                A Common API for Transparent Hybrid Multicast

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 that grants
   access to the different flavors of multicast. It proposes an
   abstract naming scheme that uses multicast URIs, and it discusses
   mapping mechanisms between different namespaces and distribution
   technologies. Additionally, this document describes the application
   of this API for building gateways that interconnect current Multicast
   Domains throughout the Internet. It reports on an implementation of
   the programming Interface, including service middleware. This
   document is a product of the Scalable Adaptive Multicast (SAM)
   Research Group.

Status of This Memo

   This document is not an Internet Standards Track specification; it is
   published for examination, experimental implementation, and
   evaluation.

   This document defines an Experimental Protocol for the Internet
   community. This document is a product of the Internet Research Task
   Force (IRTF). The IRTF publishes the results of Internet-related
   research and development activities. These results might not be
   suitable for deployment. This RFC represents the consensus of the
   Scalable Adaptive Multicast Research Group of the Internet Research
   Task Force (IRTF). Documents approved for publication by the IRSG
   are not a candidate for any level of Internet Standard; see Section 2
   of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7046.

Copyright Notice

Table of Contents

1. Introduction

   Currently, group application programmers need to choose 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 the use of DNS for Group Addresses been established. The
   standard multicast socket options [RFC3493] [RFC3678] are bound to an
   IP version by not distinguishing between the naming and addressing of
   multicast identifiers. Group communication, however,

   o is commonly implemented in different flavors, such as any-source
      multicast (ASM) vs. source-specific multicast (SSM),

   o is commonly implemented on different layers (e.g., IP vs.
      application-layer multicast), and

   o may be based on different technologies on the same tier, as seen
      with IPv4 vs. IPv6.

   The objective of this document is to provide for programmers a
   universal access to group services.

   Multicast application development should be decoupled from
   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 of 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 for 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 flexible namespace support in group addressing and
   thereby separate naming and addressing (or routing) schemes from
   the application design. This abstraction not only decouples
   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 they can also be 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 usage of
this API at gateways operating between current multicast instances
throughout the Internet is described as well. Finally, a report on
an implementation of the programming Interface, including service
middleware, is presented.

This document represents the consensus of the SAM Research Group. It
has been reviewed by the Research Group members active in the
specific area of work. In addition, this document has been
comprehensively reviewed by people who are not "in" the Research
Group but are experts in the area.

1.1. Use Cases for the Common API

The following generic use cases can be identified; these use cases
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. Thus, for a given application, they can develop a
   program that will run in every deployment scenario. The use of
   Group Names in the form of abstract metadata 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. Complexity is thereby
   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
   can thus match data received from different Interface technologies
   (e.g., IPv4, IPv6, and 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.

Uniform Access to Multicast Flavors: The URI naming scheme uniformly
   supports different flavors of group communication, such as
   any-source multicast and source-specific multicast, and selective
   broadcast, independent of their service instantiation. The
   traditional SSM model, for instance, can experience manifold
   support by directly mapping the multicast URI (i.e.,
   "group@instantiation") to an (S,G) state on the IP layer, by first
   resolving S for a subsequent Group Address query, 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 the
   system level. 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).

1.2. Illustrative Examples

1.2.1. Support of Multiple Underlying Technologies

On a very high level, the common multicast API provides the
application programmer with one single Interface to manage multicast
content independent of the technology underneath. Considering the
following simple example in Figure 1, a multicast source S is
connected via IPv4 and IPv6. It distributes one flow of multicast
content (e.g., a movie). Receivers are connected via IPv4/v6 and
Overlay Multicast (OM), respectively.

```
  +-------+        +-------+                          +-------+
  |   S   |        |  R1   |                          |  R3   |
  +-------+        +-------+                          +-------+
v6|    v4|            |v4                                |OM
  |     |             |                                  |
  |  ***|  ***  ***/  **                      ***  /***  ***   ***
   \*    |*    **  /**    *                    *   /*   **    **    *
   *\   _____/_____*__v4__+-------+      *   /           *
    *\     IPv4/v6       *      |  R2   |__OM__ *_/ Overlay Mcast  *
   *  _____*__v6__+-------+      *                      *
   *   **   **   **    *                    *    **   **   **   *
    ***  ***  ***  ***                       ***  ***  ***  ***
```
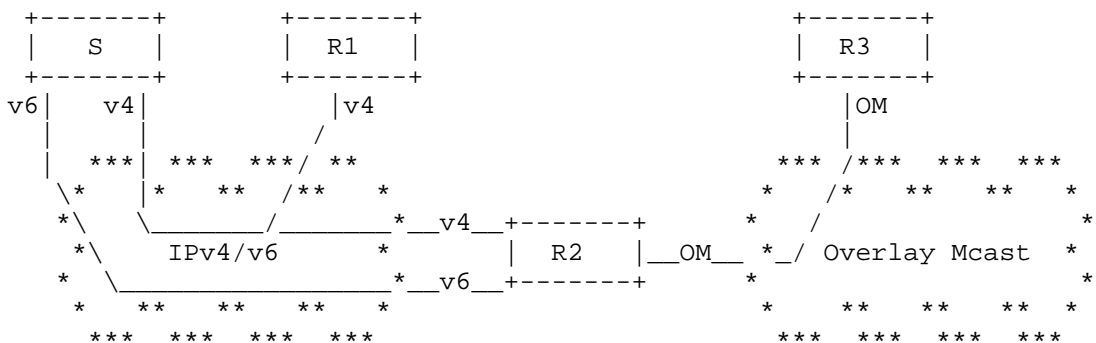
            Figure 1: Common Scenario: Source S Sends the Same Multicast Content
                          via Different Technologies

Using the current BSD socket API, the application programmer needs to
decide on the IP technologies at coding time. Additional
distribution techniques, such as overlay multicast, must be
individually integrated into the application. For each technology,
the application programmer needs to create a separate socket and

initiate a dedicated join or send. As the current socket API does
not distinguish between Group Name and Group Address, the content
will be delivered multiple times to the same receiver (cf. R2).
Whenever the source distributes content via a technology that is not
supported by the receivers or its Internet Service Provider (cf. R3),
a gateway is required. Gateway functions rely on a coherent view of
the Multicast Group states.

The common multicast API simplifies programming of multicast
applications, as it abstracts content distribution from specific
technologies. In addition to calls that implement the receiving and
sending of multicast data, the API provides service calls to grant
access to internal multicast states at the host. The API description
provided in this document defines a minimal set of programming
Interfaces to the system components at the host to operate group
communication. It is left to specific implementations to provide
additional convenience functions for programmers.

The implementation of content distribution for the example shown in
Figure 1 may then look like:

```
  //Initialize multicast socket
  MulticastSocket m = new MulticastSocket();
  //Associate all available Interfaces
  m.addInterface(getInterfaces());
  //Subscribe to Multicast Group
  m.join(URI("ham:opaque:news@cnn.com"));
  //Send to Multicast Group
  m.send(URI("ham:opaque:news@cnn.com"),message);
```

          Send/receive example using the common multicast API

The gateway function for R2 can be implemented by service calls that
look like:

```
//Initialize multicast socket
MulticastSocket m = new MulticastSocket();
//Check (a) host is designated multicast node for this Interface
//      (b) receivers exist
for all this.getInterfaces() {
  if(designatedHost(this.interface) &&
       childrenSet(this.interface,
           URI("ham:opaque:news@cnn.com")) != NULL) {
    m.addInterface(this.interface);
  }
}
while(true) {
  m.send(URI("ham:opaque:news@cnn.com"),message);
}
```

               Gateway example using the common multicast API

1.2.2. Support of Multi-Resolution Multicast

   Multi-resolution multicast adjusts the multicast stream to consider
   heterogeneous end devices. The multicast data (e.g., available by
   different compression levels) is typically announced using multiple
   multicast addresses that are unrelated to each other. Using the
   common API, multi-resolution multicast can be implemented
   transparently by an operator with the help of name-to-address
   mapping, or by systematic naming from a subscriber-centric
   perspective.

   Operator-Centric: An operator deploys a domain-specific mapping. In
      this case, any multicast receiver (e.g., mobile or DSL user)
      subscribes to the same multicast name, which will be resolved
      locally to different multicast addresses. In this case, each
      Group Address represents a different level of data quality.

   Subscriber-Centric: In a subscriber-centric example, the multicast
      receiver chooses the quality in advance, based on a predefined
      naming syntax. Consider a layered video stream "blockbuster"
      available at different qualities Q_i, each of which consists of
      the base layer plus the sum of EL_j, $j <= i$ enhancement layers.
      Each individual layer may then be accessible by a name
      "EL_j.Q_i.blockbuster", $j <= i$, while a specific quality
      aggregates the corresponding layers to "Q_i.blockbuster", and the
      full-size movie may be just called "blockbuster".

2. Terminology

   This document uses the terminology as defined for the multicast
   protocols discussed in [RFC2710], [RFC3376], [RFC3810], [RFC4601],
   and [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 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, it solely 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
      string.

   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, or an overlay routing Interface.

   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 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 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, such as the selection of an IPTV
   channel, or from dynamic session negotiations as used with the
   Session Initiation Protocol (SIP) [RFC3261] or Peer-to-Peer SIP

(P2PSIP) [SIP-RELOAD], 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 can process Group Names in their programs without the
      need to consider technological mappings that relate to designated
      deployments in target domains;

   o applications can identify packets that belong to a logically named
      group, independent of the Interface technology used for sending
      and receiving packets; this shall also hold true for multicast
      gateways.

This document considers two reference scenarios that cover the
following hybrid deployment cases displayed in Figure 2:

   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.

```
                             +-------+         +-------+
                             | Member|         | Member|
                             | Foo   |         |   G   |
                             +-------+         +-------+
                                \         ***   ***   ***   ***
                                 ***   *     **     **     **     *
                                *     *                           *
                                 *     Mcast Tech. A    *
                                *     *                           *
                                 *     **     **     **     *
                                  ***   ***   ***   ***           |
    +-------+         +-------+                              +-------+
    | Member|         | Member|                              | IMG   |
    |   G   |         | Foo   |                              +-------+
    +-------+         +-------+                                 |
       |                 |                                      |
     ***   ***   ***   ***            ***   ***   ***   ***
    *     **     **     **     *      *     **     **     **     *
   *                         *  +-------+  *                           *
   *    Mcast Tech. A    * --| IMG   |-- *   Mcast Tech. B    *    +------+
   *                         *  +-------+  *                           * -|Member|
    *     **     **     **     *      *     **     **     **     *  |   G  |
     ***   ***   ***   ***            ***   ***   ***   ***        +------+
```
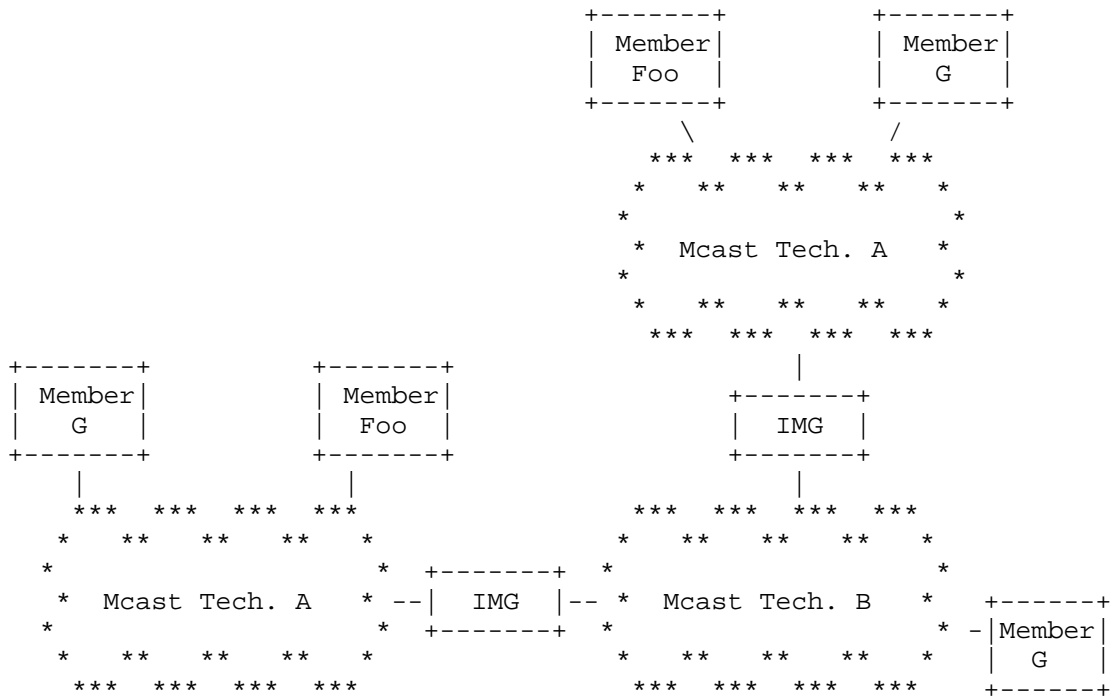
       Figure 2: Reference Scenarios for Hybrid Multicast, Interconnecting
        Group Members from Isolated Homogeneous and Heterogeneous Domains

3.2. Group Communication API and Protocol Stack

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

   Group Management Calls: provide the minimal API to instantiate an
      abstract multicast socket and manage group membership;

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

   Socket Options: provide extension calls for an explicit
      configuration of the multicast socket, 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
   from 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 provides 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 to / leaves / (de)registers 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 represents 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 and therefore
   predefine 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
   on 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. The details of service
discovery are out of scope for this document.

The extended multicast functions can be implemented by middleware, as
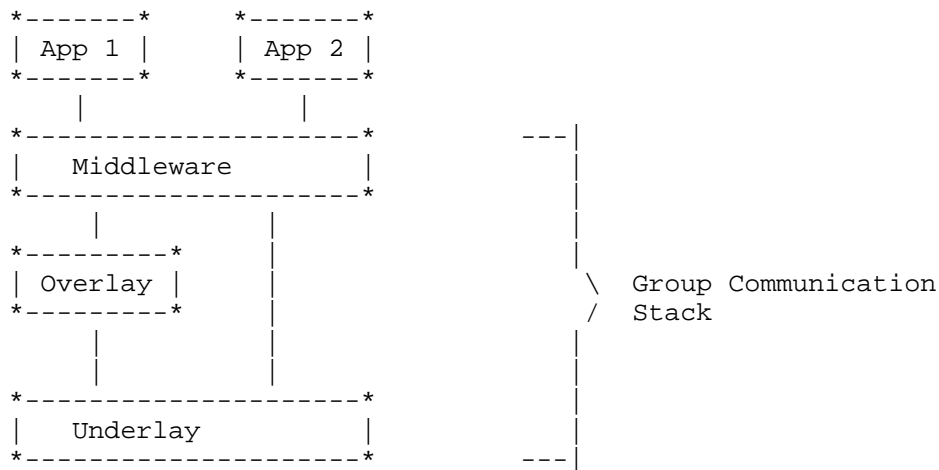conceptually presented in Figure 3.

```
          *--------*      *--------*
          | App 1 |      | App 2 |
          *-------*      *-------*
             |              |
          *---------------------*          ---|
          |    Middleware       |             |
          *---------------------*             |
             |         |                      |
          *---------*  |                      |
          | Overlay |  |                 \  Group Communication
          *---------*  |                 /  Stack
             |         |                      |
             |         |                      |
          *---------------------*             |
          |    Underlay         |             |
          *---------------------*          ---|
```

        Figure 3: Architecture of a Group Communication Stack with Middleware
            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 symbolically identical, 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 metadata type.
Throughout this document, all Group Names follow a URI notation using
the syntax defined in Section 4.2. Examples are
ham:ip:224.1.2.3:5000 for a canonical IPv4 ASM group at UDP port 5000
and ham:sip:news@cnn.com for 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. The details of automatic namespace identification or
service handling are out of scope for this document.

3.4. Namespaces

Namespace identifiers in URIs are placed in the scheme element and
characterize syntax and semantics 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 may facilitate a default mapping and a
recovery of names from addresses. When used in addresses, they
characterize the group identifier's type.

In compliance with the URI concept, namespace schemes can be added.
Examples of schemes are generic (see Section 4.2.3) or inherited from
applications (see Section 4.2.4).

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 to thereby 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 the Interfaces of the end
points. Similarly, a mapping is needed at gateways to consistently
translate between Group Addresses from different namespaces. 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, mappings can be complex and do not need to be invertible.
A mapping can be realized by embedding smaller namespaces into 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 [MCAST-v4v6-FRAMEWORK], [MCAST-v4v6], and [RFC6219]).
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, ham: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 within 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 Inter-Domain Multicast Gateways

Multicast data may arrive at an IMG via one technology and request
that the gateway 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 needs to be acquired. It may have been distributed at
source registration or may have been learned from a neighboring node,
the details of which are beyond the scope of this document.

3.6. A Note on Explicit Multicast (Xcast)

In Explicit Multicast (Xcast) [RFC5058], the multicast source
explicitly predefines the receivers. From a conceptual perspective,
Xcast is an additional distribution technology (i.e., a new
technology-specific Interface) for this API. Xcast requires
aggregated knowledge of receivers that is available at the origin of

   the distribution tree. The instantiation part of the Group Name may
   refer to such a management instance and tree root, which can be the
   source or some co-located processor.

   An implementation of Xcast then requires a topology-dependent mapping
   of the Group Name to the set of subscribers. The defining details of
   this multi-destination mapping are out of scope for this document.

3.7. MTU Handling

   This API considers a multi-technology scenario in which different
   technologies may have different Maximum Transmission Unit (MTU)
   sizes. Even if the MTU size between two hosts has been determined,
   it may change over time, as initiated by either the network (e.g.,
   path changes) or end hosts (e.g., Interface changes due to mobility).

   The design of this API is based on the objective of robust
   communication and easy application development. MTU handling and the
   implementation of fragmentation are thus guided by the following
   observations:

   Application: Application programmers need a simple way to transmit
      packets in a technology-agnostic fashion. For this, it is
      convenient at the time of coding to rely on a transparent maximum
      amount of data that can be sent in one message from a socket. A
      regular program flow should not be distracted by querying and
      changing MTU sizes. Technically, the configuration of the maximum
      message size used by the application programmer may change and
      disrupt communication when (a) Interfaces are added or excluded or
      (b) the path MTU changes during transmission and thus disables the
      corresponding Interfaces.

   Middleware: Middleware situated between application and technology
      Interfaces ensures a general packet-handling capability, which in
      turn prevents the application programmer from implementing
      fragmentation. A uniform maximum message size that cannot be
      changed during runtime shall be guaranteed by the group
      communication stack (e.g., middleware). Otherwise, this would
      conflict with a technology-agnostic application.

   Technology Interfaces: Fragmentation requirements depend on the
      technology in use. Hence, the (technology-bound) Interfaces need
      to cope with MTU sizes that may vary among Interfaces and along
      different paths.

   The concept of this API also aims at guaranteeing a maximum message
   size for the application programmer, to thereby handle fragmentation
   at the Interface level, if needed. Nevertheless, the application
   programmer should be able to determine the technology-specific atomic
   message size to optimize data distribution, or for other reasons.

   The uniform maximum message size should take realistic values (e.g.,
   following IP clients) to enable smooth and efficient services. A
   detailed selection scheme of MTU values is out of scope for this
   document.

4. Common Multicast API

4.1. Notation

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

   The corresponding C signatures are defined in Appendix A.

4.2. URI Scheme Definition

   Multicast Names and Multicast Addresses used in this API are
   represented by a URI scheme that is specified in the following
   subsections. A corresponding ham-URI denotes a multicast channel and
   may be dereferenced to retrieve data published to that channel.

4.2.1. Syntax

   The syntax of the multicast URI is specified using the Augmented
   Backus-Naur Form (ABNF) [RFC5234] and is defined as follows:

   ham-URI   = ham-scheme ":" namespace ":" group ["@" instantiation]
      [":" port] ["/" sec-credentials]

   ham-scheme   = "ham" ; hybrid adaptive multicast
   namespace   = ALPHA *(ALPHA / DIGIT / "+" / "-" / ".")
   group   = "*" / 1*unreserved ; unreserved per [RFC3986]
   instantiation   = 1*unreserved ; unreserved per [RFC3986]
   port   = 1*DIGIT
   sec-credentials = alg ";" val
   alg   = 1*unreserved ; unreserved per [RFC3986]
   val   = 1*unreserved ; unreserved per [RFC3986]

Percent-encoding is applied to distinguish between reserved and
unreserved assignments of the same character in the same ham-URI
component (cf. [RFC3986]).

4.2.2. Semantic

The semantic of the different parts of the URI is defined as follows:

ham-scheme: refers to the specification of the assigned identifier
   "ham".

namespace: takes the role of the Multicast Namespace. It defines
   the syntax of the group and instantiation part of the ham-URI. A
   basic syntax for these elements is specified in Section 4.2.1.
   The namespace may further restrict the syntax of designators.
   Example namespaces are described in Sections 4.2.3 and 4.2.4.

group: uniquely identifies the group within the Multicast Namespace
   given in the namespace. The literal "*" represents all members of
   the Multicast Group.

instantiation: identifies the entity that generates the instance of
   the group (e.g., a SIP domain or a source in SSM, a dedicated
   routing entity, or a named processor that accounts for the group
   communication), using syntax and semantics as defined by the
   namespace. 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 security mechanisms (e.g., to
   authorize Multicast Group access or authenticate multicast
   operations). This parameter is optional. "alg" represents the
   security algorithm in use. "val" represents the actual value for
   Authentication, Authorization, and Accounting (AAA). Note that
   security credentials may 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.

4.2.3. 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. The syntax of the group and instantiation follows the
      "host" definition in [RFC3986], Section 3.2.2. A processor
      dealing with the IP namespace is required to determine the syntax
      (DNS name, IP address, version) of the group and instantiation
      expression.

   SHA-2: This namespace carries address strings compliant with SHA-2
      hash digests. The syntax of the group and instantiation follows
      the "val" definition in [RFC6920], Section 3. A processor
      handling those strings is required to determine the length of the
      expressions and passes appropriate values directly to a
      corresponding overlay.

   Opaque: This namespace transparently carries strings without further
      syntactical information, meanings, or associated resolution
      mechanisms. The corresponding syntax for the group and
      instantiation part of the ham-URI is defined in Section 4.2.1.

4.2.4. 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 at the system level to
      generate a corresponding Multicast Group. The syntax of the group
      and instantiation is represented by the "userinfo" component
      [RFC3261], Section 25.1.

   RELOAD: This namespace covers address strings that are valid in a
      REsource LOcation And Discovery [RELOAD] overlay network. A
      processor handling those strings may pass these values directly to
      a corresponding overlay that may manage multicast distribution
      according to [RFC7019].

4.2.5. Future Namespaces

   The concept of the common multicast API allows for any namespace that
   complies with the superset syntax defined in Section 4.2.1. This
   document specifies a basic set of Multicast Namespaces in
   Sections 4.2.3 and 4.2.4. If additional namespaces are needed in the
   future, a registry for those namespaces should be created and should
   be defined in a future document. All namespaces defined in such a
   document should then also be assigned to the registry.

4.3. Additional Abstract Data Types

4.3.1. Interface

   The Interface denotes the layer and instance on which the
   corresponding call takes effect. 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 data
   structure:

```
       struct ifProp {
         UnsignedInt if_index; /* 1, 2, ... */
         String      *ifName;  /* "eth0", "eth1:1", "lo", ... */
         String      *ifAddr;  /* "1.2.3.4", "abc123", ... */
         String      *ifTech;  /* "ip", "overlay", ... */
       };
```

   The following function retrieves all available Interfaces from the
   system:

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

   It extends the functions for Interface identification as defined in
   [RFC3493], Section 4 and can be implemented by:

```
       struct ifProp(out IfProp <ifsProps>);
```

4.3.2. Membership Events

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

```
       eventType {
               joinEvent;
               leaveEvent;
               newSourceEvent;
       };

       event {
               EventType event;
               Uri groupName;
               Interface if;
       };
```

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

4.4. Group Management Calls

4.4.1. 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 non-operational
communication 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, it
is NULL.

4.4.2. 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 is returned.

4.4.3. Join

The join call initiates a subscription for the given Group Name.
Depending on the Interfaces that are associated with the socket, this
may result in an IGMP / Multicast Listener Discovery (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 is returned.

4.4.4. 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 argument identifies the group.

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

4.4.5. 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 --
   or may facilitate a name-to-address mapping. Likewise, it may remain
   without effect in some multicast technologies.

        srcRegister(in Socket s, in Uri groupName,
                    out 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 is returned.

4.4.6. Source Deregister

   The srcDeregister call indicates that a source no longer intends to
   send data to the Multicast Group. This call may remain without
   effect in some multicast technologies.

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

   The s argument identifies the multicast socket.

   The groupName argument identifies the Multicast Group to which a
   source has stopped sending 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 is returned.

4.5. Send and Receive Calls

4.5.1. Send

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

It is worth noting that it is the choice of the programmer to send
data via one socket per group or to use a single socket for multiple
groups.

```
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 is returned.
A message that is too long is indicated by an implementation-specific
error code (e.g., EMSGSIZE in C).

4.5.2. Receive

The receive call passes multicast data and the corresponding Group
Name to the application. This may come in a blocking or non-blocking
variant.

It is worth noting that it is the choice of the programmer to receive
data via one socket per group or to use a single socket for multiple
groups.

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

   The s argument identifies the multicast socket.

   The groupName 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 is returned.
   A message that is too long is indicated by an implementation-specific
   error code (e.g., EMSGSIZE).

4.6. Socket Options

   The following calls configure an existing multicast socket.

4.6.1. Get Interfaces

   The getInterfaces 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 is returned.

4.6.2. Add Interface

   The addInterface call adds a distribution channel to the socket.
   This may be an overlay or underlay Interface, e.g., IPv6 or
   Distributed Hash Table (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 is returned.

4.6.3. Delete Interface

The delInterface call removes the Interface 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 is returned.

4.6.4. Set TTL

The setTTL call configures the maximum hop count for the socket that
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 is returned.

4.6.5. Get TTL

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

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

The s argument identifies a multicast socket.

The if argument identifies an interface that is bound to socket s.

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

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

4.6.6. Atomic Message Size

   The getAtomicMsgSize function returns the maximum message size that
   an application is allowed to transmit per socket at once without
   fragmentation. This value depends on the Interfaces associated with
   the socket in use and thus may change during runtime.

