Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2010

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlisch
link-lab & FU Berlin
June 28, 2010

A RELOAD Usage for Distributed Conference Control (DisCo)
draft-knauf-p2psip-disco-00

Abstract

This document defines a RELOAD Usage for Distributed Conference Control (DisCo) with SIP. DisCo splits the semantic of identifier and locator of a SIP conference URI using a new Kind data structure. Conference members are enabled to select conference controllers based on proximity awareness. DisCo proposes call delegation to balance load at focus peers. The document addresses also aspects of security and trust, as well as compatibility for conference unaware clients.

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 30, 2010.

Copyright Notice

Table of Contents

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 and plain SIP user agents to participate in a managed P2P conference. DisCo defines the following functions:

- o A protocol scheme for distributed conference control
- o RELOAD Usage and definition of conferencing Kind
- o Mechanisms for conference synchronization and call delegation
- o Mechanisms for proximity-aware routing for conference participants
- o XML extension for the event package for conference state
- o A graduated trust delegation system

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 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 the demands for distributed conferences. The data structure stores the mapping of a single conference to multiple conference controllers and thereby separates the conference URI from focus instantiations.

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.

To provide secure mechanisms, which allows users to join or even control a distributed conference, this document describes a graduated trust delegation system. The proposed system guides implementors how to maintain privacy and trust to other peers in a distributed multiparty system.

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].

The terminology and definitions from der RELOAD base [I-D.ietf-p2psip-base], the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts] 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 can be 'active' if is already in signaling relations to conference participants. Otherwise it is 'potential' if it is only registered in a distributed conference data structure but not maintaining signaling relations yet.

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 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 owner O 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 multiparty manager for the RELOAD peer C and the plain SIP user agent E whereas focus peer B serves for RELOAD peer D and the RELOAD client F.

RELOAD peers and clients obtain the contact information for the conference from the owner O. In contrast, 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 establish 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.

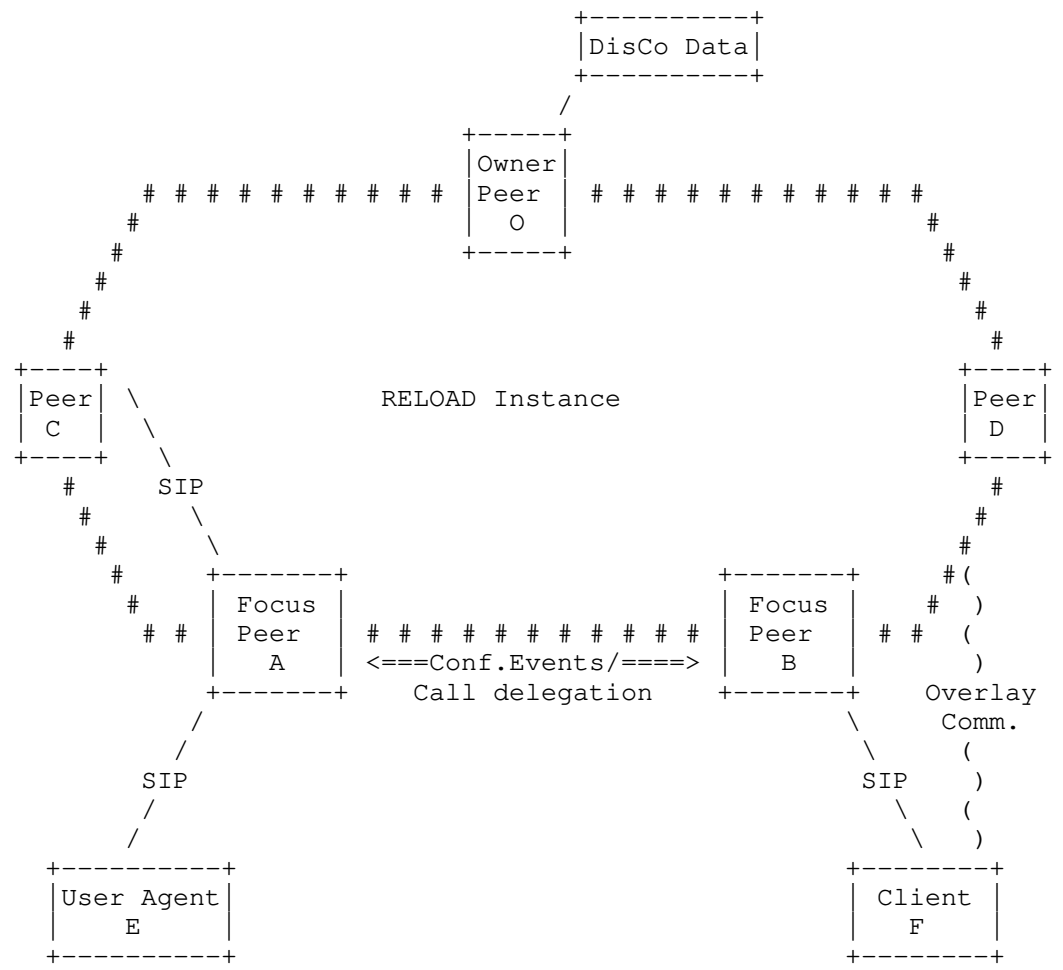


Figure 1: Reference Scenario: Focus peers A,B maintain a distributed conference

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 (Address-of-Record or Destination List) in the RELOAD overlay under DisCo-Registration Kind (cf., Figure Figure 2). The hashed conference URI is used as the Resource-ID. This data structure will later contain the contact IDs of all potential focus peers including optionally topological descriptors.

3.3. Joining a Conference

A RELOAD-aware node (cf., Bob in Figure Figure 2) intending to join an existing conference retrieves the list of potential focus peers stored in the DisCo-Registration under the conference's Resource-ID. To join the conference it selects any of the focus peers (e.g., Alice) and establishes a connection using AppAttach. 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 to contact can optionally be based on proximity information if available.

A node that is not aware of RELOAD uses common SIP signaling to retrieve the conference URI.

A conference member proposes as a focus for subsequent participants by storing a mapping of the conference URI to his Address-of-Record or Destination List in the RELOAD overlay using the conference Resource-ID. This decision should incorporate bandwidth, power, and other constraints, but details are beyond the scope of this document.

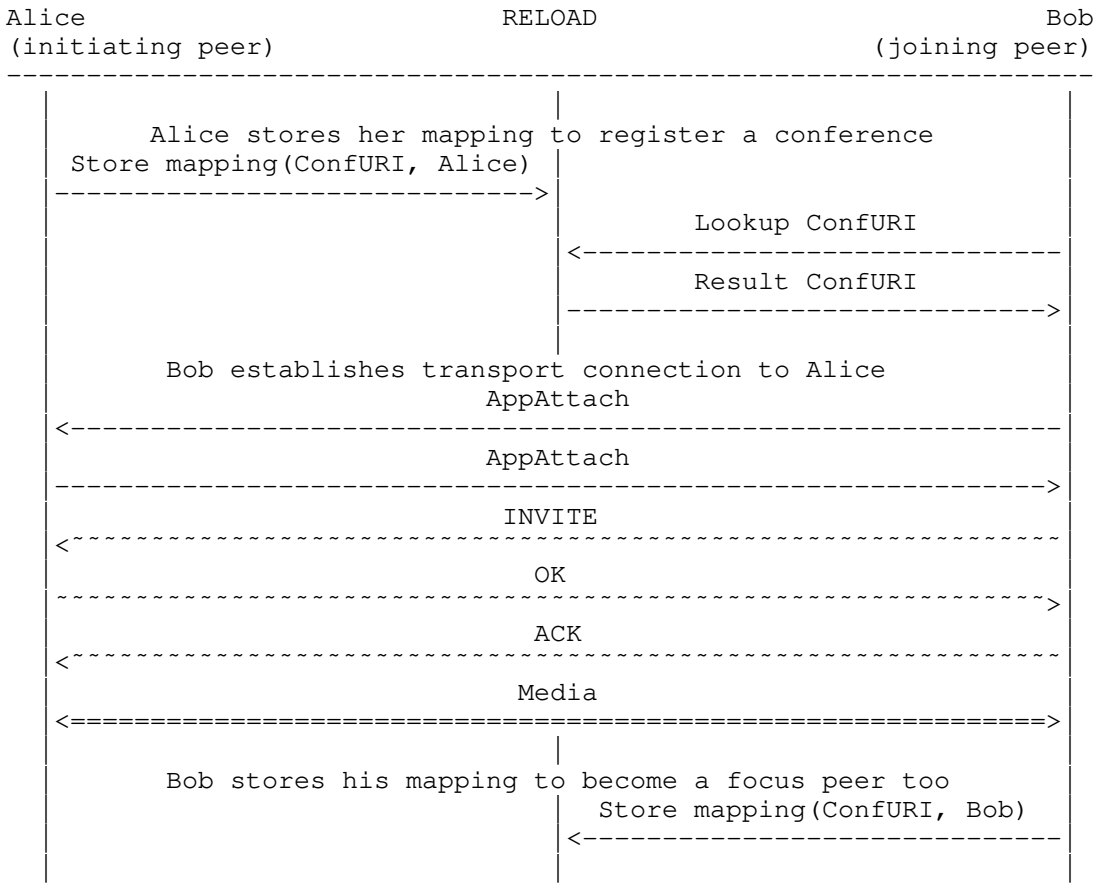


Figure 2: DisCo Usage generic Call Flow

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 conference focus peers. In DisCo, state synchronization is based on SIP specific event notifications [RFC3265].

Each focus peer can complete its view of the entire conference state among the focus peers by subscribing all other focus peers for an XML event package for distributed conferences. This is defined in this document and based on the event package for conference state [RFC4575]. Receivers of event notifications update their local conference state document to regain a valid view of current

conference state.

The event notification package for distributed conferences enables focus peers to synchronize the entire conference state. It is designed as an extension to the XML event package for conference state, which provides signaling and media parameters for each peer participating in the multiparty session. The extension defines additional XML elements and complex types (see Section 7 for more details), which allow views of the responsibilities of any focus peer in the conference. By providing these views each focus peer is enabled to perform additional load balancing operations and enhances the robustness against departures of focus peers.

3.5. Call delegation

The call delegation (seeSection 5.2.) is a feature used to transfer an incoming participation request to another focus peer. It can be applied to prevent an overloading of focus peers reaching its limit of serving new clients. Call delegation is realized through SIP REFER requests, which carry signaling and session description information of the callee to be transferred. A focus peer can decide to refer an incoming call to a less loaded remote focus. This feature is achieved transparently to 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. Resiliance

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

3.7. Topology Awareness

DisCo supports landmarking approaches based on an extension for the RELOAD XML configuration document (see Section 8) to construct topology-aware connections between focus and peers. Each peer intending to create or participate in a distributed conference SHOULD determine a topological descriptor that describes its relative position in the n-dimensional Cartesian space. Focus peers store these coordinate values as 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. Kind Data Structure

