft-peng-p2psip-snmp-05

Abstract

This document defines an SNMP Usage for REsource LOcation And Discovery(RELOAD). The objective of SNMP Usage is to provide the functionality of managing the RELOAD network. In particular, the SNMP Usage provides the following functions: (a) defining the method that allow the registrations to map a network manager's name to a host node reachable in the overlay, and (b) providing lookup service for the node hosts the network manager in the overlay. Then the AppAttach method is used to exchange addresses between nodes to establish a direct connection through which SNMP messages are exchanged.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on April 21, 2013.

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 3
3. Network Management Requirements 4
4. Basic Operations and SNMP 5
5. Overview of SNMP Usage 5
6. SNMP Usage Architecture 6
7. Abstract Service Interfaces(ASI) 7
7.1. SNMP-RELOAD Application Primitive 7
7.1.1. getNodeForResource ASI 7
7.1.2. returnNodeForResource ASI 7
7.1.3. getAddressForNode ASI 8
7.1.4. returnAddressForNode ASI 8
7.2. RELOAD Node(M-Node/O-Node) primitive 8
7.2.1. getNodeForResource ASI 8
7.2.2. returnNodeForResource ASI 9
7.2.3. exchangeCandidateAddressList ASI 9
7.2.4. registerManagerReq ASI 9
7.2.5. registerManagerAns ASI 10
8. Managed Object Definitions for RELOAD 10
9. Network Manager Registration and Lookup 15
10. An SNMP Entity Forms a Direct Connection with Another SNMP Entity 17
11. O-Node information Collection 19
12. O-Node Lookup by the Network Manager for a Resource 20
13. Definition of SNMP-REGISTRATION Kind 21
14. Security Considerations 22
15. IANA Considerations 22
16. Acknowledgments 23
17. References 23
17.1. Normative References 23
17.2. Informative References 24
Appendix A. Additional Stuff 24
Authors' Addresses 25

1. Introduction

This document defines an SNMP Usage for RELOAD, which can be used to manage the RELOAD network. It can provide important network management functions, such as changing the network configuration, monitoring the performance of the network, collecting real-time status/failure information, etc. These network management functions are essential for stable operation and high-quality services offered by the network. Since the traditional network management protocols (e.g., SNMP) cannot be directly applied to RELOAD network management, it is necessary to introduce new RELOAD usage of SNMP.

As defined in [I-D.ietf-p2psip-base], there are two kinds of network elements in RELOAD network: centralized servers, such as the Enrollment Server; distributed nodes, such as Peer and Client. The management function of centralized servers can be carried out by traditional management methods, and aren't discussed in this document. We focus on the management of the distributed nodes called as RELOAD Nodes in this draft.

When the manager starts up, it needs to register the mapping between its name and Node-ID into the RELOAD network, in order to be recognized and found by the managed RELOAD Nodes. So only the name of manager needs be fixed, and needs be known beforhand by RELOAD Nodes. Then, the RELOAD Nodes can get the manager's address and connect with it. Then the Nodes and the manager can exchange management messages through this link.

Not only RELOAD Nodes are managed object, but a RELOAD resource is a managed object as well, such as some data stored in RELOAD network by SNMP-RELOAD application.

The basic mechanism of SNMP Usage is the same as SIP Usage for RELOAD[I-D.draft-ietf-p2psip-sip]. It is easier to understood the SNMP Usage, if someone has the backgroud of SIP Usage draft.

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].

In this document, we use the definitions from Concepts and Terminology as described in the following drafts:

Peer to Peer SIP [I-D.ietf-p2psip-concepts], RELOAD Base Protocol [I-D.ietf-p2psip-base], SNMPv3 [RFC3411], TLS TM for SNMP [RFC5953] .

SNMP: Simple Network Management Protocol.

Entity: SNMP Entity, including both Manager and Agent, which resides in RELOAD Node.

Manager: SNMP Manager, which resides in RELOAD Node.

Agent: SNMP Agent, which resides in RELOAD Node.

LCD: Local Configuration Datastore.

Node: RELOAD Node, including both Peer and Client, which SNMP manager or agent resides in.

ReDiR: Recursive Distributed Rendezvous.

M-Node: Management Node, which is the RELOAD Node which SNMP Manager resides in.

O-Node: Objective Node, which is the RELOAD Node managed by a network manager, which SNMP agent resides in.

R-Node: Responsible Node, which is the RELOAD Node responsible for storing the data according to P2P algorithm.

SNMP-RELOAD Application: It provides the functions related to RELOAD for SNMP applications, such as getting available address for Node-ID and getting Node-ID for Resource ID. SNMP applications can implement the management for RELOAD network by SNMP-RELOAD Application.

3. Network Management Requirements

SNMP usage SHOULD or MAY provide the following functions and mechanisms:

i. SNMP usage for RELOAD SHOULD provide the management functions for RELOAD Nodes. Such as setting node name, software version or other configuration information, monitoring the number of the messages initiated, forwarded or processed by nodes, reporting program failure, message forwarding failure or other error on nodes.

ii. SNMP usage for RELOAD SHOULD provide the management functions for RELOAD resource. Such as tracking the RELOAD messages is forwarded, processing flows of resources.

iii. SNMP usage for RELOAD SHOULD provide mechanisms for SNMP

entities to discover each other based on RELOAD Node-ID.

iv. SNMP usage for RELOAD SHOULD provide mechanisms for SNMP entities to establish a secure connection between each other.

v. SNMP usage for RELOAD SHOULD provide mechanisms for SNMP manager to discover the RELOAD NodeID associated to a given Resource-ID.

vi. SNMP usage for RELOAD SHOULD provide mechanisms for SNMP entities to traverse the NAT in front of the SNMP entities which they will connect to.

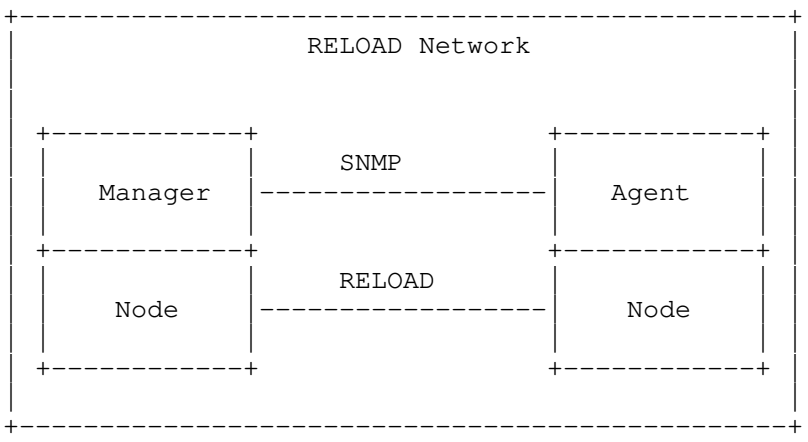
vii. SNMP usage for RELOAD MAY provide mechanisms for SNMP entities to discover the SNMP manager based on manager names or functions.

4. Basic Operations and SNMP

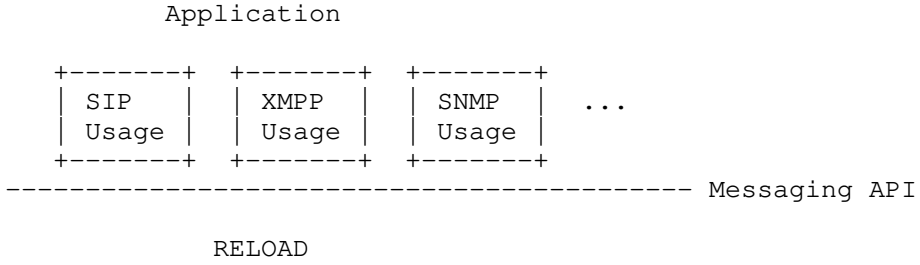
Management interactions between nodes can be abstracted into the following basic operations: a) the network manager requests data of nodes and resources; b) the network manager sets data of nodes and resources; c) nodes initiate data reports to the network manager. A variety of management functions can be carried out by these basic operations or their combinations. This document adopts SNMP as a RELOAD Usage to achieve the management of the RELOAD network. The basic operations described above can be implemented by messages defined in SNMP, such as GetRequest, GetNextRequest, GetBulkRequest, Response, SetRequest, Trap, and InformRequest.

5. Overview of SNMP Usage

The SNMP entity is deployed as an application on RELOAD Nodes in the SNMP usage for RELOAD. In other words, each SNMP entity is associated with a RELOAD Node. SNMP entities discover other entities (agents or managers) by RELOAD mechanisms and connect with other SNMP entities. Therefore, SNMP entities talk to each other using SNMP protocol on dedicated connections, while RELOAD protocols are used for Node discovery and connection setup. The following figure shows the system composition and protocol:

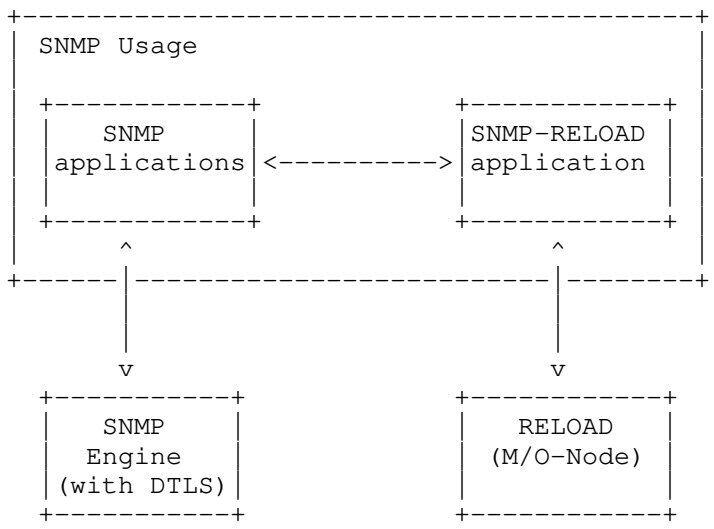


The following figure shows SNMP Usage's position in the RELOAD Architecture:



6. SNMP Usage Architecture

This document defines SNMP Usage Architeture, which includes SNMP-RELOAD application and SNMP applications in the Simple Network Management Protocol (SNMP) architecture defined in [RFC3411]. The SNMP-RELOAD Application will provide the functions related to RELOAD, such as getting available address for Node-ID and getting Node-ID for Resource-ID, to SNMP applications to implement the management for RELOAD network. This document identifies and describes some key aspects that need to be considered for SNMP usage for RELOAD. The following figure depicts SNMP usage architecture.



7. Abstract Service Interfaces(ASI)

Abstract service interfaces describe only the conceptual interfaces between SNMP-RELOAD application and the other subsystems, and it is intended to help clarify the externally observable behavior. They should not be interpreted as APIs or as requirements statements for APIs. More description about abstract service interfaces is in [RFC3411].

7.1. SNMP-RELOAD Application Primitive

7.1.1. getNodeForResource ASI

The getNodeForResource ASI is provided for SNMP applications by SNMP-RELOAD application, and it is used to get the Node-ID of the RELOAD Node that is responsible for a resource.

```
getNodeForResource(

    IN     resourceName     -- managed resource name

)
```

7.1.2. returnNodeForResource ASI

The returnNodeForResource ASI is used to return the Node-ID of RELOAD Node that is responsible for resource to SNMP applications by SNMP-RELOAD application.

```
result =     -- SUCCESS or errorIndication

returnNodeForResource(

    IN     resourceName     -- managed resource name

    IN     nodeID     -- node that responsible for managed resource
name

)
```

7.1.3. getAddressForNode ASI

The getAddressForNode ASI is provided for SNMP applications (e.g. Command Application, Notification Application) by SNMP-RELOAD application, and it is used to get the address of the other side for SNMP communication.

```
getAddressForNode(

    IN     nodeID     -- destination node

)
```

7.1.4. returnAddressForNode ASI

The returnNodeForResource ASI is used to return the address of the other side for SNMP communication.

```
result =     -- SUCCESS or errorIndication

returnAddressForNode(

    IN     nodeID     -- destination node

    IN     transportAddress     -- destination network address

)
```

7.2. RELOAD Node(M-Node/O-Node) primitive

7.2.1. getNodeForResource ASI

The getNodeForResource ASI is provided for SNMP-RELOAD application by RELOAD Node(M-Node/O-Node), and it is used to get Node-ID of RELOAD Node that is responsible for resource. The difinition of getNodeForResource is above.

7.2.2. returnNodeForResource ASI

The returnNodeForResource ASI is used to return the Node-ID of RELOAD Node that is responsible for resource to SNMP-RELOAD application by RELOAD Node. The difinition of returnNodeForResource is above.

7.2.3. exchangeCandidateAddressList ASI

The exchangeCandidateAddressList ASI is used by SNMP-RELOAD application and RELOAD Node to exchange the address list with each other for SNMP communication, and these address lists will be used for NAT traversal by the ICE.

```
exchangeCandidateAddressList(

    IN     nodeID     -- destination node

    IN     ufrag     -- the username fragment (from ICE)

    IN     password     -- the ICE password

    IN     candidateAddressList     -- sender's candidate address
list

)
```

the Elements of candidateAddress of candidateAddressList including: IP address, LinkType, etc.

In order to implement ICE, these items need to be added into LCD(Local Configuration Datastore):

Ufrag

Password

LinkType: DTLS-UDP-NO-ICE, DTLS-UDP-ICE, TLS-TCP-NO-ICE.

7.2.4. registerManagerReq ASI

The registerManagerReq ASI is provided for SNMP-RELOAD application by RELOAD M-Node, and it is used to register the Node-ID of the RELOAD Node which hosts the Manager.

```
registerManagerReq(

    IN     managerName     -- the name of Manager
```

```
      IN     nodeID     -- the Node-ID of the RELOAD Node which Manager
   resides in

   )
```

7.2.5. registerManagerAns ASI

The registerManagerAns ASI is used to return the result of registering to SNMP-RELOAD application by RELOAD M-Node.

```
result =     -- SUCCESS or errorIndication
```

8. Managed Object Definitions for RELOAD

```
SNMP-RELOAD-MIB DEFINITIONS ::= BEGIN

IMPORTS
  MODULE-IDENTITY, OBJECT-TYPE, OBJECT-IDENTITY, mib-2, Counter32,
  Gauge32, Integer32, NOTIFICATION-TYPE
    FROM SNMPv2-SMI          -- RFC 2578 or any update thereof
  MODULE-COMPLIANCE, OBJECT-GROUP,
  NOTIFICATION-GROUP
    FROM SNMPv2-CONF         -- RFC 2580 or any update thereof
  TimeStamp
    FROM SNMPv2-TC           -- RFC 2579 or any update thereof
  ;

snmpReloadMIB MODULE-IDENTITY
  LAST-UPDATED "201202280000Z"
  ORGANIZATION "P2PSIP Working Group"
  CONTACT-INFO "WG-EMail:   p2psip@ietf.org"

  DESCRIPTION  "
      SNMP Usage for RELOAD MIB

      Copyright (c) 2011 IETF Trust and the persons identified as
      the document authors.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject
      to the license terms contained in, the Simplified BSD License
      set forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (http://trustee.ietf.org/license-info)."

     REVISION     "201202280000Z"
```

```
        DESCRIPTION  "This version of this MIB module is part of
                      draft-peng-p2psip-snmp-04; see the draft itself
            for full legal notices."
      ::= { mib-2 199 }


   -- ************************************************
   -- subtrees of the SNMP-RELOAD-MIB
   -- ************************************************

   snmpReloadNotifications OBJECT IDENTIFIER ::= { snmpReloadMIB 0 }
   snmpReloadObjects       OBJECT IDENTIFIER ::= { snmpReloadMIB 1 }
   snmpReloadConformance   OBJECT IDENTIFIER ::= { snmpReloadMIB 2 }

   -- ************************************************
   -- snmpReloadObjects - Objects
   -- ************************************************

   -- Configuration Objects
   snmpReloadConfig    OBJECT IDENTIFIER ::= { snmpReloadObjects 1 }

   snmpReloadConfigVersion OBJECT-TYPE
     SYNTAX      Gauge32
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
         "The version of snmpReloadConfigItem."
     ::= { snmpReloadConfig 1 }


   snmpReloadConfigLastChanged OBJECT-TYPE
     SYNTAX      TimeStamp
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
         "The value of sysUpTime.0 when the snmpReloadConfigItem was
         last modified through any means, or 0 if it has not been
         modified since the command responder was started."
     ::= { snmpReloadConfig 2 }

   snmpReloadConfigItem OBJECT IDENTIFIER ::= { snmpReloadConfig 3 }

   snmpReloadNodeName        OBJECT-IDENTITY
      STATUS      current
      DESCRIPTION
          "The name of RELOAD Node."
      ::= { snmpReloadConfigItem 1 }
```

```
   snmpReloadNodeId        OBJECT-IDENTITY
      STATUS      current
      DESCRIPTION
          "node-id-length  This element contains the length of
      a NodeId(NodeIdLength) in bytes.  This value MUST be
      between 16 (128 bits) and 20 (160 bits).  If this
      element is not present, the default of 16 is used."
      ::= { snmpReloadConfigItem 2 }

   snmpReloadNodeType     OBJECT-TYPE
      SYNTAX      Integer32(0..9)
      MAX-ACCESS  read-write
      STATUS      current
      DESCRIPTION
          "the type of RELOAD Node.
          Definition of values as follows:
          Client(0),
          Peer(1)
          "
      ::= { snmpReloadConfigItem 3 }

   snmpReloadLogPrintLevel    OBJECT-TYPE
      SYNTAX      Integer32(0..9)
      MAX-ACCESS  read-write
      STATUS      current
      DESCRIPTION
          "the type of RELOAD Node.
           Definition of values as follows:
           debug(3),
           info(2),
           warn(1),
           error(0)
           "
      ::= { snmpReloadConfigItem 4 }

   snmpReloadNotificationEnable     OBJECT-TYPE
      SYNTAX      Integer32(0..9)
      MAX-ACCESS  read-write
      STATUS      current
      DESCRIPTION
          "Whether are notifications sent.
           Definition of values as follows:
           Disable(0),
           Enable(1)
          "
      ::= { snmpReloadConfigItem 5 }
```

```
   -- The snmpReloadFailures Group
   snmpReloadFailures  OBJECT IDENTIFIER ::= { snmpReloadObjects 2 }

   snmpReloadMessageForwardFailures OBJECT-TYPE
     SYNTAX       Counter32
     MAX-ACCESS   read-only
     STATUS      current
     DESCRIPTION
         "The number of times RELOAD message failed to be forwarded,
         for any reason."
     ::= { snmpReloadFailures 1 }

   snmpReloadDataUpdateFailures OBJECT-TYPE
     SYNTAX       Counter32
     MAX-ACCESS   read-only
     STATUS      current
     DESCRIPTION
         "The number of times RELOAD data failed to be updated,
         for any reason."
     ::= { snmpReloadFailures 2 }


   -- ************************************************
   --  snmpReloadNotifications - Notifications Information
   -- ************************************************

   snmpReloadMessageForwardFailNotification NOTIFICATION-TYPE
       OBJECTS { snmpReloadMessageForwardFailures }
       STATUS  current
       DESCRIPTION
           "Notification that RELOAD message failed to be forwarded."
       ::= { snmpReloadNotifications 1 }

   snmpReloadDataUpdateFailNotification NOTIFICATION-TYPE
       OBJECTS { snmpReloadDataUpdateFailures }
       STATUS  current
       DESCRIPTION
           "Notification that RELOAD data failed to be updated."
       ::= { snmpReloadNotifications 2 }


   -- ************************************************
   -- snmpReloadCompliances - Conformance Information
   -- ************************************************

   snmpReloadCompliances OBJECT IDENTIFIER ::= {snmpReloadConformance 1}
   snmpReloadGroups OBJECT IDENTIFIER ::= { snmpReloadConformance 2 }
```

```
   -- ************************************************
   -- Compliance statements
   -- ************************************************

   snmpReloadCompliance MODULE-COMPLIANCE
     STATUS      current
     DESCRIPTION
         "The compliance statement for SNMP engines that support
     the SNMP-RELOAD-MIB"
     MODULE
         MANDATORY-GROUPS { snmpReloadConfigGroup,
                            snmpReloadFailuresGroup,
                            snmpReloadNotificationGroup }
     ::= { snmpReloadCompliances 1 }


   -- ************************************************
   -- Units of conformance
   -- ************************************************

   snmpReloadConfigGroup OBJECT-GROUP
     OBJECTS {
     snmpReloadConfigVersion,
     snmpReloadConfigLastChanged,
     snmpReloadNodeType,
     snmpReloadLogPrintLevel,
     snmpReloadNotificationEnable
     }
     STATUS      current
     DESCRIPTION
         "A collection of objects for maintaining configuration
     information of an SNMP engine that implements
     the SNMP Usage for RELOAD."
     ::= { snmpReloadGroups 1 }

   snmpReloadFailuresGroup OBJECT-GROUP
     OBJECTS {
     snmpReloadMessageForwardFailures,
     snmpReloadDataUpdateFailures
     }
     STATUS      current
     DESCRIPTION
         "A collection of objects for failures
     information of an SNMP engine that implements
     the SNMP Usage for RELOAD."
     ::= { snmpReloadGroups 2 }
```

```
   snmpReloadNotificationGroup NOTIFICATION-GROUP
     NOTIFICATIONS {
         snmpReloadMessageForwardFailNotification,
         snmpReloadDataUpdateFailNotification
     }
     STATUS current
     DESCRIPTION
         "Notifications"
     ::= { snmpReloadGroups 3 }

   END
```

