Internet Research Task Force (IRTF)
Request for Comments: 7019
Category: Experimental
ISSN: 2070-1721

J. Buford
Avaya Labs Research
M. Kolberg, Ed.
University of Stirling

September 2013

Application-Layer Multicast Extensions to REsource LOcation And Discovery (RELOAD)

Abstract

We define a REsource LOcation And Discovery (RELOAD) Usage for Application-Layer Multicast (ALM) as well as a mapping to the RELOAD experimental message type to support ALM. The ALM Usage is intended to support a variety of ALM control algorithms in an overlay-independent way. Two example algorithms are defined, based on Scribe and P2PCast.

This document is a product of the Scalable Adaptive Multicast Research Group (SAM RG).

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/rfc7019.

Table of Contents

1. Introduction

The concept of scalable adaptive multicast includes both scaling properties and adaptability properties. Scalability is intended to cover:

- large group size
- large numbers of small groups
- rate of group membership change
- admission control for QoS
- use with network-layer QoS mechanisms
- varying degrees of reliability
- trees connecting nodes over the global Internet

Adaptability includes

- use of different control mechanisms for different multicast trees depending on initial application parameters or application classes
- changing multicast tree structure depending on changes in application requirements, network conditions, and membership

Application-Layer Multicast (ALM) has been demonstrated to be a viable multicast technology where native multicast isn't available. Many ALM designs have been proposed. This ALM Usage focuses on:

- ALM implemented in RELOAD-based overlays
- Support for a variety of ALM control algorithms
- Providing a basis for defining a separate hybrid ALM RELOAD Usage

RELOAD [RELOAD] has an application extension mechanism in which a new type of application defines a Usage. A RELOAD Usage defines a set of data types and rules for their use. In addition, this document describes additional message types and a new ALM algorithm plugin architectural component.

This document represents the consensus of the SAM RG. It was repeatedly discussed within the research group, as well as with other Application-Layer Multicast experts.

1.1. Requirements Language

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

2. Definitions

We adopt the terminology defined in Section 3 of [RELOAD], specifically the distinction between "node", "peer", and "client".

2.1. Overlay Network

Overlay network: An application-layer virtual or logical network with addressable end points that provides connectivity, routing, and messaging between end points. Overlay networks are frequently used as a substrate for deploying new network services or for providing a routing topology not available from the underlying physical network. Many peer-to-peer systems are overlay networks that run on top of the Internet. In Figure 1, "P" indicates overlay peers, and peers are connected in a logical address space. The links shown in the figure represent predecessor/successor links. Depending on the overlay routing model, additional or different links may be present.

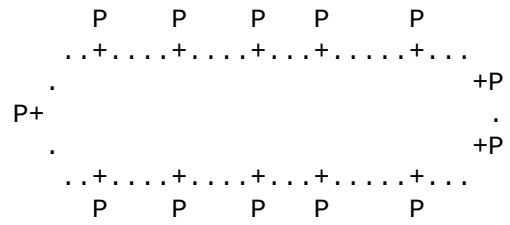

Figure 1: Overlay Network Example

2.2. Overlay Multicast

Overlay Multicast (OM): Hosts participating in a multicast session form an overlay network and utilize unicast connections among pairs of hosts for data dissemination [BUFORD2009] [KOLBERG2010] [BUFORD2008]. The hosts in overlay multicast exclusively handle group management, routing, and tree construction, without any support from Internet routers. This is also commonly known as Application-Layer Multicast (ALM) or End-System Multicast (ESM). We call systems that use proxies connected in an overlay multicast backbone "proxied overlay multicast" or POM.

2.3. Source-Specific Multicast (SSM)

SSM tree: The creator of the tree is the source. It sends data messages to the tree root that are forwarded down the tree.

2.4. Any-Source Multicast (ASM)

ASM tree: A node sending a data message sends the message to its parent and its children. Each node receiving a data message from one edge forwards it to the remaining tree edges to which it is connected.

2.5. Peer

Peer: An autonomous end system that is connected to the physical network and participates in and contributes resources to overlay construction, routing, and maintenance. Some peers may also perform additional roles such as connection relays, super nodes, NAT traversal assistance, and data storage.

3. Assumptions

3.1. Overlay

Peers connect in a large-scale overlay, which may be used for a variety of peer-to-peer applications in addition to multicast sessions. Peers may assume additional roles in the overlay beyond participation in the overlay and in multicast trees. We assume a single-structured overlay routing algorithm is used. Any of a variety of multi-hop, one-hop, or variable-hop overlay algorithms could be used.

Castro, et al. [CASTRO2003] compared multi-hop overlays and found that tree-based construction in a single overlay outperformed using separate overlays for each multicast session. We use a single overlay rather than separate overlays per multicast session.

An overlay multicast algorithm may leverage the overlay's mechanism for maintaining overlay state in the face of churn. For example, a peer may store a number of DHT (Distributed Hash Table) entries. When the peer gracefully leaves the overlay, it transfers those entries to the nearest peer. When another peer joins that is closer to some of the entries than the current peer that holds those entries, than those entries are migrated. Overlay churn affects multicast trees as well; remedies include automatic migration of the tree state and automatic rejoin operations for dislocated child nodes.

3.2. Overlay Multicast

The overlay supports concurrent multiple multicast trees. The limit on the number of concurrent trees depends on peer and network resources and is not an intrinsic property of the overlay.

3.3. RELOAD

We use RELOAD [RELOAD] as the peer-to-peer (P2P) overlay for data storage and the mechanism by which the peers interconnect and route messages. RELOAD is a generic P2P overlay, and application support is defined by profiles called Usages.

3.4. NAT

Some nodes in the overlay may be in a private address space and behind firewalls. We use the RELOAD mechanisms for NAT traversal. We permit clients to be leaf nodes in an ALM tree.

3.5. Tree Topology

All tree control messages are routed in the overlay. Two types of data or media topologies are envisioned: 1) tree edges are paths in the overlay, and 2) tree edges are direct connections between a parent and child peer in the tree, formed using the RELOAD AppAttach method.

4. Architecture Extensions to RELOAD

There are two changes as depicted in Figure 2. New ALM messages are mapped to RELOAD Message Transport using the RELOAD experimental message type. A plugin for ALM algorithms handles the ALM state and control. The ALM algorithm is under control of the application via the Group API [COMMON-API].