Each DisCo-Registration data structure stores the mappings for one conference to many focus peers and for each focus peer the related coordinates value. The data structure uses the RELOAD dictionary type whereas the DictionaryKey value is the Node-ID of the focus peer behind the dictionary entry. This allows a focus peer to update it mappings. The DisCo data structure of type DisCoRegistration is shown as follows:

```
enum {
  sip_focus_uri (1),
  sip_focus_node_id (2), (255)
} DisCoRegistrationtType;

struct {
  opaque coordinate<0..2^16-1>

  select (DisCoRegistrationtType.type) {
    case sip_focus_uri: opaque uri<0..2^16-1>

    case sip_focus_node_id: Destination destination_list<0..2^16-1>

    /* This type can be extended */
  }

} DisCoRegistrationData;

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

The content of the DisCoRegistrationData structure are as follows:

type
: type of the registration

length
: the length of the registration PDU

data
: the conference registration data

- If the DisCoRegistration is set to "sip_focus_uri", then it contains an Address-of-Record (AOR) as an opaque string and opaque "coordinates" string, that describes the relative network position. See more in section 4.4.

- o If registration type is set to "sip_focus_node" then it contains the Destination list for the peer and an opaque string "coordinates" describing the focus' relative network position.

The structure is designed for enabling a peer to contact a focus of the conference that is the nearest to itself. A joining peer MUST select the focus peer, which coordinate value matches at most (see section Section 4.4) to its own. In this manner it reduces the problem of triangle inequality as without this feature a joining peer could choose an inadequate remote conference controller causing large signaling and may streaming delays.

4.2. Determining Coordinates

Each RELOAD peer within the context of a distributed conference SHOULD be aware of it's relative position in the network topology. Those position information can support a topology-aware conference construction avoiding long signaling and media delays. Providing this the Usage for distributed conference foresees the coordinates value within the DisCo-Registration data structure that allows focus peers to store a topological descriptor. It is a generic field that describes a peer's relative position in the network as an N value long position vector in the N-dimensional Cartesian space. Focus peers store this coordinate value together with their announcement as conference focus. Joining peers likewise SHOULD determine their coordinates value and then select a focus peer whose relative position matches at most (see section Section 4.4).

Many algorithms determine topology information by measuring Round-Trip Times (RTT) towards a set on hosts serving as so called landmarks. To support such algorithms this document describes an extension to the RELOAD XML configuration document that allows to configure the set of Landmark hosts that peer must use for position estimation (see section Section 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 at conference join to select the focus peer that is relatively closest to itself.

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

4.3. Conference Creation

Before a peer registers to a new distributed conference, it is RECOMMENDED to ensure the initiating peer has a most up to date copy of the configuration document. In this way, the conference creator assures that all joining peers will equally determine their

coordinates value if such a alogithm is used. The first peer that creates a distributed conference registers it in the RELOAD overlay following the steps as described in Figure 3:

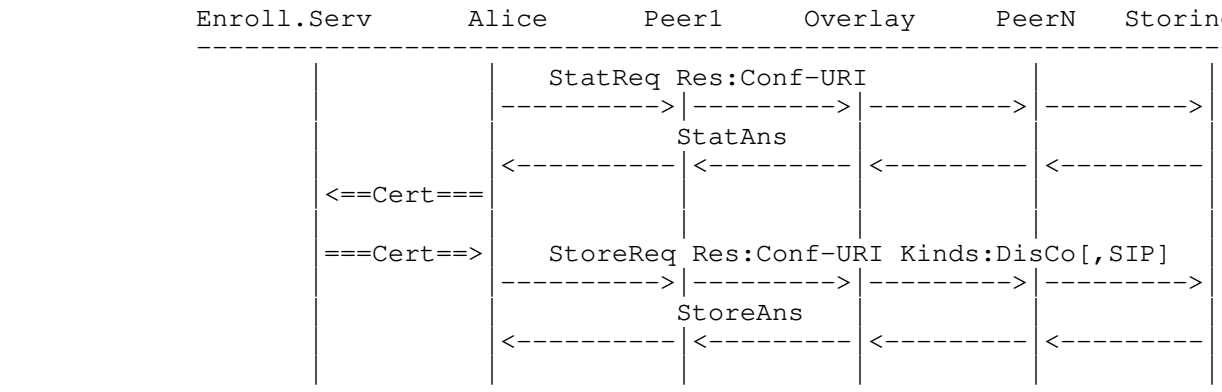


Figure 3: Creation of a Distributed Conference

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

2. The peer MUST probe whether the desired conference URI is available. It therefore generates the Resource-ID of the conference URI with the overlay hash function and sends a RELOAD StatReq towards this address. By the corresponding StatAns response the peer knows whether the desired URI is occupied by another a DisCo Kind or even a SIP-Registration Kind [I-D.ietf-p2psip-sip]. If it is, the user MUST choose another URI and repeat the availability checks. If no other DisCo or SIP-Registration Kind are stored at this Resource-ID it proceeds the registration.

3. Storing a conference registration is like to register a new virtual user that has the conference URI as its Address-of-Record. Therefore, the conference initiator MUST request the enrollment server for a new overlay certificate that contains the conference URI as user name. A sample certificate is shown below:

   User name: conference@dht.example.com
   Node-ID: 013456789abcdef
   Serial: 0815

4. The peer finally registers the DisCo data structure signed with the above certificate by a Store request towards the storing peer (the owner of the address space for the Resource-ID of the conference URI).

The additional certificate is needed for 2 major purposes:

- It separates the conference creator from the multiparty instance.
- It ensures the conference initiator's privacy. Because the DisCo data structure will be accessed by many peers using the same conference certificate. If they were using the conference creators' certificate, they were permitted to write non-shared Resources of the creator.

The conference creator MAY registers the conference URI as SIP-Registration Kind as well. In this case, it also MUST sign the Store request with the private key that matches to the certificate obtained for the conference URI. This is necessary because in the case of the departure of the conference creator, the other focus peers are permitted to redirect the mapping to another focus peer still serving the conference. The SIP-Registration SHOULD be sent in the same StoreReq as the DisCo registration.

The creator of a distributed conference MUST select on the access models as described in section Section 10.1 to define the desired privacy level of the multiparty conference.

TODO: a description how a new certificate is generated in the RELOAD instance without enrollment server

4.4. Proximity-aware Conference Participation

A RELOAD peer intending to join a distributed conference follows the steps showed in Figure 5 :

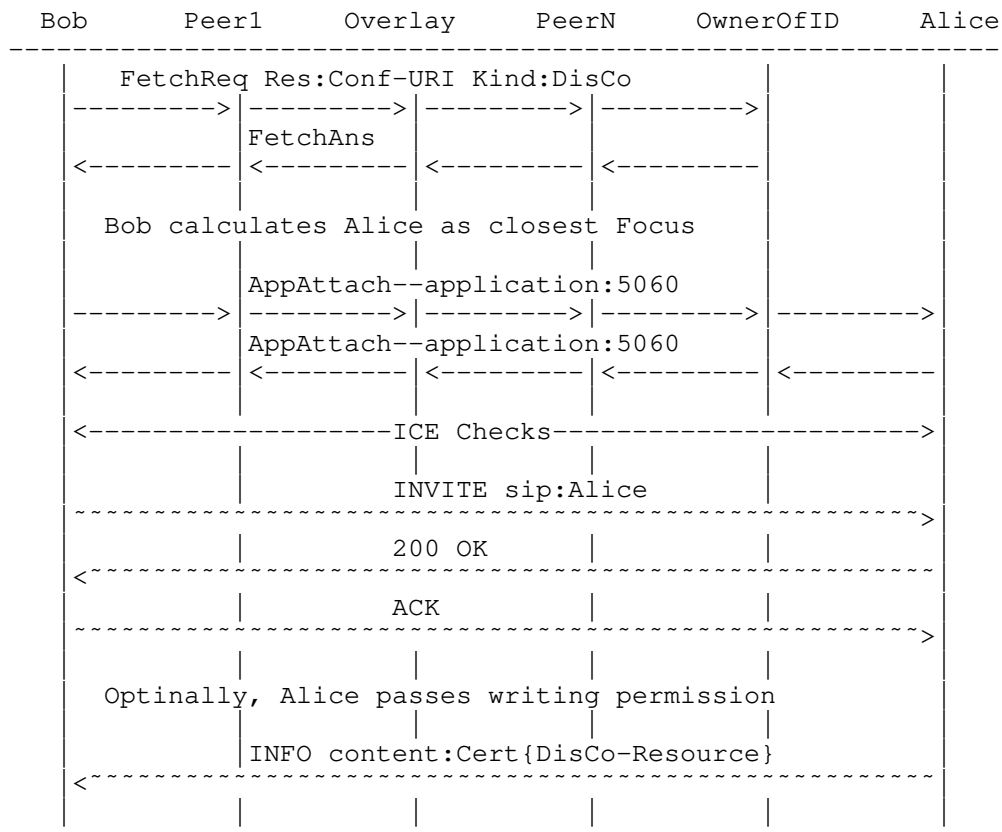


Figure 5: Participation of a Distributed Conference

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

2. The joining peer sends a FetchReq message for the DisCo Kind to the Resource-ID that corresponds to the hash over the conference URI using the overlays hash-function. The FetchReq SHOULD NOT include any specific dictionary keys thus it will receive all potential -and active focus peers of the conference.

3. Once the joining peer received the Fetch results, it calculates which of the focus peers is the relatively closest to itself by making the following calculation for each dictionary entry:

   * For each coordinate entry, calculate the each difference Di = Fi - Pi, with Fi is the coordinates vector of the Focus peer and Pi the coordinates vector of the joining peer.

* For ech Di, calculate the scalar product of Di

4. The focus with the smallest scalar product SHOULD be chosen for establishing a SIP signaling relation.

Depending on which DisCo-Registration type the selected focus has stored its mapping, the joining Peer has the following 2 possibilities:

1. If the DisCoRegistrationType is sip_focus_node_id, the joining peer uses RELOADs AppAttach request to establish a direct transport connection to the selected focus peer. The application field of the request MUST be set to 5060 indicating for SIP. This transport connection SHOULD be used to a form an ordinary SIP dialog. Further media session establishment is achieved by usual SIP mechanisms.

2. If the DisCoRegistrationType is sip_focus_uri, the joining peer MUST use the SIP-Registration [I-D.ietf-p2psip-sip] Usage to resolve the URI and form connectivity to the selected focus.

Note that in the second case a focus peer can have multiple locations for its SIP-registration. Therefore a focus MUST assure that its coordinate value corresponds to its current mapping AoR to location.

Regardless of how the focus peer has registered its mapping in the overlay a joining peer MUST add it's coordinate value base64 encoded as URI-parameter in the contact-header field of the SIP INVITE request. An example contact URI is "sip:alice@example.com;coord=YWxpY2VAZXhhbXBsZS5jb20=". The additional parameter is used by the requested focus peer as it is not capable of serving additional conference participants. It then it MUST delegate the call (see section Section 5.2) to the focus peer whose coordinate value matches next best to the coordinates of the joining peer. The focus peer therefore uses the same calculation as described in the joining process.

After the final SIP ACK request completes the signaling relation, a conference focus MAY passes the writing permission to the new participant. It therefore sends a SIP INFO request carrying the certificate for the DisCo Resource. The decision whether to pass writing permission depends on the selected security model for the distributed conference as described in section Section 10.1.

4.5. Advertising Focus Ability

All participants of a distributed conference can become a focus peer for their multiparty. The decision can depend on the capacities of

the joining peer like sufficient processing power (CPU, Memory) for the desired media type and quality of the network connectivity. Additionally, a peer intending to become focus of a conference SHOULD NOT be located behind NAT or its IP SHOULD NOT belong to the private address range. The information whether a participant is behind NAT can be obtained by ICE connectivity checks during the conference joining process.

If a participant is a candidate to become a focus of the conference it stores its mapping (Destination List or AoR) and coordinate value into the DisCo data structure. Because the DisCo Kind uses the USER-MATCH access control policy, the shared certificate passed by the participant's focus peer is sufficient to permit this peer to write the DisCo Resource. By storing the mapping into the data structure a participant becomes a potential focus.

TODO: What to do if the set of Landmark hosts changes during conference?

4.6. Resiliance in a Distributed Conference

The decentralized character of distributed conferences provide abilities to prevent the breakdown of the entire multiparty session in the case that a focus peer disappears. Two possibilities of a focus departure must be distinguished:

Friendly leave: A user whose peer is acting as conference focus decides to quit coonference participation.

Unexpected leave: Any case in which a peer serving as conference focus fails.

In the friendly case the leaving peer (lp) MUST accomplish the following procedure:

- Lp deletes its mapping in the DisCo data structure by storing the "nonexisting" value as described in the RELOAD base document [I-D.ietf-p2psip-base].

- Lp searches the conference state XML document (see section Section 7) for 'active' focus peers that have free capacities to serve further participants. Additionally, it fetches the lastest DisCo data structure for this conference to obtain all 'potential' focus peers.

- The lp then calculates for all its related participants the closest focus peer using the algorithm described in Section 4.2.

- o Based on the results from the previews step lp transfers all it's participants to their ascertained focus peers using the call delegation described in Section 5.2

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 4.4.

Assuming unfavorably circumstances it can happen that the available capacities over all potential and active focus peers are insufficient to reassemble all lost participants. In this case it RECOMMENDED to reassemble as many participants as possible in a first come first serve algorithm and to fail the rest.

5. Focus Call Control Operations

This section describes SIP call flows for third party call control for distributed conferences. Those operation comprise the call delegation and state synchronization mechanisms.

5.1. Becoming an active Focus

A conference participant that stored its mapping to the distributed DisCo data structure serves as potential focus for further participation requests by other peers. On incomming participation request a potential focus becomes an active focus and is then responsible to grant the joining peer access to the conference. Two differnt scenarios for participaton requests must be distinguished:

- A joining peer requests the potential focus
- An already active focus peer transfers a participation request to the potential focus

The second case will be dicussed as part of the call delegation in Section 5.2.

For the case that a RELOAD peer directly requests a potential focus for participation the call flow in Figure 6 describes the necessary procedure. The joing peer (JP) has already established a transport connection and sends a SIP INVITE request (1) to the contact address (IP) to its selected potential focus (PF). Note that JP is thereby unaware that PF does not serve any other participants yet. PF is participating the conference through its own active focus (AF) and is aware of the offered media types for the multiparty session. This PF offers the available media parameter to JP in (2). After finalizing the signaling in (3) and establishment of the media streams the PF in charge to synchronize the distributed conference state. As the first step of the pairwise subscrption (4) PF MUST send a SIP SUBSCRIBE [RFC3265] request to the AF that is the focus peer respoonsible for signaling with PF. It subscribes for the conference for the event package for conference state[RFC4575] with 'multi-focus' extension Section 7. This document therefore defines a new content type "application/distributed-conference-info+xml" for a MIME entity that contains conference state information for distributed conferences. After confirming the subscription (5) AF informs PF about the entire conference state by sending a 'full' XML document. It includes the list of all participants, active focus peers and used media types. In the second step for in the subscription procudure (8) AF MUST subscribe PF for distributed conference. After confirmation in (9), PF MUST inform AF about the arrival of JP and also MUST advertise its own capacities. Pf therefore sends a NOTIFY request containing a

'partial' conference state XML document that describes PF's local state (e.g. capabilities, responsibilities for JP). This step finalizes the promotion of PF to an active focus peer.

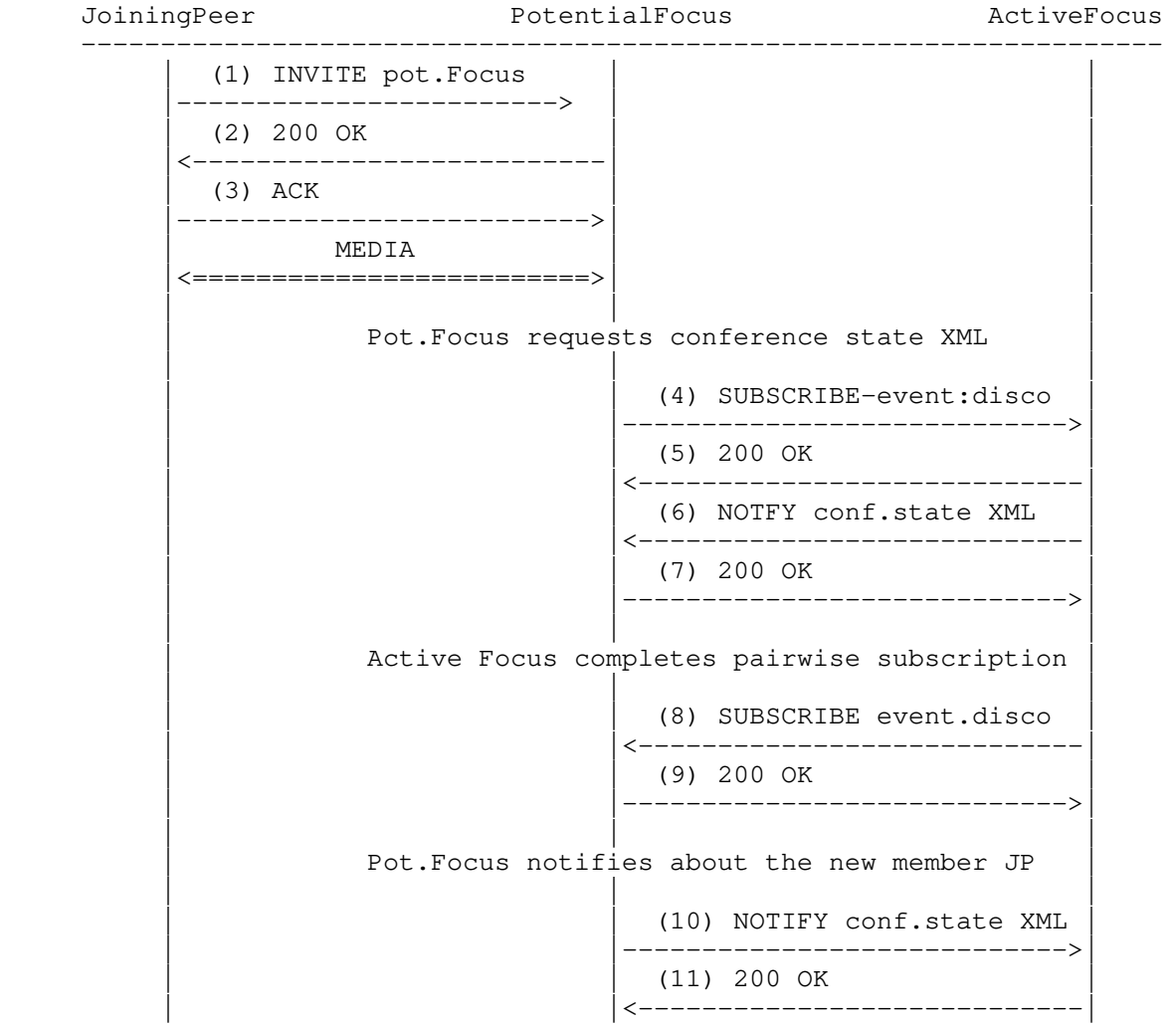


Figure 6: Pairwise subscription for conference state synchronization

Depending on whether AF has more subscriptions for the distributed conference event package it will synchronize all other focus peers sending notifications containing partial conference state XML document received from PF.

Since PF is aware of the entire conference and MAY establishes more

subscriptions to other active focus peers. This can reduce signaling delays and could serve as guideline for new routes for the media streams. A specification of how those new subscription should be done is a TODO in this document.

5.2. Delegating Calls

The call delegation feature described in this document provides focus peers the possibility to transfer an incoming participation request to another focus peer. A focus peer SHOULD delegate incoming participation requests if the number of participants it currently maintains is equal to the 'max_participants' value the focus advertised in the distributed conference XML document.

A sample scenario for call delegation in shown in Figure 7. A joining peer (JP) requests the active focus (AF) for conference participation (1). AF has no more capacity to serve JP as focus peer and has to transfer the call. AF firstly temporally accepts the call (2-3) and then selects an adequate focus peer for call delegation based on JP's coordinate value (sent base64 encoded as URI parameter in (1) see section Section 4.4). AF fetches the lastest DisCo data structure (not shown in Figure 7) to obain all available potential and active focus peers. As shown in the exmaple, AF determines the potential focus (PF) as best candidate to become JP's focus and transfers the participation request to PF sending a SIP REFER request (4). The REFER request MUST contain the session identifier from JP as payload in the request body and the call-ID of (1) as parameter in the URI of the refer-to header field, e.g., 'Refer-To: <sip:bob@dht.example;call-id=1234>'

Triggered by the REFER request PF is in charge to become JP's focus peer and to enter the conference state sychronization process. Because the call delegation operation should not interrupt JP's participation request PF MUST use the signaling and session information of (4). PF sends a re-INVITE request to JP that appears as it were orginated by AF with an additional Record-Route header field set to PF's contact address. By using this technique a distributed conference appeares as one single entity. The additional Record-Route header thereby ensures that further SIP signaling will be routed to PF. After the signaling process the negotiated media session can be established.

PF then is in charge to enter the conference state synchronization mechanism by pairwise subscribting PF->AF, AF-->PF for the event package for conference with multi-focus extension (9-14) as described Section 5.1. Note that PF could subscribe another active focus peer than AF since this is not necessarily the conference controller responsible for PF.

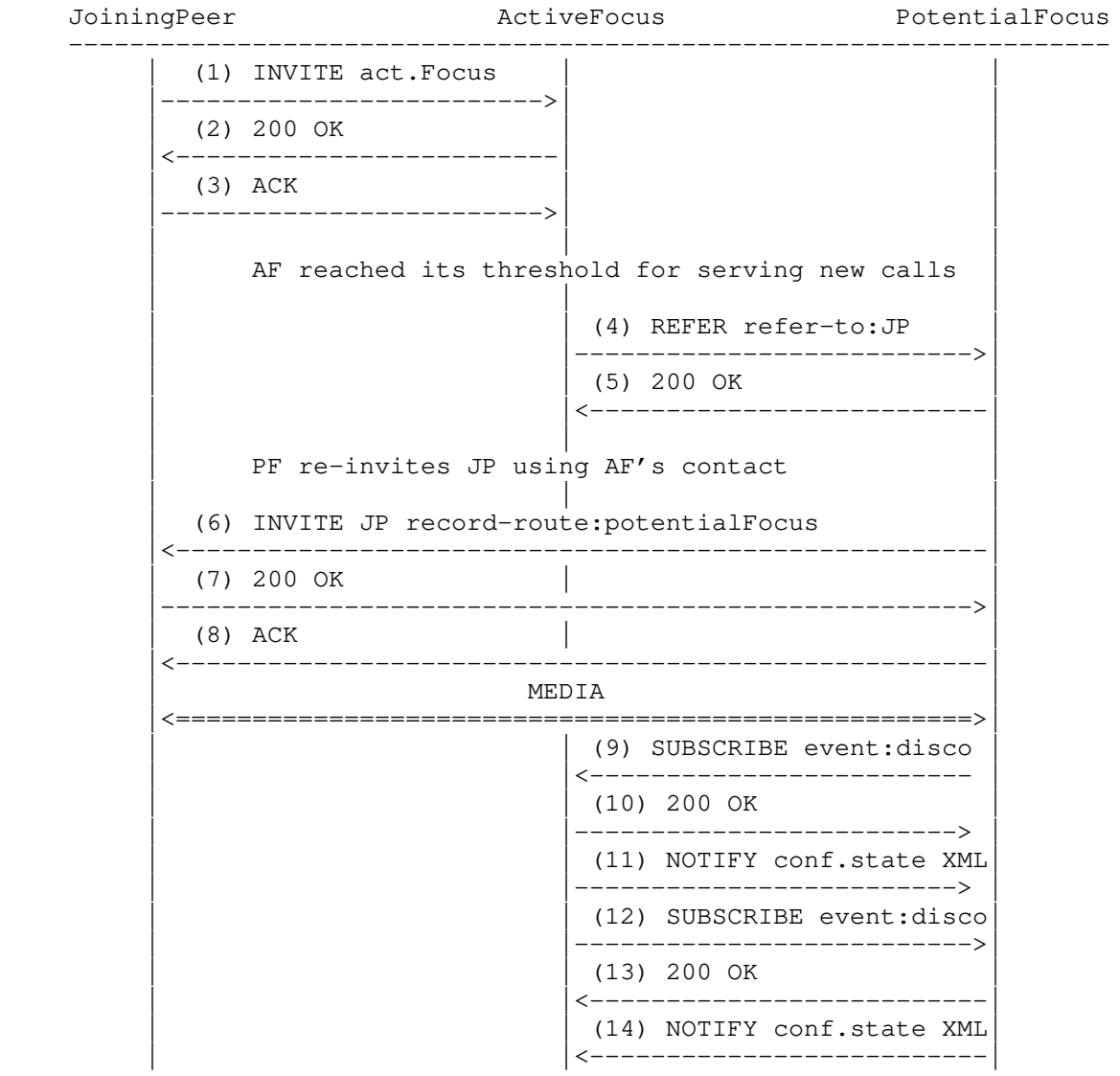


Figure 7: Call delegation to potential focus peer

5.3. Synchronizing the Conference State

The entire state of the distributed conference changes partly on every event that happens at an active focus peer. Most commenly those events refer to joins or lefts of conference participants. In order to maintain a coherent conference state all focus peers MUST send NOTIFY messages [RFC3265] to all subscribers (other focus peers)

to synchronize the conference state. The payload of the notifications MUST only contain a partial state information about the changes in the state from the previews conference state.

If the connection graph build by the pairwise subscriptions between the focus peers is not structured in a full mesh topology, state notifations MUST be forwarded by intermediate focus peers. The extension for the event package described in Section 7 therefore provides an XML element that allows every focus to reconstruct the connection graph among the focus peers.

If a state notification is received multiple times a focus peer MUST NOT forward the dublicate state information for prevent loops.

6. 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 a coordinates value describing a peers relative network position acting as focus for the conference. Additionally it contains either the peers URI or a Destination list.

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-MATCH

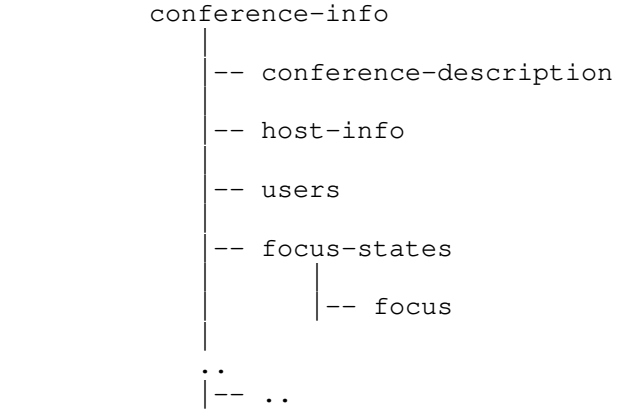
The data stored for the Kind-ID DISCO-REGISTRATION is of type DisCoRegistration. It contains a "coordinates" value, that describes the peers relative network position and
XOR one of the two following data:

sip_focus_uri
: the URI of the peer action as focus

sip_focus_node_id
: the Destination list of the peer acting as focus

7. Conference State Event Package Extension

This section presents the XML extension for the event package for conference state that enables a focus peer to have a view on the responsibilities of each other focus peer. The additional information by exentending the XML schema defined RFC4575 [RFC4575] is can be used in the case of a focus departure and call delegations. The new <focus-states> element as shown in Figure 8 is placed as child-of the root-element <conference-info>. It's child element <focus> represents every conference participant that is in the role of an active focus peer to the conference.



7.1. The <focus-states> and <focus> elements

The <focus-states> element serves as container of the <focus> sub-elements, each describing the responsibilities of a conference participant acting as focus peer.

The <focus> uses the following attributes:

entity: This attribute contains the AoR of the focus peer that is declared in the user name field in the RELOAD certificate. This AoR MUST correspond to the entity attribute defined in the focus peer's <user> element in the base conference event package. A user that whishes to lookup a focus peer's signaling information can retrieve it by looking at the corresponding <user> element with the same AoR in the entity attribute.

state: This attribute indicates whether the transmitted conference state for this focus peer is 'full', 'partial' or 'deleted' and have to be interpreted as defined in [RFC4575].

The <focus> element uses the complex focus-type that contains the

following child-elements:

<focus-capacity> : This element describes a focus peer's maximal number of participants it can serve respectively the maximal number media connections to other focus peers it can handle.

<participant> : Each participant element describes a conference member that has this focus peer as it's conference controller. It uses the 'entity' attribute that contains the AoR that this RELOAD peer uses as user name in the overlay certificate. It corresponds to the AoR in the <user> element in the base conference event XML document. Additionally, the <participants> element uses the 'state' attribute to provide the partial notification mechanism as defined in [RFC4575].

<graph> : Each <graph> element describes a conference state synchronization relation this focus peer maintains. By reference to this element each conference controller has a view of the entire synchronization topology over the focus peers. It uses the 'state' attribute as well.

7.2. XML Schema

The schema for XML extension is:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema
   targetNamespace="http://www.example.org/inet-ci-multifocus-ext"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.example.org/inet-ci-multifocus-ext"
   elementFormDefault="qualified" >
   <!--
        FOCUS STATES ELEMENT
   -->
     <xs:element name="focus-states" type="focus-states-type"/>
     <xs:complexType name="focus-states-type">
         <xs:sequence>
         <!--
             FOCUS ELEMENT
         -->
             <xs:element name="focus" type="focus-type"
             maxOccurs="unbounded" minOccurs="0"/>
             <xs:any namespace="##other" processContents="lax"/>
         </xs:sequence>
         <xs:attribute name="state" type="state-type"/>
         <xs:anyAttribute namespace="##other" processContents="lax"/>
     </xs:complexType>
     <!--
```

```
        FOCUS TYPE
    -->
    <xs:complexType name="focus-type">
        <xs:sequence>
            <xs:element name="focus-capacity"
            type="focus-capacity-type" maxOccurs="1" minOccurs="0"/>
            <xs:element name="participant" type="participant-type"
            maxOccurs="unbounded" minOccurs="0"/>
            <xs:element name="graph" type="graph-type"
            maxOccurs="unbounded" minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="entity" type="xs:anyURI"/>
        <xs:attribute name="state" type="state-type"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
        FOCUS-CAPACITY TYPE
    -->
    <xs:complexType name="focus-capacity-type">
        <xs:sequence>
            <xs:element name="max-participants" type="xs:int"
            maxOccurs="1" minOccurs="0"/>
            <xs:element name="max-focus-references" type="xs:int"
            maxOccurs="1" minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="state" type="state-type"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
        PARTICIPANT TYPE
    -->
    <xs:complexType name="participant-type">
        <xs:sequence>
            <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="entity" type="xs:anyURI"/>
        <xs:attribute name="state" type="state-type"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <!--
        GRAPH TYPE
    -->
    <xs:complexType name="graph-type">
        <xs:sequence>
            <xs:element name="ref-to-focus" type="xs:anyURI"
        maxOccurs="1" minOccurs="0"/>
```

```
                <xs:any namespace="##other" processContents="lax"/>
            </xs:sequence>
            <xs:attribute name="state" type="state-type"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:complexType>
    </xs:schema>
```

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.

8.1. The <landmark> and <Landmark-host> elements

The <landmarks> element serves as container of the <landmark-host> sub-elements each representing a single host that serves 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 responses for distance estimation.

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

8.2. Relax NG Grammar

The grammar for the Landmark configuration document extension is:

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

9. Example

TODO: Call flow examples for joining, delegating

10. Security Considerations

10.1. Layered Security

TODO: An ad hoc conference can be set up to a layered security model. Three models: open access, focus authenticate, closed access model.

10.2. Trust Aspects

TODO: Describing the privacy level for a conference instance; define whether a joining user is allowed to become a member or even focus of a conference.

11. IANA Considerations

TODO: register Kind-ID code point at the IANA

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-08 (work in progress), March 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-04 (work in progress), March 2010.

[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.

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

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

12.2. Informative References

[I-D.ietf-p2psip-concepts]
Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-03 (work in progress), July 2008.

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

Authors' Addresses

Alexander Knauf
HAW Hamburg
Berliner Tor 7
Hamburg
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 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

SAMRG
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2012

Z. Sun
H. Wang
T. Li
J. Mao
NUDT

July 12, 2011

Labelcast Protocol
draft-sunzhigang-sam-labelcast-02

Abstract

This document presents a lightweight transport protocol-Labelcast, especially for long-lived connection video streams. This protocol provides a procedure for application programs to send media data to other programs. Like the UDP protocol, the Labelcast does not provide delivery and duplicate protection. However, it provides efficient support for the monitoring of video transmission quality. And, fast forwarding mechanism based on label is also supported by the protocol. Labelcast can coexist and integrate with existing mechanisms, such as IP multicast and RTP/RTCP, while offering more flexible deployment options and scaling to support a greater number of simultaneous multicast groups.

Requirements Language

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

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 13, 2012.

Table of Contents

1. Introduction

Internet Protocol Television (IPTV) service has emerged as one of the most promising applications in the coming years. IPTV video content is delivered over IP networks and based on IP techniques. However, there is lacking efficient Internet protocols or mechanisms for IPTV services, especially between core network and access network, which is the bottleneck for IPTV data transmission.[I-D.litao-p2mpsmd-problem-statement]

Peer-to-Peer based on application layer technology just concerns the logic network rather than real network states and topology, and a great deal of redundant streams pass over the core Internet. Due to lack of administration, P2P reduces the ISPs' benefits and is not feasible to IPTV live broadcast systems. Pure clients P2P can't be further developed to the main delivery technologies.

IP multicast is insufficient to deploy largely-scale over the Internet because of their defects in scalability, user management, flow control, etc. The scalability of IP multicast is an important issue and this hampers its implementation. Routers keep every active group states and when the scale is increasing, the burden would be expanded. Another problem in IP multicast is the reliability. The UDP is used as the transport layer in IP multicast and the integrity of data will not be assured.

The transport layer protocols TCP/UDP, which are not designed for IPTV service at the beginning, can not achieve the motivation of effective video traffic transmission due to the characteristics of IPTV streams, such as long-time connection, high bandwidth consumption and so on.

TS over RTP/UDP RFC 1889 [RFC1889] are the most widely used transmission means for streaming media, however there are two problems. One problem is that RTP/UDP can not provide efficient Point-to-Multipoint IPTV distribution methods. Another is that RTP/UDP is in lack of support to monitoring the transfer quality of the video. Not only end systems are hard to realize monitoring towards their QoE (Quality of Experience), but also the intermediate routing nodes can't optimize QoS due to the lack of enough information.RFC 3550 [RFC3550]

Labelcast is a transport protocol supporting Point-to-Multipoint IPTV data distribution especially for the characteristics of long-lived connection, high bandwidth consumption and continuity. This protocol contains abundant fields that can support Point-to-Multipoint data transmission, and video quality monitoring both for intermediate routing nodes and end systems. Labelcast can efficiently meet the

requirements of the video quality transmission monitoring, failure detection and isolation of network failures.

2. Ideas of Labelcast

The basic ideas are as follows:

1. IPTV data distribution protocol based on Labelcast can support the stream characteristics of long-lived connection, high bandwidth consumption, real-time and continuity.

2. Simplify the implementation of Point-to-Multipoint data transmission by utilizing labels.

3. Reduce the processing overhead of intermediate routing nodes.

4. Support monitoring of video transmission quality.

2.1. Replacing UDP for video streaming delivery

UDP is an unconnected and unreliable transport protocol. In IPTV data transmission, only the destination port field of UDP header is functional, while the field of source port, length and checksum are not used. Labelcast defines some specific fields to support the features important to video traffic transmission, such as bandwidth and timestamp for monitoring. Labelcast sets up the transmission paths between source and receivers through label switching. The protocol supports for the packet priorities, bandwidth reservation, calculating time interval and packets loss rate, etc. Based on the state information, the video transmission quality can be monitored and the scheduling strategy can be optimized.

Besides, the evaluation of the video transmission quality at intermediate routing nodes and end systems can take advantage of MDI index RFC 4445 [RFC4445]. DF and MLR values are parts of MDI. DF value reflects the delay jitter of the video traffic and MLR value reflects the loss rates of the video traffic. The bandwidth of the transmission requirement, loss rate and the delay jitters can also be calculated by intermediate routing nodes, and the transmission QoS can be controlled.

2.2. Relationship to RTP

RTP is designed for supporting multimedia application, and it can not provide reliable insurance for sequenced data streams, nor flow or congestion control mechanism. These are all implemented by RTCP. RTCP sends periodically control messages to the group members. The

members can get the condition of network through feedback data. RTCP control flow increases linearly to the number of participants, and it consumes more bandwidth in larger groups. In this situation, Labelcast can replace UDP to work with RTP/RTCP as a transport layer protocol as it provide aboundant information for video quality monitor, failure detection and flow control . Moreover, Labelcast is responsible for monitoring network nodes (routers/switches) in the Internet, while RTP/RTCP is responsible for monitoring end hosts. Labelcast and RTP/RTCP can work together to get the complete view of the whole network.

2.3. Relationship to IP multicast

IP multicast is considered as the most effective technology to solve the bandwidth problem in IPTV data transmission. However, IP multicast is not a network-aware mechanism and can not diagnose the transmission quality. Most streams are concentrated in the minority paths (the shortest ones). In IP multicast, the packets are forwarded through fixed paths, even if there is congestion between two nodes. Besides, IP multicast can not guarantee the quality of transmission as it uses UDP protocol in transport layer.

As a supplement of the IP multicast technology, Labelcast can co-exist with IP multicast in the operational network. Routers can be configured to decide whether a packet is forwarded by IP multicast or labelcast labels, with the infomation of protocol field in IP packet. In Labelcast, the forwarding is depend on the label field, and the packets are forwarded by the label switching. Label aggregation techniques can be used to reduce the forwarding states. IP multicast address is considered as different group ID in Labelcast. If the forwarding node is not a Labelcast node, the packets is forwarded based on the layer 3 processing.

In this way we are able to combine the two multicast technologies together even though it seems that they are conflicting with each other at the first glance. Labelcast can replace IP multicast if all the intermediate nodes support it. Compare to IP multicast, Labelcast can accelerate the forwarding speed and reduce the total routing overhead among the transmitting path. In a hybrid network of IP multicast and Labelcast, for example, IP tunnel mechanism or gateway mechanism can be utilized for the connection. Two solutions will be described later in chapter 6.

2.4. Lightweight protocol

Labelcast is a lightweight protocol with two reasons. Firstly, it can be configured by utilizing Openflow technique [McKeown], and the labels are allocated through an external controller. The way to

calculate the label can be either centralized or distributed. Secondly, only a Labelcast module is added into the router, and the protocol stack in the hosts is with minor change.

3. Labelcast Protocol

3.1. Labelcast header format

Labelcast header has the length of 8 bytes with seven fields which is illustrated in figure 1.

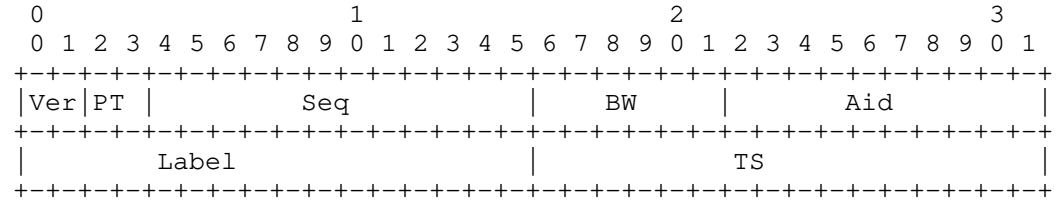


Figure 1: Head Format

(1) Ver [0:1] is the version of Labelcast protocol and it is defined "01" initially.

(2) PT [0:1] denotes the type of frame, which decides the priority to discard packets. whether the playload it carries is I frame or B frame or P frame reflect the importance of packets. I frames own the highest priority and P frames own the lowest.11: having the highest discarding priority when there is a congestion (The first to be dropped). 00: having the lowest discarding level if there is a congestion (The last to be dropped). When network congestion occurs, flow control based on priorities can be realized.

(3) Seq [0:11] denotes the sequences of packets which is used for the detection of the packets loss. It denotes the sequences of packets in the flow. It is used to monitor the packets loss and order. Suppose the sequence field is k bits, the length of packets is n bytes, and the bandwidth of the flow is Mbps, then the period of the sequence is Tseq = 2^k * n/M.

(4) BW [0:5] denotes that the average bandwidth of flow is BW*128Kbps. The maximum value is 8Mbps. BW which has the value of 0 denotes that bandwidth is unknown.

(5) Aid [0:7] denotes the application ID.

(6) Label [0:15] denotes labels related to the packets routing. The

processing actions of nodes are determined by this field. The nodes which support the label determine the next hop forwarding nodes by looking up the label table.

(7) TS [0:15] denotes the sending timestamp of packets, of which the unit is us(microsecond). It is used to account the delay between two nodes.

3.2. Forward label table

The forward label table is composed of four tuples, which include ingress port number, ingress label, egress port number and egress label. The label entry is modified by outside controller through control protocol.

In some special conditions, the tuple should be expanded. When there is a non-Labelcast node between two Labelcast nodes, it may receive two different video flows which have the same ingress port and label. However, the two different flows can not be distinguished by the above-mentioned forwarding label table. So some tuples are needed to identify flow ID, such as IP address of the source.

3.3. The Requirement for IP Protocol Fields

The value of Labelcast protocol which is identified in IP header field is 123.

4. Requirement of Protocol

4.1. Processing Requirement

4.1.1. Building the Label Paths

Similar to ATM and MPLS RFC 5332 [RFC5332], the virtual paths are established between the source and each receiver with extra means before IPTV data transmission. The extra means could be the signaling protocol of MPLS LDP RFC 3031 [RFC3031], or centralized control manners [I-D.kellil-sam-mtocp] of static calculation and configuration.

4.1.2. Requirement of the Source

(1) If source is unaware of the contents of applications, the Pri field is filled with 00 uniformly.

(2) Seq field could be filled when the packets are sent from source.

(3) BW field could be filled when the packets are sent from source.

(4) The Aid field should be filled according to the application requirements.

(5) Label field could be filled when the packets are sent from source.

(6) The TS field could be filled when the packets are sent from source.

4.1.3. Processing Requirement of Labelcast Forwarding Nodes

(1) Ver, Pri, Seq, BW and Aid can't be modified by Labelcast forwarding nodes.

(2) Label field should be modified according to the next hop distribution nodes. When packets arrive at the Label nodes, Label table is looked up at first. From the label table, the local processing actions and the next egress labels are known .

(3) According to the extra configure, TS field could be modified or keep unchanged by the forwarding nodes.

(4) The nodes which don't support Labelcast protocol can forward the packets based on IP address.

In order to prevent the initial virtual paths being deviated by IP forwarding from their inherent orientations, IP tunnels should be adopted. The packets are encapsulated with the address of next hop Labelcast nodes as the destination address, and then they can be sent to the next hop Label nodes directly.

4.1.4. Requirement of End Systems

(1) The end systems submit the payload of packets to different applications according to Aid field.

(2) The video transmission quality can be evaluated with BW, Seq, TS parameters etc.

4.2. Impact on protocol stack

4.2.1. Source serve

The communication interface will be negotiated between server and client before data transmission, Labelcast packets are identified by Aid. The session manager in Source Server addes the Labelcast module

and it can support TCP, UDP and Labelcast. Stream processor can provide RTSP/RTP/UDP/HTTP/Labelcast and it encapsulates the transport layer header with Labelcast protocol form.

4.2.2. Client

Client receives Labelcast packets with Raw Socket, it resolves Labelcast packets and sends the payload to the applications. Clients sample the video content and send the related information such as BW, Seq, TS to monitor.

4.2.3. Forwarding Node

When a intermediate Labelcast node receives the Labelcast packets, it mainly has three processes: 1. Modify the TTL options in the header and recompute the checksum of IP header. 2. Modify the timestamp of the header, and rewirte the local time. 3. look up the label table, get the next hop, and replace the label.

5. Application Example

5.1. Point-to-Multipoint Data Distribution Based on Labels

The label paths are composed of many labelcast nodes in series, and labels are assigned before the transmission of video traffic. The label tables are established according to their forwarding paths. The next hop Label nodes (one or multiple) can be determined by looking up the labels of packets in label table.

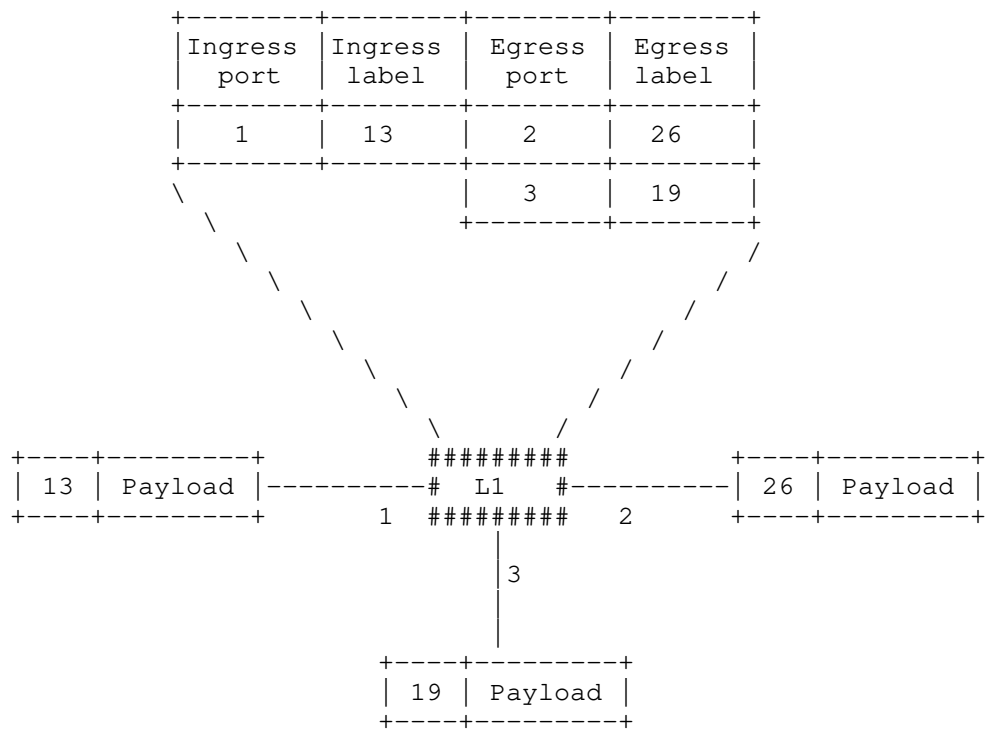


Figure 2: Label Processing

Figure 2 shows the label processing. When the packets with label 13 arrive at port 1 of Labelcast node L1, L1 looks up the lable table,then determine the forwarding ports and their corresponding new labels. Since there are two forwarding ports for packets arriving port 1 with label 13, the packets are replicated at L1, and forwarded to port 2 and port3. Before the forwarding process, the labels of packets are modified with new labels of 26 and 19 respectively.

5.2. Video-awre Network Processing

Labelcast protocol can simplify the video traffic identifications at network nodes and can guarantee application-awree QoS through the Pri field which is initilized at the source. The video transmission quality can be monitored through Bw, TS, Seq fields, and correspondingly the distribution paths are optimized by the monitoring results. For example, the arrival intervals can be calculated by the Bw and length of packets, and then the delay jitters of the successive arriving packets can be concluded. Based on this, the processing actions of forwarding nodes can be optimized.

5.3. Detecting Network Status

The characteristics of video transmission are high bandwidth-occupied, time-orderly, so they have high network requirement for Internet. For its continuity and time-orderly, video traffic itself is a good manner of network measurement, and network information is placed in the header of Labelcast protocol. Network status can be known by the Labelcast protocol. Through the analysis of TS and Seq fields, the performance of network can be acquired.

The timestamps are marked by each Label nodes passing by, and whether or not the upstream links are congested can be inferred through the delay jitters between packets. t1,t2 denote the timestamps of two packets with N intervals in previous hop Label node, and the timestamps of local Label node are t3 and t4, then formula a=(t4-t3)/(t2-t1) reflects the delay jitter of previous hop.

6. Labelcast Deployment

Labelcast requires minor modification to the underlay network which can be deployed gradually. Labelcast processing module can be added into router by ISP just like a value-added module.. Labelcast nodes can coexist with the normal IP nodes in the network, which is unlike of MPLS or IP multicast where all the routers must provide support.

6.1. Relationship to Common API

Considering a hybrid multicast network, a common multicast API [I-D.waehlisch-sam-common-api] which is suitable for transparent communication in underlay and overlay, and grants access to the different multicast flavors. Common API offers calls to transmit and receive multicast data independent of the supporting layer and the underlying technological details. So the deployment of Labelcast will have little influence for users since it is transparent to application layer if with support of common API.

6.2. IP tunnel

IP tunnel can be used in a hybrid network where IP nodes and Labelcast nodes are connected with each other. If a Labelcast node A needs to go through an IP node B to another Labelcast node C, it will firstly look up the unicast or multicast table to get the forwarding port. Then, node A encapsulates the packet with the IP header of destination C and send it to B. Node B will forward the packet to C according to the IP route table. After node C receives the packet, it will remove the encapsulated IP header.

6.3. Gateway

Gateway can be used to connect an IP multicast network with a Labelcast network. It can map a UDP header to a Labelcast header for transmitting through these two different networks.

7. Security Considerations

The security issues brought by Labelcast is what we need take into consideration.

8. IANA Considerations

The value of Labelcast protocol which is identified in IP header field is 123.

9. Acknowledgments

The authors would like to acknowledge Dilife Tchnology, Inc. for help in Labelcast implementation of server and clients during project.

10. References

10.1. Normative References

[RFC1889] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

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

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, January 2001.

10.2. Informative References

[I-D.kellil-sam-mtocp]
Kellil, M., Janneteau, C., and P. Roux, "Multiparty Transport Overlay Control Protocol(MTOCP)", draft-kellil-sam-mtocp-01 (work in progress), July 2010.

[I-D.litao-p2mpsmd-problem-statement]
Sun, Z., Li, T., Wang, H., and C. Jia, "P2MP Streaming

Media Delivery", draft-litao-p2mpsmd-problem-statement-00 (work in progress), March 2011.

[I-D.waehlisch-sam-common-api] Waehlisch, M., Schmidt, T., and S. Venaas, "A Common API for Transparent Hybrid Multicast", draft-waehlisch-sam-common-api-01 (work in progress), October 2009.

[McKeown] McKeown, N., Anderson, T., and H. Balakrishnan, "OpenFlow: enabling innovation in campus networks.", Computer Communication Review, Vol38 2008.

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

[RFC4445] Welch, J. and J. Clark, "A Proposed Media Delivery Index (MDI)", RFC 4445, April 2006.

[RFC5332] Eckert, T., Rosen, E., Aggarwal, R., and Y. Rekhter, "MPLS Multicast Encapsulations", RFC 5332, August 2008.

Authors' Addresses

Zhi Gang. Sun
NUDT
47 Yanwachi St
Changsha, Hunan 410073
China

Phone: +86-13875903721
Email: sunzhigang@nudt.edu.cn

Hui. Wang
NUDT
47 Yanwachi St
Changsha, Hunan 410073
China

Phone: +86-13467578342
Email: wanghuinudt@gmail.com

Tao. Li
NUDT
47 Yanwachi St
Changsha, Hunan 410073
China

Phone: +86-13487568531
Email: taoli.nudt@gmail.com

Jian Biao. Mao
NUDT
47 Yanwachi St
Changsha, Hunan 410073
China

Phone: +86-13618464415
Email: mjn198812@sina.com

SAM Research Group
Internet-Draft
Intended status: Informational
Expires: January 29, 2011

M. Waehlisch
link-lab & FU Berlin
T C. Schmidt
HAW Hamburg
S. Venaas
cisco Systems
July 28, 2010

A Common API for Transparent Hybrid Multicast
draft-waehlisch-sam-common-api-04

Abstract

Group communication services exist in a large variety of flavors, and technical implementations at different protocol layers. Multicast data distribution is most efficiently performed on the lowest available layer, but a heterogeneous deployment status of multicast technologies throughout the Internet requires an adaptive service binding at runtime. Today, it is difficult to write an application that runs everywhere and at the same time makes use of the most efficient multicast service available in the network. Facing robustness requirements, developers are frequently forced to using a stable, upper layer protocol controlled by the application itself. This document describes a common multicast API that is suitable for transparent communication in underlay and overlay, and grants access to the different multicast flavors. It proposes an abstract naming by multicast URIs and discusses mapping mechanisms between different namespaces and distribution technologies. Additionally, it describes the application of this API for building gateways that interconnect current multicast domains throughout the Internet.

Table of Contents

1. Introduction

Currently, group application programmers need to make the choice of the distribution technology that the application will require at runtime. There is no common communication interface that abstracts multicast transmission and subscriptions from the deployment state at runtime. The standard multicast socket options [RFC3493], [RFC3678] are bound to an IP version and do not distinguish between naming and addressing of multicast identifiers. Group communication, however, is commonly implemented in different flavors such as any source (ASM) vs. source specific mutlicast (SSM), on different layers (e.g., IP vs. application layer multicast), and may be based on different technologies on the same tier as with IPv4 vs. IPv6. It is the objective of this document to provide a universal access to group services.

Multicast application development should be decoupled of technological deployment throughout the infrastructure. It requires a common multicast API that offers calls to transmit and receive multicast data independent of the supporting layer and the underlying technological details. For inter-technology transmissions, a consistent view on multicast states is needed, as well. This document describes an abstract group communication API and core functions necessary for transparent operations. Specific implementation guidelines with respect to operating systems or programming languages are out-of-scope of this document.

In contrast to the standard multicast socket interface, the API introduced in this document abstracts naming from addressing. Using a multicast address in the current socket API predefines the corresponding routing layer. In this specification, the multicast name used for joining a group denotes an application layer data stream that is identified by a multicast URI, independent of its binding to a specific distribution technology. Such a group name can be mapped to variable routing identifiers.

The aim of this common API is twofold:

- Enable any application programmer to implement group-oriented data communication independent of the underlying delivery mechanisms. In particular, allow for a late binding of group applications to multicast technologies that makes applications efficient, but robust with respect to deployment aspects.

- Allow for a flexible namespace support in group addressing, and thereby separate naming and addressing/routing schemes from the application design. This abstraction does not only decouple programs from specific aspects of underlying protocols, but may

open application design to extend to specifically flavored group services.

Multicast technologies may be of various P2P kinds, IPv4 or IPv6 network layer multicast, or implemented by some other application service. Corresponding namespaces may be IP addresses or DNS naming, overlay hashes or other application layer group identifiers like <sip:*@peanuts.org>, but also names independently defined by the applications. Common namespaces are introduced later in this document, but follow an open concept suitable for further extensions.

This document also proposes and discusses mapping mechanisms between different namespaces and forwarding technologies. Additionally, the multicast API provides internal interfaces to access current multicast states at the host. Multiple multicast protocols may run in parallel on a single host. These protocols may interact to provide a gateway function that bridges data between different domains. The application of this API at gateways operating between current multicast instances throughout the Internet is described, as well.

1.1. Use Cases for the Common API

Four generic use cases can be identified that require an abstract common API for multicast services:

Application Programming Independent of Technologies Application programmers are provided with group primitives that remain independent of multicast technologies and its deployment in target domains. They are thus enabled to develop programs once that run in every deployment scenario. The employment of group names in the form of abstract meta data types allows applications to remain namespace-agnostic in the sense that the resolution of namespaces and name-to-address mappings may be delegated to a system service at runtime. Thereby, the complexity is minimized as developers need not care about how data is distributed in groups, while the system service can take advantage of extended information of the network environment as acquired at startup.

Global Identification of Groups Groups can be identified independent of technological instantiations and beyond deployment domains. Taking advantage of the abstract naming, an application is thus enabled to match data received from different interface technologies (e.g., IPv4, IPv6, or overlays) to belong to the same group. This not only increases flexibility, an application may for instance combine heterogeneous multipath streams, but simplifies the design and implementation of gateways and translators.

Simplified Service Deployment through Generic Gateways The API allows for an implementation of abstract gateway functions with mappings to specific technologies residing at a system level. Such generic gateways may provide a simple bridging service and facilitate an inter-domain deployment of multicast.

Mobility-agnostic Group Communication Group naming and management as foreseen in the API remain independent of locators. Naturally, applications stay unaware of any mobility-related address changes. Handover-initiated re-addressing is delegated to the mapping services at the system level and may be designed to smoothly interact with mobility management solutions provided at the network or transport layer.

2. Terminology

This document uses the terminology as defined for the multicast protocols [RFC2710],[RFC3376],[RFC3810],[RFC4601],[RFC4604]. In addition, the following terms will be used.

Group Address: A Group Address is a routing identifier. It represents a technological specifier and thus reflects the distribution technology in use. Multicast packet forwarding is based on this ID.

Group Name: A Group Name is an application identifier that is used by applications to manage communication in a multicast group (e.g., join/leave and send/receive). The Group Name does not predefine any distribution technologies, even if it syntactically corresponds to an address, but represents a logical identifier.

Multicast Namespace: A Multicast Namespace is a collection of designators (i.e., names or addresses) for groups that share a common syntax. Typical instances of namespaces are IPv4 or IPv6 multicast addresses, overlay group ids, group names defined on the application layer (e.g., SIP or Email), or some human readable strings.

Multicast Domain: A Multicast Domain hosts nodes and routers of a common, single multicast forwarding technology and is bound to a single namespace.

Interface An Interface is a forwarding instance of a distribution technology on a given node. For example, the IP interface 192.168.1.1 at an IPv4 host.

Inter-domain Multicast Gateway: An Inter-domain Multicast Gateway (IMG) is an entity that interconnects different multicast domains. Its objective is to forward data between these domains, e.g., between IP layer and overlay multicast.

3. Overview

3.1. Objectives and Reference Scenarios

The default use case addressed in this document targets at applications that participate in a group by using some common identifier taken from some common namespace. This group name is typically learned at runtime from user interaction like the selection of an IPTV channel, from dynamic session negotiations like in the Session Initiation Protocol (SIP), but may as well have been predefined for an application as a common group name. Technology-specific system functions then transparently map the group name to group addresses such that

- programmers are enabled to process group names in their programs without the need to consider technological mappings to designated deployments in target domains;

- applications are enabled to identify packets that belong to a logically named group, independent of the interface technology used for sending and receiving packets. The latter shall also hold for multicast gateways.

This document refers to a reference scenario that covers the following two hybrid deployment cases displayed in Figure 1:

1. Multicast domains running the same multicast technology but remaining isolated, possibly only connected by network layer unicast.

2. Multicast domains running different multicast technologies, but hosting nodes that are members of the same multicast group.

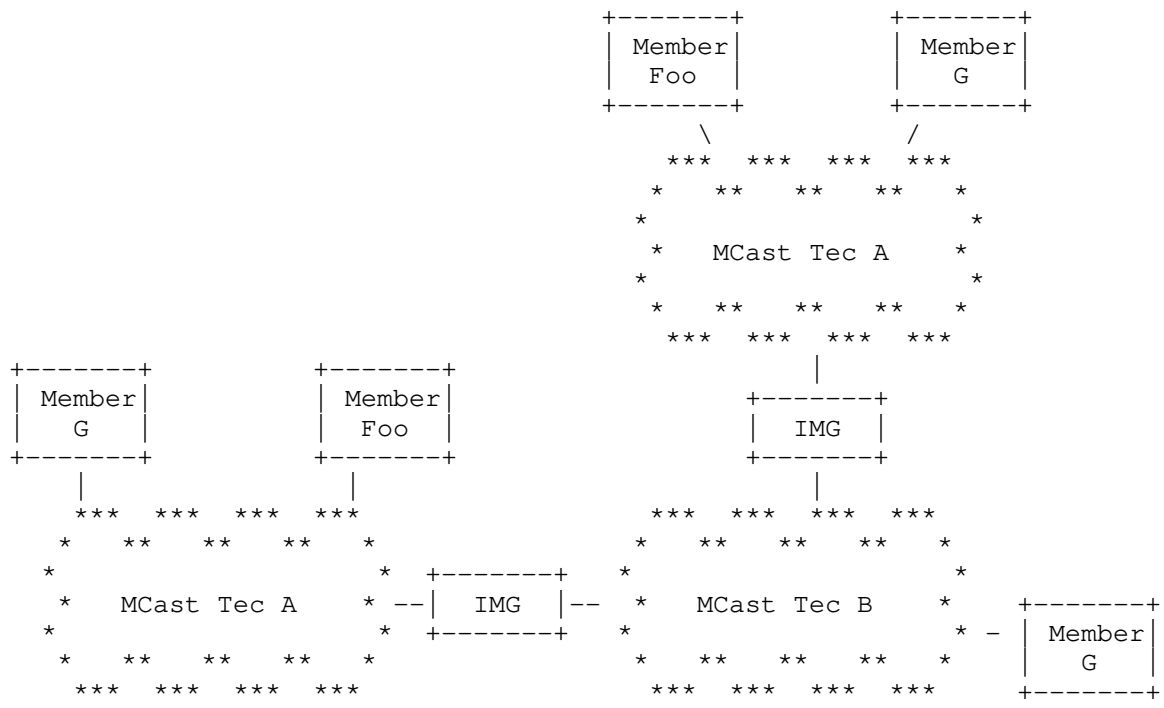


Figure 1: Reference scenarios for hybrid multicast, interconnecting group members from isolated homogeneous and heterogeneous domains.

It is assumed throughout the document that the domain composition, as well as the node attachment to a specific technology remain unchanged during a multicast session.

3.2. Group Communication API & Protocol Stack

The group communication API consists of four parts. Two parts combine the essential communication functions, while the remaining two offer optional extensions for an enhanced management:

Group Management Calls provide the minimal API to instantiate a multicast socket and manage group membership.

Send/Receive Calls provide the minimal API to send and receive multicast data in a technology-transparent fashion.

Socket Options provide extension calls for an explicit configuration of the multicast socket like setting hop limits or associated interfaces.

Service Calls provide extension calls that grant access to internal multicast states of an interface such as the multicast groups under subscription or the multicast forwarding information base.

Multicast applications that use the common API require assistance by a group communication stack. This protocol stack serves two needs:

- It provides system-level support to transfer the abstract functions of the common API, including namespace support, into protocol operations at interfaces.

- It bridges data distribution between different multicast technologies.

A general initiation of a multicast communication in this setting proceeds as follows:

1. An application opens an abstract multicast socket.

2. The application subscribes/leaves/(de)registers to a group using a logical group identifier.

3. An intrinsic function of the stack maps the logical group ID (Group Name) to a technical group ID (Group Address). This function may make use of deployment-specific knowledge such as available technologies and group address management in its domain.

4. Packet distribution proceeds to and from one or several multicast-enabled interfaces.

The multicast socket describes a group communication channel composed of one or multiple interfaces. A socket may be created without explicit interface association by the application, which leaves the choice of the underlying forwarding technology to the group communication stack. However, an application may also bind the socket to one or multiple dedicated interfaces, which predefines the forwarding technology and the namespace(s) of the Group Address(es).

Applications are not required to maintain mapping states for Group Addresses. The group communication stack accounts for the mapping of the Group Name to the Group Address(es) and vice versa. Multicast data passed to the application will be augmented by the corresponding Group Name. Multiple multicast subscriptions thus can be conducted

on a single multicast socket without the need for Group Name encoding at the application side.

Hosts may support several multicast protocols. The group communication stack discovers available multicast-enabled communication interfaces. It provides a minimal hybrid function that bridges data between different interfaces and multicast domains. Details of service discovery are out-of-scope of this document.

The extended multicast functions can be implemented by a middleware as conceptually visualized in Figure 2.

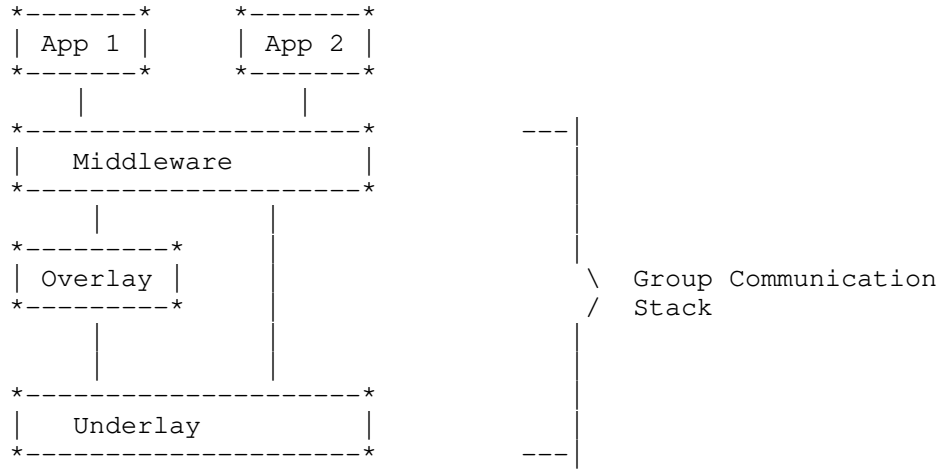


Figure 2: A middleware for offering uniform access to multicast in underlay and overlay

3.3. Naming and Addressing

Applications use Group Names to identify groups. Names can uniquely determine a group in a global communication context and hide technological deployment for data distribution from the application. In contrast, multicast forwarding operates on Group Addresses. Even though both identifiers may be identical in symbols, they carry different meanings. They may also belong to different namespaces. The namespace of a Group Address reflects a routing technology, while the namespace of a Group Name represents the context in which the application operates.

URIs [RFC3986] are a common way to represent namespace-specific identifiers in applications in the form of an abstract meta-data type. Throughout this document, any kind of Group Name follows a URI notation with the syntax defined in Section 4.1.1. Examples are,

ip://224.1.2.3:5000 for a canonical IPv4 ASM group, sip://news@cnn.com for an application-specific naming with service instantiator and default port selection.

An implementation of the group communication middleware can provide convenience functions that detect the namespace of a Group Name and use it to optimize service instantiation. In practice, such a library would provide support for high-level data types to the application, similar to the current socket API (e.g., InetAddress in Java). Using this data type could implicitly determine the namespace. Details of automatic namespace identification is out-of-scope of this document.

3.4. Mapping

All group members subscribe to the same Group Name taken from a common namespace and thereby identify the group in a technology-agnostic way.

Group Names require a mapping to addresses prior to service instantiation at an Interface. Similarly, a mapping is needed at gateways to translate between Group Addresses from different namespaces. Some namespaces facilitate a canonical transformation to default address spaces. For example, ip://224.1.2.3:5000 has an obvious correspondence to 224.1.2.3 in the IPv4 multicast address space. Note that in this example the multicast URI can be completely recovered from any data packet received from this group.

However, mapping in general can be more complex and need not be invertible. Mapping functions can be stateless in some contexts, but may require states in others. The application of such functions depends on the cardinality of the namespaces, the structure of address spaces, and possible address collisions. For example, it is not obvious how to map a large identifier space (e.g., IPv6) to a smaller, collision-prone set like IPv4.

Two (or more) Multicast Addresses from different namespaces may belong to

a. the same logical group (i.e., same Multicast Name)

b. different multicast channels (i.e., different technical IDs).

This decision can be based on invertible mappings. However, the application of such functions depends on the cardinality of the namespaces and thus does not hold in general. It is not obvious how to map a large identifier space (e.g., IPv6) to a smaller set (e.g., IPv4).

A mapping can be realized by embedding smaller in larger namespaces or selecting an arbitrary, unused ID in the target space. The relation between logical and technical ID is maintained by mapping functions which can be stateless or stateful. The middleware thus queries the mapping service first, and creates a new technical group ID only if there is no identifier available for the namespace in use. The Group Name is associated with one or more Group Addresses, which belong to different namespaces. Depending on the scope of the mapping service, it ensures a consistent use of the technical ID in a local or global domain.

4. Common Multicast API

4.1. Abstract Data Types

4.1.1. Multicast URI

Multicast Names and Multicast Addresses used in this API follow an URI scheme that defines a subset of the generic URI specified in [RFC3986] and is compliant with the guidelines in [RFC4395].

The multicast URI is defined as follows:

scheme "://" group "@" instantiation ":" port "/" sec-credentials

The parts of the URI are defined as follows:

scheme refers to the specification of the assigned identifier [RFC3986] which takes the role of the namespace.

group identifies the group uniquely within the namespace given in scheme.

instantiation identifies the entity that generates the instance of the group (e.g., a SIP domain or a source in SSM) using the namespace given in scheme.

port identifies a specific application at an instance of a group.

sec-credentials used to implement security credentials (e.g., to authorize a multicast group access).

4.1.2. Interface

The interface denotes the layer and instance on which the corresponding call will be effective. In agreement with [RFC3493] we identify an interface by an identifier, which is a positive integer

starting at 1.

Properties of an interface are stored in the following struct:

```
struct if_prop {
  unsigned int if_index; /* 1, 2, ... */
  char        *if_name;  /* "eth0", "eth1:1", "lo", ... */
  char        *if_addr;  /* "1.2.3.4", "abc123" ... */
  char        *if_tech;  /* "ip", "overlay", ... */
};
```

The following function retrieves all available interfaces from the system:

```
struct if_prop *if_prop(void);
```

It extends the functions for Interface Identification in [RFC3493] (cf., Section 4).

4.2. Group Management Calls

4.2.1. Create

The create call initiates a multicast socket and provides the application programmer with a corresponding handle. If no interfaces will be assigned based on the call, the default interface will be selected and associated with the socket. The call may return an error code in the case of failures, e.g., due to a non-operational middleware.

```
int createMSocket(uint32_t *if);
```

The if argument denotes a list of interfaces (if_indexes) that will be associated with the multicast socket. This parameter is optional.

On success a multicast socket identifier is returned, otherwise NULL.

4.2.2. Destroy

The destroy call removes the multicast socket.

```
int destroyMSocket(int s);
```

The s argument identifies the multicast socket for destruction.

On success the value 0 is returned, otherwise -1.

4.2.3. Join

The join call initiates a subscription for the given group. Depending on the interfaces that are associated with the socket, this may result in an IGMP/MLD report or overlay subscription.

```
int join(int s, const uri group_name);
```

The s argument identifies the multicast socket.

The group_name argument identifies the group.

On success the value 0 is returned, otherwise -1.

4.2.4. Leave

The leave call results in an unsubscription for the given Group Name.

```
int leave(int s, const uri group_name);
```

The s argument identifies the multicast socket.

The group_name identifies the group.

On success the value 0 is returned, otherwise -1.

4.2.5. Source Register

The srcRegister call registers a source for a Group on all active interfaces of the socket s. This call may assist group distribution in some technologies, the creation of sub-overlays, for example. Not all multicast technologies require his call.

```
int srcRegister(int s, const uri group_name,
                uint_t num_ifs, uint_t *ifs);
```

The s argument identifies the multicast socket.

The group_name argument identifies the multicast group to which a source intends to send data.

The num_ifs argument holds the number of elements in the ifs array. This parameter is optional.

The ifs argument points to the list of interface indexes for which the source registration failed. If num_ifs was 0 on output, a NULL pointer is returned. This parameter is optional.

If source registration succeeded for all interfaces associated with the socket, the value 0 is returned, otherwise -1.

4.2.6. Source Deregister

The srcDeregister indicates that a source does no longer intend to send data to the multicast group. This call may remain without effect in some multicast technologies.

```
int srcDeregister(int s, const uri group_name,
                  uint_t num_ifs, uint_t *ifs);
```

The s argument identifies the multicast socket.

The group_name argument identifies the multicast group to which a source has stopped to send multicast data.

The num_ifs argument holds the number of elements in the ifs array.

The ifs argument points to the list of interfaces for which the source deregistration failed. If num_ifs was 0 on output, a NULL pointer is returned.

If source deregistration succeeded for all interfaces associated with the socket, the value 0 is returned, otherwise -1.

4.3. Send and Receive Calls

4.3.1. Send

The send call passes multicast data for a Multicast Name from the application to the multicast socket.

```
int send(int s, const uri group_name,
         size_t msg_len, const void *buf);
```

The s argument identifies the multicast socket.

The group_name argument identifies the group to which data will be sent.

The msg_len argument holds the length of the message to be sent.

The buf argument passes the multicast data to the multicast socket.

On success the value 0 is returned, otherwise -1.

4.3.2. Receive

The receive call passes multicast data and the corresponding Group Name to the application.

```
int receive(int s, const uri group_name,
            size_t msg_len, msg *msg_buf);
```

The s argument identifies the multicast socket.

The group_name argument identifies the multicast group for which data was received.

The msg_len argument holds the length of the received message.

The msg_buf argument points to the payload of the received multicast data.

On success the value 0 is returned, otherwise -1.

4.4. Socket Options

The following calls configure an existing multicast socket.

4.4.1. Get Interfaces

The getInterface call returns an array of all available multicast communication interfaces associated with the multicast socket.

```
int getInterfaces(int s, uint_t num_ifs, uint_t *ifs);
```

The s argument identifies the multicast socket.

The num_ifs argument holds the number of interfaces in the ifs list.

The ifs argument points to an array of interface index identifiers.

On success the value 0 or lager is returned, otherwise -1.

4.4.2. Add Interface

The addInterface call adds a distribution channel to the socket. This may be an overlay or underlay interface, e.g., IPv6 or DHT. Multiple interfaces of the same technology may be associated with the socket.

```
int addInterface(int s, uint32_t if);
```

The s and if arguments identify a multicast socket and interface, respectively.

On success the value 0 is returned, otherwise -1.

4.4.3. Delete Interface

The delnterface call removes the interface if from the multicast socket.

```
int delInterface(int s, uint32_t if);
```

The s and if arguments identify a multicast socket and interface, respectively.

On success the value 0 is returned, otherwise -1.

4.4.4. Set TTL

The setTTL call configures the maximum hop count for the socket a multicast message is allowed to traverse.

```
int setTTL(int s, int h, uint_t num_ifs, uint_t *ifs);
```

The s and h arguments identify a multicast socket and the maximum hop count, respectively.

The num_ifs argument holds the number of interfaces in the ifs list. This parameter is optional.

The ifs argument points to an array of interface index identifiers. This parameter is optional.

On success the value 0 is returned, otherwise -1.

4.5. Service Calls

4.5.1. Group Set

The groupSet call returns all multicast groups registered at a given interface. This information can be provided by group management states or routing protocols. The return values distinguish between sender and listener states.

```
int groupSet(uint32_t if, uint_t *num_groups,
             struct groupSet *groupSet);

struct groupSet {
  uri group_name; /* registered multicast group */
  int type;       /* 0 = listener state, 1 = sender state,
                     2 = sender & listener state */
```

The if argument identifies the interface for which states are maintained.

The num_groups argument holds the number of groups in the groupSet array.

The groupSet argument points to an array of group states.

On success the value 0 is returned, otherwise -1.

4.5.2. Neighbor Set

The neighborSet function returns the set of neighboring nodes for a given interface as seen by the multicast routing protocol.

```
int neighborSet(uint32_t if, uint_t *num_neighbors,
                const uri *neighbor_address);
```

The if argument identifies the interface for which neighbors are inquired.

The num_neighbors argument holds the number of addresses in the neighbor_address array.

The neighbor_address argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.3. Children Set

The childrenSet function returns the set of child nodes that receive multicast data from a specified interface for a given group. For a common multicast router, this call retrieves the multicast forwarding information base per interface.

```
int childrenSet(uint32_t if, const uri group_name,
                uint_t *num_children, const uri *child_address);
```

The if argument identifies the interface for which children are inquired.

The group_name argument defines the multicast group for which distribution is considered.

The num_children argument holds the number of addresses in the child_address array.

The child_address argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.4. Parent Set

The parentSet function returns the set of neighbors from which the current node receives multicast data at a given interface for the specified group.

```
int parentSet(uint32_t if, const uri group_name, uint_t *num_parents,
              const uri *parent_address);
```

The if argument identifies the interface for which parents are inquired.

The group_name argument defines the multicast group for which distribution is considered.

The num_parents argument holds the number of addresses in the parent_address array.

The parent_address argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.5. Designated Host

The designatedHost function inquires whether the host has the role of a designated forwarder resp. querier, or not. Such an information is provided by almost all multicast protocols to prevent packet duplication, if multiple multicast instances serve on the same subnet.

```
int designatedHost(uint32_t if, const uri *group_name);
```

The if argument identifies the interface for which designated forwarding is inquired.

The group_name argument specifies the group for which the host may attain the role of designated forwarder.

The function returns 1 if the host is a designated forwarder or querier, otherwise 0. The return value -1 indicates an error.

4.5.6. Update Listener

The updateListener function is invoked to inform a group service about a change of listener states for a group. This is the result of receiver new subscriptions or leaves. The group service may call groupSet to get updated information.

```
const uri *updateListener();
```

On success the updateListener function points to the Group Name that experienced a state change, otherwise NULL is returned.

5. Functional Details

In this section, we describe specific functions of the API and the associated system middleware in detail.

5.1. Namespaces

Namespace identifiers in URIs are placed in the scheme element and characterize syntax and semantic of the group identifier. They enable the use of convenience functions and high-level data types while processing URIs. When used in names, they may facilitate a default mapping and a recovery of names from addresses. They characterize its type, when used in addresses.

Compliant to the URI concept, namespace-schemes can be added. Examples of schemes and functions currently foreseen include

IP This namespace is comprised of regular IP node naming, i.e., DNS names and addresses taken from any version of the Internet Protocol. A processor dealing with the IP namespace is required to determine the syntax (DNS name, IP address version) of the group expression.

OLM This namespace covers address strings immediately valid in an overlay network. A processor handling those strings need not be aware of the address generation mechanism, but may pass these values directly to a corresponding overlay.

SIP The SIP namespace is an example of an application-layer scheme that bears inherent group functions (conferencing). SIP conference URIs may be directly exchanged and interpreted at the application, and mapped to group addresses on the system level to generate a corresponding multicast group.

Opaque This namespace transparently carries strings without further syntactical information, meanings or associated resolution mechanism.

5.2. Mapping

Group Name to Group Address, SSM/ASM TODO

6. IANA Considerations

This document makes no request of IANA.

7. Security Considerations

This draft does neither introduce additional messages nor novel protocol operations. TODO

8. Acknowledgements

We would like to thank the HAMcast-team, Dominik Charousset, Gabriel Hege, Fabian Holler, Alexander Knauf, Sebastian Meiling, and Sebastian Woelke, at the HAW Hamburg for many fruitful discussions and for their continuous critical feedback while implementing API and a hybrid multicast middleware.

This work is partially supported by the German Federal Ministry of Education and Research within the HAMcast project, which is part of G-Lab.

9. Informative References

[I-D.ietf-mboned-auto-multicast]
Thaler, D., Talwar, M., Aggarwal, A., Vicisano, L., and T.

Pusateri, "Automatic IP Multicast Without Explicit Tunnels (AMT)", draft-ietf-mboned-auto-multicast-10 (work in progress), March 2010.

[RFC1075] Waitzman, D., Partridge, C., and S. Deering, "Distance Vector Multicast Routing Protocol", RFC 1075, November 1988.

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

[RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.

[RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3678] Thaler, D., Fenner, B., and B. Quinn, "Socket Interface Extensions for Multicast Source Filters", RFC 3678, January 2004.

[RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.

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

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.

[RFC4604] Holbrook, H., Cain, B., and B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast", RFC 4604, August 2006.

[RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano,

"Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.

Appendix A. Practical Example of the API

```
-- Application above middleware:

//Initialize multicast socket;
//the middleware selects all available interfaces
MulticastSocket m = new MulticastSocket();

m.join(URI("ip://224.1.2.3:5000"));
m.join(URI("ip://[FF02:0:0:0:0:0:0:3]:6000"));
m.join(URI("sip://news@cnn.com"));

-- Middleware:

join(URI mcAddress) {
  //Select interfaces in use
  for all this.interfaces {
    switch (interface.type) {
      case "ipv6":
        //... map logical ID to routing address
        Inet6Address rtAddressIPv6 = new Inet6Address();
        mapNametoAddress(mcAddress,rtAddressIPv6);
        interface.join(rtAddressIPv6);
      case "ipv4":
        //... map logical ID to routing address
        Inet4Address rtAddressIPv4 = new Inet4Address();
        mapNametoAddress(mcAddress,rtAddressIPv4);
        interface.join(rtAddressIPv4);
      case "sip-session":
        //... map logical ID to routing address
        SIPAddress rtAddressSIP = new SIPAddress();
        mapNametoAddress(mcAddress,rtAddressSIP);
        interface.join(rtAddressSIP);
      case "dht":
        //... map logical ID to routing address
        DHTAddress rtAddressDHT = new DHTAddress();
        mapNametoAddress(mcAddress,rtAddressDHT);
        interface.join(rtAddressDHT);
       //...
    }
  }
}
```

Appendix B. Deployment Use Cases for Hybrid Multicast

This section describes the application of the defined API to implement an IMG.

B.1. DVMRP

The following procedure describes a transparent mapping of a DVMRP-based any source multicast service to another many-to-many multicast technology.

An arbitrary DVMRP [RFC1075] router will not be informed about new receivers, but will learn about new sources immediately. The concept of DVMRP does not provide any central multicast instance. Thus, the IMG can be placed anywhere inside the multicast region, but requires a DVMRP neighbor connectivity. The group communication stack used by the IMG is enhanced by a DVMRP implementation. New sources in the underlay will be advertised based on the DVMRP flooding mechanism and received by the IMG. Based on this the updateSender() call is triggered. The relay agent initiates a corresponding join in the native network and forwards the received source data towards the overlay routing protocol. Depending on the group states, the data will be distributed to overlay peers.

DVMRP establishes source specific multicast trees. Therefore, a graft message is only visible for DVMRP routers on the path from the new receiver subnet to the source, but in general not for an IMG. To overcome this problem, data of multicast senders will be flooded in the overlay as well as in the underlay. Hence, an IMG has to initiate an all-group join to the overlay using the namespace extension of the API. Each IMG is initially required to forward the received overlay data to the underlay, independent of native multicast receivers. Subsequent prunes may limit unwanted data distribution thereafter.

B.2. PIM-SM

The following procedure describes a transparent mapping of a PIM-SM-based any source multicast service to another many-to-many multicast technology.

The Protocol Independent Multicast Sparse Mode (PIM-SM) [RFC4601] establishes rendezvous points (RP). These entities receive listener and source subscriptions of a domain. To be continuously updated, an IMG has to be co-located with a RP. Whenever PIM register messages are received, the IMG must signal internally a new multicast source using updateSender(). Subsequently, the IMG joins the group and a shared tree between the RP and the sources will be established, which

may change to a source specific tree after a sufficient number of data has been delivered. Source traffic will be forwarded to the RP based on the IMG join, even if there are no further receivers in the native multicast domain. Designated routers of a PIM-domain send receiver subscriptions towards the PIM-SM RP. The reception of such messages invokes the updateListener() call at the IMG, which initiates a join towards the overlay routing protocol. Overlay multicast data arriving at the IMG will then transparently be forwarded in the underlay network and distributed through the RP instance.

B.3. PIM-SSM

The following procedure describes a transparent mapping of a PIM-SSM-based source specific multicast service to another one-to-many multicast technology.

PIM Source Specific Multicast (PIM-SSM) is defined as part of PIM-SM and admits source specific joins (S,G) according to the source specific host group model [RFC4604]. A multicast distribution tree can be established without the assistance of a rendezvous point.

Sources are not advertised within a PIM-SSM domain. Consequently, an IMG cannot anticipate the local join inside a sender domain and deliver a priori the multicast data to the overlay instance. If an IMG of a receiver domain initiates a group subscription via the overlay routing protocol, relaying multicast data fails, as data are not available at the overlay instance. The IMG instance of the receiver domain, thus, has to locate the IMG instance of the source domain to trigger the corresponding join. In the sense of PIM-SSM, the signaling should not be flooded in underlay and overlay.

One solution could be to intercept the subscription at both, source and receiver sites: To monitor multicast receiver subscriptions (updateListener()) in the underlay, the IMG is placed on path towards the source, e.g., at a domain border router. This router intercepts join messages and extracts the unicast source address S, initializing an IMG specific join to S via regular unicast. Multicast data arriving at the IMG of the sender domain can be distributed via the overlay. Discovering the IMG of a multicast sender domain may be implemented analogously to AMT [I-D.ietf-mboned-auto-multicast] by anycast. Consequently, the source address S of the group (S,G) should be built based on an anycast prefix. The corresponding IMG anycast address for a source domain is then derived from the prefix of S.

B.4. BIDIR-PIM

The following procedure describes a transparent mapping of a BIDIR-PIM-based any source multicast service to another many-to-many multicast technology.

Bidirectional PIM [RFC5015] is a variant of PIM-SM. In contrast to PIM-SM, the protocol pre-establishes bidirectional shared trees per group, connecting multicast sources and receivers. The rendezvous points are virtualized in BIDIR-PIM as an address to identify on-tree directions (up and down). However, routers with the best link towards the (virtualized) rendezvous point address are selected as designated forwarders for a link-local domain and represent the actual distribution tree. The IMG is to be placed at the RP-link, where the rendezvous point address is located. As source data in either cases will be transmitted to the rendezvous point address, the BIDIR-PIM instance of the IMG receives the data and can internally signal new senders towards the stack via updateSender(). The first receiver subscription for a new group within a BIDIR-PIM domain needs to be transmitted to the RP to establish the first branching point. Using the updateListener() invocation, an IMG will thereby be informed about group requests from its domain, which are then delegated to the overlay.

Appendix C. Change Log

The following changes have been made from draft-waehlisch-sam-common-api-03

1. Use cases added for illustration.

2. Service calls added for inquiring on the multicast distribution system.

3. Namespace examples added.

4. Clarifications and editorial improvements.

The following changes have been made from draft-waehlisch-sam-common-api-02

1. Rename init() in createMSocket().

2. Added calls srcRegister()/srcDeregister().

3. Rephrased API calls in C-style.

4. Cleanup code in "Practical Example of the API".

5. Partial reorganization of the document.

6. Many editorial improvements.

The following changes have been made from draft-waehlisch-sam-common-api-01

1. Document restructured to clarify the realm of document overview and specific contributions s.a. naming and addressing.

2. A clear separation of naming and addressing was drawn. Multicast URIs have been introduced.

3. Clarified and adapted the API calls.

4. Introduced Socket Option calls.

5. Deployment use cases moved to an appendix.

6. Simple programming example added.

7. Many editorial improvements.

Authors' Addresses

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

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

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

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

Stig Venaas
cisco Systems
Tasman Drive
San Jose, CA 95134
USA

Email: stig@cisco.com