9. Network Manager Registration and Lookup

The Node-ID of the network manager which acts as a provider of management service should be discovered by agents on RELOAD nodes, so that the agents can send messages to the manager. The Node-ID of network manager may not be fixed or predefined in advance. So a recognizable name is necessary and the managed nodes should find the Node-ID of the manager through this fixed name. Therefore, it is necessary for the manager to register itself in the network after joining the network. In other words, the manager needs to store the mapping between its name and its Node-ID in the RELOAD network. When an agent wants to contact the manager, it needs to first look up the manager's Node-ID corresponding to the predefined management service name. This registration is achieved by storing the name of the network manager and the structure of SnmpRegistration into the RELOAD network. The corresponding SNMP-REGISTRATION Kind-ID will be formally defined in the following chapter. It is proposed to store the mapping of the manager's name to a destination list in this document. Therefore, a single Node-ID as a special case for a destination list. The contents of a SnmpRegistration structure are as follows

```
   struct {

      opaque        contact_prefs<0..2^16-1>;

      Destination        destination_list<0..2^16-1>;

   } SnmpRegistrationData;

   struct {

      uint16        length;
```

```
   SnmpRegistrationData        data;

} SnmpRegistration;
```

The contents of the SnmpRegistration PDU are:

length

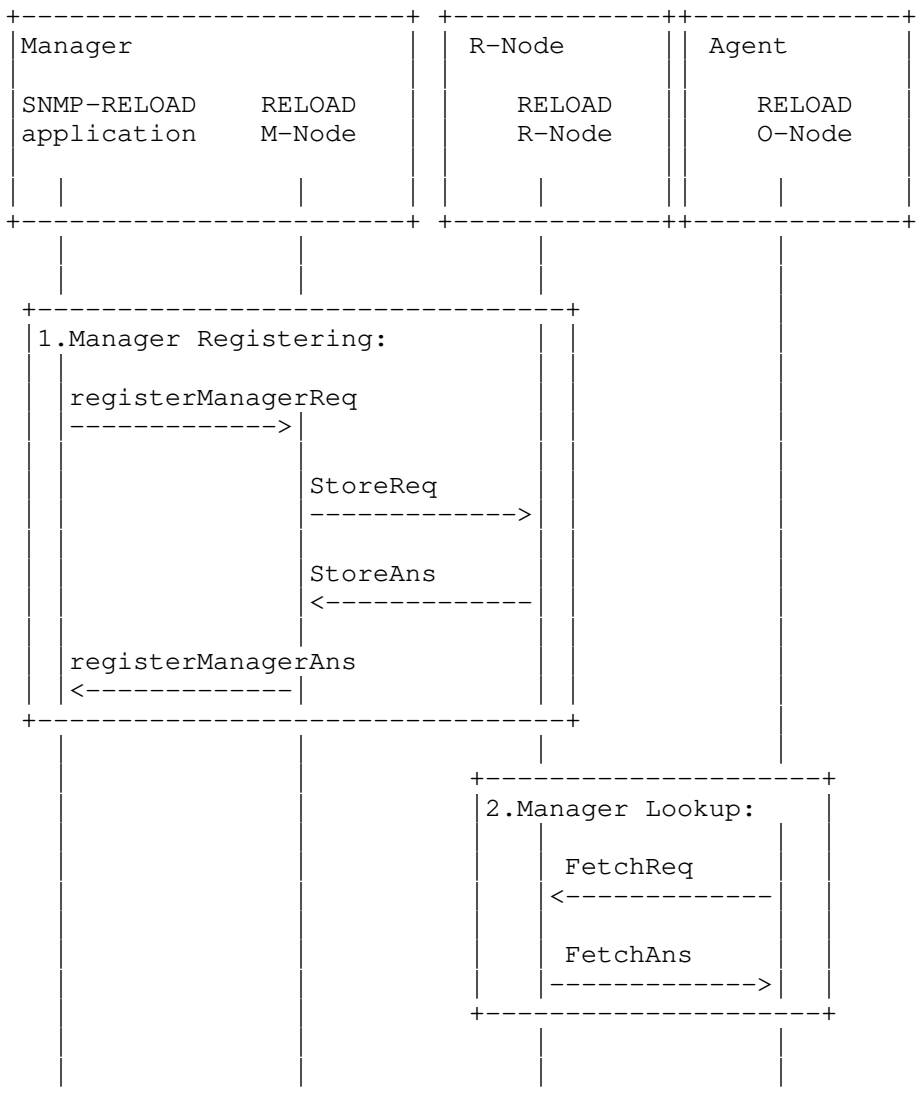
the length of the rest of the PDU

data

the contents of the registration data is an opaque string containing the network manager's contact preferences and a destination list for the peer.

When an agent needs to contact a network manager, it must perform a query of SnmpRegistration by FetchReq message to get the manager's Node-ID.

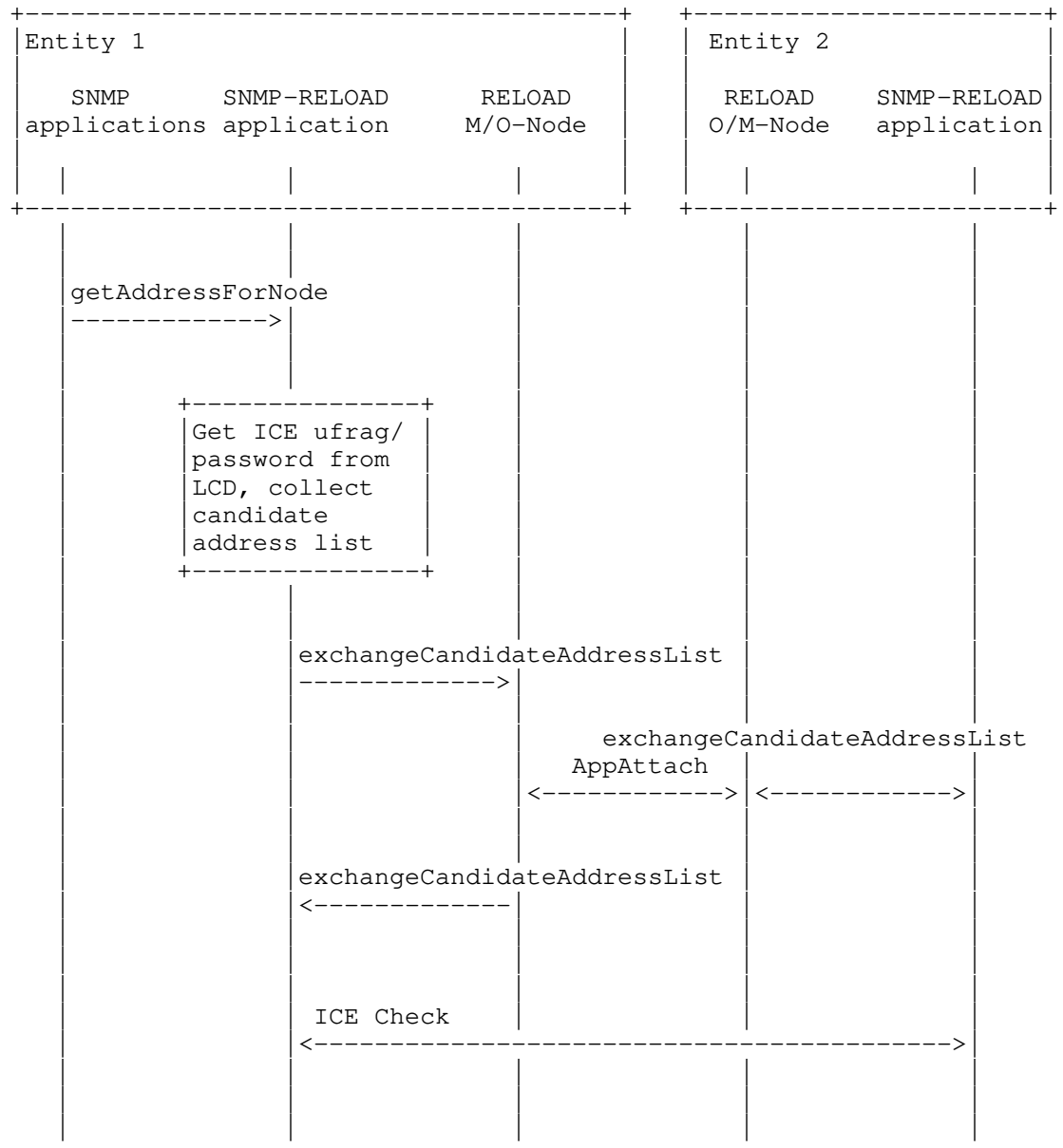
The process for Network Manager Registration and Lookup is as shown the following figure:

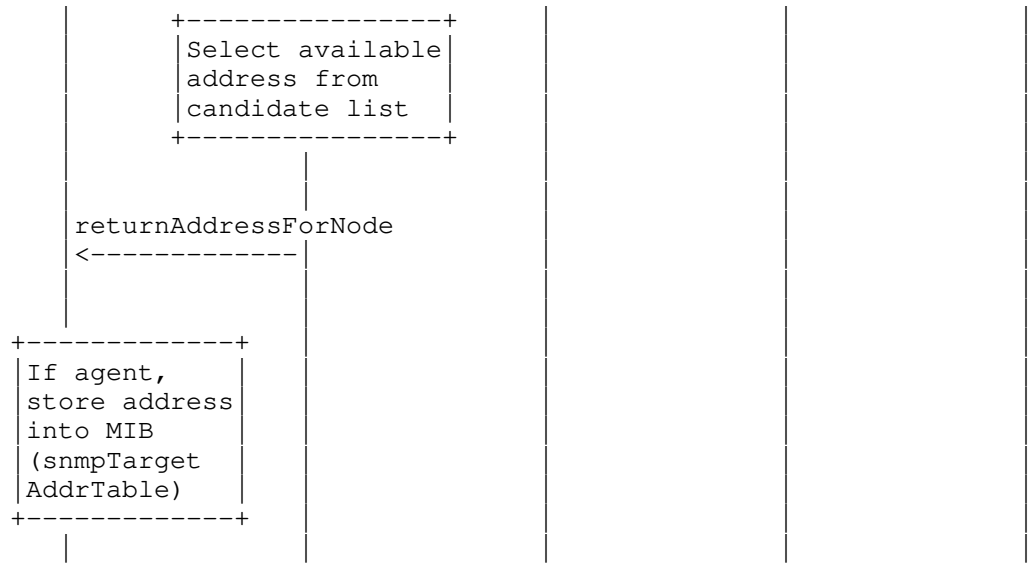


10. An SNMP Entity Forms a Direct Connection with Another SNMP Entity

Note that the targets of the management tasks and reports of RELOAD network are Node-ID of RELOAD or snmeEngineID of SNMP. (Note: In this document, snmeEngineID constructed from Node-ID.) When a SNMP Entity needs to send SNMP messages to another SNMP Entity, it must get the other side of available IP address firstly. Due to the existence of NAT, they need to exchange ICE addresses with each other and check for connectivity, and then selects a pair of available IP

address to establish the connection(of course, if a connection has been established between this pair of IP address, the initiator node will directly send messages to the target node.) The process of establishing a direct connection between SNMP Entities is as shown below:





11. O-Node information Collection

Before a network manager performs management tasks for nodes, it must first collect the Node-ID and the status information of managed nodes. The manager collects the information about RELOAD nodes (including Peer and Client) using the following method: when an agent starts up(or is activated), its associated RELOAD node joins the RELOAD network, and it needs to obtain the name of a network manager by some method. Such as: a) the name of the network manager is set in the configuration file in configuration server, and the managed nodes can obtain the name from the configuration file, b) build the tree struture of the names of the network manager according to ReDiR, and the managed nodes can obtain the name by the method of ReDiR service discovery (note: an entry is added in ReDiR Namespaces Registry, its detail is in the section of IANA Considerations), c) the name of the network manager is set in the LCD in the managed node, and the managed node can obtain the name from its LCD. Then this node connects to the network manager and registers its own information, such as node name, Node-ID, status, etc., to the manager. The procedure for finding the manager and connecting to it has been introduced in the previous section. There are many other ways to collect the information about managed nodes, which could be studied further in future.

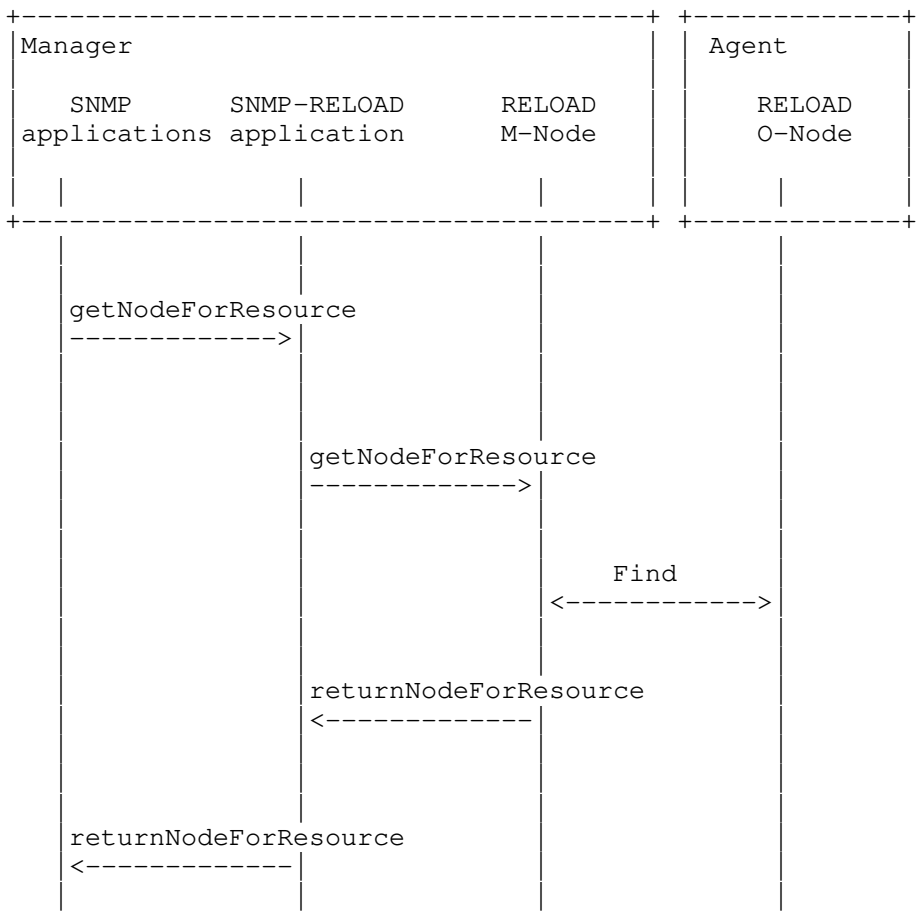
12. O-Node Lookup by the Network Manager for a Resource

When a network manager needs to send a management task for resource, it is necessary that the network manager first gets the Node-ID of the O-Node responsible for the resource in order to determine whether there is a connection with the O-Node. One way for the manager to get the Node-ID of the O-Node responsible for the resource is to acquire the Node-ID of the O-NODE responsible for the target resource through the via_list of Forwarding Header in FindAns. The process is as follows:

First, the network manager sends a FindReq to the RELOAD network with the target Reource-ID put into the destination_list of the FindReq. Then the RELOAD network routes FindReq to the node responsible for the target Resource ID according to its routing algorithm.

Second, the O-Node returns FindAns to the network manager through the RELOAD network. The first Node-ID in the via_list of the Forwarding Header of the FindAns is the Node-ID of the O-Node responsible for the target resource.

The process which Network Manger utilizes when Looking up the O-Node for a Resource is as shown below:



After the network manager gets the Node-ID of O-Node, it can determine whether there is a connection between itself and the O-Node. If the connection exists, the network manager may directly send SNMP message to the O-Node, otherwise it is required to establish a new connection to the O-Node.

13. Definition of SNMP-REGISTRATION Kind

This section defines the SNMP-REGISTRATION kind.

Name: SNMP-REGISTRATION

Kind ID: The Resource Name for the SNMP-REGISTRATION Kind-ID is the Name of the network manager. The data stored is a SnmpRegistrationData, which can contain a destination list and

contact preferences to the peer which is acting for the network manager.

Data Model: The data model for the SNMP-REGISTRATION Kind-ID is single value.

Access Control: USER-NODE-MATCH.

Data stored under the SNMP-REGISTRATION kind is of type SnmpRegistration. A destination list can be used to reach the network manager.

14. Security Considerations

The threats to SNMP Usage for RELOAD are the same with the SNMP, and which are described specifically in RFC5953. We won't repeat it in this document.

There are three solutions can solve the security issues in SNMP Usage for RELOAD. The first option is to use a shared key based solution which is utilized in SNMPv3 security solution (USM). The second option is a PKI based security solution, which is to use the certificate of RELOAD to authenticate and encrypt the SNMP messages. The third option is (D)TLS based security solution, which uses the secure (D)TLS links to transfer the SNMP message.

USM was designed to be independent of other existing security infrastructures. USM therefore uses a separate principal and key management infrastructure. Many operators have reported that deploying another principal and key management infrastructure in order to use SNMPv3 is a deterrent to deploying SNMPv3[RFC5590]. We note that the second option may not be as efficient as expected by the service providers. So we recommend the third option.

The special detail of (D)TLS based security for SNMP is defined in RFC5953, and it won't be described again in this document. In short, we propose to use RELOAD certificate for setting up the connection using (D)TLS based security. When the Mapping of certificate's subjectAltName to a tmSecurityName is used in the SNMP-TLS-TM-MIB's snmpTlstmCertToTSNTable, tmSecurityName is derived from the user name value of the SubjectAltName field in RELOAD certificate.

15. IANA Considerations

In case of multiple managers and single administration domain, the managed Nodes may get the manager's name by the method of ReDiR

service discovery. And an entry added in ReDiR Namespaces Registry is below.

Namespace	RFC
snmp-manager	RFC-AAAA

16. Acknowledgments

This draft is based on "REsource LOcation And Discovery (RELOAD) Base Protocol" draft by C. Jennings, B. Lowekamp, E. Rescorla, S. Baset and H. Schulzrinne.

This draft make a reference to "A SIP Usage for RELOAD" draft by C. Jennings, B. Lowekamp, Ed., E. Rescorla, S. Baset, H. Schulzrinne.

This draft is based on "Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks" RFC by Harrington, D., Presuhn, R., and B. Wijnen.

This draft is based on "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)" RFC by Hardaker, W..

Thanks to David Harrington, Juergen Schoenwaelder, Dan Romascanu, Tom Petch, Marc Petit-Huguenin, and others in P2PSIP and Network WG who offered significant advice on earlier versions of this draft.

Thank the many people of the IETF P2PSIP WG and Network WG whose many drafts and RFCs we have learned.

17. References

17.1. 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", August 2010.

[I-D.ietf-p2psip-service-discovery]
Maenpaa, J. and G. Camarillo, "Service Discovery Usage for REsource LOcation And Discovery (RELOAD)", October 2012.

[I-D.ietf-p2psip-sip]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", July 2010.

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

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC5953] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5953, August 2010.

17.2. 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", July 2008.

[I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC4181] Heard, C., "Guidelines for Authors and Reviewers of MIB Documents", BCP 111, RFC 4181, September 2005.

Appendix A. Additional Stuff

Authors' Addresses

YongLin Peng
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13776637274
Email: peng.yonglin@zte.com.cn

Wei Wang
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13851658076
Email: wang.wei108@zte.com.cn

ZhenWu Hao
ZTE Corporation
Nanjing, 210012
China

Phone: +86 13382087596
Email: hao.zhenwu@zte.com.cn

Yu Meng
ZTE Corporation
Nanjing, 210012
China

Phone: +86 18651806839
Email: meng.yu@zte.com.cn

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2012

M. Petit-Huguenin
Stonyfish, Inc.

July 5, 2011

Configuration of Access Control Policy in REsource LOcation And Discovery (RELOAD) Base Protocol
draft-petithuguenin-p2psip-access-control-03

Abstract

This document describes an extension to the REsource LOcation And Discovery (RELOAD) base protocol to distribute the code of new Access Control Policies without having to upgrade the RELOAD implementations in an overlay.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 6, 2012.

Copyright Notice

Copyright (c) 2011 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. Processing 4
4. Security Considerations 6
5. IANA Considerations 7
6. Acknowledgements 7
7. References 7
7.1. Normative References 7
7.2. Informative References 7
Appendix A. Examples 8
A.1. Standard Access Control Policies 8
A.1.1. USER-MATCH 8
A.1.2. NODE-MATCH 8
A.1.3. USER-NODE-MATCH 8
A.1.4. NODE-MULTIPLE 9
A.2. Service Discovery Access Control Policy NODE-ID-MATCH . . 9
A.3. VIPR Access Control Policy 11
Appendix B. Release notes 11
B.1. Modifications between -03 and -02 11
B.2. Modifications between -02 and -01 12
B.3. Modifications between -01 and -00 12
B.4. Running Code Considerations 12
B.5. TODO List 12
Author's Address 13

1. Introduction

The RELOAD base protocol specifies an Access Control Policy as "defin[ing] whether a request from a given node to operate on a given value should succeed or fail." The paragraph continues saying that "[i]t is anticipated that only a small number of generic access control policies are required", but there is indications that this assumption will not hold. On all the RELOAD Usages defined in other documents than the RELOAD base protocol, roughly 50% defines a new Access Control Policy.

The problem with a new Access Control Policy is that, because it is executed when a Store request is processed, it needs to be implemented by all the peers and so requires an upgrade of the software. This is something that is probably not possible in large overlays or on overlays using different implementations. For this reason, this document proposes an extension to the RELOAD configuration document that permits to transport the code of a new Access Control Policy to each peer.

This extension defines a new element <access-control-code> that can be optionally added to a <configuration> element in the configuration document. The <access-control-code> element contains ECMAScript [ECMA-262] code that will be called for each StoredData object that use this access control policy. The code receives four parameters, corresponding to the Resource-ID, Signature, Kind and StoredDataValue of the value to store. The code returns true or false to signal to the implementation if the request should succeed or fail.

For example the USER-MATCH Access Control Policy defined in the base protocol could be redefined by inserting the following code in an <access-control-code> element:

```
return resource.equalsHash(signer.user_name.bytes());
```

The <kind> parameters are also passed to the code, so the NODE-MULTIPLE Access Control Policy could be implemented like this:

```
for (var i = 0; i < kind.max_node_multiple; i++) {
    if (resource.equalsHash(signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

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 [RFC2119].

3. Processing

A peer receiving a configuration document containing one or more <access-control-code> elements, either by retrieving it from the configuration server or in a ConfigUpdateReq message, MUST reject this configuration if is not is not signed or if the signature verification fails.

The Compact Relax NG Grammar for this element is:

```
namespace acp = "http://implementers.org/access-control"

parameter &= element acp:access-control-code {
  attribute name { xsd:string },
  xsd:base64Binary
}?
```

The "name" attribute defines the access control policy and can then be used in a <kind> element as if it was defined by IANA.

If the <access-control-code> element is present in the namespace allocated to this specification, and the Access Control Policy is not natively implemented, then the code inside the element MUST be called for each DataValue found in a received StoreReq for a Kind that is defined with this access control policy. The content of the <access-control-code> element MUST be decoded using the base64 [RFC4648] encoding, uncompressed using gzip [RFC1952] then converted to characters using UTF-8. <access-control-code> elements that are not encoded using UTF-8, compressed with gzip or finally converted to the base64 format MUST be ignored. For each call to the code, the following ECMAScript objects, properties and functions MUST be available:

configuration.instance_name: The name of the overlay, as a String object.
configuration.topology_plugin: The overlay algorithm, as a String object.

configuration.node_id_length: The length of a NodeId in bytes, as a Number object.

configuration.evaluate(String, String, String): A function that evaluates the first parameter as an XPath expression against the configuration element, and returns the result as a String object. The second parameter must contain a namespace prefix and the third parameter must contain a namespace.

kind.id: The id of the Kind associated with the entry, as a Number object.

kind.name: If the Kind associated with the entry is registered by IANA, contains the name as a String object. If not, this property is undefined.

kind.data_model: The name of the Data Model associated with the entry, as a String object.

kind.access_control: The name of the Access Control Policy associated with the entry, as a String object.

kind.max_count: The value of the max-count element in the configuration file, as a Number object.

kind.max_size: The value of the max-size element in the configuration file as a Number object.

kind.max_node_multiple: If the Access Control is MULTIPLE-NODE, contains the value of the max-node-multiple element in the configuration file, as a Number object. If not, this property is undefined.

kind.evaluate(String, String, String): A function that evaluates the first parameter as an XPath expression against the kind element, and returns the result as a String object. The second parameter must contain a namespace prefix and the third parameter must contain a namespace.

resource: An opaque object representing the Resource-ID, as an array of bytes.

resource.equalsHash(Object...): A function that returns true if hashing the concatenation of the arguments according to the mapping function of the overlay algorithm is equal to the Resource-ID. Each argument is an array of bytes.

signer.user_name: The rfc822Name stored in the certificate that was used to sign the request, as a String object.

signer.node_id: The Node-ID stored in the certificate that was used to sign the request, as an array of bytes.

entry.index: If the Data Model is ARRAY, contains the index of the entry, as a Number object. If not, this property is undefined.

entry.key: If the Data Model is DICTIONARY, contains the key of the entry, as an array of bytes. If not, this property is undefined.

entry.storage_time: The date and time used to store the entry, as a Date object.

entry.lifetime: The validity for the entry in seconds, as a Number object.
entry.exists: Indicates if the entry value exists, as Boolean object.
entry.value: This property contains an opaque object that represents the whole data, as an array of bytes.

The properties SHOULD NOT be modifiable or deletable and if they are, modifying or deleting them MUST NOT modify or delete the equivalent internal values (in other words, the code cannot be used to modify the elements that will be stored).

The value returned by the code is evaluated to true or false, according to the ECMAScript rules. If the return value of one of the call to the code is evaluated to false, then the StoreReq fails, the state MUST be rolled back and an Error_Forbidden MUST be returned.

4. Security Considerations

Because the configuration document containing the ECMAScript code is under the responsability of the same entity that will sign it, using a scripting language does not introduce any additional risk if the RELOAD implementers follow the rules in this document (no side effect when modifying the parameters, only base classes of ECMAScript implemented, etc...). It is even possible to deal with less than perfect implementations as long as they do not accept a configuration file that is not signed correctly. One way for the signer to enforce this would be to deliberately send in a ConfigUpdate an incorrectly signed version of the configuration file and blacklist all the nodes that accepted it in a newly issued configuration file.

By permitting multiple overlay implementations to interoperate inside one overlay, RELOAD helps build overlays that are not only resistant to hardware or communication failures, but also to programmer errors. Distributing the access control policy code inside the configuration document reintroduces this single point of failure. To mitigate this problem, new access control policies should be implemented natively as soon as possible, but if all implementations uses the script as a blueprint for the native code, an hidden bug can be duplicated. This is why developers should implement new access control policies from the normative text instead of using the code. That is anyway probably not legal under most copyright laws but to help developers do the right thing the code in the configuration is obfuscated by compressing and encoding it as a base64 character string.

5. IANA Considerations

If this document is accepted as a standard track document this section will request an URN in the "XML Namespaces" class of the "IETF XML Registry" from IANA. Until this is done, implementions should use the following URN:

http://implementers.org/access-control

6. Acknowledgements

This document was written with the xml2rfc tool described in [RFC2629].

7. References

7.1. Normative References

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[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.

[ECMA-262]
Ecma, "ECMAScript Language Specification 3rd Edition", December 2009.

7.2. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[I-D.ietf-p2psip-service-discovery]
Maenpaa, J. and G. Camarillo, "Service Discovery Usage for REsource LOcation And Discovery (RELOAD)",

draft-ietf-p2psip-service-discovery-03 (work in progress), July 2011.

[I-D.petithuguenin-vipr-reload-usage]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification", draft-petithuguenin-vipr-reload-usage-01 (work in progress), June 2011.

[I-D.knauf-p2psip-share]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A Usage for Shared Resources in RELOAD (ShaRe)", draft-knauf-p2psip-share-00 (work in progress), March 2011.

Appendix A. Examples

A.1. Standard Access Control Policies

This section shows the ECMAScript code that could be used to implement the standard Access Control Policies defined in [I-D.ietf-p2psip-base].

A.1.1. USER-MATCH

```
String.prototype['bytes'] = function() {
    var bytes = [];
    for (var i = 0; i < this.length; i++) {
        bytes[i] = this.charCodeAt(i);
    }
    return bytes;
};

return resource.equalsHash(signer.user_name.bytes());
```

A.1.2. NODE-MATCH

```
return resource.equalsHash(signer.node_id);
```

A.1.3. USER-NODE-MATCH

```
String.prototype['bytes'] = function() {
    var bytes = [];
    for (var i = 0; i < this.length; i++) {
        bytes[i] = this.charCodeAt(i);
    }
    return bytes;
};

var equals = function(a, b) {
    if (a.length !== b.length) return false;
    for (var i = 0; i < a.length; i++) {
        if (a[i] !== b[i]) return false;
    }
    return true;
};

return resource.equalsHash(signer.user_name.bytes())
  && equals(entry.key, signer.node_id);
```

A.1.4. NODE-MULTIPLE

```
Number.prototype['width'] = function(w) {
    var bytes = [];
    for (var i = 0; i < w; i++) {
        bytes[i] = (this >>> ((w - i - 1) * 8)) & 255;
    }
    return bytes;
};

for (var i = 0; i < kind.max_node_multiple; i++) {
    if (resource.equalsHash(signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

[[Note that base-15 still does not state exactly the length of i when concatenated in the hash input]]

A.2. Service Discovery Access Control Policy NODE-ID-MATCH

[I-D.ietf-p2psip-service-discovery] defines a specific Access Control Policy (NODE-ID-MATCH) that need to access the content of the entry to be written. If implemented as specified by this document, the ECMAScript code would look something like this:

```
/* Insert here the code from
```

```
      http://jsfromhell.com/classes/bignumber
    */

   var toBigNumber = function(node_id) {
      var bignum = new BigNumber(0);
      for (var i = 0; i < node_id.length; i++) {
          bignum = bignum.multiply(256).add(node_id[i]);
      }
      return bignum;
   };

   var checkIntervals = function(node_id, level, node, factor) {
      var size = new BigNumber(2).pow(128);
      var node = toBigNumber(node_id);
      for (var f = 0; f < factor; f++) {
          var temp = size.multiply(new BigNumber(f)
            .pow(new BigNumber(level).negate()));
          var min = temp.multiply(node.add(new BigNumber(f)
            .divide(factor)));
          var max = temp.multiply(node.add(new BigNumber(f + 1)
            .divide(factor)));
          if (node.compare(min) === -1 || node.compare(max) == 1
            || node.compare(max) == 0) return false;
      }
      return true;
   };

   var equals = function(a, b) {
      if (a.length !== b.length) return false;
      for (var i = 0; i < a.length; i++) {
          if (a[i] !== b[i]) return false;
      }
      return true;
   };

   var level = function(value) {
      var length = value[16] * 256 + value[17];
      return value[18 + length] * 256 + value[18 + length + 1];
   };

   var node = function(value) {
      var length = value[16] * 256 + value[17];
      return value[18 + length + 2] * 256
        + value[18 + length + 3];
   };

   var namespace = function(value) {
      var length = value[16] * 256 + value[17];
```

```
      return String.fromCharCode.apply(null,
        value.slice(18, length + 18));
  };

  var branching_factor =
    kind.evaluate('/branching-factor',
    'redir', 'urn:ietf:params:xml:ns:p2p:redir');
  return equals(entry.key, signer.node_id)
   && (!entry.exists || checkIntervals(entry.key,
     level(entry.value), node(entry.value),
     branching_factor))
   && (!entry.exists
     || resource.equalsHash(namespace(entry.value),
       level(entry.value), node(entry.value)));
```

Note that the code for the BigNumber object was removed from this example, as the licensing terms are unclear. The code is available at <http://jsfromhell.com/classes/bignumber>.

A.3. VIPR Access Control Policy

[I-D.petithuguenin-vipr-reload-usage] defines a specific Access Control Policy. If implemented as specified by this document, the ECMAScript code would look something like this:

```
var equals = function(a, b) {
  if (a.length !== b.length) return false;
  for (var i = 0; i < a.length; i++) {
    if (a[i] !== b[i]) return false;
  }
  return true;
};
var length = configuration.node_id_length;
return equals(entry.key.slice(0, length),
  entry.value.slice(4, length + 4))
  && equals(entry.key.slice(0, length), signer.node_id);
```

Appendix B. Release notes

This section must be removed before publication as an RFC.

B.1. Modifications between -03 and -02

- Moved the access-control-code element fom the kind element to the configuration element so the code can be shared between kinds. A new "name" attribute is used to name the access control policy.

- Added configuration object to pass information about the whole overlay.
- Added evaluate functions to retrieve extensions parameters.
- Renamed the signature attribute to signer.
- Filled Security section.
- Added temporary namespace to IANA section.
- The content of the access-control-code is now UTF-8 encoded, compressed with gzip and converted back to characters with base64.
- Fixed the implementation of the service discovery access control policy.
- Added code for VIPR policy.

B.2. Modifications between -02 and -01

- Made clear that an unsigned kind with this extension must be rejected.
- Removed the kind.params array, and converted the max-count, max-size and max-node-multiple as Number objects. Fixed the examples.
- Removed the parsing of extensions in the kind element. The former system did not work with namespaces or attributes, and the right solution (xpath) is probably too complex. The value of the parameters can still be manually mirrored in the script, so there is perhaps no need for the added complexity. Also fixed the examples.
- Reference draft-p2psip-share instance of draft-p2psip-disco.
- Added a "Running Code Considerations" section that contain the reference to the reference implementation and script tester.
- Nits

B.3. Modifications between -01 and -00

- Changed reference from JavaScript to ECMAScript.
- Changed signature from equals() to equalsHash().
- Fixed the examples following implementation.
- Replaced automatic decoding of value by ECMAScript code.
- Added the type of each property.
- Specified that the code cannot be used to modify the value stored.

B.4. Running Code Considerations

- Reference Implementation and Access Control Policy script tester (<http://debian.implementers.org/testing/source/reload.tar.gz>). Marc Petit-Huguenin. Implements version -03.

B.5. TODO List

- The access control policy in ShaRe [I-D.knauf-p2psip-share] requires an access to the list of all values stored at the same Resource-ID than the value currently verified. A major update of this specification is needed for this, so more time is needed to do it right.

Author's Address

Marc Petit-Huguenin
Stonyfish, Inc.

Email: petithug@acm.org

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2013

M. Petit-Huguenin
Impedance Mismatch

October 22, 2012

Configuration of Access Control Policy in REsource LOcation And Discovery (RELOAD) Base Protocol
draft-petithuguenin-p2psip-access-control-05

Abstract

This document describes an extension to the REsource LOcation And Discovery (RELOAD) base protocol to distribute the code of new Access Control Policies without having to upgrade the RELOAD implementations in an overlay.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on April 25, 2013.

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. Processing 4
4. Security Considerations 6
5. IANA Considerations 7
6. References 7
6.1. Normative References 7
6.2. Informative References 7
Appendix A. Examples 8
A.1. Standard Access Control Policies 8
A.1.1. USER-MATCH 8
A.1.2. NODE-MATCH 8
A.1.3. USER-NODE-MATCH 8
A.1.4. NODE-MULTIPLE 9
A.2. Service Discovery Access Control Policy NODE-ID-MATCH . . 9
A.3. VIPR Access Control Policy 11
A.4. ShaRe Access Control Policy USER-CHAIN-ACL 11
Appendix B. Release notes 12
B.1. Modifications between -05 and -04 12
B.2. Running Code Considerations 12
B.3. TODO List 12
Author's Address 12

1. Introduction

The RELOAD base protocol specifies an Access Control Policy as "defin[ing] whether a request from a given node to operate on a given value should succeed or fail." The paragraph continues saying that "[i]t is anticipated that only a small number of generic access control policies are required", but there is indications that this assumption will not hold. On all the RELOAD Usages defined in other documents than the RELOAD base protocol, roughly 50% defines a new Access Control Policy.

The problem with a new Access Control Policy is that, because it is executed when a Store request is processed, it needs to be implemented by all the peers and so requires an upgrade of the software. This is something that is probably not possible in large overlays or on overlays using different implementations. For this reason, this document proposes an extension to the RELOAD configuration document that permits to transport the code of a new Access Control Policy to each peer.

This extension defines a new element <access-control-code> that can be optionally added to a <configuration> element in the configuration document. The <access-control-code> element contains ECMAScript [ECMA-262] code that will be called for each StoredData object that use this access control policy. The code receives four parameters, corresponding to the Resource-ID, Signature, Kind and StoredDataValue of the value to store. The code returns true or false to signal to the implementation if the request should succeed or fail.

For example the USER-MATCH Access Control Policy defined in the base protocol could be redefined by inserting the following code in an <access-control-code> element:

```
return resource.equalsHash(signer.user_name.bytes());
```

The <kind> parameters are also passed to the code, so the NODE-MULTIPLE Access Control Policy could be implemented like this:

```
for (var i = 0; i < kind.max_node_multiple; i++) {
    if (resource.equalsHash(signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

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 [RFC2119] .

"SHOULD", "SHOULD NOT", "RECOMMENDED", and "NOT RECOMMENDED" are appropriate when valid exceptions to a general requirement are known to exist or appear to exist, and it is infeasible or impractical to enumerate all of them. However, they should not be interpreted as permitting implementors to fail to implement the general requirement when such failure would result in interoperability failure.

3. Processing

A peer receiving a configuration document containing one or more <access-control-code> elements, either by retrieving it from the configuration server or in a ConfigUpdateReq message, MUST reject this configuration if is not is not signed or if the signature verification fails.

The Compact Relax NG Grammar for this element is:

```
namespace acp = "http://implementers.org/access-control"

parameter &= element acp:access-control-code {
  attribute name { xsd:string },
  xsd:base64Binary
}?
```

The "name" attribute defines the access control policy and can then be used in a <kind> element as if it was defined by IANA.

If the <access-control-code> element is present in the namespace allocated to this specification, and the Access Control Policy is not natively implemented, then the code inside the element MUST be called for each DataValue found in a received StoreReq for a Kind that is defined with this access control policy. The content of the <access-control-code> element MUST be decoded using the base64 [RFC4648] encoding, uncompressed using gzip [RFC1952] then converted to characters using UTF-8. <access-control-code> elements that are not encoded using UTF-8, compressed with gzip or finally converted to the base64 format MUST be ignored. For each call to the code, the following ECMAScript objects, properties and functions MUST be available:

configuration.instance_name: The name of the overlay, as a String object.
configuration.topology_plugin: The overlay algorithm, as a String object.
configuration.node_id_length: The length of a NodeId in bytes, as a Number object.
configuration.kinds: An array of kinds (with the same definition than the kind object), indexed by id and eventually by name.
configuration.evaluate(String, String, String): A function that evaluates the first parameter as an XPath expression against the configuration element, and returns the result as a String object. The second parameter must contain a namespace prefix and the third parameter must contain a namespace.
kind.id: The id of the Kind associated with the entry, as a Number object.
kind.name: If the Kind associated with the entry is registered by IANA, contains the name as a String object. If not, this property is undefined.
kind.data_model: The name of the Data Model associated with the entry, as a String object.
kind.access_control: The name of the Access Control Policy associated with the entry, as a String object.
kind.max_count: The value of the max-count element in the configuration file, as a Number object.
kind.max_size: The value of the max-size element in the configuration file as a Number object.
kind.max_node_multiple: If the Access Control is MULTIPLE-NODE, contains the value of the max-node-multiple element in the configuration file, as a Number object. If not, this property is undefined.
kind.evaluate(String, String, String): A function that evaluates the first parameter as an XPath expression against the kind element, and returns the result as a String object. The second parameter must contain a namespace prefix and the third parameter must contain a namespace.
resource: An opaque object representing the Resource-ID, as an array of bytes.
resource.entries: An array of arrays of entry objects, with the first array level indexed by Kind-Id and kind names, and the second level indexed by index, key or nothing, depending on the data model of the kind. This permits to retrieve all the values of all Kinds stored at the same Resource-ID than the entry currently processed.
resource.equalsHash(Object...): A function that returns true if hashing the concatenation of the arguments according to the mapping function of the overlay algorithm is equal to the Resource-ID. Each argument is an array of bytes.

entry.index: If the Data Model is ARRAY, contains the index of the entry, as a Number object. If not, this property is undefined.
entry.key: If the Data Model is DICTIONARY, contains the key of the entry, as an array of bytes. If not, this property is undefined.
entry.storage_time: The date and time used to store the entry, as a Date object.
entry.lifetime: The validity for the entry in seconds, as a Number object.
entry.exists: Indicates if the entry value exists, as Boolean object.
entry.value: This property contains an opaque object that represents the whole data, as an array of bytes.
entry.signer.user_name: The rfc822Name stored in the certificate that was used to sign the request, as a String object.
entry.signer.node_id: The Node-ID stored in the certificate that was used to sign the request, as an array of bytes.

The properties SHOULD NOT be modifiable or deletable and if they are, modifying or deleting them MUST NOT modify or delete the equivalent internal values (in other words, the code cannot be used to modify the elements that will be stored).

The value returned by the code is evaluated to true or false, according to the ECMAScript rules. If the return value of one of the call to the code is evaluated to false, then the StoreReq fails, the state MUST be rolled back and an Error_Forbidden MUST be returned.

4. Security Considerations

Because the configuration document containing the ECMAScript code is under the responsability of the same entity that will sign it, using a scripting language does not introduce any additional risk if the RELOAD implementers follow the rules in this document (no side effect when modifying the parameters, only base classes of ECMAScript implemented, etc...). It is even possible to deal with less than perfect implementations as long as they do not accept a configuration file that is not signed correctly. One way for the signer to enforce this would be to deliberately send in a ConfigUpdate an incorrectly signed version of the configuration file and blacklist all the nodes that accepted it in a newly issued configuration file.

By permitting multiple overlay implementations to interoperate inside one overlay, RELOAD helps build overlays that are not only resistant to hardware or communication failures, but also to programmer errors. Distributing the access control policy code inside the configuration document reintroduces this single point of failure. To mitigate this problem, new access control policies should be implemented natively

as soon as possible, but if all implementations uses the script as a blueprint for the native code, an hidden bug can be duplicated. This is why developers should implement new access control policies from the normative text instead of using the code. That is anyway probably not legal under most copyright laws so to help developers do the right thing the code in the configuration is obfuscated by compressing and encoding it as a base64 character string.

5. IANA Considerations

If this document is accepted as a standard track document this section will request an URN in the "XML Namespaces" class of the "IETF XML Registry" from IANA. Until this is done, implementions should use the following URN:

http://implementers.org/access-control

6. References

6.1. Normative References

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[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-22 (work in progress), July 2012.

[ECMA-262] Ecma, "ECMAScript Language Specification 3rd Edition", December 2009.

6.2. Informative References

[I-D.ietf-p2psip-service-discovery] Maenpaa, J. and G. Camarillo, "Service Discovery Usage for REsource LOcation And Discovery (RELOAD)", draft-ietf-p2psip-service-discovery-06 (work in progress),

October 2012.

[I-D.petithuguenin-vipr-reload-usage]
Petit-Huguenin, M., Rosenberg, J., and C. Jennings, "A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification", draft-petithuguenin-vipr-reload-usage-04 (work in progress), March 2012.

[I-D.ietf-p2psip-share]
Knauf, A., Schmidt, T., Hege, G., and M. Waehlisch, "A Usage for Shared Resources in RELOAD (ShaRe)", draft-ietf-p2psip-share-00 (work in progress), October 2012.

Appendix A. Examples

A.1. Standard Access Control Policies

This section shows the ECMAScript code that could be used to implement the standard Access Control Policies defined in [I-D.ietf-p2psip-base] .

A.1.1. USER-MATCH

```
String.prototype['bytes'] = function () {
    var bytes = [];
    for (var i = 0; i < this.length; i++) {
        bytes[i] = this.charCodeAt(i);
    }
    return bytes;
};

return resource.equalsHash(entry.signer.user_name.bytes());
```

A.1.2. NODE-MATCH

```
return resource.equalsHash(entry.signer.node_id);
```

A.1.3. USER-NODE-MATCH

```
String.prototype['bytes'] = function () {
    var bytes = [];
    for (var i = 0; i < this.length; i++) {
        bytes[i] = this.charCodeAt(i);
    }
    return bytes;
};

var equals = function (a, b) {
    if (a.length !== b.length) return false;
    for (var i = 0; i < a.length; i++) {
        if (a[i] !== b[i]) return false;
    }
    return true;
};

return resource.equalsHash(entry.signer.user_name.bytes())
  && equals(entry.key, entry.signer.node_id);
```

A.1.4. NODE-MULTIPLE

```
Number.prototype['width'] = function (w) {
    var bytes = [];
    for (var i = 0; i < w; i++) {
        bytes[i] = (this >>> ((w - i - 1) * 8)) & 255;
    }
    return bytes;
};

for (var i = 0; i < kind.max_node_multiple; i++) {
    if (resource.equalsHash(entry.signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

A.2. Service Discovery Access Control Policy NODE-ID-MATCH

[I-D.ietf-p2psip-service-discovery] defines a specific Access Control Policy (NODE-ID-MATCH) that need to access the content of the entry to be written. If implemented as specified by this document, the ECMAScript code would look something like this:

```
/* Insert here the code from
   http://jsfromhell.com/classes/bignumber
 */

var toBigNumber = function (node_id) {
```

```
      var bignum = new BigNumber(0);
      for (var i = 0; i < node_id.length; i++) {
          bignum = bignum.multiply(256).add(node_id[i]);
      }
      return bignum;
   };

   var checkIntervals = function (node_id, level, node, factor) {
      var size = new BigNumber(2).pow(128);
      var node = toBigNumber(node_id);
      for (var f = 0; f < factor; f++) {
          var temp = size.multiply(new BigNumber(f)
            .pow(new BigNumber(level).negate()));
          var min = temp.multiply(node.add(new BigNumber(f)
            .divide(factor)));
          var max = temp.multiply(node.add(new BigNumber(f + 1)
            .divide(factor)));
          if (node.compare(min) === -1 || node.compare(max) == 1
            || node.compare(max) == 0) return false;
      }
      return true;
   };

   var equals = function (a, b) {
      if (a.length !== b.length) return false;
      for (var i = 0; i < a.length; i++) {
          if (a[i] !== b[i]) return false;
      }
      return true;
   };

   var level = function (value) {
      var length = value[16] * 256 + value[17];
      return value[18 + length] * 256 + value[18 + length + 1];
   };

   var node = function (value) {
      var length = value[16] * 256 + value[17];
      return value[18 + length + 2] * 256
        + value[18 + length + 3];
   };

   var namespace = function (value) {
      var length = value[16] * 256 + value[17];
      return String.fromCharCode.apply(null,
        value.slice(18, length + 18));
   };
```

```
var branching_factor =
  kind.evaluate('/branching-factor',
  'redir', 'urn:ietf:params:xml:ns:p2p:redir');
return equals(entry.key, entry.signer.node_id)
 && (!entry.exists || checkIntervals(entry.key,
   level(entry.value), node(entry.value),
   branching_factor))
 && (!entry.exists
   || resource.equalsHash(namespace(entry.value),
     level(entry.value), node(entry.value)));
```

Note that the code for the BigNumber object was removed from this example, as the licensing terms are unclear. The code is available at <http://jsfromhell.com/classes/bignumber> .

A.3. VIPR Access Control Policy

[I-D.petithuguenin-vipr-reload-usage] defines a specific Access Control Policy. If implemented as specified by this document, the ECMAScript code would look something like this:

```
var equals = function (a, b) {
  if (a.length !== b.length) return false;
  for (var i = 0; i < a.length; i++) {
    if (a[i] !== b[i]) return false;
  }
  return true;
};
var length = configuration.node_id_length;
return equals(entry.key.slice(0, length),
  entry.value.slice(4, length + 4))
  && equals(entry.key.slice(0, length), entry.signer.node_id);
```

A.4. ShaRe Access Control Policy USER-CHAIN-ACL

[I-D.ietf-p2psip-share] defines a new Access Control Policies, USER-CHAIN-ACL. If implemented as specified by this document, the ECMAScript code would look something like this:

```
   var pattern = kind.evaluate('/share:pattern',
     'share', 'urn:ietf:params:xml:ns:p2p:config-share');
   var username = entry.signer.user_name.match(/^([^@]+)@(.+)$/);
   var new_pattern = new RegExp(
     pattern.replace('$USER', username[1])
       .replace('$DOMAIN', username[2]));
   var length =  entry.value[0] * 256 + entry.value[1];
   var resource_name = String.fromCharCode.apply(null,
     entry.value.slice(2, length + 2));
   return new_pattern.test(resource_name);\n"));
```

[[Note: the code is incomplete]]

Appendix B. Release notes

This section must be removed before publication as an RFC.

B.1. Modifications between -05 and -04

- Resurrected the draft.

B.2. Running Code Considerations

- Reference Implementation and Access Control Policy script tester (<http://debian.implementers.org/testing/source/reload.tar.gz>). Marc Petit-Huguenin. Implements version -03.

B.3. TODO List

- Finish the code for ShaRe.

Author's Address

Marc Petit-Huguenin
Impedance Mismatch

Email: petithug@acm.org

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2012

J. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish
July 4, 2011

Proportional Quota in REsource LOcation And Discovery (RELOAD) draft-petithuguenin-p2psip-proportional-quota-01

Abstract

This document defines an extension to RELOAD [I-D.ietf-p2psip-base] that limits the number of a specific kind element that can be stored by one RELOAD peer.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 5, 2012.

Copyright Notice

Copyright (c) 2011 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 3
3. Quota Algorithm 3
4. Overlay Configuration Document Extension 4
5. Security Considerations 5
6. IANA Considerations 5
7. Acknowledgements 5
8. References 5
8.1. Normative References 5
8.2. Informative References 5
Appendix A. Release notes 5
A.1. Modifications between draft-petithuguenin-p2psip-proportional-quota-01 and draft-petithuguenin-p2psip-proportional-quota-00 5
A.2. Modifications between draft-petithuguenin-p2psip-proportional-quota-00 and draft-petithuguenin-vipr-reload-usage-00 6
A.3. Running Code Considerations 6
Authors' Addresses 6

1. Introduction

The base specification of RELOAD defines two variables to set the storage quota. The first one is the maximum size of an element of a specific kind, the second one is the maximum number of elements of a specific kind that can be stored on a specific peer. The combination of the two variables permits to limit the quantity of data that a peer have to store.

For kinds that are used with an access control policy that does not restrict the Resource-IDs, a different algorithm is needed to force storing nodes to behave. This document describes a quota algorithm that limits the number of elements of a specific kind that a node can store in the overlay, independently of the Resource-ID used. Another way to look at this quota algorithm is that an entity must provide a number of peers that is proportional to the number of elements of a specific kind that this entity wants to store.

It is important to understand that this quota works only for an overlay comprising only peers, so with a configuration file containing a <clients-permitted> element set to false.

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 [RFC2119].

3. Quota Algorithm

A peer responsible for storing kinds using the quota algorithm described in this document MUST maintain a count of the number of unique entries being stored per signer for each kind. This operation does not require to add a field containing the Node-ID as the Node-ID of the signer is always available in the signature field of each element stored.

For example if a peer is storing 5 Resource-IDs and at each of those 5 there are two keys whose first 16 bytes correspond to a particular Node-ID, it means this node is currently storing 10 unique dictionary entries for that Node-ID.

When performing the quota checks for an entry, the peer starts by verifying that the size of the entry is consistent. It then takes the <max-count> configuration parameter for this overlay, which measures the amount of entries of a specific kind a particular node

can store when a <max-count-per>SIGNER</max-count-per> configuration parameter is also present. That value is multiplied by by the number of replicas used by the topology plugin (i.e. 3 for Chord) and then divided by the fraction of the overlay owned by this peer. If the result is less than one, it is rounded up to two. This is the maximum number of unique entries that can be stored for this signer. If the peer is not yet storing this many entries for that Node-ID, the store is allowed.

Note that when evaluating a Store Request containing multiple entries per kind, the count of unique entries used for the evaluation is incremented after each successful check, but the count will be reset to its initial value if one of the check fails.

The algorithm takes in account only the duplications made by the topology plugin. If another level of duplication is done at the application level, the <max-count> value must be adjusted accordingly.

Note that because of imperfect distribution of Resource-IDs across the ring, new entries can be rejected even if the total count is under the limit. It is the responsibility of the storing entity to calculate the maximum acceptable probability of rejection and to increase the number of peers accordingly.

4. Overlay Configuration Document Extension

This document extends the overlay configuration document by defining a new element in the "urn:ietf:params:xml:ns:p2p:quota" namespace.

The <max-count-per> element changes the meaning of the <max-count> variable. The value "PEER" forces the <max-count> to be interpreted as been per storing peer, which is the default quota algorithm when this extension is not used. The value "SIGNER" forces the <max-count> to be interpreted as been per signer, which is the algorithm defined by this document.

The Compact Relax NG Grammar for this element is:

```
namespace pqt = "urn:ietf:params:xml:ns:p2p:quota"

kind-parameter &= element pqt:max-count-per { max-count-per-type }
max-count-per-type |= "PEER"
max-count-per-type |= "SIGNER"
max-count-per-type |= xsd:string # extensions
```

5. Security Considerations

TBD

6. IANA Considerations

This document adds the following URN to the "XML Namespaces" class of the "IETF XML Registry":

urn:ietf:params:xml:ns:p2p:quota

7. Acknowledgements

This document was written with the xml2rfc tool described in [RFC2629].

8. References

8.1. Normative References

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

8.2. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between draft-petithuguenin-p2psip-proportional-quota-01 and draft-petithuguenin-p2psip-proportional-quota-00

- Changed "storing peer" to "signer" as this works also for replica peers.
- The default quota algorithm is per storing peer, not per Resource-ID.
- Changed the constants in the XML extension accordingly.
- Added running code considerations for reference implementation.

A.2. Modifications between draft-petithuguenin-p2psip-proportional-quota-00 and draft-petithuguenin-vipr-reload-usage-00

- Instead of having a StorageQuota parameter that gives the maximum number of entries, reused the max-count parameter (that is mandatory anyway) and changes its meaning.
- Removed the 3x multiplier to account for the application layer COPY, as it is application specific.
- Removed the additional 3x multiplier to compensate for imperfect distribution, and moved the responsibility to the storing nodes.

A.3. Running Code Considerations

- Reference Implementation (<http://debian.implementers.org/testing/source/reload.tar.gz>). Marc Petit-Huguenin. Implements version -01.

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: December 23, 2011

N. Zong
X. Jiang
R. Even
Huawei Technologies
Y. Zhang
China Mobile

June 21, 2011

An extension to RELOAD to support Direct Response Routing
draft-zong-p2psip-drr-00

Abstract

This document proposes an optional extension to RELOAD to support direct response routing mode. RELOAD recommends symmetric recursive routing for routing messages. The new optional extension provides a shorter route for responses reducing the overhead on intermediary peers and describes the potential cases where this extension can be used.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 23, 2011.

Copyright Notice

Copyright (c) 2011 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
1.1. Backgrounds 4
2. Terminology 4
3. Problem Statement 5
3.1. Overview 5
3.1.1. Symmetric Recursive Routing (SRR) 5
3.1.2. Direct Response Routing (DRR) 6
3.2. Scenarios Where DRR can be Used 7
3.2.1. Managed or Closed P2P System 7
3.2.2. Wireless Scenarios 7
4. Relationship Between SRR and DRR 8
4.1. How DRR Works 8
4.2. How SRR and DRR Work Together 8
5. Comparison on cost of SRR and DRR 8
5.1. Closed or managed networks 9
5.2. Open networks 10
6. Extensions to RELOAD 10
6.1. Basic Requirements 10
6.2. Modification To RELOAD Message Structure 11
6.2.1. State-keeping Flag 11
6.2.2. Extensive Routing Mode 11
6.3. Creating a Request 12
6.3.1. Creating a Request for DRR 12
6.4. Request And Response Processing 12
6.4.1. Destination Peer: Receiving a Request And Sending a Response 13
6.4.2. Sending Peer: Receiving a Response 13
7. Optional Methods to Investigate Node Connectivity 13
7.1. Getting Addresses To Be Used As Candidates for DRR 14
7.2. Public Reacheability Test 15
8. Security Considerations 16
9. IANA Considerations 16
9.1. A new RELOAD Forwarding Option 16
10. Acknowledgements 16
11. References 16
11.1. Normative References 16
11.2. Informative References 17
Authors' Addresses 17

1. Introduction

1.1. Backgrounds

RELOAD [I-D.ietf-p2psip-base] recommends symmetric recursive routing (SRR) for routing messages and describes the extensions that would be required to support additional routing algorithms. Other than SRR, two other routing options: direct response routing (DRR) and relay peer routing (RPR) are also discussed in Appendix D in [I-D.ietf-p2psip-base]. As we show in section 3, DRR is advantageous over SRR in some scenarios by reducing load (CPU and link BW) on intermediary peers . For example, in a closed network where every node is in the same address realm, DRR performs better than SRR. In other scenarios, using a combination of DRR and SRR together is more likely to bring benefits than if SRR is used alone. Some discussion on connectivity is in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf].

Note that in this draft, we focus on DRR routing mode and its extensions to RELOAD to produce a standalone solution. Please refer to RPR draft [I-D.zong-p2psip-rpr] for RPR routing mode.

We first discuss the problem statement in Section 3, then how to combine DRR and SRR is presented in Section 4. In Section 5, we give comparison on the cost of SRR and DRR in both managed and open networks. An extension to RELOAD to support DRR is proposed in Section 6. Some optional methods to check node connectivity is introduced in Section 7, as informational text.

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 Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] draft extensively in this document. We also use terms defined in NAT behavior discovery [I-D.ietf-behave-nat-behavior-discovery]. Other terms used in this document are defined inline when used and are also defined below for reference.

There are two types of roles in the RELOAD architecture: peer and client. Node is used when describing both peer and client. In discussions specific to behavior of a peer or client, the term peer or client is used instead.

Publicly Reachable: A node is publicly reachable if it can receive unsolicited messages from any other node in the same overlay. Note: "publicly" does not mean that the nodes must be on the public Internet, because the RELOAD protocol may be used in a closed system.

Direct Response Routing (DRR): refers to a routing mode in which responses to P2PSIP requests are returned to the sending peer directly from the destination peer based on the sending peer's own local transport address(es). For simplicity, the abbreviation DRR is used instead in the following text.

Symmetric Recursive Routing(SRR): refers to a routing mode in which responses follow the request path in the reverse order to get back to the sending peer. For simplicity, the abbreviation SRR is used instead in the following text.

3. Problem Statement

RELOAD is expected to work under a great number of application scenarios. The situations where RELOAD is to be deployed differ greatly. For instance, some deployments are global, such as a Skype-like system intended to provide public service. Some run in closed networks of small scale. SRR works in any situation, but DRR may work better in some specific scenarios.

3.1. Overview

RELOAD is a simple request-response protocol. After sending a request, a node waits for a response from a destination node. There are several ways for the destination node to send a response back to the source node. In this section, we will provide detailed information on two routing modes: SRR and DRR.

Some assumptions are made in the following illustrations.

1) Peer A sends a request destined to a peer who is the responsible peer for Resource-ID k;

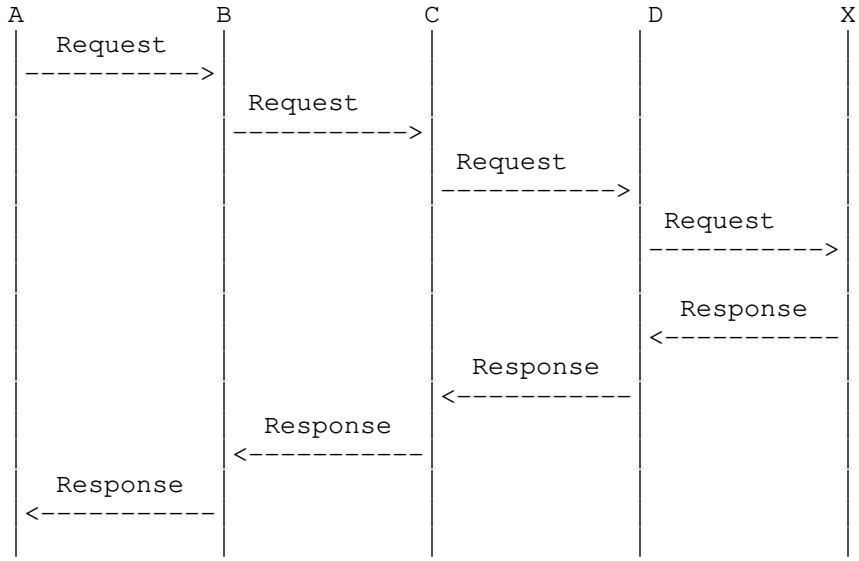
2) Peer X is the root peer being responsible for resource k;

3) The intermediate peers for the path from A to X are peer B, C, D.

3.1.1. Symmetric Recursive Routing (SRR)

For SRR, when the request sent by peer A is received by an intermediate peer B, C or D, each intermediate peer will insert information on the peer from whom they got the request in the via-

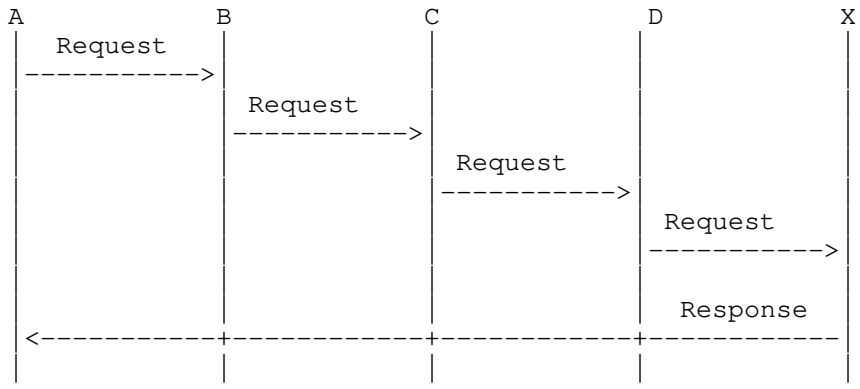
list as described in RELOAD. As a result, the destination peer X will know the exact path which the request has traversed. Peer X will then send back the response in the reverse path by constructing a destination list based on the via-list in the request.



SRR works in any situation, especially when there are NATs or firewalls. A downside of this solution is that the message takes several hops to return to the client, increasing the bandwidth usage and CPU/battery load of multiple nodes.

3.1.2. Direct Response Routing (DRR)

In DRR, peer X receives the request sent by peer A through intermediate peer B, C and D, as in SRR. However, peer X sends the response back directly to peer A based on peer A's local transport address. In this case, the response won't be routed through intermediate peers. Shorter route means less overhead on intermediary peers, especially in the case of wireless network where the CPU and uplink BW is limited. In the absence of NATs or other connectivity issues, this is the optimal routing technique. Note that establishing a secure connection requires multiple round trips. Please refer to Section 5 for cost comparison between SRR and DRR.



3.2. Scenarios Where DRR can be Used

This section lists several scenarios where using DRR would work, and when the increased efficiency would be advantageous.

3.2.1. Managed or Closed P2P System

The properties that make P2P technology attractive, such as the lack of need for centralized severs, self-organization, etc. are attractive for managed systems as well as unmanaged systems. Many of these systems are deployed on private network where nodes are in the same address realm and/or can directly route to each other. In such a scenario, the network administrator can indicate preference for DRR in the peer's configuration file. Peers in such a system would always try DRR first, but peers must also support SRR in case DRR fails. If during the process of establishing a direct connection with the sending peer, the responding peer receives a response with SRR as the preferred routing mode (or it fails to establish the direct connection), the responding peer should not use DRR but switch to SRR. A node can keep a list of unreachable nodes based on trying DRR and use only SRR for these nodes. The advantage in using DRR is on the network stability since it puts less overhead on the intermediary peers that will not route the responses. The intermediary peers will need to route less messages and save CPU resources as well as the link bandwidth usage.

3.2.2. Wireless Scenarios

While some mobile deployments may use clients, in mobile networks with full peers, there is an advantage to using DRR in order to reduce the load on intermediary nodes. Using DRR helps with reducing radio battery usage and bandwidth by the intermediary peers. The service provider may recommend in the configuration using DRR based on his knowledge of the topology.

4. Relationship Between SRR and DRR

4.1. How DRR Works

DRR is very simple. The only requirement is for the source peers to provide their (publically reachable) transport address to the destination peers, so that the destination peer knows where to send the response. Responses are sent directly to the requesting peer.

4.2. How SRR and DRR Work Together

DRR is not intended to replace SRR. As seen from Section 3, DRR has better performance in some scenarios, but have limitations as well, see for example section 4.3 in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf]. As a result, it is better to use these two modes together to adapt to each peer's specific situation. In this section, we give some informative suggestions on how to transition between the routing modes in RELOAD.

A peer can collect statistical data on the success of the different routing modes based on previous transactions and keep a list of non-reachable addresses. Based on the data, the peer will have a clearer view about the success rate of different routing modes. Other than the success rate, the peer can also get data of fine granularity, for example, the number of retransmission the peer needs to achieve a desirable success rate.

A typical strategy for the node is as follows. A node chooses to start with DRR. Based on the success rate as seen from the lost message statistics or responses that used DRR, the node can either continue to offer DRR first or switch to SRR.

The node can decide whether to try DRR based on other information such as configuration file information. If an overlay runs within a private network and all nodes in the system can reach each other directly, nodes may send most of the transactions with DRR.

5. Comparison on cost of SRR and DRR

The major advantages in using DRR are in going through less intermediary peers on the response. By doing that it reduces the load on those peers' resources like processing and communication bandwidth.

5.1. Closed or managed networks

As described in Section 3, many P2P systems run in a closed or managed environment (e.g. carrier networks) so that network administrators would know that they could safely use DRR.

SRR brings out more routing hops than DRR. Assuming that there are N nodes in the P2P system and Chord is applied for routing, the number of hops for a response in SRR and DRR are listed in the following table. Establishing a secure connection between sending peer and responding peer with (D)TLS requires multiple messages. Note that establishing (D)TLS secure connections for P2P overlay is not optimal in some cases, e.g. direct response routing where (D)TLS is heavy for temporary connections. Instead, some alternate security techniques, e.g. using public keys of the destination to encrypt the messages, signing timestamps to prevent reply attacks can be adopted. Therefore, in the following table, we show the cases of: 1) no (D)TLS in DRR; 2) still using DTLS in DRR as sub-optimal and, as the worst-cost case, 7 messages are used during the DTLS handshaking [DTLS]. (TLS Handshake is two round-trip negotiation protocol while DTLS handshake is three round-trip negotiation protocol.)

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
DRR	Yes	1	1
DRR(DTLS)	Yes	1	7+1

From the above comparison, it is clear that:

1) In most cases of N > 2 (2^1), DRR has fewer hops than SRR. Shorter route means less overhead and resource usage on intermediary peers, which is an important consideration for adopting DRR in the cases where the resource such as CPU and BW is limited, e.g. the case of mobile, wireless network.

2) In the cases of N > 256 (2^8), DRR also has fewer messages than SRR.

3) In the cases where N < 256, DRR has more messages than SRR (but still has fewer hops than SRR). So the consideration to use DRR or SRR depends on other factors like using less resources (bandwidth and processing) from the intermediaries peers. Section 4 provides use cases where DRR has better chance to work or where the intermediary resources considerations are important.

5.2. Open networks

In open network where DRR is not guaranteed, DRR can fall back to SRR If it fails after trial, as described in Section 4. Based on the same settings in Section 5.1, the number of hops, number of messages for a response in SRR and DRR are listed in the following table.

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
DRR	Yes	1	1
	Fail&Fall back to SRR	1+logN	1+logN
DRR(DTLS)	Yes	1	7+1
	Fail&Fall back to SRR	1+logN	8+logN

From the above comparison, it can be observed that:

1) Trying DRR would still have a good chance of fewer hops than SRR. Suppose that P peers are publicly reachable, the number of hops in DRR and SRR is P*1+(N-P)*(1+logN), N*logN, respectively. The condition for fewer hops in DRR is P*1+(N-P)*(1+logN) < N*logN, which is P/N > 1/logN. This means that when the number of peers N grows, the required ratio of publicly reachable peers P/N for fewer hops in DRR decreases. Therefore, the chance of trying DRR with fewer hops than SRR becomes better as the scale of the network increases.

2) In the cases of large network and the success rate of DRR is good, it is still possible that DRR has fewer messages than SRR. Otherwise, the consideration to use DRR or SRR depends on other factors like using less resources from the intermediaries peers.

6. Extensions to RELOAD

Adding support for DRR requires extensions to the current RELOAD protocol. In this section, we define the changes required to the protocol, including changes to message structure and to message processing.

6.1. Basic Requirements

All peers implementing DRR MUST support SRR.

All peers MUST be able to process requests for routing in SRR, and MAY support DRR routing requests.

6.2. Modification To RELOAD Message Structure

RELOAD provides an extensible framework to accommodate future extensions. In this section, we define a ForwardingOption structure to support DRR mode. Additionally we present a state- keeping flag to inform intermediate peers if they are allowed to not maintain state for a transaction.

6.2.1. State-keeping Flag

RELOAD allows intermediate peers to maintain state in order to implement SRR, for example for implementing hop-by-hop retransmission. If DRR is used, the response will not follow the reverse path, and the state in the intermediate peers won't be cleared until such state expires. In order to address this issue, we propose a new flag, state-keeping flag, in the message header to indicate whether the state keeping is not required in the intermediate peers.

flag : 0x3 IGNORE-STATE-KEEPING

If IGNORE-STATE-KEEPING is set, any peer receiving this message and which is not the destination of the message SHOULD forward the message with the full VIA list and SHOULD not maintain any internal state.

6.2.2. Extensive Routing Mode

This draft introduces a new forwarding option for an extensive routing mode. This option conforms to the description in section 5.3.2.3 in [I-D.ietf-p2psip-base].

We first define a new type to define the new option, EXTENSIVE_ROUTING_MODE_TYPE:

The option value will be illustrated in the following figure, defining the ExtensiveRoutingModeOption structure:

```
enum { 0x0, 0x01 (DRR), 0x02(RPR), 255} RouteMode;
struct {
        RouteMode               routemode;
        OverlayLink             transport;
        IpAddressPort          ipaddressport;
        Destination             destination<1..2>;
} ExtensiveRoutingModeOption;
```

The above structure reuses: Transport, Destination and IpAddressPort structure defined in section 5.3.1.1 and 5.3.2.2 in [I-D.ietf-p2psip-

base].

Route mode: refers to which type of routing mode is indicated to the destination peer. Currently, only DRR and RPR (specified in RPR draft [I-D.zong-p2psip-rpr]) are defined.

Transport: refers to the transport type which is used to deliver responses from the destination peer to the sending peer.

IpAddressPort: refers to the transport address that the destination peer should use to send the response to. This will be a sending node address for DRR.

Destination: refers to the sending node itself. If the routing mode is DRR, then the destination only contains the sending node's node-id.

6.3. Creating a Request

6.3.1. Creating a Request for DRR

When using DRR for a transaction, the sending peer MUST set the IGNORE-STATE-KEEPING flag in the ForwardingHeader. Additionally, the peer MUST construct and include a ForwardingOptions structure in the ForwardingHeader. When constructing the ForwardingOption structure, the fields MUST be set as follows:

1) The type MUST be set to EXTENSIVE_ROUTING_MODE_TYPE.

2) The ExtensiveRoutingModeOption structure MUST be used for the option field within the ForwardingOptions structure. The fields MUST be defined as follows:

2.1) RouteMode set to 0x01 (DRR).

2.2) Transport set as appropriate for the sender.

2.3) IPAddressPort set to the peer's associated transport address.

2.4) The destination structure MUST contain one value, defined as type peer and set with the sending peer's own values.

6.4. Request And Response Processing

This section gives normative text for message processing after DRR is introduced. Here, we only describe the additional procedures for supporting DRR. Please refer to [I-D.ietf-p2psip-base] for RELOAD base procedures.

6.4.1. Destination Peer: Receiving a Request And Sending a Response

When the destination peer receives a request, it will check the options in the forwarding header. If the destination peer can not understand extensive_routing_mode option in the request, it MUST attempt to use SRR to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer.

If the routing mode is DRR, the peer MUST construct the Destination list for the response with only one entry, using the sending peer's node-id from the option in the request as the value.

In the event that the routing mode is set to DRR and there is not exactly one destination, the destination peer MUST try to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer using SRR.

After the peer constructs the destination list for the response, it sends the response to the transport address which is indicated in the IpAddressPort field in the option using the specific transport mode in the Forwardingoption. If the destination peer receives a retransmit with SRR preference on the message it is trying to response to now, the responding peer should abort the DRR response and use SRR.

6.4.2. Sending Peer: Receiving a Response

Upon receiving a response, the peer follows the rules in [I-D.ietf-p2psip-base]. The peer should note if DRR worked in order to decide if to offer DRR again. If the peer does not receive a response until the timeout it SHOULD resend the request using SRR.

7. Optional Methods to Investigate Node Connectivity

This section is for informational purposes only for providing some mechanisms that can be used when the configuration information does not specify if DRR can be used. It summarizes some methods which can be used for a node to determine its own network location compared with NAT. These methods may help a node to decide which routing mode it may wish to try. Note that there is no foolproof way to determine if a node is publically reachable, other than via out-of-band mechanisms. As such, peers using these mechanisms may be able to optimize traffic, but must be able to fall back to SRR routing if the other routing mechanisms fail.

For DRR to function correctly, a node may attempt to determine whether it is publicly reachable. If it is not, the peers should fall back to SRR. If the peer believes it is publically reachable, DRR may be attempted. NATs and firewalls are two major contributors preventing DRR from functioning properly. There are a number of techniques by which a node can get its reflexive address on the public side of the NAT. After obtaining the reflexive address, a peer can perform further tests to learn whether the reflexive address is publicly reachable. If the address appears to be publicly reachable, the nodes to which the address belongs can use DRR for responses.

Some conditions are unique in P2PSIP architecture which could be leveraged to facilitate the tests. In P2P overlay network, each node only has partial a view of the whole network, and knows of a few nodes in the overlay. P2P routing algorithms can easily deliver a request from a sending node to a peer with whom the sending node has no direct connection. This makes it easy for a node to ask other nodes to send unsolicited messages back to the requester.

In the following sections, we first introduce several ways for a node to get the addresses needed for the further tests. Then a test for learning whether a peer may be publicly reachable is proposed.

7.1. Getting Addresses To Be Used As Candidates for DRR

In order to test whether a peer may be publicly reachable, the node should first get one or more addresses which will be used by other nodes to send him messages directly. This address is either a local address of a node or a translated address which is assigned by a NAT to the node.

STUN is used to get a reflexive address on the public side of a NAT with the help of STUN servers. There is also a STUN usage [I-D.ietf-behave-nat-behavior-discovery] to discover NAT behavior. Under RELOAD architecture, a few infrastructure servers can be leveraged for this usage, such as enrollment servers, diagnostic servers, bootstrap servers, etc.

The node can use a STUN Binding request to one of STUN servers to trigger a STUN Binding response which returns the reflexive address from the server's perspective. If the reflexive transport address is the same as the source address of the Binding request, the node can determine that there likely is no NAT between him and the chosen infrastructure server. (Certainly, in some rare cases, the allocated address happens to be the same as the source address. Further tests will detect this case and rule it out in the end.). Usually, these infrastructure severs are publicly reachable in the overlay, so the

node can be considered publicly reachable. On the other hand, with the techniques in [I-D.ietf-behave-nat-behavior-discovery], a node can also decide whether it is behind NAT with endpoint-independent mapping behavior. If the node is behind a NAT with endpoint-independent mapping behavior, the reflexive address should also be a candidate for further tests.

UPnP-IGD is a mechanism that a node can use to get the assigned address from its residential gateway and after obtaining this address to communicate it with other nodes, the node can receive unsolicited messages from outside, even though it is behind a NAT. So the address obtained through the UPnP mechanism should also be used for further tests.

Another way that a node behind NAT can use to learn its assigned address by NAT is NAT-PMP. Like in UPnP-IGD, the address obtained using this mechanism should also be tested further.

The above techniques are not exhaustive. These techniques can be used to get candidate transport addresses for further tests.

7.2. Public Reacheability Test

Using the transport addresses obtained by the above techniques, a node can start a test to learn whether the candidate transport address is publicly reachable. The basic idea for the test is for a node to send a request and expect another node in the overlay to send back a response. If the response is received by the sending node successfully and also the node giving the response has no direct connection with the sending node, the sending node can determine that the address is probably publicly reachable and hence the node may be publicly reachable at the tested transport address.

In P2P overlay, a request is routed through the overlay and finally a destination peer will terminate the request and give the response. In a large system, there is a high probability that the destination peer has no direct connection with the sending node. Especially in RELOAD architecture, every node maintains a connection table. So it is easier for a node to check whether it has direct connection with another node.

Note: Currently, no existing message in base RELOAD can achieve the test. In our opinion, this kind of test is within diagnostic scope, so authors hope WG can define a new diagnostic message to do that. We don’t plan to define the message in this document, for the objective of this draft is to propose an extension to support DRR. The following text is informative.

If a node wants to test whether its transport address is publicly reachable, it can send a request to the overlay. The routing for the test message would be different from other kinds of requests because it is not for storing/fetching something to/from the overlay or locating a specific node, instead it is to get a peer who can deliver the sending node an unsolicited response and which has no direct connection with him. Each intermediate peer receiving the request first checks whether it has a direct connections with the sending peer. If there is a direct connection, the request is routed to the next peer. If there is no direct connection, the intermediate peer terminates the request and sends the response back directly to the sending node with the transport address under test.

After performing the test, if the peer determines that it may be publicly reachable, it can try DRR in subsequent transaction.

8. Security Considerations

TBD

9. IANA Considerations

9.1. A new RELOAD Forwarding Option

A new RELOAD Forwarding Option type is add to the Registry.

Type: 0x1 - extensive_routing_mode

10. Acknowledgements

David Bryan has helped extensively with this document, and helped provide some of the text, analysis, and ideas contained here. The authors would like to thank Ted Hardie, Narayanan Vidya, Dondeti Lakshminath and Bruce Lowekamp for their constructive comments.

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-12 (work in progress), March 2010.

[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-03 (work in progress), October 2010.

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

11.2. Informative References

[ChurnDHT] Rhea, S., "Handling Churn in a DHT", Proceedings of the USENIX Annual Technical Conference. Handling Churn in a DHT, June 2004.

[DTLS] Modadugu, N., Rescorla, E., "The Design and Implementation of Datagram TLS", 11th Network and Distributed System Security Symposium (NDSS), 2004.

[I-D.ietf-behave-nat-behavior-discovery] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using STUN", draft-ietf-behave-nat-behavior-discovery-04 (work in progress), July 2008.

[I-D.ietf-behave-tcp] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", draft-ietf-behave-tcp-08 (work in progress), September 2008.

[I-D.lowekamp-mmusic-ice-tcp-framework] Lowekamp, B. and A. Roach, "A Proposal to Define Interactive Connectivity Establishment for the Transport Control Protocol (ICE-TCP) as an Extensible Framework", draft-lowekamp-mmusic-ice-tcp-framework-00 (work in progress), October 2008.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[I-D.zong-p2psip-rpr] Zong, N., Jiang, X., Even, R. and Zhang, Y., "An extension to RELOAD to support Relay Peer Routing", draft-zong-p2psip-rpr-00, May 2011.

Authors' Addresses

Ning Zong
Huawei Technologies

Email: zongning@huawei.com

Xingfeng Jiang
Huawei Technologies

Email: jiang.x.f@huawei.com

Roni Even
Huawei Technologies

Email: even.roni@huawei.com

Yunfei Zhang
China Mobile

Email: zhangyunfei@chinamobile.com

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: March 22, 2012

N. Zong
X. Jiang
R. Even
Huawei Technologies
Y. Zhang
China Mobile

September 19, 2011

An extension to RELOAD to support Direct Response Routing
draft-zong-p2psip-drr-01

Abstract

This document proposes an optional extension to RELOAD to support direct response routing mode. RELOAD recommends symmetric recursive routing for routing messages. The new optional extension provides a shorter route for responses reducing the overhead on intermediary peers and describes the potential cases where this extension can be used.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on March 22, 2012.

Copyright Notice

Copyright (c) 2011 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
1.1. Backgrounds 4
2. Terminology 4
3. Problem Statement 5
3.1. Overview 5
3.1.1. Symmetric Recursive Routing (SRR) 5
3.1.2. Direct Response Routing (DRR) 6
3.2. Scenarios Where DRR can be Used 7
3.2.1. Managed or Closed P2P System 7
3.2.2. Wireless Scenarios 7
4. Relationship Between SRR and DRR 8
4.1. How DRR Works 8
4.2. How SRR and DRR Work Together 8
5. Comparison on cost of SRR and DRR 8
5.1. Closed or managed networks 9
5.2. Open networks 10
6. Extensions to RELOAD 10
6.1. Basic Requirements 10
6.2. Modification To RELOAD Message Structure 11
6.2.1. State-keeping Flag 11
6.2.2. Extensive Routing Mode 11
6.3. Creating a Request 12
6.3.1. Creating a Request for DRR 12
6.4. Request And Response Processing 12
6.4.1. Destination Peer: Receiving a Request And Sending a Response 13
6.4.2. Sending Peer: Receiving a Response 13
7. Optional Methods to Investigate Node Connectivity 13
7.1. Getting Addresses To Be Used As Candidates for DRR 14
7.2. Public Reacheability Test 15
8. Security Considerations 16
9. IANA Considerations 16
9.1. A new RELOAD Forwarding Option 16
10. Acknowledgements 16
11. References 16
11.1. Normative References 16
11.2. Informative References 17
Authors' Addresses 17

1. Introduction

1.1. Backgrounds

RELOAD [I-D.ietf-p2psip-base] recommends symmetric recursive routing (SRR) for routing messages and describes the extensions that would be required to support additional routing algorithms. Other than SRR, two other routing options: direct response routing (DRR) and relay peer routing (RPR) are also discussed in Appendix D in [I-D.ietf-p2psip-base]. As we show in section 3, DRR is advantageous over SRR in some scenarios by reducing load (CPU and link BW) on intermediary peers . For example, in a closed network where every node is in the same address realm, DRR performs better than SRR. In other scenarios, using a combination of DRR and SRR together is more likely to bring benefits than if SRR is used alone. Some discussion on connectivity is in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf].

Note that in this draft, we focus on DRR routing mode and its extensions to RELOAD to produce a standalone solution. Please refer to RPR draft [I-D.zong-p2psip-rpr] for RPR routing mode.

We first discuss the problem statement in Section 3, then how to combine DRR and SRR is presented in Section 4. In Section 5, we give comparison on the cost of SRR and DRR in both managed and open networks. An extension to RELOAD to support DRR is proposed in Section 6. Some optional methods to check node connectivity is introduced in Section 7, as informational text.

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 Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] draft extensively in this document. We also use terms defined in NAT behavior discovery [I-D.ietf-behave-nat-behavior-discovery]. Other terms used in this document are defined inline when used and are also defined below for reference.

There are two types of roles in the RELOAD architecture: peer and client. Node is used when describing both peer and client. In discussions specific to behavior of a peer or client, the term peer or client is used instead.

Publicly Reachable: A node is publicly reachable if it can receive unsolicited messages from any other node in the same overlay. Note: "publicly" does not mean that the nodes must be on the public Internet, because the RELOAD protocol may be used in a closed system.

Direct Response Routing (DRR): refers to a routing mode in which responses to P2PSIP requests are returned to the sending peer directly from the destination peer based on the sending peer's own local transport address(es). For simplicity, the abbreviation DRR is used instead in the following text.

Symmetric Recursive Routing (SRR): refers to a routing mode in which responses follow the request path in the reverse order to get back to the sending peer. For simplicity, the abbreviation SRR is used instead in the following text.

3. Problem Statement

RELOAD is expected to work under a great number of application scenarios. The situations where RELOAD is to be deployed differ greatly. For instance, some deployments are global, such as a Skype-like system intended to provide public service. Some run in closed networks of small scale. SRR works in any situation, but DRR may work better in some specific scenarios.

3.1. Overview

RELOAD is a simple request-response protocol. After sending a request, a node waits for a response from a destination node. There are several ways for the destination node to send a response back to the source node. In this section, we will provide detailed information on two routing modes: SRR and DRR.

Some assumptions are made in the following illustrations.

1) Peer A sends a request destined to a peer who is the responsible peer for Resource-ID k;

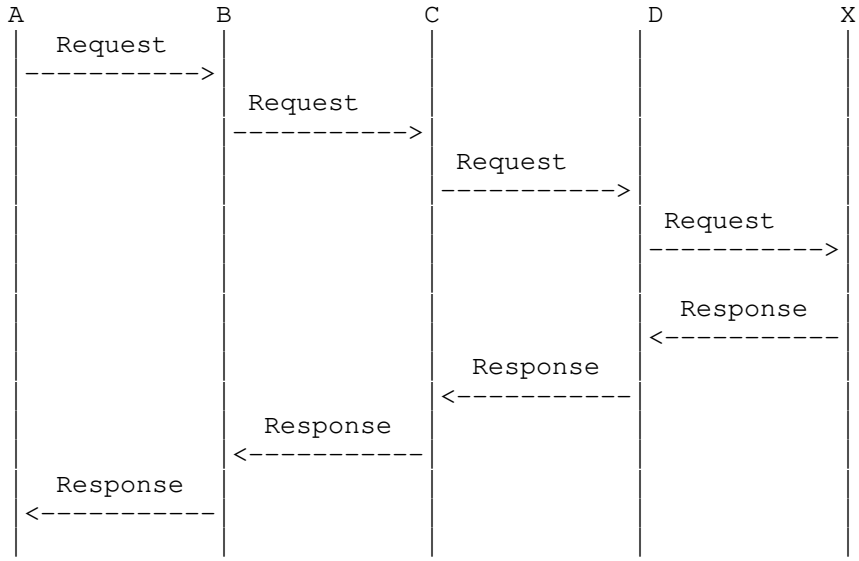
2) Peer X is the root peer being responsible for resource k;

3) The intermediate peers for the path from A to X are peer B, C, D.

3.1.1. Symmetric Recursive Routing (SRR)

For SRR, when the request sent by peer A is received by an intermediate peer B, C or D, each intermediate peer will insert information on the peer from whom they got the request in the via-

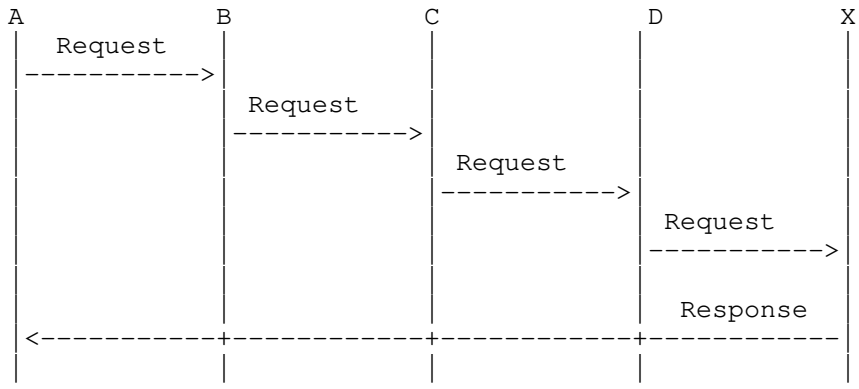
list as described in RELOAD. As a result, the destination peer X will know the exact path which the request has traversed. Peer X will then send back the response in the reverse path by constructing a destination list based on the via-list in the request.



SRR works in any situation, especially when there are NATs or firewalls. A downside of this solution is that the message takes several hops to return to the client, increasing the bandwidth usage and CPU/battery load of multiple nodes.

3.1.2. Direct Response Routing (DRR)

In DRR, peer X receives the request sent by peer A through intermediate peer B, C and D, as in SRR. However, peer X sends the response back directly to peer A based on peer A's local transport address. In this case, the response won't be routed through intermediate peers. Shorter route means less overhead on intermediary peers, especially in the case of wireless network where the CPU and uplink BW is limited. In the absence of NATs or other connectivity issues, this is the optimal routing technique. Note that establishing a secure connection requires multiple round trips. Please refer to Section 5 for cost comparison between SRR and DRR.



3.2. Scenarios Where DRR can be Used

This section lists several scenarios where using DRR would work, and when the increased efficiency would be advantageous.

3.2.1. Managed or Closed P2P System

The properties that make P2P technology attractive, such as the lack of need for centralized severs, self-organization, etc. are attractive for managed systems as well as unmanaged systems. Many of these systems are deployed on private network where nodes are in the same address realm and/or can directly route to each other. In such a scenario, the network administrator can indicate preference for DRR in the peer's configuration file. Peers in such a system would always try DRR first, but peers must also support SRR in case DRR fails. If during the process of establishing a direct connection with the sending peer, the responding peer receives a response with SRR as the preferred routing mode (or it fails to establish the direct connection), the responding peer should not use DRR but switch to SRR. A node can keep a list of unreachable nodes based on trying DRR and use only SRR for these nodes. The advantage in using DRR is on the network stability since it puts less overhead on the intermediary peers that will not route the responses. The intermediary peers will need to route less messages and save CPU resources as well as the link bandwidth usage.

3.2.2. Wireless Scenarios

While some mobile deployments may use clients, in mobile networks with full peers, there is an advantage to using DRR in order to reduce the load on intermediary nodes. Using DRR helps with reducing radio battery usage and bandwidth by the intermediary peers. The service provider may recommend in the configuration using DRR based on his knowledge of the topology.

4. Relationship Between SRR and DRR

4.1. How DRR Works

DRR is very simple. The only requirement is for the source peers to provide their (publically reachable) transport address to the destination peers, so that the destination peer knows where to send the response. Responses are sent directly to the requesting peer.

4.2. How SRR and DRR Work Together

DRR is not intended to replace SRR. As seen from Section 3, DRR has better performance in some scenarios, but have limitations as well, see for example section 4.3 in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf]. As a result, it is better to use these two modes together to adapt to each peer's specific situation. In this section, we give some informative suggestions on how to transition between the routing modes in RELOAD.

A peer can collect statistical data on the success of the different routing modes based on previous transactions and keep a list of non-reachable addresses. Based on the data, the peer will have a clearer view about the success rate of different routing modes. Other than the success rate, the peer can also get data of fine granularity, for example, the number of retransmission the peer needs to achieve a desirable success rate.

A typical strategy for the node is as follows. A node chooses to start with DRR. Based on the success rate as seen from the lost message statistics or responses that used DRR, the node can either continue to offer DRR first or switch to SRR.

The node can decide whether to try DRR based on other information such as configuration file information. If an overlay runs within a private network and all nodes in the system can reach each other directly, nodes may send most of the transactions with DRR.

5. Comparison on cost of SRR and DRR

The major advantages in using DRR are in going through less intermediary peers on the response. By doing that it reduces the load on those peers' resources like processing and communication bandwidth.

5.1. Closed or managed networks

As described in Section 3, many P2P systems run in a closed or managed environment (e.g. carrier networks) so that network administrators would know that they could safely use DRR.

SRR brings out more routing hops than DRR. Assuming that there are N nodes in the P2P system and Chord is applied for routing, the number of hops for a response in SRR and DRR are listed in the following table. Establishing a secure connection between sending peer and responding peer with (D)TLS requires multiple messages. Note that establishing (D)TLS secure connections for P2P overlay is not optimal in some cases, e.g. direct response routing where (D)TLS is heavy for temporary connections. Instead, some alternate security techniques, e.g. using public keys of the destination to encrypt the messages, signing timestamps to prevent reply attacks can be adopted. Therefore, in the following table, we show the cases of: 1) no (D)TLS in DRR; 2) still using DTLS in DRR as sub-optimal and, as the worst-cost case, 7 messages are used during the DTLS handshaking [DTLS]. (TLS Handshake is two round-trip negotiation protocol while DTLS handshake is three round-trip negotiation protocol.)

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
DRR	Yes	1	1
DRR(DTLS)	Yes	1	7+1

From the above comparison, it is clear that:

1) In most cases of N > 2 (2^1), DRR has fewer hops than SRR. Shorter route means less overhead and resource usage on intermediary peers, which is an important consideration for adopting DRR in the cases where the resource such as CPU and BW is limited, e.g. the case of mobile, wireless network.

2) In the cases of N > 256 (2^8), DRR also has fewer messages than SRR.

3) In the cases where N < 256, DRR has more messages than SRR (but still has fewer hops than SRR). So the consideration to use DRR or SRR depends on other factors like using less resources (bandwidth and processing) from the intermediaries peers. Section 4 provides use cases where DRR has better chance to work or where the intermediary resources considerations are important.

5.2. Open networks

In open network where DRR is not guaranteed, DRR can fall back to SRR If it fails after trial, as described in Section 4. Based on the same settings in Section 5.1, the number of hops, number of messages for a response in SRR and DRR are listed in the following table.

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
DRR	Yes	1	1
	Fail&Fall back to SRR	1+logN	1+logN
DRR(DTLS)	Yes	1	7+1
	Fail&Fall back to SRR	1+logN	8+logN

From the above comparison, it can be observed that:

1) Trying DRR would still have a good chance of fewer hops than SRR. Suppose that P peers are publicly reachable, the number of hops in DRR and SRR is P*1+(N-P)*(1+logN), N*logN, respectively. The condition for fewer hops in DRR is P*1+(N-P)*(1+logN) < N*logN, which is P/N > 1/logN. This means that when the number of peers N grows, the required ratio of publicly reachable peers P/N for fewer hops in DRR decreases. Therefore, the chance of trying DRR with fewer hops than SRR becomes better as the scale of the network increases.

2) In the cases of large network and the success rate of DRR is good, it is still possible that DRR has fewer messages than SRR. Otherwise, the consideration to use DRR or SRR depends on other factors like using less resources from the intermediaries peers.

6. Extensions to RELOAD

Adding support for DRR requires extensions to the current RELOAD protocol. In this section, we define the changes required to the protocol, including changes to message structure and to message processing.

6.1. Basic Requirements

All peers implementing DRR MUST support SRR.

All peers MUST be able to process requests for routing in SRR, and MAY support DRR routing requests.

6.2. Modification To RELOAD Message Structure

RELOAD provides an extensible framework to accommodate future extensions. In this section, we define a ForwardingOption structure to support DRR mode. Additionally we present a state- keeping flag to inform intermediate peers if they are allowed to not maintain state for a transaction.

6.2.1. State-keeping Flag

RELOAD allows intermediate peers to maintain state in order to implement SRR, for example for implementing hop-by-hop retransmission. If DRR is used, the response will not follow the reverse path, and the state in the intermediate peers won't be cleared until such state expires. In order to address this issue, we propose a new flag, state-keeping flag, in the message header to indicate whether the state keeping is not required in the intermediate peers.

flag : 0x8 IGNORE-STATE-KEEPING

If IGNORE-STATE-KEEPING is set, any peer receiving this message and which is not the destination of the message SHOULD forward the message with the full VIA list and SHOULD not maintain any internal state.

6.2.2. Extensive Routing Mode

This draft introduces a new forwarding option for an extensive routing mode. This option conforms to the description in section 5.3.2.3 in [I-D.ietf-p2psip-base].

We first define a new type to define the new option, EXTENSIVE_ROUTING_MODE_TYPE:

The option value will be illustrated in the following figure, defining the ExtensiveRoutingModeOption structure:

```
enum { 0x0, 0x01 (DRR), 0x02(RPR), 255} RouteMode;
struct {
        RouteMode               routemode;
        OverlayLink             transport;
        IpAddressPort          ipaddressport;
        Destination             destination<1..2^8-1>;
} ExtensiveRoutingModeOption;
```

The above structure reuses: Transport, Destination and IpAddressPort structure defined in section 5.3.1.1 and 5.3.2.2 in [I-D.ietf-p2psip-

base].

Route mode: refers to which type of routing mode is indicated to the destination peer. Currently, only DRR and RPR (specified in RPR draft [I-D.zong-p2psip-rpr]) are defined.

Transport: refers to the transport type which is used to deliver responses from the destination peer to the sending peer.

IpAddressPort: refers to the transport address that the destination peer should use to send the response to. This will be a sending node address for DRR.

Destination: refers to the sending node itself. If the routing mode is DRR, then the destination only contains the sending node's node-id.

6.3. Creating a Request

6.3.1. Creating a Request for DRR

When using DRR for a transaction, the sending peer MUST set the IGNORE-STATE-KEEPING flag in the ForwardingHeader. Additionally, the peer MUST construct and include a ForwardingOptions structure in the ForwardingHeader. When constructing the ForwardingOption structure, the fields MUST be set as follows:

1) The type MUST be set to EXTENSIVE_ROUTING_MODE_TYPE.

2) The ExtensiveRoutingModeOption structure MUST be used for the option field within the ForwardingOptions structure. The fields MUST be defined as follows:

2.1) RouteMode set to 0x01 (DRR).

2.2) Transport set as appropriate for the sender.

2.3) IPAddressPort set to the peer's associated transport address.

2.4) The destination structure MUST contain one value, defined as type peer and set with the sending peer's own values.

6.4. Request And Response Processing

This section gives normative text for message processing after DRR is introduced. Here, we only describe the additional procedures for supporting DRR. Please refer to [I-D.ietf-p2psip-base] for RELOAD base procedures.

6.4.1. Destination Peer: Receiving a Request And Sending a Response

When the destination peer receives a request, it will check the options in the forwarding header. If the destination peer can not understand extensive_routing_mode option in the request, it MUST attempt to use SRR to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer.

If the routing mode is DRR, the peer MUST construct the Destination list for the response with only one entry, using the sending peer's node-id from the option in the request as the value.

In the event that the routing mode is set to DRR and there is not exactly one destination, the destination peer MUST try to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer using SRR.

After the peer constructs the destination list for the response, it sends the response to the transport address which is indicated in the IpAddressPort field in the option using the specific transport mode in the Forwardingoption. If the destination peer receives a retransmit with SRR preference on the message it is trying to response to now, the responding peer should abort the DRR response and use SRR.

6.4.2. Sending Peer: Receiving a Response

Upon receiving a response, the peer follows the rules in [I-D.ietf-p2psip-base]. The peer should note if DRR worked in order to decide if to offer DRR again. If the peer does not receive a response until the timeout it SHOULD resend the request using SRR.

7. Optional Methods to Investigate Node Connectivity

This section is for informational purposes only for providing some mechanisms that can be used when the configuration information does not specify if DRR can be used. It summarizes some methods which can be used for a node to determine its own network location compared with NAT. These methods may help a node to decide which routing mode it may wish to try. Note that there is no foolproof way to determine if a node is publically reachable, other than via out-of-band mechanisms. As such, peers using these mechanisms may be able to optimize traffic, but must be able to fall back to SRR routing if the other routing mechanisms fail.

For DRR to function correctly, a node may attempt to determine whether it is publicly reachable. If it is not, the peers should fall back to SRR. If the peer believes it is publically reachable, DRR may be attempted. NATs and firewalls are two major contributors preventing DRR from functioning properly. There are a number of techniques by which a node can get its reflexive address on the public side of the NAT. After obtaining the reflexive address, a peer can perform further tests to learn whether the reflexive address is publicly reachable. If the address appears to be publicly reachable, the nodes to which the address belongs can use DRR for responses.

Some conditions are unique in P2PSIP architecture which could be leveraged to facilitate the tests. In P2P overlay network, each node only has partial a view of the whole network, and knows of a few nodes in the overlay. P2P routing algorithms can easily deliver a request from a sending node to a peer with whom the sending node has no direct connection. This makes it easy for a node to ask other nodes to send unsolicited messages back to the requester.

In the following sections, we first introduce several ways for a node to get the addresses needed for the further tests. Then a test for learning whether a peer may be publicly reachable is proposed.

7.1. Getting Addresses To Be Used As Candidates for DRR

In order to test whether a peer may be publicly reachable, the node should first get one or more addresses which will be used by other nodes to send him messages directly. This address is either a local address of a node or a translated address which is assigned by a NAT to the node.

STUN is used to get a reflexive address on the public side of a NAT with the help of STUN servers. There is also a STUN usage [I-D.ietf-behave-nat-behavior-discovery] to discover NAT behavior. Under RELOAD architecture, a few infrastructure servers can be leveraged for this usage, such as enrollment servers, diagnostic servers, bootstrap servers, etc.

The node can use a STUN Binding request to one of STUN servers to trigger a STUN Binding response which returns the reflexive address from the server's perspective. If the reflexive transport address is the same as the source address of the Binding request, the node can determine that there likely is no NAT between him and the chosen infrastructure server. (Certainly, in some rare cases, the allocated address happens to be the same as the source address. Further tests will detect this case and rule it out in the end.). Usually, these infrastructure severs are publicly reachable in the overlay, so the

node can be considered publicly reachable. On the other hand, with the techniques in [I-D.ietf-behave-nat-behavior-discovery], a node can also decide whether it is behind NAT with endpoint-independent mapping behavior. If the node is behind a NAT with endpoint-independent mapping behavior, the reflexive address should also be a candidate for further tests.

UPnP-IGD is a mechanism that a node can use to get the assigned address from its residential gateway and after obtaining this address to communicate it with other nodes, the node can receive unsolicited messages from outside, even though it is behind a NAT. So the address obtained through the UPnP mechanism should also be used for further tests.

Another way that a node behind NAT can use to learn its assigned address by NAT is NAT-PMP. Like in UPnP-IGD, the address obtained using this mechanism should also be tested further.

The above techniques are not exhaustive. These techniques can be used to get candidate transport addresses for further tests.

7.2. Public Reacheability Test

Using the transport addresses obtained by the above techniques, a node can start a test to learn whether the candidate transport address is publicly reachable. The basic idea for the test is for a node to send a request and expect another node in the overlay to send back a response. If the response is received by the sending node successfully and also the node giving the response has no direct connection with the sending node, the sending node can determine that the address is probably publicly reachable and hence the node may be publicly reachable at the tested transport address.

In P2P overlay, a request is routed through the overlay and finally a destination peer will terminate the request and give the response. In a large system, there is a high probability that the destination peer has no direct connection with the sending node. Especially in RELOAD architecture, every node maintains a connection table. So it is easier for a node to check whether it has direct connection with another node.

Note: Currently, no existing message in base RELOAD can achieve the test. In our opinion, this kind of test is within diagnostic scope, so authors hope WG can define a new diagnostic message to do that. We don’t plan to define the message in this document, for the objective of this draft is to propose an extension to support DRR. The following text is informative.

If a node wants to test whether its transport address is publicly reachable, it can send a request to the overlay. The routing for the test message would be different from other kinds of requests because it is not for storing/fetching something to/from the overlay or locating a specific node, instead it is to get a peer who can deliver the sending node an unsolicited response and which has no direct connection with him. Each intermediate peer receiving the request first checks whether it has a direct connections with the sending peer. If there is a direct connection, the request is routed to the next peer. If there is no direct connection, the intermediate peer terminates the request and sends the response back directly to the sending node with the transport address under test.

After performing the test, if the peer determines that it may be publicly reachable, it can try DRR in subsequent transaction.

8. Security Considerations

TBD

9. IANA Considerations

9.1. A new RELOAD Forwarding Option

A new RELOAD Forwarding Option type is add to the Registry.

Type: 0x2 - extensive_routing_mode

10. Acknowledgements

David Bryan has helped extensively with this document, and helped provide some of the text, analysis, and ideas contained here. The authors would like to thank Ted Hardie, Narayanan Vidya, Dondeti Lakshminath and Bruce Lowekamp for their constructive comments.

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-18 (work in progress), August 2011.

[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-03 (work in progress), October 2010.

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

11.2. Informative References

[ChurnDHT] Rhea, S., "Handling Churn in a DHT", Proceedings of the USENIX Annual Technical Conference. Handling Churn in a DHT, June 2004.

[DTLS] Modadugu, N., Rescorla, E., "The Design and Implementation of Datagram TLS", 11th Network and Distributed System Security Symposium (NDSS), 2004.

[I-D.ietf-behave-nat-behavior-discovery] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using STUN", draft-ietf-behave-nat-behavior-discovery-04 (work in progress), July 2008.

[I-D.ietf-behave-tcp] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", draft-ietf-behave-tcp-08 (work in progress), September 2008.

[I-D.lowekamp-mmusic-ice-tcp-framework] Lowekamp, B. and A. Roach, "A Proposal to Define Interactive Connectivity Establishment for the Transport Control Protocol (ICE-TCP) as an Extensible Framework", draft-lowekamp-mmusic-ice-tcp-framework-00 (work in progress), October 2008.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[I-D.zong-p2psip-rpr] Zong, N., Jiang, X., Even, R. and Zhang, Y., "An extension to RELOAD to support Relay Peer Routing", draft-zong-p2psip-rpr-01, September 2011.

Authors' Addresses

Ning Zong
Huawei Technologies

Email: zongning@huawei.com

Xingfeng Jiang
Huawei Technologies

Email: jiang.x.f@huawei.com

Roni Even
Huawei Technologies

Email: even.roni@huawei.com

Yunfei Zhang
China Mobile

Email: zhangyunfei@chinamobile.com

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: December 23, 2011

N. Zong
X. Jiang
R. Even
Huawei Technologies
Y. Zhang
China Mobile

June 21, 2011

An extension to RELOAD to support Relay Peer Routing
draft-zong-p2psip-rpr-00

Abstract

This document proposes an optional extension to RELOAD to support relay peer routing mode. RELOAD recommends symmetric recursive routing for routing messages. The new optional extension provides a shorter route for responses reducing the overhead on intermediary peers and describes the potential cases where this extension can be used.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 23, 2011.

Copyright Notice

Copyright (c) 2011 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
1.1. Backgrounds 4
2. Terminology 4
3. Problem Statement 5
3.1. Overview 5
3.1.1. Relay Peer Routing (RPR) 6
3.2. Scenarios Where RPR Benefits 6
3.2.1. Managed or Closed P2P System 6
3.2.2. Using Bootstrap Peers as Relay Peers 7
3.2.3. Wireless Scenarios 7
4. Relationship Between SRR and RPR 7
4.1. How RPR Works 7
4.2. How SRR and RPR Work Together 8
5. Comparison on cost of SRR and RPR 8
5.1. Closed or managed networks 8
5.2. Open networks 9
6. Extensions to RELOAD 9
6.1. Basic Requirements 9
6.2. Modification To RELOAD Message Structure 10
6.2.1. State-keeping Flag 10
6.2.2. Extensive Routing Mode 10
6.3. Creating a Request 10
6.3.1. Creating a request for RPR 10
6.4. Request And Response Processing 11
6.4.1. Destination Peer: Receiving a Request And Sending a Response 11
6.4.2. Sending Peer: Receiving a Response 12
6.4.3. Relay Peer Processing 12
7. Discovery Of Relay Peer 12
8. Optional Methods to Investigate Node Connectivity 12
9. Security Considerations 13
10. IANA Considerations 13
11. Acknowledgements 13
12. References 14
12.1. Normative References 14
12.2. Informative References 14
Authors' Addresses 15

1. Introduction

1.1. Backgrounds

RELOAD [I-D.ietf-p2psip-base] recommends symmetric recursive routing (SRR) for routing messages and describes the extensions that would be required to support additional routing algorithms. Other than SRR, two other routing options: direct response routing (DRR) and relay peer routing (RPR) are also discussed in Appendix D in [I-D.ietf-p2psip-base]. DRR is specified in [I-D.zong-p2psip-drr]. As we show in section 3, RPR is advantageous over SRR in some scenarios reducing load (CPU and link BW) on intermediary peers . RPR works better in a network where relay peers are provisioned in advance so that relay peers are publicly reachable in the P2P system. In other scenarios, using a combination of RPR and SRR together is more likely to bring benefits than if SRR is used alone. Some discussion on connectivity is in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf].

Note that in this draft, we focus on RPR routing mode and its extensions to RELOAD. Some text such as modification to RELOAD message structure, optional methods to investigate node connectivity described in DRR draft [I-D.zong-p2psip-drr] are also relevent to RPR.

We first discuss the problem statement in Section 3, then how to combine RPR and SRR is presented in Section 4. In Section 5, we give comparison on the cost of SRR and RPR in both managed and open networks. An extension to RELOAD to support RPR is proposed in Section 6. Discovery of relay peers is introduced in Section 7. Some optional methods to check node connectivity is introduced in Section 8.

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 Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] draft extensively in this document. We also use terms defined in NAT behavior discovery [I-D.ietf-behave-nat-behavior-discovery]. Other terms used in this document are defined inline when used and are also defined below for reference.

There are two types of roles in the RELOAD architecture: peer and

client. Node is used when describing both peer and client. In discussions specific to behavior of a peer or client, the term peer or client is used instead.

Publicly Reachable: A node is publicly reachable if it can receive unsolicited messages from any other node in the same overlay. Note: "publicly" does not mean that the nodes must be on the public Internet, because the RELOAD protocol may be used in a closed system.

Relay Peer: A type of publicly reachable peer that can receive unsolicited messages from all other nodes in the overlay and forward the responses from destination peers towards the request sender.

Relay Peer Routing (RPR): refers to a routing mode in which responses to P2PSIP requests are sent by the destination peer to a relay peer transport address who will forward the responses towards the sending peer. For simplicity, the abbreviation RPR is used instead in the following text.

Symmetric Recursive Routing(SRR): refers to a routing mode in which responses follow the request path in the reverse order to get back to the sending peer. For simplicity, the abbreviation SRR is used instead in the following text.

3. Problem Statement

RELOAD is expected to work under a great number of application scenarios. The situations where RELOAD is to be deployed differ greatly. For instance, some deployments are global, such as a Skype-like system intended to provide public service. Some run in closed networks of small scale. SRR works in any situation, but RPR may work better in some specific scenarios.

3.1. Overview

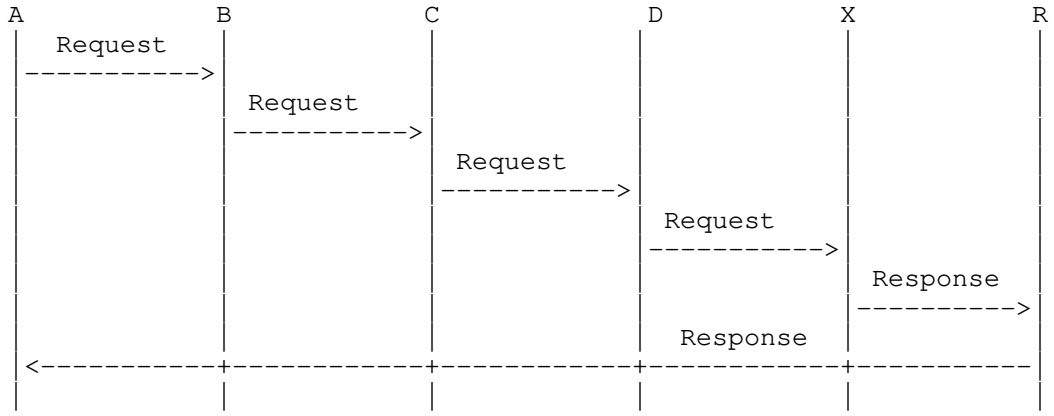
RELOAD is a simple request-response protocol. After sending a request, a node waits for a response from a destination node. There are several ways for the destination node to send a response back to the source node. In this section, we will provide detailed information on RPR.

Note that the same illustrative settings can be found in DRR draft [I-D.zong-p2psip-drr].

3.1.1. Relay Peer Routing (RPR)

If peer A knows it is behind a NAT or NATs, and knows one or more relay peers with whom they have a prior connections, peer A can try RPR. Assume A is associated with relay peer R. When sending the request, peer A includes information describing peer R transport address in the request. When peer X receives the request, peer X sends the response to peer R, which forwards it directly to Peer A on the existing connection. Note that RPR also allows a shorter route for responses compared to SRR, which means less overhead on intermediary peers. Establishing a connection to the relay with TLS requires multiple round trips. Please refer to Section 5 for cost comparison between SRR and RPR.

This technique relies on the relative population of nodes such as A that require relay peers and peers such as R that are capable of serving as a relay peers. It also requires mechanism to enable peers to know which nodes can be used as their relays. This mechanism may be based on configuration, for example as part of the overlay configuration an initial list of relay peers can be supplied. Another option is in a response to ATTACH request the peer can signal that it can be used as a relay peer.



3.2. Scenarios Where RPR Benefits

In this section, we will list several scenarios where using RPR would provide improved performance.

3.2.1. Managed or Closed P2P System

As described in Section 3.2.1, many P2P systems run in a closed or managed environment so that network administrators can better manage their system. For example, the network administrator can deploy

several relay peers which are publicly reachable in the system and indicate their presence in the configuration file. After learning where these relay peers are, peers behind NATs can use RPR with the help from these relay peers. Peers must also support SRR in case RPR fails.

Another usage is to install relay peers on the managed network boundary allowing external peers to send responses to peers inside the managed network.

3.2.2. Using Bootstrap Peers as Relay Peers

Bootstrap peers must be publicly reachable in a RELOAD architecture. As a result, one possible architecture would be to use the bootstrap peers as relay peers for use with RPR. The requirements for being a relay peer are publicly accessible and maintaining a direct connection with its client. As such, bootstrap peers are well suited to play the role of relay peers.

3.2.3. Wireless Scenarios

While some mobile deployments may use clients, in mobile networks using peers, RPR may reduce radio battery usage and bandwidth usage by the intermediary peers. The service provider may recommend in the configuration using RPR based on his knowledge of the topology. Such relay peers may also help connectivity to external networks.

4. Relationship Between SRR and RPR

4.1. How RPR Works

Peers using RPR must maintain a connection with their relay peer(s). This can be done in the same way as establishing a neighbor connection between peers by using the Attach method.

A requirement for RPR is for the source peer to convey their relay peer (or peers) transport address in the request, so the destination peer knows where the relay peer are and send the response to a relay peer first. The request should include also the requesting peer information enabling the relay peer to route the response back to the right peer.

(Editor's Note: Being a relay peer does not require that the relay peer have more functionality than an ordinary peer. As discussed later, relay peers comply with the same procedure as an ordinary peer to forward messages. The only difference is that there may be a larger traffic burden on relay peers. Relay peers can decide whether

to accept a new connection based on their current burden.)

4.2. How SRR and RPR Work Together

RPR is not intended to replace SRR. As seen from Section 3, RPR has better performance in some scenarios, but have limitations as well, see for example section 4.3 in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf]. As a result, it is better to use these two modes together to adapt to each peer's specific situation. Note that the informative suggestions on how to transition between SRR and RPR (e.g. compute success rate of RPR, fall back to SRR, etc) are same with that on DRR and RPR. Please refer to DRR draft [I-D.zong-p2psip-drr] for more details. Similarly, the node can decide whether to try RPR based on other information such as configuration file information. If a relay peer is provided by the service provider, nodes may prefer RPR over SRR.

5. Comparison on cost of SRR and RPR

The major advantages in using RPR are in going through less intermediary peers on the response. By doing that it reduces the load on those peers' resources like processing and communication bandwidth.

5.1. Closed or managed networks

As described in Section 3, many P2P systems run in a closed or managed environment (e.g. carrier networks) so that network administrators would know that they could safely use RPR.

The number of hops for a response in SRR and RPR are listed in the following table. Note that the same illustrative settings can be found in DRR draft [I-D.zong-p2psip-drr].

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
RPR	Yes	2	2
RPR(DTLS)	Yes	2	7+2

From the above comparison, it is clear that:

1) In most cases of N > 4 (2^2), RPR has fewer hops than SRR. Shorter route means less overhead and resource usage on intermediary peers, which is an important consideration for adopting RPR in the cases where the resource such as CPU and BW is limited, e.g. the case of mobile, wireless network.

2) In the cases of N > 512 (2^9), RPR also has fewer messages than SRR.

3) In the cases where N < 512, RPR has more messages than SRR (but still has fewer hops than SRR). So the consideration to use RPR or SRR depends on other factors like using less resources (bandwidth and processing) from the intermediaries peers. Section 4 provides use cases where RPR has better chance to work or where the intermediary resources considerations are important.

5.2. Open networks

In open network where RPR is not guaranteed, RPR can fall back to SRR If it fails after trial, as described in Section 4. Based on the same settings in Section 5.1, the number of hops, number of messages for a response in SRR and RPR are listed in the following table.

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
RPR	Yes	2	2
	Fail&Fall back to SRR	2+logN	2+logN
RPR(DTLS)	Yes	2	7+2
	Fail&Fall back to SRR	2+logN	9+logN

From the above comparison, it can be observed that:

1) Trying RPR would still have a good chance of fewer hops than SRR. The detailed analysis is same as DRR case and can be found in DRR draft [I-D.zong-p2psip-drr].

2) In the cases of large network and the success rate of RPR is good, it is still possible that RPR has fewer messages than SRR. Otherwise, the consideration to use RPR or SRR depends on other factors like using less resources from the intermediaries peers.

6. Extensions to RELOAD

Adding support for RPR requires extensions to the current RELOAD protocol. In this section and in DRR[I-D.zong-p2psip-drr], we define the changes required to the protocol, including changes to message structure and to message processing.

6.1. Basic Requirements

The basic requirements to peers for supporting RPR are same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr].

6.2. Modification To RELOAD Message Structure

RELOAD provides an extensible framework to accommodate future extensions. In this section and in DRR[I-D.zong-p2psip-drr], we define a ForwardingOption structure to support RPR mode.

6.2.1. State-keeping Flag

The state-keeping flag to support RPR is same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr].

6.2.2. Extensive Routing Mode

The ForwardingOption structure to support RPR is same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr]. The definition of the fields is as follow:

Route mode: refers to which type of routing mode is indicated to the destination peer. Currently, only DRR (specified in DRR draft [I-D.zong-p2psip-drr]) and RPR are defined.

Transport: refers to the transport type which is used to deliver responses from the destination peer to the relay peer.

IpAddressPort: refers to the transport address that the destination peer should use to send the response to. This will be a relay peer address for RPR.

Destination: refers to the relay peer itself. If the routing mode is RPR, then the destination contains two destinations, which are the relay peer's node-id and the sending node's node-id.

6.3. Creating a Request

6.3.1. Creating a request for RPR

When using RPR for a transaction, the sending peer MUST set the IGNORE-STATE-KEEPING flag in the ForwardingHeader. Additionally, the peer MUST construct and include a ForwardingOptions structure in the ForwardingHeader. When constructing the ForwardingOption structure, the fields MUST be set as follows:

1) The type MUST be set to EXTENSIVE_ROUTING_MODE_TYPE.

2) The ExtensiveRoutingModeOption structure MUST be used for the option field within the ForwardingOptions structure. The fields MUST be defined as follows:

2.1) RouteMode set to 0x02 (RPR).

2.2) Transport set as appropriate for the relay peer.

2.3) IPAddressPort set to the transport address of the relay peer that the sender wishes the message to be relayed through.

2.4) Destination structure MUST contain two values. The first MUST be defined as type peer and set with the values for the relay peer. The second MUST be defined as type peer and set with the sending peer's own values.

6.4. Request And Response Processing

This section gives normative text for message processing after RPR is introduced. Here, we only describe the additional procedures for supporting RPR. Please refer to [I-D.ietf-p2psip-base] for RELOAD base procedures.

6.4.1. Destination Peer: Receiving a Request And Sending a Response

When the destination peer receives a request, it will check the options in the forwarding header. If the destination peer can not understand extensive_routing_mode option in the request, it MUST attempt to use SRR to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer.

If the routing mode is RPR, the destination peer MUST construct a Destination list for the response with two entries. The first MUST be set to the relay peer node-id from the option in the request and the second MUST be the sending node node-id from the option of the request.

In the event that the routing mode is set to RPR and there are not exactly two destinations the destination peer MUST try to send an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer using SRR.

After the peer constructs the destination list for the response, it sends the response to the transport address which is indicated in the IpAddressPort field in the option using the specific transport mode in the Forwardingoption. If the destination peer receives a retransmit with SRR preference on the message it is trying to response to now, the responding peer should abort the RPR response and use SRR.

6.4.2. Sending Peer: Receiving a Response

Upon receiving a response, the peer follows the rules in [I-D.ietf-p2psip-base]. If the sender used RPR and does not get a response until the timeout, it MAY either resend the message using RPR but with a different relay peer (if available), or resend the message using SRR.

6.4.3. Relay Peer Processing

Relay peers are designed to forward responses to nodes who are not publicly reachable. For the routing of the response, this draft still uses the destination list. The only difference from SRR is that the destination list is not the reverse of the via-list, instead it is constructed from the forwarding option as described below.

When a relay peer receives a response, it MUST follow the rules in [I-D.ietf-p2psip-base]. It receives the response, validates the message, re-adjust the destination-list and forward the response to the next hop in the destination list based on the connection table. There is no added requirement for relay peer.

7. Discovery Of Relay Peer

There are several ways to distribute the information about relay peers throughout the overlay. P2P network providers can deploy some relay peers and advertise them in the configuration file. With the configuration file at hand, peers can get relay peers to try RPR. Another way is to consider relay peer as a service and then some service advertisement and discovery mechanism can also be used for discovering relay peers, for example, using the same mechanism as used in TURN server discovery in base RELOAD [I-D.ietf-p2psip-base]. Another option is to let a peer advertise his capability to be a relay in the response to ATTACH or JOIN.

Editor note: This section will be extended if we adopt RPR, but like other configuration information, there may be many ways to obtain this.

8. Optional Methods to Investigate Node Connectivity

This section is for informational purposes only for providing some mechanisms that can be used when the configuration information does not specify if RPR can be used. It summarizes some methods which can be used for a node to determine its own network location compared with NAT. These methods may help a node to decide which routing mode

it may wish to try. Note that there is no foolproof way to determine if a node is publically reachable, other than via out- of-band mechanisms. As such, peers using these mechanisms may be able to optimize traffic, but must be able to fall back to SRR routing if the other routing mechanisms fail.

For RPR to function correctly, a node may attempt to determine whether it is publicly reachable. If it is not, RPR may be chosen to route the response with the help from relay peers, or the peers should fall back to SRR. NATs and firewalls are two major contributors preventing RPR from functioning properly. There are a number of techniques by which a node can get its reflexive address on the public side of the NAT. After obtaining the reflexive address, a peer can perform further tests to learn whether the reflexive address is publicly reachable. If the address appears to be publicly reachable, the nodes to which the address belongs can be a candidate to serve as a relay peer. Nodes which are not publicly reachable may still use RPR to shorten the response path with the help from relay peers.

Some conditions are unique in P2PSIP architecture which could be leveraged to facilitate the tests. In P2P overlay network, each node only has partial a view of the whole network, and knows of a few nodes in the overlay. P2P routing algorithms can easily deliver a request from a sending node to a peer with whom the sending node has no direct connection. This makes it easy for a node to ask other nodes to send unsolicited messages back to the requester.

The approaches for a node to get the addresses needed for the further tests, as well as the test for learning whether a peer may be publicly reacheable is same as the DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr] for more details.

9. Security Considerations

TBD

10. IANA Considerations

No IANA action is needed.

11. Acknowledgements

David Bryan has helped extensively with this document, and helped provide some of the text, analysis, and ideas contained here. The

authors would like to thank Ted Hardie, Narayanan Vidya, Dondeti Lakshminath and Bruce Lowekamp for their constructive comments.

12. References

12.1. 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-12 (work in progress), March 2010.

[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-03 (work in progress), October 2010.

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

[I-D.zong-p2psip-drr] Zong, N., Jiang, X., Even, R. and Zhang, Y., "An extension to RELOAD to support Direct Response Routing", draft-zong-p2psip-drr-00, May 2011.

12.2. Informative References

[ChurnDHT] Rhea, S., "Handling Churn in a DHT", Proceedings of the USENIX Annual Technical Conference. Handling Churn in a DHT, June 2004.

[DTLS] Modadugu, N., Rescorla, E., "The Design and Implementation of Datagram TLS", 11th Network and Distributed System Security Symposium (NDSS), 2004.

[I-D.ietf-behave-nat-behavior-discovery] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using STUN", draft-ietf-behave-nat-behavior-discovery-04 (work in progress), July 2008.

[I-D.ietf-behave-tcp] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", draft-ietf-behave-tcp-08 (work in progress), September 2008.

[I-D.lowekamp-mmusic-ice-tcp-framework] Lowekamp, B. and A. Roach, "A Proposal to Define Interactive Connectivity Establishment for the Transport Control Protocol (ICE-TCP) as an Extensible Framework", draft-lowekamp-mmusic-ice-tcp-framework-00 (work in progress), October 2008.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

Authors' Addresses

Ning Zong
Huawei Technologies

Email: zongning@huawei.com

Xingfeng Jiang
Huawei Technologies

Email: jiang.x.f@huawei.com

Roni Even
Huawei Technologies

Email: even.roni@huawei.com

Yunfei Zhang
China Mobile

Email: zhangyunfei@chinamobile.com

P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: March 22, 2012

N. Zong
X. Jiang
R. Even
Huawei Technologies
Y. Zhang
China Mobile

September 19, 2011

An extension to RELOAD to support Relay Peer Routing
draft-zong-p2psip-rpr-01

Abstract

This document proposes an optional extension to RELOAD to support relay peer routing mode. RELOAD recommends symmetric recursive routing for routing messages. The new optional extension provides a shorter route for responses reducing the overhead on intermediary peers and describes the potential cases where this extension can be used.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on March 22, 2012.

Copyright Notice

Copyright (c) 2011 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
1.1. Backgrounds 4
2. Terminology 4
3. Problem Statement 5
3.1. Overview 5
3.1.1. Relay Peer Routing (RPR) 6
3.2. Scenarios Where RPR Benefits 6
3.2.1. Managed or Closed P2P System 6
3.2.2. Using Bootstrap Peers as Relay Peers 7
3.2.3. Wireless Scenarios 7
4. Relationship Between SRR and RPR 7
4.1. How RPR Works 7
4.2. How SRR and RPR Work Together 8
5. Comparison on cost of SRR and RPR 8
5.1. Closed or managed networks 8
5.2. Open networks 9
6. Extensions to RELOAD 9
6.1. Basic Requirements 9
6.2. Modification To RELOAD Message Structure 10
6.2.1. State-keeping Flag 10
6.2.2. Extensive Routing Mode 10
6.3. Creating a Request 10
6.3.1. Creating a request for RPR 10
6.4. Request And Response Processing 11
6.4.1. Destination Peer: Receiving a Request And Sending a Response 11
6.4.2. Sending Peer: Receiving a Response 12
6.4.3. Relay Peer Processing 12
7. Discovery Of Relay Peer 12
8. Optional Methods to Investigate Node Connectivity 12
9. Security Considerations 13
10. IANA Considerations 13
11. Acknowledgements 13
12. References 14
12.1. Normative References 14
12.2. Informative References 14
Authors' Addresses 15

1. Introduction

1.1. Backgrounds

RELOAD [I-D.ietf-p2psip-base] recommends symmetric recursive routing (SRR) for routing messages and describes the extensions that would be required to support additional routing algorithms. Other than SRR, two other routing options: direct response routing (DRR) and relay peer routing (RPR) are also discussed in Appendix D in [I-D.ietf-p2psip-base]. DRR is specified in [I-D.zong-p2psip-drr]. As we show in section 3, RPR is advantageous over SRR in some scenarios reducing load (CPU and link BW) on intermediary peers . RPR works better in a network where relay peers are provisioned in advance so that relay peers are publicly reachable in the P2P system. In other scenarios, using a combination of RPR and SRR together is more likely to bring benefits than if SRR is used alone. Some discussion on connectivity is in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf].

Note that in this draft, we focus on RPR routing mode and its extensions to RELOAD. Some text such as modification to RELOAD message structure, optional methods to investigate node connectivity described in DRR draft [I-D.zong-p2psip-drr] are also relevent to RPR.

We first discuss the problem statement in Section 3, then how to combine RPR and SRR is presented in Section 4. In Section 5, we give comparison on the cost of SRR and RPR in both managed and open networks. An extension to RELOAD to support RPR is proposed in Section 6. Discovery of relay peers is introduced in Section 7. Some optional methods to check node connectivity is introduced in Section 8.

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 Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] draft extensively in this document. We also use terms defined in NAT behavior discovery [I-D.ietf-behave-nat-behavior-discovery]. Other terms used in this document are defined inline when used and are also defined below for reference.

There are two types of roles in the RELOAD architecture: peer and

client. Node is used when describing both peer and client. In discussions specific to behavior of a peer or client, the term peer or client is used instead.

Publicly Reachable: A node is publicly reachable if it can receive unsolicited messages from any other node in the same overlay. Note: "publicly" does not mean that the nodes must be on the public Internet, because the RELOAD protocol may be used in a closed system.

Relay Peer: A type of publicly reachable peer that can receive unsolicited messages from all other nodes in the overlay and forward the responses from destination peers towards the request sender.

Relay Peer Routing (RPR): refers to a routing mode in which responses to P2PSIP requests are sent by the destination peer to a relay peer transport address who will forward the responses towards the sending peer. For simplicity, the abbreviation RPR is used instead in the following text.

Symmetric Recursive Routing (SRR): refers to a routing mode in which responses follow the request path in the reverse order to get back to the sending peer. For simplicity, the abbreviation SRR is used instead in the following text.

3. Problem Statement

RELOAD is expected to work under a great number of application scenarios. The situations where RELOAD is to be deployed differ greatly. For instance, some deployments are global, such as a Skype-like system intended to provide public service. Some run in closed networks of small scale. SRR works in any situation, but RPR may work better in some specific scenarios.

3.1. Overview

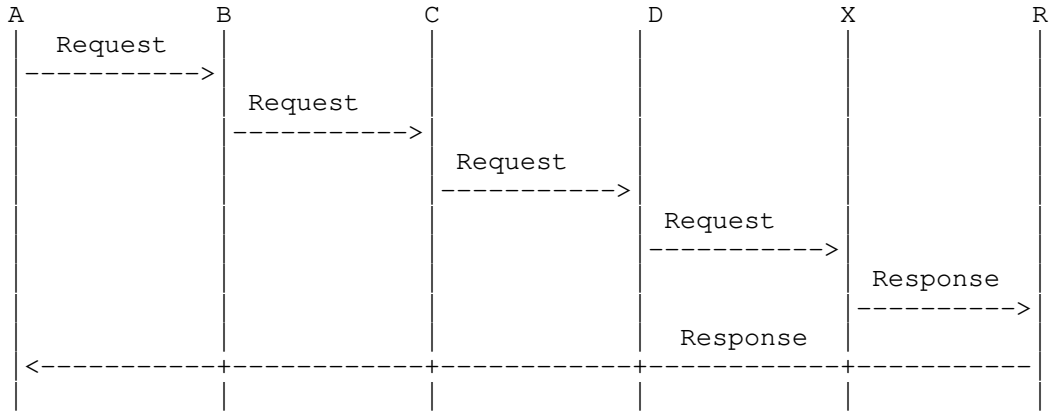
RELOAD is a simple request-response protocol. After sending a request, a node waits for a response from a destination node. There are several ways for the destination node to send a response back to the source node. In this section, we will provide detailed information on RPR.

Note that the same illustrative settings can be found in DRR draft [I-D.zong-p2psip-drr].

3.1.1. Relay Peer Routing (RPR)

If peer A knows it is behind a NAT or NATs, and knows one or more relay peers with whom they have a prior connections, peer A can try RPR. Assume A is associated with relay peer R. When sending the request, peer A includes information describing peer R transport address in the request. When peer X receives the request, peer X sends the response to peer R, which forwards it directly to Peer A on the existing connection. Note that RPR also allows a shorter route for responses compared to SRR, which means less overhead on intermediary peers. Establishing a connection to the relay with TLS requires multiple round trips. Please refer to Section 5 for cost comparison between SRR and RPR.

This technique relies on the relative population of nodes such as A that require relay peers and peers such as R that are capable of serving as a relay peers. It also requires mechanism to enable peers to know which nodes can be used as their relays. This mechanism may be based on configuration, for example as part of the overlay configuration an initial list of relay peers can be supplied. Another option is in a response to ATTACH request the peer can signal that it can be used as a relay peer.



3.2. Scenarios Where RPR Benefits

In this section, we will list several scenarios where using RPR would provide improved performance.

3.2.1. Managed or Closed P2P System

As described in Section 3.2.1, many P2P systems run in a closed or managed environment so that network administrators can better manage their system. For example, the network administrator can deploy

several relay peers which are publicly reachable in the system and indicate their presence in the configuration file. After learning where these relay peers are, peers behind NATs can use RPR with the help from these relay peers. Peers must also support SRR in case RPR fails.

Another usage is to install relay peers on the managed network boundary allowing external peers to send responses to peers inside the managed network.

3.2.2. Using Bootstrap Peers as Relay Peers

Bootstrap peers must be publicly reachable in a RELOAD architecture. As a result, one possible architecture would be to use the bootstrap peers as relay peers for use with RPR. The requirements for being a relay peer are publicly accessible and maintaining a direct connection with its client. As such, bootstrap peers are well suited to play the role of relay peers.

3.2.3. Wireless Scenarios

While some mobile deployments may use clients, in mobile networks using peers, RPR may reduce radio battery usage and bandwidth usage by the intermediary peers. The service provider may recommend in the configuration using RPR based on his knowledge of the topology. Such relay peers may also help connectivity to external networks.

4. Relationship Between SRR and RPR

4.1. How RPR Works

Peers using RPR must maintain a connection with their relay peer(s). This can be done in the same way as establishing a neighbor connection between peers by using the Attach method.

A requirement for RPR is for the source peer to convey their relay peer (or peers) transport address in the request, so the destination peer knows where the relay peer are and send the response to a relay peer first. The request should include also the requesting peer information enabling the relay peer to route the response back to the right peer.

(Editor's Note: Being a relay peer does not require that the relay peer have more functionality than an ordinary peer. As discussed later, relay peers comply with the same procedure as an ordinary peer to forward messages. The only difference is that there may be a larger traffic burden on relay peers. Relay peers can decide whether

to accept a new connection based on their current burden.)

4.2. How SRR and RPR Work Together

RPR is not intended to replace SRR. As seen from Section 3, RPR has better performance in some scenarios, but have limitations as well, see for example section 4.3 in Non-Transitive Connectivity and DHTs [http://srhea.net/papers/ntr-worlds05.pdf]. As a result, it is better to use these two modes together to adapt to each peer's specific situation. Note that the informative suggestions on how to transition between SRR and RPR (e.g. compute success rate of RPR, fall back to SRR, etc) are same with that on DRR and RPR. Please refer to DRR draft [I-D.zong-p2psip-drr] for more details. Similarly, the node can decide whether to try RPR based on other information such as configuration file information. If a relay peer is provided by the service provider, nodes may prefer RPR over SRR.

5. Comparison on cost of SRR and RPR

The major advantages in using RPR are in going through less intermediary peers on the response. By doing that it reduces the load on those peers' resources like processing and communication bandwidth.

5.1. Closed or managed networks

As described in Section 3, many P2P systems run in a closed or managed environment (e.g. carrier networks) so that network administrators would know that they could safely use RPR.

The number of hops for a response in SRR and RPR are listed in the following table. Note that the same illustrative settings can be found in DRR draft [I-D.zong-p2psip-drr].

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
RPR	Yes	2	2
RPR(DTLS)	Yes	2	7+2

From the above comparison, it is clear that:

1) In most cases of N > 4 (2^2), RPR has fewer hops than SRR. Shorter route means less overhead and resource usage on intermediary peers, which is an important consideration for adopting RPR in the cases where the resource such as CPU and BW is limited, e.g. the case of mobile, wireless network.

2) In the cases of N > 512 (2^9), RPR also has fewer messages than SRR.

3) In the cases where N < 512, RPR has more messages than SRR (but still has fewer hops than SRR). So the consideration to use RPR or SRR depends on other factors like using less resources (bandwidth and processing) from the intermediaries peers. Section 4 provides use cases where RPR has better chance to work or where the intermediary resources considerations are important.

5.2. Open networks

In open network where RPR is not guaranteed, RPR can fall back to SRR If it fails after trial, as described in Section 4. Based on the same settings in Section 5.1, the number of hops, number of messages for a response in SRR and RPR are listed in the following table.

Mode	Success	No. of Hops	No. of Msgs
SRR	Yes	logN	logN
RPR	Yes	2	2
	Fail&Fall back to SRR	2+logN	2+logN
RPR(DTLS)	Yes	2	7+2
	Fail&Fall back to SRR	2+logN	9+logN

From the above comparison, it can be observed that:

1) Trying RPR would still have a good chance of fewer hops than SRR. The detailed analysis is same as DRR case and can be found in DRR draft [I-D.zong-p2psip-drr].

2) In the cases of large network and the success rate of RPR is good, it is still possible that RPR has fewer messages than SRR. Otherwise, the consideration to use RPR or SRR depends on other factors like using less resources from the intermediaries peers.

6. Extensions to RELOAD

Adding support for RPR requires extensions to the current RELOAD protocol. In this section and in DRR[I-D.zong-p2psip-drr], we define the changes required to the protocol, including changes to message structure and to message processing.

6.1. Basic Requirements

The basic requirements to peers for supporting RPR are same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr].

6.2. Modification To RELOAD Message Structure

RELOAD provides an extensible framework to accommodate future extensions. In this section and in DRR[I-D.zong-p2psip-drr], we define a ForwardingOption structure to support RPR mode.

6.2.1. State-keeping Flag

The state-keeping flag to support RPR is same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr].

6.2.2. Extensive Routing Mode

The ForwardingOption structure to support RPR is same as DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr]. The definition of the fields is as follow:

Route mode: refers to which type of routing mode is indicated to the destination peer. Currently, only DRR (specified in DRR draft [I-D.zong-p2psip-drr]) and RPR are defined.

Transport: refers to the transport type which is used to deliver responses from the destination peer to the relay peer.

IpAddressPort: refers to the transport address that the destination peer should use to send the response to. This will be a relay peer address for RPR.

Destination: refers to the relay peer itself. If the routing mode is RPR, then the destination contains two destinations, which are the relay peer's node-id and the sending node's node-id.

6.3. Creating a Request

6.3.1. Creating a request for RPR

When using RPR for a transaction, the sending peer MUST set the IGNORE-STATE-KEEPING flag in the ForwardingHeader. Additionally, the peer MUST construct and include a ForwardingOptions structure in the ForwardingHeader. When constructing the ForwardingOption structure, the fields MUST be set as follows:

1) The type MUST be set to EXTENSIVE_ROUTING_MODE_TYPE.

2) The ExtensiveRoutingModeOption structure MUST be used for the option field within the ForwardingOptions structure. The fields MUST be defined as follows:

2.1) RouteMode set to 0x02 (RPR).

2.2) Transport set as appropriate for the relay peer.

2.3) IPAddressPort set to the transport address of the relay peer that the sender wishes the message to be relayed through.

2.4) Destination structure MUST contain two values. The first MUST be defined as type peer and set with the values for the relay peer. The second MUST be defined as type peer and set with the sending peer's own values.

6.4. Request And Response Processing

This section gives normative text for message processing after RPR is introduced. Here, we only describe the additional procedures for supporting RPR. Please refer to [I-D.ietf-p2psip-base] for RELOAD base procedures.

6.4.1. Destination Peer: Receiving a Request And Sending a Response

When the destination peer receives a request, it will check the options in the forwarding header. If the destination peer can not understand extensive_routing_mode option in the request, it MUST attempt to use SRR to return an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer.

If the routing mode is RPR, the destination peer MUST construct a Destination list for the response with two entries. The first MUST be set to the relay peer node-id from the option in the request and the second MUST be the sending node node-id from the option of the request.

In the event that the routing mode is set to RPR and there are not exactly two destinations the destination peer MUST try to send an "Error_Unknown_Extension" response (defined in Section 5.3.3.1 and Section 13.9 in [I-D.ietf-p2psip-base]) to the sending peer using SRR.

After the peer constructs the destination list for the response, it sends the response to the transport address which is indicated in the IpAddressPort field in the option using the specific transport mode in the Forwardingoption. If the destination peer receives a retransmit with SRR preference on the message it is trying to response to now, the responding peer should abort the RPR response and use SRR.

6.4.2. Sending Peer: Receiving a Response

Upon receiving a response, the peer follows the rules in [I-D.ietf-p2psip-base]. If the sender used RPR and does not get a response until the timeout, it MAY either resend the message using RPR but with a different relay peer (if available), or resend the message using SRR.

6.4.3. Relay Peer Processing

Relay peers are designed to forward responses to nodes who are not publicly reachable. For the routing of the response, this draft still uses the destination list. The only difference from SRR is that the destination list is not the reverse of the via-list, instead it is constructed from the forwarding option as described below.

When a relay peer receives a response, it MUST follow the rules in [I-D.ietf-p2psip-base]. It receives the response, validates the message, re-adjust the destination-list and forward the response to the next hop in the destination list based on the connection table. There is no added requirement for relay peer.

7. Discovery Of Relay Peer

There are several ways to distribute the information about relay peers throughout the overlay. P2P network providers can deploy some relay peers and advertise them in the configuration file. With the configuration file at hand, peers can get relay peers to try RPR. Another way is to consider relay peer as a service and then some service advertisement and discovery mechanism can also be used for discovering relay peers, for example, using the same mechanism as used in TURN server discovery in base RELOAD [I-D.ietf-p2psip-base]. Another option is to let a peer advertise his capability to be a relay in the response to ATTACH or JOIN.

Editor note: This section will be extended if we adopt RPR, but like other configuration information, there may be many ways to obtain this.

8. Optional Methods to Investigate Node Connectivity

This section is for informational purposes only for providing some mechanisms that can be used when the configuration information does not specify if RPR can be used. It summarizes some methods which can be used for a node to determine its own network location compared with NAT. These methods may help a node to decide which routing mode

it may wish to try. Note that there is no foolproof way to determine if a node is publically reachable, other than via out- of-band mechanisms. As such, peers using these mechanisms may be able to optimize traffic, but must be able to fall back to SRR routing if the other routing mechanisms fail.

For RPR to function correctly, a node may attempt to determine whether it is publicly reachable. If it is not, RPR may be chosen to route the response with the help from relay peers, or the peers should fall back to SRR. NATs and firewalls are two major contributors preventing RPR from functioning properly. There are a number of techniques by which a node can get its reflexive address on the public side of the NAT. After obtaining the reflexive address, a peer can perform further tests to learn whether the reflexive address is publicly reachable. If the address appears to be publicly reachable, the nodes to which the address belongs can be a candidate to serve as a relay peer. Nodes which are not publicly reachable may still use RPR to shorten the response path with the help from relay peers.

Some conditions are unique in P2PSIP architecture which could be leveraged to facilitate the tests. In P2P overlay network, each node only has partial a view of the whole network, and knows of a few nodes in the overlay. P2P routing algorithms can easily deliver a request from a sending node to a peer with whom the sending node has no direct connection. This makes it easy for a node to ask other nodes to send unsolicited messages back to the requester.

The approaches for a node to get the addresses needed for the further tests, as well as the test for learning whether a peer may be publicly reacheable is same as the DRR case. Please refer to DRR draft [I-D.zong-p2psip-drr] for more details.

9. Security Considerations

TBD

10. IANA Considerations

No IANA action is needed.

11. Acknowledgements

David Bryan has helped extensively with this document, and helped provide some of the text, analysis, and ideas contained here. The

authors would like to thank Ted Hardie, Narayanan Vidya, Dondeti Lakshminath and Bruce Lowekamp for their constructive comments.

12. References

12.1. 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-18 (work in progress), August 2011.

[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-03 (work in progress), October 2010.

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

[I-D.zong-p2psip-drr] Zong, N., Jiang, X., Even, R. and Zhang, Y., "An extension to RELOAD to support Direct Response Routing", draft-zong-p2psip-drr-01, September 2011.

12.2. Informative References

[ChurnDHT] Rhea, S., "Handling Churn in a DHT", Proceedings of the USENIX Annual Technical Conference. Handling Churn in a DHT, June 2004.

[DTLS] Modadugu, N., Rescorla, E., "The Design and Implementation of Datagram TLS", 11th Network and Distributed System Security Symposium (NDSS), 2004.

[I-D.ietf-behave-nat-behavior-discovery] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using STUN", draft-ietf-behave-nat-behavior-discovery-04 (work in progress), July 2008.

[I-D.ietf-behave-tcp] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", draft-ietf-behave-tcp-08 (work in progress), September 2008.

[I-D.lowekamp-mmusic-ice-tcp-framework] Lowekamp, B. and A. Roach, "A Proposal to Define Interactive Connectivity Establishment for the Transport Control Protocol (ICE-TCP) as an Extensible Framework", draft-lowekamp-mmusic-ice-tcp-framework-00 (work in progress), October 2008.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

Authors' Addresses

Ning Zong
Huawei Technologies

Email: zongning@huawei.com

Xingfeng Jiang
Huawei Technologies

Email: jiang.x.f@huawei.com

Roni Even
Huawei Technologies

Email: even.roni@huawei.com

Yunfei Zhang
China Mobile

Email: zhangyunfei@chinamobile.com