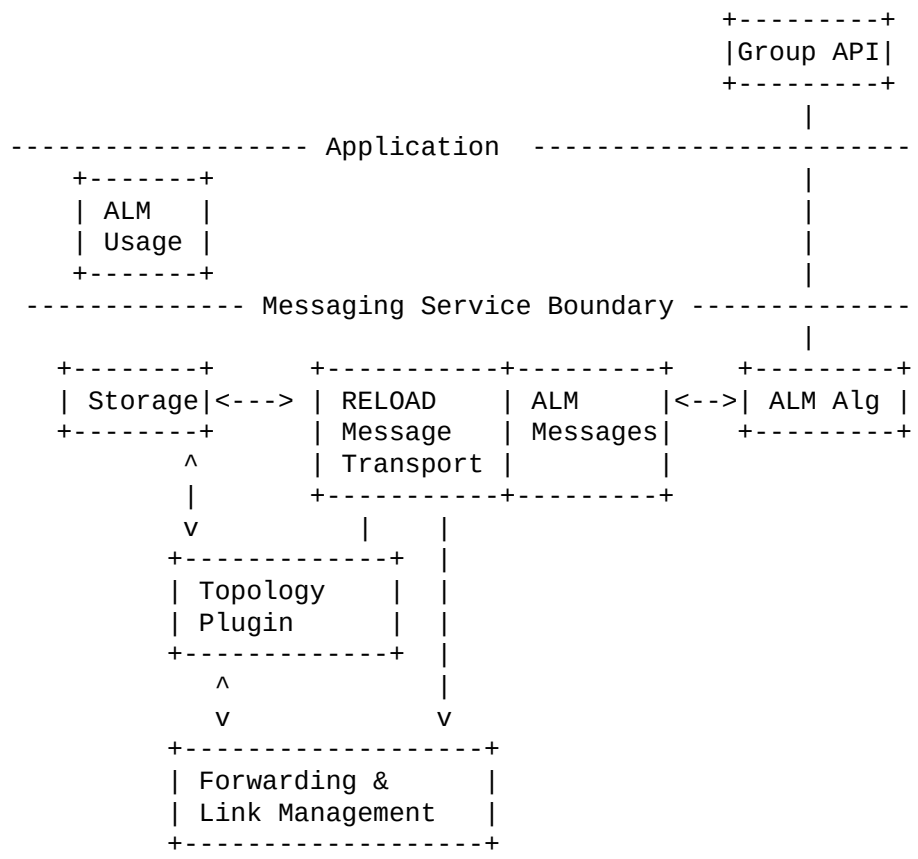

Figure 2: RELOAD Architecture Extensions

The ALM components interact with RELOAD as follows:

- ALM uses the RELOAD data storage functionality to store an ALMTree instance when a new ALM tree is created in the overlay and to retrieve ALMTree instance(s) for existing ALM trees.

- ALM applications and management tools may use the RELOAD data storage functionality to store diagnostic information about the operation of trees, including average number of trees, delay from source to leaf nodes, bandwidth use, and packet loss rate. In addition, diagnostic information may include statistics specific to the tree root or to any node in the tree.

5. RELOAD ALM Usage

Applications of RELOAD are restricted in the data types that can be stored in the DHT. The profile of accepted data types for an application is referred to as a Usage. RELOAD is designed so that new applications can easily define new Usages. New RELOAD Usages are needed for multicast applications since the data types in base RELOAD and existing Usages are not sufficient.

We define an ALM Usage in RELOAD. This ALM Usage is sufficient for applications that require ALM functionality in the overlay. Figure 2 shows the internal structure of the ALM Usage. This contains the Group API ([COMMON-API]), an ALM algorithm plugin (e.g., Scribe), and the ALM messages that are then sent out to the RELOAD network.

A RELOAD Usage is required [RELOAD] to define the following:

- Kind-ID and code points
- data structures for each Kind
- access control rules for each Kind
- the Resource Name used to hash to the Resource ID that determines where the Kind is stored
- address restoration after recovery from a network partition (to form a single coherent network)
- the types of connections that can be initiated using AppConnect

An ALM group_id is a RELOAD node_id. The owner of an ALM group creates a RELOAD node_id as specified in [RELOAD]. This means that a group_id is used as a RELOAD Destination for overlay routing purposes.

6. ALM Tree Control Signaling

Peers use the overlay to support ALM operations such as:

- CreateALMTree
- Join
- Leave
- Reform or optimize tree

There are a variety of algorithms for peers to form multicast trees in the overlay. The approach presented here permits multiple such algorithms to be supported in the overlay since different algorithms may be more suitable for certain application requirements; the approach also supports experimentation. Therefore, overlay messaging corresponding to the set of overlay multicast operations MUST carry algorithm identification information.

For example, for small groups, the join point might be directly assigned by the rendezvous point, while for large trees the Join request might be propagated down the tree with candidate parents forwarding their position directly to the new node.

Here is a simplistic notation for forming a multicast tree in the overlay. Its main advantage is the use of the overlay for routing both control and data messages. The group creator does not have to be the root of the tree or even in the tree. It does not consider per-node load, admission control, or alternative paths. After the creation of a tree, the group_id is expected to be advertised or distributed out of band, perhaps by publishing in the DHT. Similarly, joining peers will discover the group_id out of band, perhaps by a lookup in the tree.

As stated earlier, multiple algorithms will coexist in the overlay.

1. Peer that initiates multicast group:

```
group_id = create();  // Allocate a unique group_id.
                      // The root is the nearest
                      // peer in the overlay.
```

2. Any joining peer:

```
joinTree(group_id); // sends "join group_id" message
```

   The overlay routes the Join request using the overlay routing mechanism toward the peer with the nearest ID to the group_id. This peer is the root. Peers on the path to the root join the tree as forwarding points.

3. Leave Tree:

```
leaveTree(group_id); // removes this node from the tree
```

Propagates a Leave request to each child node and to the parent node. If the parent node is a forwarding node and this is its last child, then it propagates a Leave request to its parent. A child node receiving a Leave request from a parent sends a Join request to the group_id.

4. Message forwarding:

multicastMsg(group_id, msg);

For message forwarding, both Any-Source Multicast (ASM) and Source-Specific Multicast (SSM) approaches may be used.

7. ALM Messages Mapped to RELOAD

7.1. Introduction

In this document, we define messages for overlay multicast tree creation, using an existing protocol (RELOAD) in the P2P-SIP WG [RELOAD] for a universal structured peer-to-peer overlay protocol. RELOAD provides the mechanism to support a number of overlay topologies. Hence, the overlay multicast framework defined in this document can be used with P2P-SIP and makes the Scalable Adaptive Multicast (SAM) framework overlay agnostic.

As discussed in the SAM requirements document [SAM-GENERIC], there are a variety of ALM tree formation and tree maintenance algorithms. The intent of this specification is to be algorithm agnostic, similar to how RELOAD is overlay algorithm agnostic. We assume that all control messages are propagated using overlay routed messages.

The message types needed for ALM behavior are divided into the following categories:

- Tree lifecycle (Create, Join, Leave, Reform, Heartbeat)

- Peer region and multicast properties

The message codes are defined in Section 14.2 of this document. Messages are mapped to the RELOAD experimental message type.

In the following sections, the protocol messages as mapped to RELOAD are discussed. Detailed example message flows are provided in Section 11.

In the following descriptions, we use the datatype Dictionary, which is a set of opaque values indexed by an opaque key with one value for each key. A single dictionary entry is represented by a DictionaryEntry as defined in Section 7.2.3 of the RELOAD document [RELOAD]. The Dictionary datatype is defined as follows:

```
struct {
  DictionaryEntry elements<0..2^16-1>;
  } Dictionary;
```

7.2. Tree Lifecycle Messages

Peers use the overlay to transmit ALM operations defined in this section.

7.2.1. CreateALMTree

A new ALM tree is created in the overlay with the identity specified by group_id. The common interpretation in a DHT-based overlay of group_id is that the peer with a peer_id closest to and less than the group_id is the root of the tree. However, other overlay types are supported. The tree has no children at the time it is created.

The group_id is generated from a well-known session key to be used by other peers to address the multicast tree in the overlay. The generation of the group_id from the session_key MUST be done using the overlay's ID-generation mechanism.

```
struct {
  node_id peer_id;
  opaque session_key<0..2^32-1>;
  node_id group_id;
  Dictionary options;
} ALMTree;
```

peer_id: overlay address of the peer that creates the multicast tree.

session_key: a well-known string that when hashed using the overlay's ID-generation algorithm produces the group_id.

group_id: overlay address of the root of the tree.

options: name-value list of properties to be associated with the tree, such as the maximum size of the tree, restrictions on peers joining the tree, latency constraints, preference for distributed or centralized tree formation and maintenance, and Heartbeat interval.

Tree creation is subject to access control since it involves a Store operation. The NODE-MATCH access policy defined in Section 7.3.2 of [RELOAD] is used.

A successful CreateALMTree causes an ALMTree structure to be stored in the overlay at the node G responsible for the group_id. This node G performs the RELOAD-defined StoreReq operation as a side effect of performing the CreateALMTree. If the StoreReq fails, the CreateALMTree fails too.

After a successful CreateALMTree, peers can use the RELOAD Fetch method to retrieve the ALMTree struct at address group_id. The ALMTree Kind is defined in Section 12.1.

7.2.2. CreateALMTreeResponse

After receiving a CreateALMTree message from node S, the peer sends a CreateALMTreeResponse to node S.

```
struct {
  Dictionary options;
} CreateALMTreeResponse;
```

options: A node may provide algorithm-dependent parameters about the created tree to the requesting node.

7.2.3. Join

Join causes the distributed algorithm for peer join of a specific ALM group to be invoked. The definition of the Join request is shown below. If successful, the joining peer is notified of one or more candidate parent peers in one or more JoinAccept messages. The particular ALM join algorithm is not specified in this protocol.

```
struct {
  node_id peer_id;
  node_id group_id;
  Dictionary options;
} Join;
```

peer_id: overlay address of joining/leaving peer

group_id: overlay address of the root of the tree

options: name-value list of options proposed by joining peer

RELOAD is a request-response protocol. Consequently, the messages JoinAccept and JoinReject (defined below) are matching responses for Join. If JoinReject is received, then no further action on this request is carried out. If JoinAccept is received, then either a JoinConfirm or a JoinDecline message (see below) is sent. The matching response for JoinConfirm is JoinConfirmResponse. The matching response for JoinDecline is JoinDeclineResponse.

The following list shows the matching request-responses according to the request-response mechanism defined in [RELOAD].

Join -- JoinAccept: Node C sends a Join request to node P. If node P accepts, it responds with JoinAccept.

Join -- JoinReject: Node C sends a Join request to node P. If node P does not accept the Join request, it responds with JoinReject.

JoinConfirm -- JoinConfirmResponse: If node P sent node C a JoinAccept and node C confirms with a JoinConfirm request, then node P responds with a JoinConfirmResponse.

JoinDecline -- JoinDeclineResponse: If node P sent node C a JoinAccept and node C declines with a JoinDecline request, then node P responds with a JoinDeclineResponse.

Thus, Join, JoinConfirm, and JoinDecline are treated as requests as defined in RELOAD, are mapped to the RELOAD exp_a_req message, and are therefore retransmitted until either a retry limit is reached or a matching response received. JoinAccept, JoinReject, JoinConfirmResponse, and JoinDeclineResponse are treated as message responses as defined above and are mapped to the RELOAD exp_a_ans message.

The Join behavior can be described as follows:

```
if(checkAccept(msg)) {
    recvJoins.add(msg.source, msg.group_id)
    SEND(JoinAccept(node_id, msg.source, msg.group_id))
}
```

7.2.4. JoinAccept (Join Response)

JoinAccept tells the requesting joining peer that the indicated peer is available to act as its parent in the ALM tree specified by group_id, with the corresponding options specified. A peer MAY receive more than one JoinAccept from different candidate parent peers in the group_id tree. The peer accepts a peer as parent using

a JoinConfirm message. A JoinAccept that receives neither a JoinConfirm nor JoinDecline message MUST expire. RELOAD implementations are able to read a local configuration file for settings. It is assumed that this file contains the timeout value to be used.

```
struct {
  node_id parent_peer_id;
  node_id child_peer_id;
  node_id group_id;
  Dictionary options;
} JoinAccept;
```

parent_peer_id: overlay address of a peer that accepts the joining peer

child_peer_id: overlay address of joining peer

group_id: overlay address of the root of the tree

options: name-value list of options accepted by parent peer

7.2.5. JoinReject (Join Response)

A peer receiving a Join request responds with a JoinReject response to indicate the request is rejected.

7.2.6. JoinConfirm

A peer receiving a JoinAccept message that it wishes to accept MUST explicitly accept it using a JoinConfirm message before the expiration of a timer for the JoinAccept message. The joining peer MUST include only those options from the JoinAccept that it also accepts, completing the negotiation of options between the two peers.

```
struct {
  node_id child_peer_id;
  node_id parent_peer_id;
  node_id group_id;
  Dictionary options;
} JoinConfirm;
```

child_peer_id: overlay address of joining peer that is a child of the parent peer

parent_peer_id: overlay address of the peer that is the parent of the joining peer

group_id: overlay address of the root of the tree

options: name-value list of options accepted by both peers

The JoinConfirm message behavior is described below:

```
if(recvJoins.contains(msg.source,msg.group_id)){
   if !(groups.contains(msg.group_id)) {
      groups.add(msg.group_id)
      SEND(msg,msg.group_id)
   }
   groups[msg.group_id].children.add(msg.source)
   recvJoins.del(msg.source, msg.group_id)
}
```

7.2.7. JoinConfirmResponse

A peer receiving a JoinConfirm message responds with a JoinConfirmResponse message.

7.2.8. JoinDecline

A peer receiving a JoinAccept message that it does not wish to accept MAY explicitly decline it using a JoinDecline message.

```
   struct {
     node_id peer_id;
     node_id parent_peer_id;
     node_id group_id;
   } JoinDecline;
```

peer_id: overlay address of joining peer that declines the JoinAccept

parent_peer_id: overlay address of the peer that issued a JoinAccept to this peer

group_id: overlay address of the root of the tree

The behavior of the JoinDecline message is described as follows:

```
if(recvJoins.contains(msg.source,msg.group_id))
   recvJoins.del(msg.source, msg.group_id)
```

7.2.9. JoinDeclineResponse

A peer receiving a JoinConfirm message responds with a JoinDeclineResponse message.

7.2.10. Leave

A peer that is part of an ALM tree identified by group_id that intends to detach from either a child or parent peer SHOULD send a Leave request to the peer from which it wishes to detach. A peer receiving a Leave request from a peer that is neither in its parent nor child lists SHOULD ignore the message.

```
struct {
  node_id peer_id;
  node_id group_id;
  Dictionary options;
} Leave;
```

peer_id: overlay address of leaving peer

group_id: overlay address of the root of the tree

options: name-value list of options

The behavior of the Leave request can be described as:

```
groups[msg.group_id].children.remove(msg.source)
if (groups[msg.group].children = 0)
      SEND(msg,groups[msg.group_id].parent)
```

7.2.11. LeaveResponse

A peer receiving a Leave request responds with a LeaveResponse message.

7.2.12. Reform or Optimize Tree

This triggers a reorganization of either the entire tree or only a subtree. It MAY include hints to specific peers of recommended parent or child peers to which to reconnect. A peer receiving this message MAY ignore it, MAY propagate it to other peers in its subtree, and MAY invoke local algorithms for selecting preferred parent and/or child peers.

```
struct {
  node_id group_id;
  node_id peer_id;
  Dictionary options;
} Reform;
```

group_id: overlay address of the root of the tree

peer_id: if omitted, then the tree is reorganized starting from the root; otherwise, it is reorganized only at the subtree identified by peer_id.

options: name-value list of options

7.2.13. ReformResponse

A peer receiving a Reform message responds with a ReformResponse.

```
struct {
  Dictionary options;
} ReformResponse;
```

options: algorithm-dependent information about the results of the Reform operation

7.2.14. Heartbeat

A child node signals to its adjacent parent nodes in the tree that it is alive. If a parent node does not receive a Heartbeat message within N Heartbeat time intervals, it MUST treat this as an explicit Leave request from the unresponsive peer. N is configurable. RELOAD implementations are able to read a local configuration file for settings. It is assumed that this file contains the value for N to be used.

```
struct {
  node_id peer_id_src;
  node_id peer_id_dst;
  node_id group_id;
  Dictionary options;
} Heartbeat;
```

peer_id_src: source of Heartbeat

peer_id_dst: destination of Heartbeat

group_id: overlay address of the root of the tree

options: an algorithm may use the Heartbeat message to provide state information to adjacent nodes in the tree

7.2.15. Heartbeat Response

A parent node responds with a HeartbeatResponse to a Heartbeat from a child node indicating that it has received the Heartbeat message.

7.2.16. NodeQuery

The NodeQuery message is used to obtain information about the state and performance of the tree on a per-node basis. A set of nodes could be queried to construct a centralized view of the multicast trees, similar to a web crawler.

```
struct {
  node_id peer_id_src;
  node_id peer_id_dst;
} NodeQuery;
```

peer_id_src: source of query

peer_id_dst: destination of query

7.2.17. NodeQueryResponse

The response to a NodeQuery message contains a NodeStatistics instance for this node.

```
public struct {
   uint32        node_lifetime;
   uint32        total_number_trees;
   uint16        number_algorithms_supported;
   uint8         algorithms_supported[32];
   TreeData      max_tree_data;
   uint16        number_active_trees;
   TreeData      tree_data<0..2^8-1>;
   ImplementationInfo impl_info;
}  NodeStatistics;
```

node_lifetime: time the node has been alive in seconds since last restart

total_number_trees: total number of trees this node has been part of during the node lifetime

number_algorithms_supported: value between 0..2^16-1 corresponding to the number of algorithms supported

algorithms_supported: list of algorithms, each byte encoded using the corresponding algorithm code

max_tree_data: data about tree with largest number of nodes that this node was part of. NodeQuery can be used to crawl all the nodes in an ALM tree to fill this field. This is intended to support monitoring, algorithm design, and general experimentation with ALM in RELOAD.

number_active_trees: current number of trees that the node is part of

tree_data: details of each active tree; the number of such is specified by number_active_trees

impl_info: information about the implementation of this Usage

```
public struct {
  uint32       tree_id;
  uint8        algorithm;
  node_id      tree_root;
  uint8        number_parents;
  node_id      parent<0..2^8-1>;
  uint16       number_child_nodes;
  node_id      children<0..2^16-1>;
  uint32       path_length_to_root;
  uint32       path_delay_to_root;
  uint32       path_delay_to_child;
} TreeData;
```

tree_id: the ID of the tree

algorithm: code identifying the multicast algorithm used by this tree

tree_root: node_id of tree root, or 0 if unknown

number_parents: 0 .. 2^8-1 indicates number of parent nodes for this node

parent: the RELOAD node_id of each parent node

number_child_nodes: 0..2^16-1 indicates number of children

children: the RELOAD node_id of each child node

path_length_to_root: number of overlay hops to the root of the tree

path_delay_to_root: RTT in milliseconds to root node

path_delay_to_child: last measured RTT in milliseconds to child node with largest RTT

```
public struct {
  uint32       join_confirm_timeout;
  uint32       heartbeat_interval;
  uint32       heartbeat_response_timeout;
  uint16       info_length;
  uint8        info<0..2^16-1>;
} ImplementationInfo;
```

join_confirm_timeout: The default time for JoinConfirm/JoinDecline, intended to provide sufficient time for a Join request to receive all responses and confirm the best choice. Default value is 5000 msec. An implementation can change this value.

heartbeat_interval: The default Heartbeat interval is 2000 msec. Different interoperating implementations could use different intervals.

heartbeat_response_timeout: The default Heartbeat timeout is 5000 msec and is the max time between Heartbeat reports from an adjacent node in the tree at which point the Heartbeat is missed.

info_length: length of the info field

info: implementation-specific information, such as name of implementation, build version, and implementation-specific features

7.2.18. Push

A peer sends arbitrary multicast data to other peers in the tree. Nodes in the tree forward this message to adjacent nodes in the tree in an algorithm-dependent way.

```
struct {
  node_id group_id;
  uint8  priority;
  uint32 length;
  uint8  data<0..2^32-1>;
} Push;
```

group_id: overlay address of root of the ALM tree

priority: the relative priority of the message; highest priority is 255. A node may ignore this field.

length: length of the data field in bytes

data: the data

In pseudocode, the behavior of Push can be described as:

```
foreach(groups[msg.group_id].children as node_id)
     SEND(msg,node_id)
if memberOf(msg.group_id)
     invokeMessageHandler(msg.group_id, msg)
```

7.2.19. PushResponse

After receiving a Push message from node S, the receiving peer sends a PushResponse to node S.

```
struct {
  Dictionary options;
} PushResponse;
```

options: A node may provide feedback to the sender about previous Push messages in some window, for example, the last N Push messages. The feedback could include, for each Push message received, the number of adjacent nodes that were forwarded the Push message and the number of adjacent nodes from which a PushResponse was received.

8. Scribe Algorithm

8.1. Overview

Figure 3 shows a mapping between RELOAD ALM messages (as defined in Section 5 of this document) and Scribe messages as defined in [CASTRO2002].

Section	RELOAD ALM Message	Scribe Message
7.2.1	CreateALMTree	Create
7.2.3	Join	Join
7.2.4	JoinAccept	
7.2.6	JoinConfirm	
7.2.8	JoinDecline	
7.2.10	Leave	Leave
7.2.12	Reform	
7.2.14	Heartbeat	
7.2.16	NodeQuery	
7.2.18	Push	Multicast
	Note 1	deliver
	Note 1	forward
	Note 1	route
	Note 1	send

Figure 3: Mapping to Scribe Messages

Note 1: These Scribe messages are handled by RELOAD messages.

The following sections describe the Scribe algorithm in more detail.

8.2. Create

This message will create a group with group_id. This message MUST be delivered to the node whose node_id is closest to the group_id. This node becomes the rendezvous point and root for the new multicast tree. Groups MAY have multiple sources of multicast messages.

8.3. Join

To join a multicast tree, a node SHOULD send a Join request with the group_id as the key. This message gets routed by the overlay to the rendezvous point of the tree. If an intermediate node is already a forwarder for this tree, it SHOULD add the joining node as a child. Otherwise, the node SHOULD create a child table for the group and add the joining node. It SHOULD then send the Join request towards the rendezvous point terminating the Join request from the child.

To adapt the Scribe algorithm to the ALM Usage proposed here, after a Join request is accepted, a JoinAccept message MUST be returned to the joining node.

8.4. Leave

When leaving a multicast group, a node SHOULD change its local state to indicate that it left the group. If the node has no children in its table, it MUST send a Leave request to its parent, from where it SHOULD travel up the multicast tree and stop at a node that still has children remaining after removing the leaving node.

8.5. JoinConfirm

This message is not part of the Scribe protocol but is required by the basic protocol proposed in this document. Thus, the Usage MUST send this message to confirm a joining node accepting its parent node.

8.6. JoinDecline

Like JoinConfirm, this message is not part of the Scribe protocol. Thus, the Usage MUST send this message if a peer receiving a JoinAccept message wishes to decline it.

8.7. Multicast

A message to be multicast to a group MUST be sent to the rendezvous node from where it is forwarded down the tree. If a node is a member of the tree rather than just a forwarder, it SHOULD pass the multicast data up to the application.

9. P2PCast Algorithm

9.1. Overview

P2PCast [P2PCAST] creates a forest of related trees to increase load balancing. P2PCast is independent of the underlying P2P substrate. Its goals and approach are similar to SplitStream [SPLITSTREAM] (which assumes Pastry as the P2P overlay). In P2PCast, the content provider splits the stream of data into f stripes. Each tree in the forest of multicast trees is an (almost) full tree of arity f. These trees are conceptually separate: every node of the system appears once in each tree, with the content provider being the source in all of them. To ensure that each peer contributes as much bandwidth as it receives, every node is a leaf in all the trees except for one, in which the node will serve as an internal node (proper tree of this node). To reduce the complexity of the discussion that follows, the remainder of this section will assume that f = 2. However, the algorithm scales for any number f.

P2PCast distinguishes the following types of nodes:

- Incomplete Node: A node with less than f children in its proper stripe
- Only-Child Node: A node whose parent (in any multicast tree) is an incomplete node
- Complete Node: A node with exactly f children in its proper stripe
- Special Node: A single node that is a leaf in all multicast trees of the forest

9.2. Message Mapping

Figure 4 shows a mapping between RELOAD ALM messages (as defined in Section 5 of this document) and P2PCast messages as defined in [P2PCAST].

Section	RELOAD ALM Message	P2PCast Message
7.2.1	CreateALMTree	Create
7.2.3	Join	Join
7.2.4	JoinAccept	
7.2.6	JoinConfirm	
7.2.8	JoinDecline	
7.2.10	Leave	Leave
7.2.12	Reform	Takeon Substitute Search Replace Direct Update
7.2.14	Heartbeat	
7.2.16	NodeQuery	
7.2.18	Push	Multicast

Figure 4: Mapping to P2PCast Messages

The following sections describe the mapping of the P2PCast messages in more detail.

9.3. Create

This message will create a group with group_id. This message MUST be delivered to the node whose node_id is closest to the group_id. This node becomes the rendezvous point and root for the new multicast tree. The rendezvous point will maintain f subtrees.

9.4. Join

To join a multicast tree, a joining node N MUST send a Join request to a random node A already part of the tree. Depending on the type of A, the joining algorithm continues as follows:

- Incomplete Node: Node A will arbitrarily select for which tree it wants to serve as an internal node and adopt N in that tree. In the other tree, node N will adopt node A as a child (taking node A's place in the tree), thus becoming an internal node in the stripe that node A didn't choose.

- Only-Child Node: As this node has a parent that is an incomplete node, the joining node will be redirected to the parent node and will handle the request as detailed above.

- Complete Node: The contacted node A must be a leaf in the other tree. If node A is a leaf node in Stripe 1, node N will become an internal node in Stripe 1, taking the place of node A and adopting it at the same time. To find a place for itself in the other stripe, node N starts a random walk down the subtree rooted at the sibling of node A (if node A is the root and thus does not have siblings, node N is sent directly to a leaf in that tree), which ends as soon as node N finds an incomplete node or a leaf. In this case, node N is adopted by the incomplete node.

- Special Node: as this node is a leaf in all subtrees, the joining node MAY adopt the node in one tree and become a child in the other.

P2PCast uses defined messages for communication between nodes during reorganization. To use P2PCast in this context, these messages are encapsulated by the message type Reform. In doing so, the P2PCast message is to be included in the options parameter of Reform. The following reorganization messages are defined by P2PCast:

Takeon: To take another peer as a child

Substitute: To take the place of a child of some peer

Search: To obtain the child of a node in a particular stripe

Replace: Different from Substitute in that the calling node that makes a node its child sheds off a random child

Direct: To direct a node to its would-be parent

Update: A node sends its updated state to its children

To adapt the P2PCast algorithm to the ALM Usage proposed here, after a Join request is accepted, a JoinAccept message MUST be returned to the joining node (one for every subtree).

9.5. Leave

When leaving a multicast group, a node will change its local state to indicate that it left the group. Disregarding the case where the leaving node is the root of the tree, the leaving node must be complete or incomplete in its proper tree. In the other trees, the node is a leaf and can just disappear by notifying its parent. For the proper tree, if the node is incomplete, it is replaced by its child. However, if the node is complete, a gap is created that is filled by a random child. If this child is incomplete, it can simply fill the gap. However, if it is complete, it needs to shed a random child. This child is directed to its sibling, which sheds a random child. This process ripples down the tree until the next-to-last level is reached. The shed node is then taken as a child by the parent of the deleted node in the other stripe.

Again, for the reorganization of the tree, the Reform message type is used as defined in the previous section.

9.6. JoinConfirm

This message is not part of the P2PCast protocol but is required by the basic protocol defined in this document. Thus, the Usage MUST send this message to confirm a joining node accepting its parent node. As with Join and JoinAccept, this MUST be carried out for every subtree.

9.7. Multicast

A message to be multicast to a group MUST be sent to the rendezvous node from where it is forwarded down the tree by being split into k stripes. Each stripe is then sent via a subtree. If a receiving node is a member of the tree rather than just a forwarder, it MAY pass the multicast data up to the application.

10. Message Format

All messages are mapped to the RELOAD experimental message type. The mapping is shown in Figure 5. The message codes are listed in Section 14.2. The format of the body of a message is provided in [RELOAD].

Message	RELOAD Code Point
CreateALMTree	exp_a_req
CreateALMTreeResponse	exp_a_ans
Join	exp_a_req
JoinAccept	exp_a_ans
JoinReject	exp_a_ans
JoinConfirm	exp_a_req
JoinConfirmResponse	exp_a_ans
JoinDecline	exp_a_req
JoinDeclineResponse	exp_a_ans
Leave	exp_a_req
LeaveResponse	exp_a_ans
Reform	exp_a_req
ReformResponse	exp_a_ans
Heartbeat	exp_a_req
HeartbeatResponse	exp_a_ans
NodeQuery	exp_a_req
NodeQueryResponse	exp_a_ans
Push	exp_a_req
PushResponse	exp_a_ans

Figure 5: RELOAD Message Code Mapping

For Data Kind-IDs, the RELOAD specification [RELOAD] states: "Code points in the range 0xF0000001 to 0xFFFFFFFE are reserved for private use". ALM Usage Kind-IDs are defined in the private use range.

All ALM Usage messages map to the RELOAD Message Extension mechanism.

Code points for the Kinds defined in this document MUST NOT conflict with any defined code points for RELOAD. RELOAD defines exp_a_req and exp_a_ans for experimental purposes. This specification uses only these message types for all ALM messages. RELOAD defines the MessageContents data structure. The ALM mapping uses the fields as follows:

- message_code: exp_a_req for requests and exp_a_ans for responses
- message_body: contains one instance of ALMHeader followed by one instance of ALMMessageContents
- extensions: unused

10.1. ALMHeader Definition

```
struct {
   uint32           sam_token;
   uint16           alm_algorithm_id;
   uint8            version;
} ALMHeader;
```

The fields in ALMHeader are used as follows:

sam_token: The first four bytes identify this message as an ALM message. This field MUST contain the value 0xD3414D42 (the string "SAMB" with the high bit of the first byte set).

alm_algorithm_id: The ALM Algorithm ID of the ALM algorithm being used. Each multicast tree uses only one algorithm. Trees with different ALM algorithms can coexist and can share the same nodes. ALM Algorithm ID codes are defined in Section 14.1.

version: The version of the ALM protocol being used. This is a fixed-point integer between 0.1 and 25.4. This document describes version 1.0 with a value of 0xA.

10.2. ALMMessageContents Definition

```
struct {
   uint16       alm_message_code;
   opaque       alm_message_body;
} ALMMessageContents;
```

The fields in ALMMessageContents are used as follows:

alm_message_code: This indicates the message being sent. The message codes are listed in Section 14.2.

alm_message_body: The message body itself, represented as a variable-length string of bytes. The bytes themselves are dependent on the code value. See Sections 8 and 9, which describe the various ALM methods for the definitions of the payload contents.

10.3. Response Codes

Response codes are defined in Section 6.3.3.1 of [RELOAD]. This specification maps to RELOAD ErrorResponse as follows:

ErrorResponse.error_code = Error_Exp_A;

Error_info contains an ALMErrorResponse instance.

```
public struct {
   uint16   alm_error_code;
   opaque   alm_error_info<0..2^16-1>;
} ALMErrorResponse;
```

alm_error_code: The following error code values are defined. Numeric values for these are defined in Section 14.3.

Error_Unknown_Algorithm: The multicast algorithm is not known or not supported.

Error_Child_Limit_Reached: The maximum number of child nodes has been reached for this node.

Error_Node_Bandwidth_Reached: The overall data bandwidth limit through this node has been reached.

Error_Node_Conn_Limit_Reached: The total number of connections to this node has been reached.

Error_Link_Cap_Limit_Reached: The capacity of a link has been reached.

Error_Node_Mem_Limit_Reached: An internal memory capacity of the node has been reached.

Error_Node_CPU_Cap_Limit_Reached: An internal processing capacity of the node has been reached.

Error_Path_Limit_Reached: The maximum path length in hop count over the multicast tree has been reached.

Error_Path_Delay_Limit_Reached: The maximum path length in message delay over the multicast tree has been reached.

Error_Tree_Fanout_Limit_Reached: The maximum fanout of a multicast tree has been reached.

Error_Tree_Depth_Limit_Reached: The maximum height of a multicast tree has been reached.

Error_Other: A human-readable description is placed in the alm_error_info field.

11. Examples

All peers in the examples are assumed to have completed bootstrapping. "Pn" refers to peer N. "group_id" refers to a peer responsible for storing the ALMTree instance with group_id.

11.1. Create Tree

A node with "NODE-MATCH" rights sends a CreateALMTree request to the group_id node, which also has NODE-MATCH rights for its own address. The group_id node determines whether to create the new tree and, if so, performs a local StoreReq. If the CreateALMTree succeeds, the ALMTree instance can be retrieved using Fetch. An example message flow for creating a tree is depicted in Figure 6.

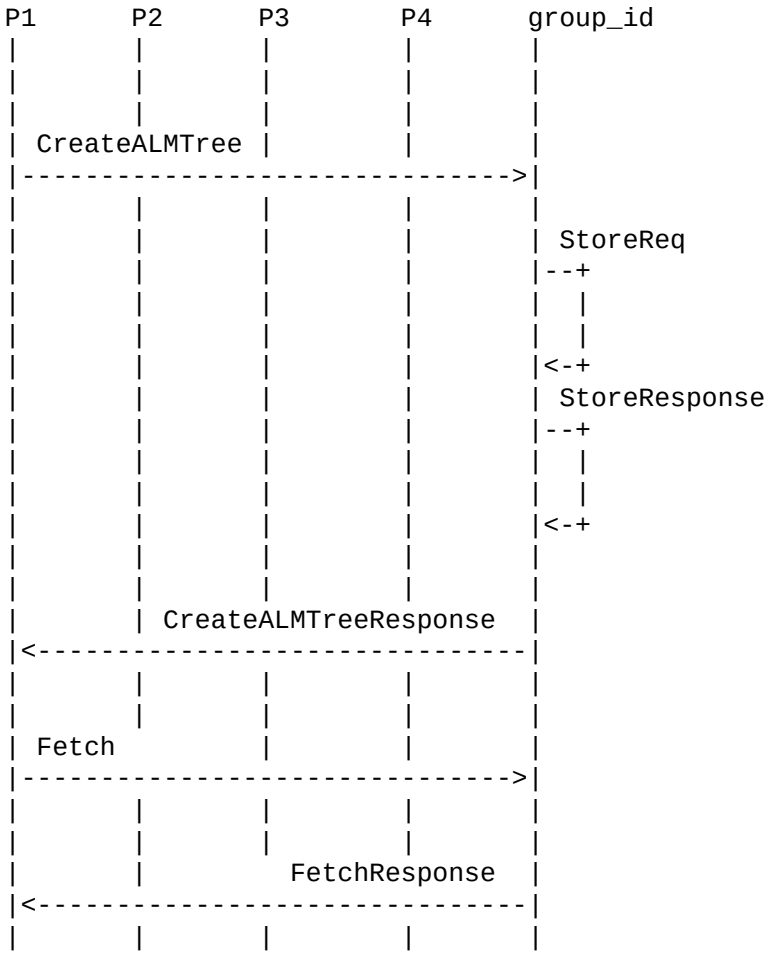

Figure 6: Message Flow Example for CreateALMTree

11.2. Join Tree

P1 joins node group_id as child node. P2 joins the tree as a child of P1. P4 joins the tree as a child of P1. The corresponding message flow is shown in Figure 7.

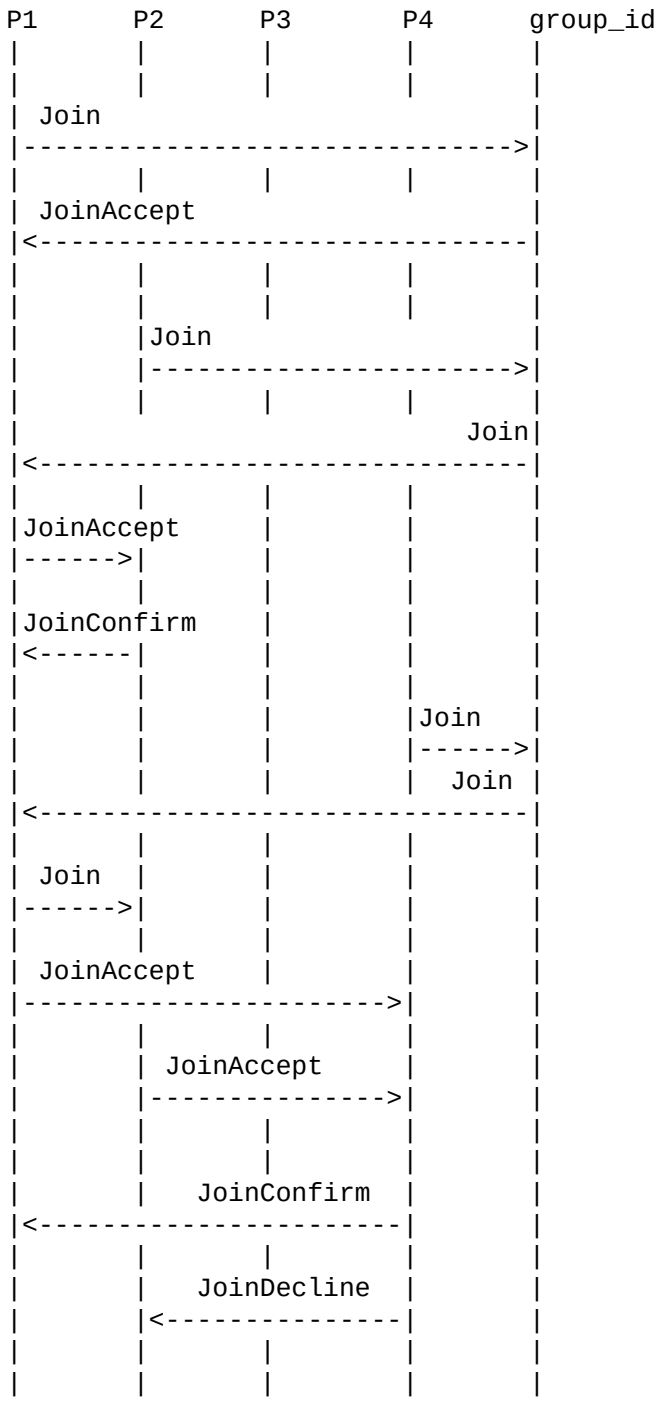

Figure 7: Message Flow Example for Tree Join

11.3. Leave Tree

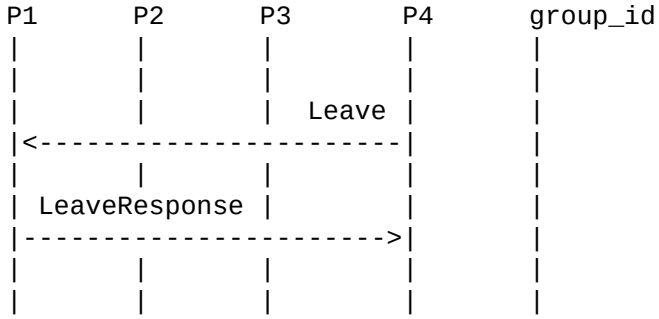

Figure 8: Message Flow Example for Leave Tree

11.4. Push Data

The multicast data is pushed recursively P1 => group_id => P1 => P2, P4 following the tree topology created in the Join example above. An example message flow is shown in Figure 9.

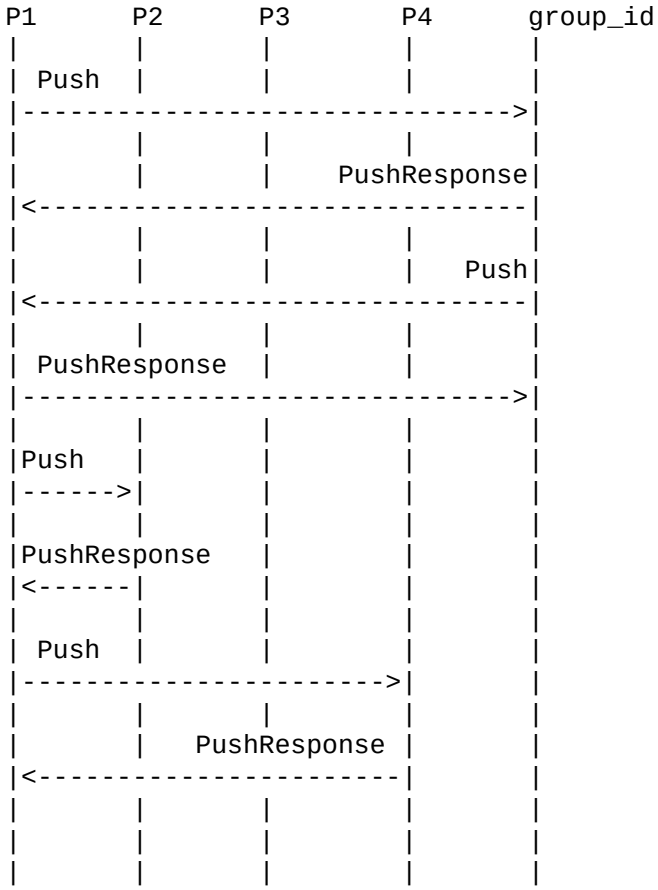

Figure 9: Message Flow Example for Pushing Data

12. Kind Definitions

12.1. ALMTree Kind Definition

This section defines the ALMTree Kind per Section 7.4.5 of [RELOAD]. An instance of the ALMTree Kind is stored in the overlay for each ALM tree instance. It is stored at the address group_id.

Kind-ID: 0xF0000001. (This is a private-use code point per Section 14.6 of [RELOAD].) The Resource Name for the ALMTree Kind-ID is the session_key used to identify the ALM tree.

Data Model: The data model is the ALMTree structure.

Access Control: NODE-MATCH. The node performing the store operation is required to have NODE-MATCH access.

Meaning: The meaning of the fields is given in Section 7.2.1.

```
struct {
  node_id peer_id;
  opaque session_key<0..2^32-1>;
  node_id group_id;
  Dictionary options;
} ALMTree;
```

13. RELOAD Configuration File Extensions

There are no ALM parameters defined for the RELOAD configuration file.

14. IANA Considerations

This section contains the new code points registered by this document.

14.1. ALM Algorithm Types

IANA has created the "SAM ALM Algorithm IDs" registry. Entries in this registry are 16-bit integers denoting Application-Layer Multicast algorithms as described in Section 10.1 of this document. Code points in the range 0x0003 to 0x7FFF SHALL be registered via RFC 5226 [RFC5226] Expert Review. Code points in the range 0x8000 to 0xFFFF are reserved for private use. The initial contents of this registry are:

Algorithm Name	ALM Algorithm ID	RFC
INVALID-ALG	0x0000	RFC 7019
SCRIBE-SAM	0x0001	RFC 7019
P2PCAST-SAM	0x0002	RFC 7019
Reserved	0x8000-0xFFFF	RFC 7019

Figure 10: "SAM ALM Algorithm IDs" