

SAM Research Group
Internet-Draft
Intended status: Informational
Expires: January 27, 2012

M. Waehlich
link-lab & FU Berlin
T C. Schmidt
HAW Hamburg
S. Venaas
cisco Systems
July 26, 2011

A Common API for Transparent Hybrid Multicast
draft-irtf-samrg-common-api-03

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 use a stable, upper layer protocol provided 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.

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 27, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Use Cases for the Common API	5
2. Terminology	6
3. Overview	7
3.1. Objectives and Reference Scenarios	7
3.2. Group Communication API & Protocol Stack	8
3.3. Naming and Addressing	10
3.4. Namespaces	11
3.4.1. Generic Namespaces	11
3.4.2. Application-centric Namespaces	12
3.5. Name-to-Address Mapping	12
3.5.1. Canonical Mapping	13
3.5.2. Mapping at End Points	13
3.5.3. Mapping at Interdomain Multicast Gateways	13
4. Common Multicast API	13
4.1. Notation	13
4.2. Abstract Data Types	13
4.2.1. Multicast URI	14
4.2.2. Interface	14
4.2.3. Membership Events	15
4.3. Group Management Calls	15
4.3.1. Create	15
4.3.2. Delete	16
4.3.3. Join	16
4.3.4. Leave	16
4.3.5. Source Register	17
4.3.6. Source Deregister	17
4.4. Send and Receive Calls	17
4.4.1. Send	18
4.4.2. Receive	18

4.5. Socket Options	18
4.5.1. Get Interfaces	19
4.5.2. Add Interface	19
4.5.3. Delete Interface	19
4.5.4. Set TTL	19
4.5.5. Get TTL	20
4.6. Service Calls	20
4.6.1. Group Set	20
4.6.2. Neighbor Set	21
4.6.3. Children Set	21
4.6.4. Parent Set	21
4.6.5. Designated Host	22
4.6.6. Enable Membership Events	22
4.6.7. Disable Membership Events	23
5. IANA Considerations	23
6. Security Considerations	23
7. Acknowledgements	23
8. Informative References	23
Appendix A. C Signatures	25
Appendix B. Practical Example of the API	27
Appendix C. Deployment Use Cases for Hybrid Multicast	28
C.1. DVMRP	29
C.2. PIM-SM	29
C.3. PIM-SSM	30
C.4. BIDIR-PIM	30
Appendix D. Change Log	31
Authors' Addresses	33

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, nor has been the use of DNS for group addresses established. The standard multicast socket options [RFC3493], [RFC3678] are bound to an IP version by not distinguishing between naming and addressing of multicast identifiers. Group communication, however, is commonly implemented in different flavors such as any source (ASM) vs. source specific multicast (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 for programmers 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:

- o 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.
- o Allow for a flexible namespace support in group addressing, and thereby separate naming and addressing resp. 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 peer-to-peer 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 discusses mapping mechanisms between different namespaces and forwarding technologies and proposes expressions of defaults for an intended binding. 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

The following 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 their deployment in target domains. They are thus enabled to develop programs once that run in every deployment scenario. The use 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 also

simplifies the design and implementation of gateways and translators.

Uniform Access to Multicast Flavors: The URI naming scheme uniformly supports different flavors of group communication such as any source and source specific multicast, selective broadcast etc., independent of their service instantiation. The traditional SSM model for instance can experience manifold support, either by directly mapping the multicast URI (i.e., "group@instantiation") to an (S,G) state on the IP layer, or by first resolving S for a subsequent group address query, or by transferring this process to any of the various source specific overlay schemes, or by delegating to a plain replication server. The application programmer can invoke any of these underlying mechanisms with the same line of code.

Simplified Service Deployment through Generic Gateways: The common multicast 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 common multicast 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 (see [RFC5757] for mobility-related aspects).

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 address.

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.

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.

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

Inter-domain Multicast Gateway (IMG): 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 an 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

- o programmers are enabled to process group names in their programs without the need to consider technological mappings to designated deployments in target domains;
- o 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 considers two reference scenarios that cover the following 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.

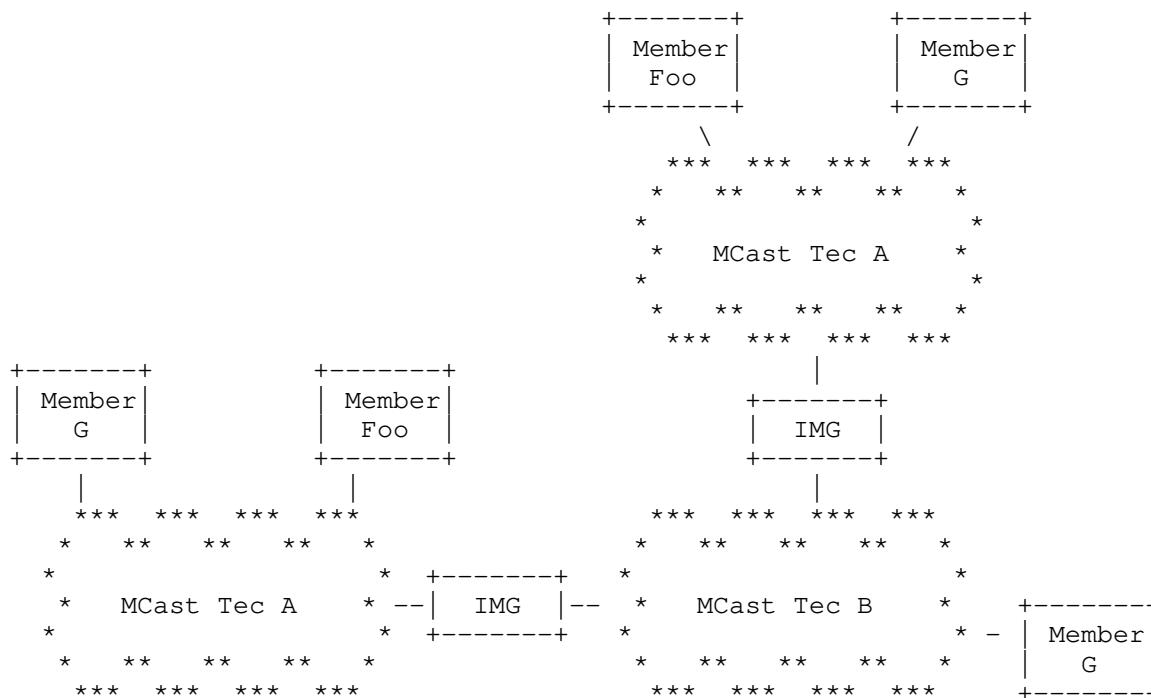


Figure 1: Reference scenarios for hybrid multicast, interconnecting group members from isolated homogeneous and heterogeneous domains.