```
     getAtomicMsgSize(in Socket s,
                      out Int return);
```

   On success, the function returns a positive value of appropriate
   message size; otherwise, -1 is returned.

4.7. Service Calls

4.7.1. 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 and 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 is returned.

4.7.2. 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 information
   regarding neighbors is requested.

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

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

4.7.3. 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 information
   regarding children is requested.

   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 is returned.

4.7.4. 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 information
   regarding parents is requested.

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 is returned.

4.7.5. Designated Host

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

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

The if argument identifies the Interface for which information
regarding designated forwarding is requested.

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. The return value -1 indicates an error. Otherwise, 0 is
returned.

4.7.6. 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 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 4.3.2).

The multicast protocol does not have to support membership tracking
in order to enable this feature. This function can also be
implemented at the middleware layer.

4.7.7. 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.

4.7.8. Maximum Message Size

   The getMaxMsgSize function returns the maximum message size that an
   application is allowed to transmit per socket at once. This value is
   statically guaranteed by the group communication stack.

        getMaxMsgSize(out Int return);

   On success, the function returns a positive value of allowed message
   size; otherwise, -1 is returned.

5. Implementation

   A reference implementation of the Common API for Transparent Hybrid
   Multicast is available with the HAMcast stack [HAMcast-DEV] [GC2010]
   [LCN2012]. This open-source software supports the multicast API (C++
   and Java library) for group application development, the middleware
   as a user space system service, and several multicast-technology
   modules. The middleware is implemented in C++.

   This API is verified and adjusted based on the real-world experiences
   gathered in the HAMcast project, and by additional users of the
   stack.

6. IANA Considerations

   This document specifies the "ham" URI scheme that has been registered
   by IANA as one of the "Provisional URI Schemes" according to
   [RFC4395].

   URI scheme name   ham

   Status   provisional

   URI scheme syntax   See Section 4.2.1.

   URI scheme semantics   See Section 4.2.2.

```
Encoding                See Section 4.2.1
considerations

Applications/protocols  The scheme is used by multicast applications
that use this URI       to access multicast content.
scheme name

Interoperability        None
considerations

Security                See Section 7.
considerations

Contact                 Matthias Waehlisch, mw@link-lab.net

Author/Change           IRTF
controller

References               As specified in this document.
```

7. Security Considerations

   This document does not introduce additional messages or novel
   protocol operations.

8. Acknowledgements

   We would like to thank the HAMcast team at the HAW Hamburg -- Nora
   Berg, Gabriel Hege, Fabian Holler, Alexander Knauf, Sebastian
   Meiling, Sebastian Woelke, and Sebastian Zagaria -- for many fruitful
   discussions and for their continuous critical feedback while
   implementing the common multicast API and hybrid multicast
   middleware. Special thanks to Dominik Charousset of the HAMcast team
   for in-depth perspectives on the matter of code. We gratefully
   acknowledge WeeSan, Mario Kolberg, and John Buford for reviewing and
   their suggestions to improve the document. We would like to thank
   the Name-Based Socket BoF (in particular Dave Thaler) for clarifying
   insights into the question of meta-function calls. We thank Lisandro
   Zambenedetti Granville and Tony Li for very careful reviews of the
   pre-final versions of this document. Barry Leiba and Graham Klyne
   provided very constructive input to find a suitable URI scheme. They
   are gratefully acknowledged.

   This work is partially supported by the German Federal Ministry of
   Education and Research within the HAMcast project (see
   <http://hamcast.realmv6.org>), which is part of G-Lab.

9. References

9.1. Normative References

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

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

   [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.

   [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.

   [RFC5058] Boivie, R., Feldman, N., Imai, Y., Livens, W., and D.
              Ooms, "Explicit Multicast (Xcast) Concepts and Options",
              RFC 5058, November 2007.

   [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234, January 2008.

   [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B.,
              Keranen, A., and P. Hallam-Baker, "Naming Things with
              Hashes", RFC 6920, April 2013.

9.2. Informative References

   [AMT]   Bumgardner, G., "Automatic Multicast Tunneling", Work
              in Progress, October 2013.

   [GC2010]   Meiling, S., Charousset, D., Schmidt, T., and M.
              Waehlisch, "System-assisted Service Evolution for a Future
              Internet - The HAMcast Approach to Pervasive Multicast",
              Proc. IEEE GLOBECOM 2010 Workshops, MCS 2010, pp. 913-917,
              Piscataway, NJ, USA, IEEE Press, December 2010.

   [HAMcast-DEV]
              "HAMcast developers",
              <http://hamcast.realmv6.org/developers>.

   [LCN2012] Meiling, S., Schmidt, T., and M. Waehlisch, "Large-Scale
              Measurement and Analysis of One-Way Delay in Hybrid
              Multicast Networks", Proc. 37th Annual IEEE Conference on
              Local Computer Networks (LCN 2012), Piscataway, NJ, USA,
              IEEE Press, October 2012.

   [MCAST-v4v6]
              Venaas, S., Asaeda, H., SUZUKI, S., and T. Fujisaki, "An
              IPv4 - IPv6 multicast translator", Work in Progress,
              December 2010.

   [MCAST-v4v6-FRAMEWORK]
              Venaas, S., Li, X., and C. Bao, "Framework for IPv4/IPv6
              Multicast Translation", Work in Progress, June 2011.

   [RELOAD]   Jennings, C., Lowekamp, B., Ed., Rescorla, E., Baset, S.,
              and H. Schulzrinne, "REsource LOcation And Discovery
              (RELOAD) Base Protocol", Work in Progress, February 2013.

   [RFC5757] Schmidt, T., Waehlisch, M., and G. Fairhurst, "Multicast
              Mobility in Mobile IP Version 6 (MIPv6): Problem Statement
              and Brief Survey", RFC 5757, February 2010.

   [RFC6219] Li, X., Bao, C., Chen, M., Zhang, H., and J. Wu, "The
              China Education and Research Network (CERNET) IVI
              Translation Design and Deployment for the IPv4/IPv6
              Coexistence and Transition", RFC 6219, May 2011.

   [RFC7019] Buford, J. and M. Kolberg, "Application-Layer Multicast
              Extensions to REsource LOcation And Discovery (RELOAD)",
              RFC 7019, September 2013.

   [SIP-RELOAD]
              Jennings, C., Lowekamp, B., Rescorla, E., Baset, S.,
              Schulzrinne, H., and T. Schmidt, Ed., "A SIP Usage for
              RELOAD", Work in Progress, July 2013.

Appendix A. C Signatures

   This section describes the C signatures of the common multicast API
   (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,
                    uint32_t* ifs);

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

    int getAtomicMsgSize(int msock);
```

```
    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 enableEvents();

       int disableEvents();

    int getMaxMsgSize();
```

Appendix B. Use Case for the API

   For the sake of readability, we demonstrate development of
   applications using the API based on a high-level Java-like syntax; we
   do not consider error handling.

     -- Application above middleware:

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

     m.join(URI("ham:ip:224.1.2.3:5000"));
     m.join(URI("ham:ip:[ff02:0:0:0:0:0:0:3]:6000"));
     m.join(URI("ham: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, e.g., an overlay.

   An arbitrary Distance Vector Multicast Routing Protocol (DVMRP)
   [RFC1075] router will not be informed of 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 the IMG requires a DVMRP neighbor
   connectivity. Thus, 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 to DVMRP routers on the path from the
   new receiver subnet to the source, but in general not to an IMG. To
   overcome this problem, data of multicast senders in the overlay may
   become noticeable via the Source Register call, as well as by an IMG
   that initiates an all-group join in the overlay using the namespace
   extension of the API. Each IMG is initially required to forward the
   data received in the overlay 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, e.g., an overlay.

   The Protocol Independent Multicast - Sparse Mode (PIM-SM) [RFC4601]
   establishes rendezvous points (RPs). These entities receive listener
   subscriptions and source registering of a domain. For a continuous
   update, an IMG has to be co-located with an 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; this shared tree may change to a source-
specific tree after PIM switches to phase three. 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 be transparently
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, e.g., an overlay.

   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 is
   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 agreement with the
   objectives of PIM-SSM, the signaling should not be flooded in the
   underlay and overlay.

   A solution can be to intercept the subscription at both source sites
   and receiver sites: To monitor multicast receiver subscriptions
   ("join_event" or "leave_event") in the underlay, the IMG is placed on
   the 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 Automatic Multicast
   Tunneling [AMT] 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, e.g., an overlay.

   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). 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 case
   will be transmitted to the RP link, 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 of group requests from
   its domain, which are then delegated to the overlay.

Authors' Addresses

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

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

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

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

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

   EMail: stig@cisco.com