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 monitoring and management:

Group Management Calls provide the minimal API to instantiate a multicast socket and to 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 such as 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:

- o It provides system-level support to transfer the abstract functions of the common API, including namespace support, into protocol operations at Interfaces.
- o It group communication services across different multicast technologies at the local host.

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 Group Name.
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 abstract 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 Multicast 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 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.

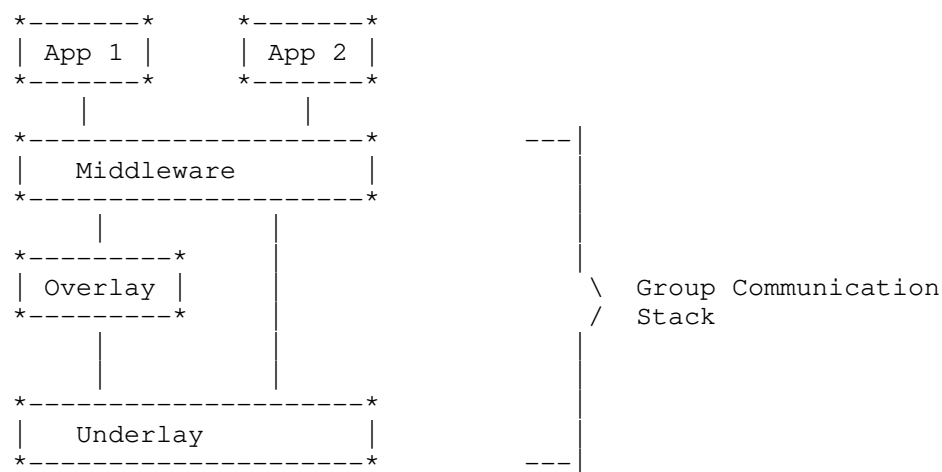


Figure 2: A middleware for offering uniform access to multicast in underlay and overlay

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 Multicast 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, all Group Names follows a URI notation with the syntax defined in Section 4.2.1. 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 stack can provide convenience functions that detect the Namespace of a Group Name or further optimize service instantiation. In practice, such a library would provide support for high-level data types to the application, similar to some versions of the current socket API (e.g., `InetAddress` in Java). Using this data type could implicitly determine the Namespace. Details of automatic Namespace identification or service handling are out of scope of this document.

3.4. 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 indicate an application context, or 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 are generic or inherited from applications.

3.4.1. Generic Namespaces

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.

SHA-2 This namespace carries address strings compliant to SHA-2 hash digests. A processor handling those strings is required to determine the length of the group expression and passes appropriate values directly to a corresponding overlay.

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

3.4.2. Application-centric Namespaces

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.

RELOAD This namespace covers address strings immediately valid in a RELOAD [I-D.ietf-p2psip-base] overlay network. A processor handling those strings may pass these values directly to a corresponding overlay.

3.5. Name-to-Address Mapping

The multicast communication paradigm requires all group members to subscribe to the same Group Name, taken from a common Multicast Namespace, and thereby to identify the group in a technology-agnostic way. Following this common API, a sender correspondingly registers a Group Name prior to transmission.

At communication end points, Group Names require a mapping to Group Addresses prior to service instantiation at its Interface(s). Similarly, a mapping is needed at gateways to translate between Group Addresses from different namespaces consistently. Two requirements need to be met by a mapping function that translates between Multicast Names and Addresses.

- a. For a given Group Name, identify an Address that is appropriate for a local distribution instance.
- b. For a given Group Address, invert the mapping to recover the Group Name.

In general, mapping can be complex and need not be invertible. A mapping can be realized by embedding smaller in larger namespaces or selecting an arbitrary, unused ID in a smaller target namespace. For example, it is not obvious how to map a large identifier space (e.g., IPv6) to a smaller, collision-prone set like IPv4 (see [I-D.venaas-behave-v4v6mc-framework][I-D.venaas-behave-mcast46]). 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. However, some namespaces facilitate a canonical, invertible transformation to default address spaces.

3.5.1. Canonical Mapping

Some Multicast Namespaces defined in Section 3.4 can express a canonical default mapping. For example, `ip://224.1.2.3:5000` indicates the correspondence to 224.1.2.3 in the default IPv4 multicast address space at port 5000. This default mapping is bound to a technology and may not always be applicable, e.g., in the case of address collisions. Note that under canonical mapping, the multicast URI can be completely recovered from any data message received from this group.

3.5.2. Mapping at End Points

Multicast listeners or senders require a Name-to-Address conversion for all technologies they actively run in a group. Even though a mapping applies to the local Multicast Domain only, end points may need to learn a valid Group Address from neighboring nodes, e.g., from a gateway in the collision-prone IPv4 domain. Once set, an end point will always be aware of the Name-to-Address correspondence and thus can autonomously invert the mapping.

3.5.3. Mapping at Interdomain Multicast Gateways

Multicast data may arrive at an IMG in one technology, requesting the gateway to re-address packets for another distribution system. At initial arrival, the IMG may not have explicit knowledge of the corresponding Multicast Group Name. To perform a consistent mapping, the group name potentially needs to be acquired out of band from a neighboring node.

4. Common Multicast API

4.1. Notation

The following description of the common multicast API is described in pseudo syntax. Variables that are passed to function calls are declared by "in", return values are declared by "out". A list of elements is denoted by <>. The pseudo syntax assumes that lists include an attribute which represents the number of elements.

The corresponding C signatures are defined in Appendix A.

4.2. Abstract Data Types

4.2.1. 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 Multicast 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 syntax and semantic as defined by the Namespace given in `scheme`. This parameter is optional. Note that ambiguities (e.g., identical node addresses in multiple overlay instances) can be distinguished by ports.

`port` identifies a specific application at an instance of a group. This parameter is optional.

`sec-credentials` used to implement optional security credentials (e.g., to authorize a multicast group access). Note that security credentials carry a distinct technical meaning w.r.t. AAA schemes and may differ between group members. Hence the `sec-credentials` are not considered part of the Group Name.

4.2.2. 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:

```
getInterfaces(out Interface <ifs>);
```

It extends the functions for Interface Identification defined in Section 4 of [RFC3493] and can be implemented by:

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

4.2.3. Membership Events

A membership event is triggered by a multicast state change, which is observed by the current node. It is related to a specific Group Name and may be receiver or source oriented.

```
event_type {
    join_event;
    leave_event;
    new_source_event;
};

event {
    event_type event;
    Uri group_name;
    Interface if;
};
```

An event will be created by the group communication stack and passed to applications that have registered for events.

4.3. Group Management Calls

4.3.1. 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 returns an error code in the case of failures, e.g., due to a non-operational middleware.

```
createMSocket(in Interface <ifs>,  
              out Socket s);
```

The ifs 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.

4.3.2. Delete

The delete call removes the multicast socket.

```
deleteMSocket(in Socket s, out Int error);
```

The s argument identifies the multicast socket for destruction.

On success the out parameter error is 0, otherwise -1.

4.3.3. 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, for example.

```
join(in Socket s, in Uri groupName, out Int error);
```

The s argument identifies the multicast socket.

The groupName argument identifies the group.

On success the out parameter error is 0, otherwise -1.

4.3.4. Leave

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

```
leave(in Socket s, in Uri groupName, out Int error);
```

The s argument identifies the multicast socket.

The groupName identifies the group.

On success the out parameter error is 0, otherwise -1.

4.3.5. 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, for example the creation of sub-overlays. It may remain without effect in some multicast technologies.

```
srcRegister(in Socket s, in Uri groupName,  
            in Interface <ifs>, out Int error);
```

The `s` argument identifies the multicast socket.

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

The `ifs` argument points to the list of Interface indexes for which the source registration failed. A NULL pointer is returned, if the list is empty. This parameter is optional.

If source registration succeeded for all Interfaces associated with the socket, the out parameter `error` is 0, otherwise -1.

4.3.6. 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.

```
srcDeregister(in Socket s, in Uri groupName,  
              in Interface <ifs>, out Int error);
```

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 `ifs` argument points to the list of Interfaces for which the source deregistration failed. A NULL pointer is returned, if the list is empty.

If source deregistration succeeded for all Interfaces associated with the socket, the out parameter `error` is 0, otherwise -1.

4.4. Send and Receive Calls

4.4.1. Send

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

```
send(in Socket s, in Uri groupName,  
     in Size msgLen, in Msg msgBuf,  
     out Int error);
```

The s argument identifies the multicast socket.

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

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

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

On success the out parameter error is 0, otherwise -1.

4.4.2. Receive

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

```
receive(in Socket s, out Uri groupName,  
        out Size msgLen, out Msg msgBuf,  
        out Int error);
```

The s argument identifies the multicast socket.

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

The msgLen argument holds the length of the received message.

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

On success the out parameter error is 0, otherwise -1.

4.5. Socket Options

The following calls configure an existing multicast socket.

4.5.1. Get Interfaces

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

```
getInterfaces(in Socket s,  
             out Interface <ifs>, out Int error);
```

The `s` argument identifies the multicast socket.

The `ifs` argument points to an array of Interface index identifiers.

On success the out parameter `error` is 0, otherwise -1.

4.5.2. 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.

```
addInterface(in Socket s, in Interface if,  
            out Int error);
```

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

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

4.5.3. Delete Interface

The `delInterface` call removes the Interface `if` from the multicast socket.

```
delInterface(in Socket s, Interface if,  
            out Int error);
```

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

On success the out parameter `error` is 0, otherwise -1.

4.5.4. Set TTL

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

```
setTTL(in Socket s, in Int h,  
      in Interface <ifs>,  
      out Int error);
```

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

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

On success the out parameter error is 0, otherwise -1.

4.5.5. Get TTL

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

```
getTTL(in Socket s,  
      out Int h, out Int error);
```

The s argument identifies a multicast socket.

The h argument holds the maximum number of hops associated with socket s.

On success the out parameter error is 0, otherwise -1.

4.6. Service Calls

4.6.1. 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.

```
struct GroupSet {  
    uri groupName; /* registered multicast group */  
    int type;       /* 0 = listener state, 1 = sender state,  
                   2 = sender & listener state */  
}  
  
groupSet(in Interface if,  
      out GroupSet <groupSet>, out Int error);
```

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

The groupSet argument points to a list of group states.

On success the out parameter error is 0, otherwise -1.

4.6.2. Neighbor Set

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

```
neighborSet(in Interface if,  
            out Uri <neighborsAddresses>, out Int error);
```

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

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

On success the out parameter error is 0, otherwise -1.

4.6.3. 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.

```
childrenSet(in Interface if, in Uri groupName,  
            out Uri <childrenAddresses>, out Int error);
```

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

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

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

On success the out parameter error is 0, otherwise -1.

4.6.4. 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.

```
parentSet(in Interface if, in Uri groupName,  
          out Uri <parentsAddresses>, out Int error);
```

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

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

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

On success the out parameter error is 0, otherwise -1.

4.6.5. Designated Host

The designatedHost function inquires whether this 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.

```
designatedHost(in Interface if, in Uri groupName  
              out Int return);
```

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

The groupName 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.

4.6.6. Enable Membership Events

The enableEvents function registers an application at the group communication stack to receive information about group changes. State changes are the result of new receiver subscriptions or leaves as well as of source changes. Upon receiving an event, the group service may obtain additional information from further service calls.

```
enableEvents();
```

Calling this function, the stack starts to pass membership events to the application. Each event includes an event type identifier and a Group Name (cf., Section 4.2.3).

4.6.7. Disable Membership Events

The `disableEvents` function deactivates the information about group state changes.

```
disableEvents();
```

On success the stack will not pass membership events to the application.

5. IANA Considerations

This document makes no request of IANA.

6. Security Considerations

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

7. 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 the common multicast 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.

8. Informative References

[I-D.ietf-mboned-auto-multicast]

Thaler, D., Talwar, M., Aggarwal, A., Vicisano, L., Pusateri, T., and T. Morin, "Automatic IP Multicast Tunneling", draft-ietf-mboned-auto-multicast-11 (work in progress), July 2011.

[I-D.ietf-p2psip-base]

Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-17 (work in progress), July 2011.

- [I-D.venaas-behave-mcast46]
Venaas, S., Asaeda, H., SUZUKI, S., and T. Fujisaki, "An IPv4 - IPv6 multicast translator", draft-venaas-behave-mcast46-02 (work in progress), December 2010.
- [I-D.venaas-behave-v4v6mc-framework]
Venaas, S., Li, X., and C. Bao, "Framework for IPv4/IPv6 Multicast Translation", draft-venaas-behave-v4v6mc-framework-03 (work in progress), June 2011.
- [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.
- [RFC5757] Schmidt, T., Waehlich, M., and G. Fairhurst, "Multicast Mobility in Mobile IP Version 6 (MIPv6): Problem Statement and Brief Survey", RFC 5757, February 2010.

Appendix A. C Signatures

This section describes the C signatures of the common multicast API, which are defined in Section 4.

```
int createMSocket(int* result, size_t num_ifs, const uint32_t* ifs);

int deleteMSocket(int s);

int join(int msock, const char* group_uri);

int leave(int msock, const char* group_uri);

int srcRegister(int msock,
                const char* group_uri,
                size_t num_ifs,
                const uint32_t *ifs);

int srcDeregister(int msock,
                  const char* group_uri,
                  size_t num_ifs,
                  const uint32_t *ifs);

int send(int msock,
         const char* group_uri,
         size_t buf_len,
         const void* buf);
```

```
int receive(int msock,
            const char* group_uri,
            size_t buf_len,
            void* buf);

int getInterfaces(int msock,
                 size_t* num_ifs,
                 uint32_t** ifs);

int addInterface(int msock, uint32_t iface);

int delInterface(int msock, uint32_t iface);

int setTTL(int msock, uint8_t value,
           size_t num_ifs, uint32_t* ifs);

int getTTL(int msock, uint8_t* result);

typedef struct {
    char* group_uri; /* registered mcast group */
    int type; /* 0: listener state,
               1: sender state
               2: sender and listener state */
}
GroupSet;

int groupSet(uint32_t iface,
            size_t* num_groups,
            GroupSet** groups);

int neighborSet(uint32_t iface,
               const char* group_name,
               size_t* num_neighbors,
               char** neighbor_uris);

int childrenSet(uint32_t iface,
               const char* group_name,
               size_t* num_children,
               char** children_uris);
```

```
int parentSet(uint32_t iface,
               const char* group_name,
               size_t* num_parents,
               char** parents_uris);

int designatedHost(uint32_t iface,
                   const char* group_name);

typedef void (*MembershipEventCallback)(int,          /* event type */
                                       uint32_t,      /* interface id */
                                       const char*); /* group uri */

int registerEventCallback(MembershipEventCallback callback);

int disableEvents();
```

Appendix B. 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 C. Deployment Use Cases for Hybrid Multicast

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

C.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 event "new_source_event" is created and passed to the application. 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.

C.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 the event "new_source_event". 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 initiates the event "join_event" 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.

C.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 ("join_event" or "leave_event") 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.

C.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 the "new_source_event". 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 "join_event", an IMG will thereby be informed about group requests from its domain, which are then delegated to the overlay.

Appendix D. Change Log

The following changes have been made from
draft-irtf-samrg-common-api-02

1. Added use case of multicast flavor support
2. Restructured Section 3
3. Major update on namespaces and on mapping
4. C signatures completed
5. Many clarifications and editorial improvements

The following changes have been made from
draft-irtf-samrg-common-api-01

1. Pseudo syntax for lists objects changed
2. Editorial improvements

The following changes have been made from
draft-irtf-samrg-common-api-00

1. Incorrect pseudo code syntax fixed
2. Minor editorial improvements

The following changes have been made from

draft-waehlisch-sam-common-api-06

1. no changes; draft adopted as WG document (previous draft-waehlisch-sam-common-api-06, now draft-irtf-samrg-common-api-00)

The following changes have been made from draft-waehlisch-sam-common-api-05

1. Description of the Common API using pseudo syntax added
2. C signatures of the Comon API moved to appendix
3. updateSender() and updateListener() calls replaced by events
4. Function destroyMSocket renamed as deleteMSocket.

The following changes have been made from draft-waehlisch-sam-common-api-04

1. updateSender() added.

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-waehlisches-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 Waehlisches
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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlich
link-lab & FU Berlin
October 31, 2011

A RELOAD Usage for Distributed Conference Control (DisCo)
draft-knauf-p2psip-disco-04

Abstract

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

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

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

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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

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

1. Introduction

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

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

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

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

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

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

2. Terminology

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

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

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

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

3. Overview of DisCo

3.1. Reference Scenario

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

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

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

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

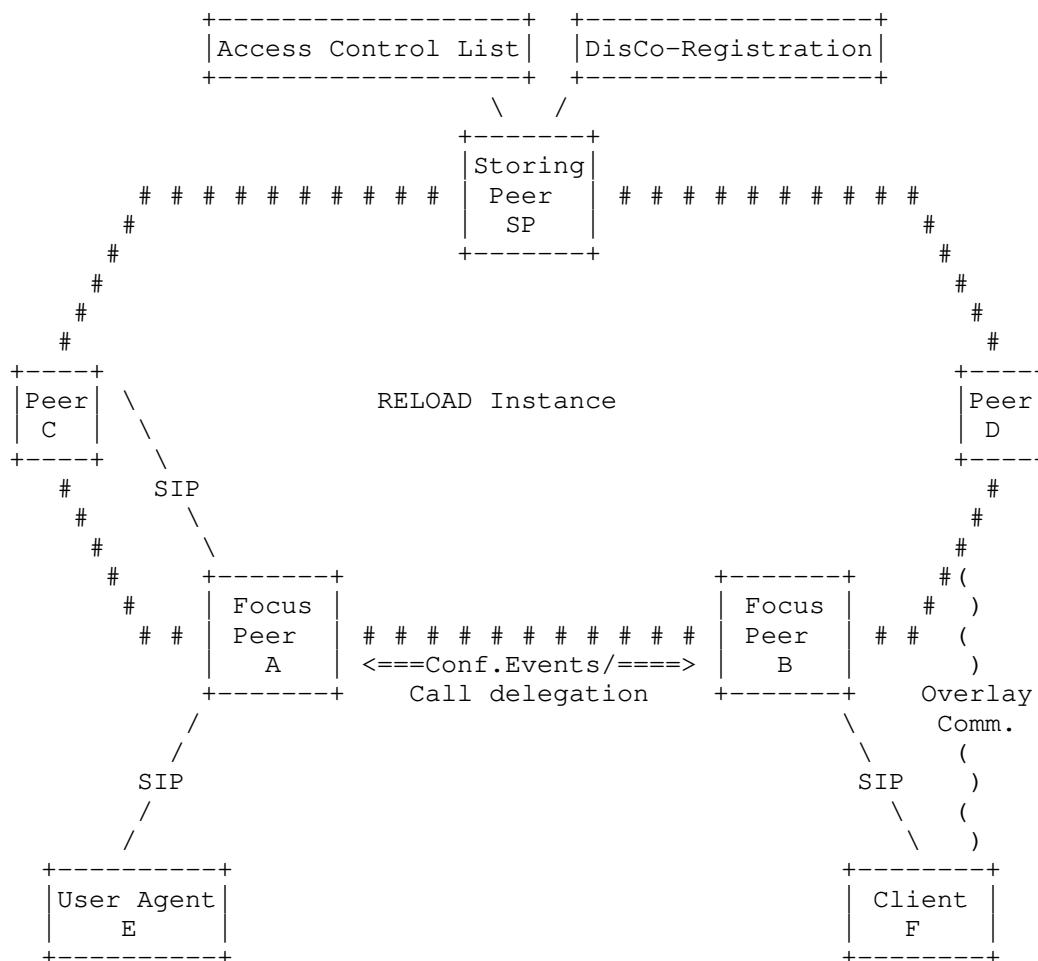


Figure 1: Reference Scenario: Focus peers A,B maintain a distributed conference

3.2. Initiating a Distributed Conference

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

3.3. Joining a Conference

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

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

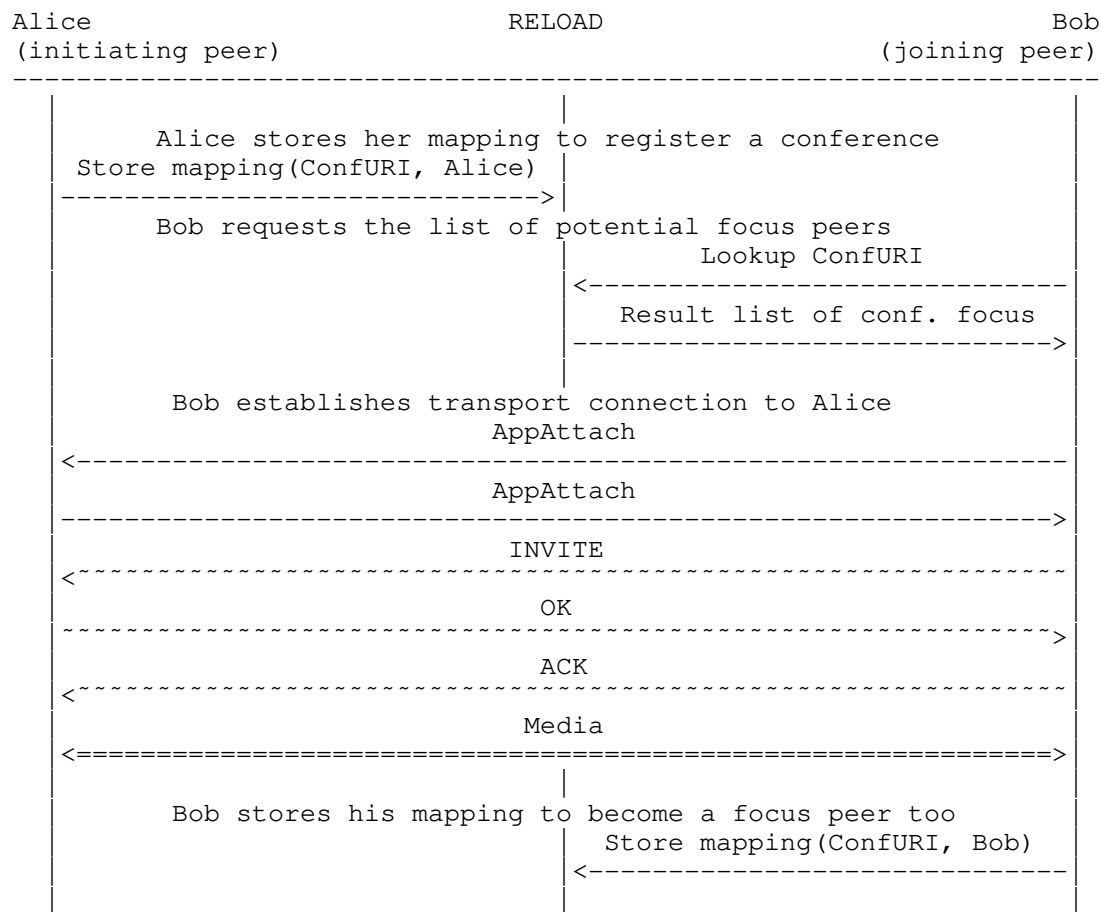


Figure 2: DisCo Usage generic Call Flow

3.4. Conference State Synchronization

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

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

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

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

3.5. Call delegation

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

3.6. Resilience

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

3.7. Topology Awareness

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

4. RELOAD Usage for Distributed Conference Control

4.1. Shared Resource DisCo-Registration

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

DisCo complies with the following requirements for Shared Resources:

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

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

4.2. Kind Data Structure

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

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

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

The DisCoRegistration structure is composed of the following values:

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

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

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

4.3. Variable Conference Identifier

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

4.4. Conference Creation

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

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

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

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

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

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

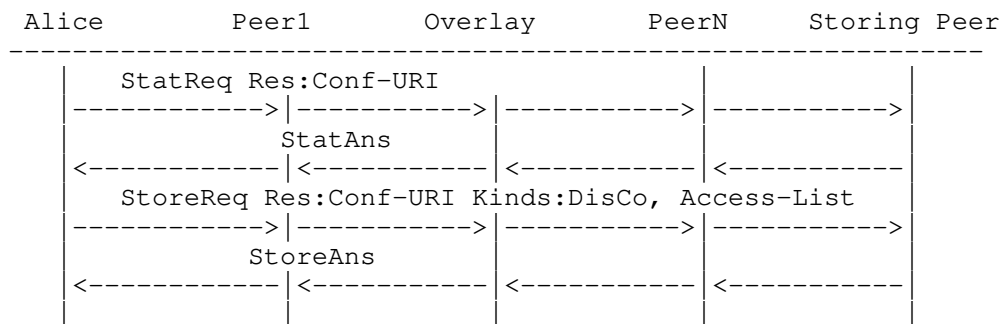


Figure 3: Initial creation of a Distributed Conference

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

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

4.5. Advertising Focus Ability

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

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

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

4.6. Determining Coordinates

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

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

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

4.7. Proximity-aware Conference Participation

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

1. Resolution of the conference identifier.
2. Establishment of of transport connection.
3. SIP signaling to join a conference.

Figure 4 and the following specifications give a more detailed view

on the joining procedure.

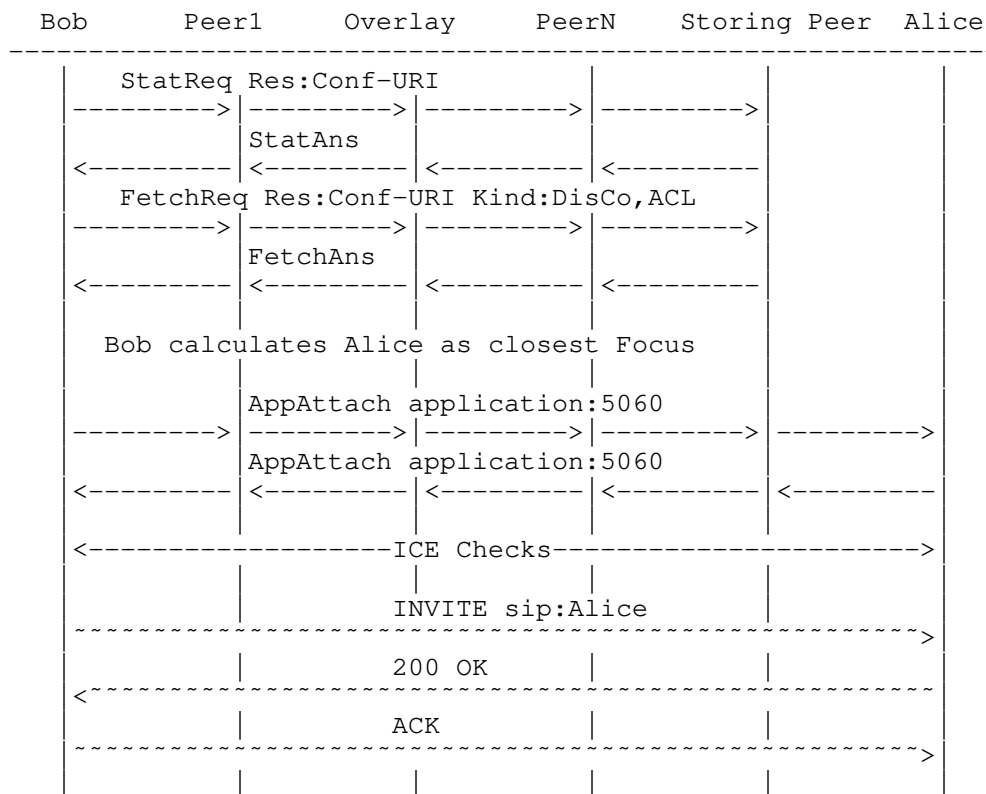


Figure 4: Participation of a Distributed Conference

1. The joining peer MAY determine its own coordinate value (if used).
2. The joining peer sends a StatReq message to obtain all indexes of the Access Control List (ACL) Kinds stored.
3. The joining peer sends a FetchReq message for the DisCo and ACL Kind to the Resource-ID of the conference URI. The FetchReq SHOULD NOT include any specific dictionary keys, but SHOULD fetch for those array ranges previously determined the StatReq. With the ACL items, the joining peer is able to verify whether DisCo-Registrations are stored by authorized focus peers (see [I-D.knauf-p2psip-share]).
4. Using the retrieved coordinates values of the DisCo-Registrations, the joining peer MAY calculate which of than

opaque <0..2¹⁶-1> initial field in the Kind data structure focus peers is the relatively closest to itself.

5. The joining peer then establishes a transport using RELOADs AppAttach, respectively, ICE procedures to create a transport to the chosen focus peer.
6. Finally, the established transport is used to create a SIP dialog from the joining peer to the focus peers.

The SIP INVITE request MAY contain the joining peer's topological descriptor as a URI-parameter called 'coord' in the contact-header in base64 encoded form [RFC4648]. An example contact URI is "sip:alice@example.com;coord=PEknbsBhIHRvcG9sb2dpY2FsIGRlc2NyaXB0b3I+". When the called focus is full and needs to delegate the call it uses the contents of the 'coord'-parameter. It determines the next available focus closest to the calling peer (Section 4.6) using the received descriptor and the other focuses' descriptors from the conference state synchronization document and delegates the call accordingly (see Section 6.1).

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

4.8. Configuration Document Extension

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

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

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

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

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

5. Conference State Synchronization

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

5.1. Event Package Overview

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

```

distributed-conference
|
|-- version-vector
|   |-- version
|   |-- version
|
|-- conference-description
|
|-- focus
|   |-- focus-state
|       |-- user-count
|       |-- coordinate
|       |-- maximum-user-count
|       |-- active
|       |-- locked
|       |-- conf-uris
|       |-- available-media
|
|   |-- users
|       |-- user
|           |-- endpoint
|               |-- media
|               |-- call-info
|
|   |-- relations
|       |-- relation
|
|-- focus
|   |-- ...

```

Figure 5: Overview of the event package for distributed conferences

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

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

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

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

5.2. <distributed-conference>

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

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

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

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

5.3. <version-vector>/<version>

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

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

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

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

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

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

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

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

5.4. <conference-description>

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

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

<subject>: Contains the subject of the conference

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

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

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

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

5.5. <focus>

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

The <focus> element uses the following attributes:

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

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

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

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

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

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

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

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

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

Following, a detailed description of the remaining sub elements.

5.5.1. <focus-state>

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

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

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

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

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

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

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

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

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

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

5.5.2. <users>/<user>

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

5.5.3. <relations>/<relation>

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

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

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

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

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

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

5.6. Distribution of Change Events

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

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

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

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

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

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

5.7. Translation to Conference-Info Event Package

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

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

5.7.1. <conference-info>

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

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

5.7.2. <conference-description>

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

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

5.7.3. <host-info>

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

5.7.4. <conference-state>

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

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

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

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

5.7.5. `<users>/<user>`

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

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

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

6. Distributed Conference Control with SIP

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

6.1. Call delegation

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

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

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

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

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

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

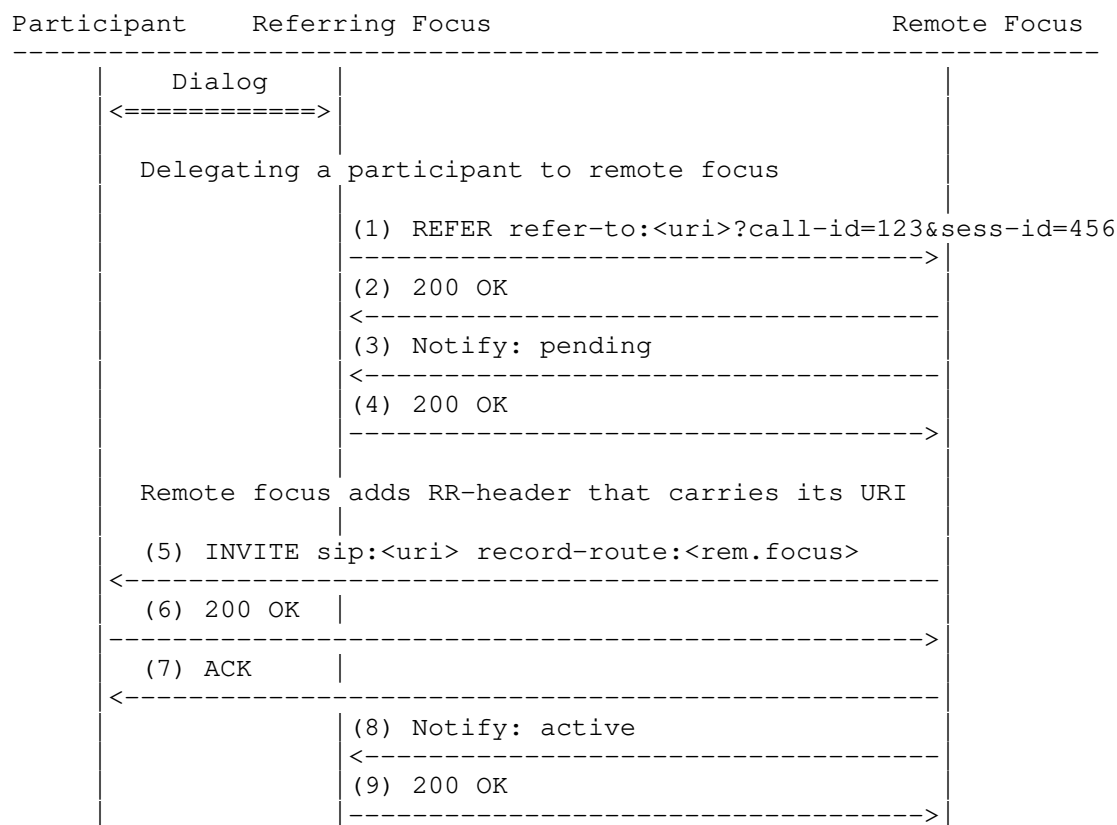


Figure 6: Delegating a participant with SIP REFER

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

6.2. Conference Access

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

Thus it must be decided whether:

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

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

6.3. Media Negotiation and Distribution

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

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

6.3.1. Offer/Answer

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

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

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

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

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

6.4. Restructuring a Conference

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

6.4.1. On Graceful Leave

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

- o LP deletes its mapping in the DisCo-Registration by storing the "non-existing" value as described in the RELOAD base document [I-D.ietf-p2psip-base]. Afterwards, it fetches the lasted version of the DisCo-Registration to obtain all potential focus peers.
- o LP calculates for all its participants the closest focus among all active and potential focus peer using the algorithm described in Section 4.6. LP then delegates all participants to those focus peers.
- o LP then announces its leave by sending a NOTIFY to all its subscribers for the extended conference event package, setting its <focus> state to 'deleted'. Thereafter, it ends its own SIP

conference dialog by sending by to its related focus peer.

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

6.4.2. On Unexpected Leave

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

7. DisCo Kind Definition

This section formally defines the DisCo kind.

Name

DISCO-REGISTRATION

Kind IDs

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

Data Model

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

Access Control

USER-CHAIN-ACL

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

8. XML Schema

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

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

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

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

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


9. Relax NG Grammar

The grammar for the Landmark configuration document extension is:

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

10. Security Considerations

10.1. Trust Aspects

TODO

11. IANA Considerations

TODO: register Kind-ID code point at the IANA

12. Acknowledgments

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

13. References

13.1. Normative References

- [I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-19 (work in progress), October 2011.
- [I-D.knauf-p2psip-share]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlis, "A Usage for Shared Resources in RELOAD (ShaRe)", draft-knauf-p2psip-share-02 (work in progress), October 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
- [RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.
- [RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.
- [RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

13.2. Informative References

[I-D.ietf-p2psip-concepts]

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

[I-D.ietf-p2psip-sip]

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

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

[landmarks-infocomm02]

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

[timestamps-acsc88]

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

Appendix A. Change Log

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

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

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

1. DisCo-Registration uses now only the USER-CHAIN-ACL access control policy.
2. Adapted mechanisms for storing DisCo-Registrations to new requirements of Shared Resources draft [I-D.knauf-p2psip-share]

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

1. The conference registration is now based on the Shared Resources draft [I-D.knauf-p2psip-share]:
 - * DisCo-Registration Kind now meets the requirements for ShaRe.
 - * Conference creation procedure now uses the ShaRe Access List.
 - * Replaced USER-CHAIN-MATCH access policy for DisCo-Registration. Now uses USER-CHAIN-ACL or USER-PATTERN-MATCH.
2. Allow focus peers behind NAT
3. Added a 'node-id' attribute to the event package XML <version> element.
4. Added a 'node-id' attribute to the event package XML <focus> element.
5. Added a 'coordinate' child element to the event package XML <focus> element.
6. Corrected typos/wording

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

1. Updated references.

2. Corrected typos.
3. New Section: Conference State Synchronization
4. XML Event Package for Distributed Conferences
5. New mechanism for generating chained conference certificates
6. Allow shared writing of resources using Access Control Policy
USER-CHAIN-MATCH
7. Media Negotiation mechanism in a distributed conference
8. New Section: Distributed Conference Control with SIP

Authors' Addresses

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

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

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

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

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

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

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

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

P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

A. Knauf
G. Hege
T C. Schmidt
HAW Hamburg
M. Waehlich
link-lab & FU Berlin
October 24, 2011

A Usage for Shared Resources in RELOAD (ShaRe)
draft-knauf-p2psip-share-02

Abstract

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

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

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

Table of Contents

1. Introduction	3
2. Terminology	4
3. Shared Resources in RELOAD	5
3.1. Mechanisms for Isolating Stored Data	6
4. Access Control List Definition	7
4.1. Overview	7
4.2. Data Structure	8
5. Extension for Variable Resource Names	10
5.1. Overview	10
5.2. Data Structure	10
5.3. Overlay Configuration Document Extension	11
6. Access Control to Shared Resources	13
6.1. Granting Write Access	13
6.2. Revoking Write Access	14
6.3. Validating Write Access through an ACL	14
6.4. Operations of Storing Peers	15
6.5. Operations of Accessing Peers	15
6.6. USER-CHAIN-ACL Access Policy	15
7. ACL Kind Definition	17
8. Security Considerations	18
8.1. Resource Exhaustion	18
8.2. Malicious or Misbehaving Storing Peer	18
8.3. Privacy Issues	18
9. IANA Considerations	19
10. Acknowledgments	20
11. References	21
11.1. Normative References	21
11.2. Informative References	21
Appendix A. Change Log	22
Authors' Addresses	23

1. Introduction

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

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

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

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

2. Terminology

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

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

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

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

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

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

3. Shared Resources in RELOAD

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

1. Storage of the Resource/Kind pairs to be shared.
2. Storage of an Access Control List (ACL) associated with those Kinds.

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

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

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

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

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

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

3.1. Mechanisms for Isolating Stored Data

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

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

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

1. Obtain the Node-ID of the certificate that will be used to sign the stored data
2. Take the least significant 24 bits of that Node-ID
3. Concatenate an 8 bit long short individual index value to those 24 bit of the Node-ID

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

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

4. Access Control List Definition

4.1. Overview

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

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

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

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

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

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

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

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

4.2. Data Structure

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

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

The AccessControllListItem structure is composed of:

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

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

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

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

5. Extension for Variable Resource Names

5.1. Overview

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

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

Example Pattern:

```
.*-conf-$USER@$DOMAIN
```

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

5.2. Data Structure

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

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

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

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

The content of the ResourceNameExtension consist of

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

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

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

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

5.3. Overlay Configuration Document Extension

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

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

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

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

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

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

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

<!-- VARIABLE RESOURCE NAMES ELEMENT -->

kind-parameter &= element share:variable-resource-names {
    attribute enable { xsd:boolean }

    <!-- PATTERN ELEMENT -->

    element pattern { xsd:string }*
}?

```

6. Access Control to Shared Resources

6.1. Granting Write Access

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

- o The Resource Owner stores an ACL root item at the Resource-ID of the Shared Resource. The root item contains the resource name extension field (see Section 5.2), the username of the Resource Owner and Kind-ID of the Shared Resource. The allow_delegation flag is set to 1. The index of array data structure MUST be generated as described in Section 3.1
- o Further ACL items for this Kind-ID stored by the Resource Owner will delegate write access to Authorized Peers. These ACL items contain the same resource name extension field, the username of the Authorized Peer and the Kind-Id of the Shared Resource. Optionally, the Resource Owner sets the "ad" to 1 (the default equals 0) to enable the Authorized Peer to further delegate write access. Each succeeding ACL item created by the Resource Owner can be stored in the numerical order of the array index starting with the index of the root item incremented by one.

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

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

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

6.2. Revoking Write Access

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

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

6.3. Validating Write Access through an ACL

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

1. Obtain the username of the certificate used for signing the data stored at the Shared Resource.
2. Validate that an item of the corresponding ACL (i.e., for this Resource/Kind pair) contains a "to_user" field whose value equals the username obtained in step 1. If the Shared Resource under examination is an Access Control List Kind, further validate if the "ad" flag is set to 1.
3. Select the username of the certificate that was used to sign the ACL item obtained in step 2.
4. Validate that an item of the corresponding ACL contains a "to_user" field whose value equals the username obtained in step 3. Additionally validate that the "ad" flag is set to 1.
5. Repeat steps 3 and 4 until the "to_user" value is equal to the username of the signer of the previously selected ACL item. This final ACL item is expected to be the root item of this ACL which SHALL be further validated by verifying that the root item was signed by the owner of the ACL Resource.

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

Shared Resource is an Authorized Peer.

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

6.4. Operations of Storing Peers

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

6.5. Operations of Accessing Peers

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

1. Send a Stat request to the Resource-ID of the Shared Resource to obtain all array indexes of stored ACL Kinds.
2. Fetch all indexes of existing ACL items at this Resource-ID by using the array ranges retrieved in the Stat request answer.

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

6.6. USER-CHAIN-ACL Access Policy

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

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

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

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

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

7. ACL Kind Definition

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

Name: ACCESS-CONTROL-LIST

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

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

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

8. Security Considerations

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

8.1. Resource Exhaustion

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

8.2. Malicious or Misbehaving Storing Peer

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

8.3. Privacy Issues

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

9. IANA Considerations

TODO: register Kind-ID code point at the IANA

10. Acknowledgments

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

11. References

11.1. Normative References

- [I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-18 (work in progress), August 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

11.2. Informative References

- [I-D.ietf-p2psip-concepts]
Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-03 (work in progress), October 2010.
- [I-D.knauf-p2psip-disco]
Knauf, A., Hege, G., Schmidt, T., and M. Waehlich, "A RELOAD Usage for Distributed Conference Control (DisCo)", draft-knauf-p2psip-disco-03 (work in progress), July 2011.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

Appendix A. Change Log

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

1. Simplified the ACL data structure in response to WG feedback
2. Added ResourceNameExtension data structure to simplify the use of variable resource names
3. Restructured document
4. Many editorial improvements

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

1. Integrated the USER-PATTERN-MATCH access policy into USER-CHAIN-ACL
2. Access Control List Kind uses USER-CHAIN-ACL exclusively
3. Resources to be shared use USER-CHAIN-ACL exclusively
4. More precise specification of mandatory User_name and Resource_name fields for Shared Resources
5. Added mechanism for isolating stored data to prevent race conditions while concurrent storing
6. XML Extension for variable resource names uses its own namespace
7. Many editorial improvements

Authors' Addresses

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

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

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

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

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

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

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

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

This Internet-Draft, draft-samrg-sam-baseline-protocol-00.txt, has expired, and has been deleted from the Internet-Drafts directory. An Internet-Draft expires 185 days from the date that it is posted unless it is replaced by an updated version, or the Secretariat has been notified that the document is under official review by the IESG or has been passed to the RFC Editor for review and/or publication as an RFC. This Internet-Draft was not published as an RFC.

Internet-Drafts are not archival documents, and copies of Internet-Drafts that have been deleted from the directory are not available. The Secretariat does not have any information regarding the future plans of the authors or working group, if applicable, with respect to this deleted Internet-Draft. For more information, or to request a copy of the document, please contact the authors directly.

Draft Authors:

John Buford<buford@avaya.com>

Mario Kolberg<mkolberg@ieee.org>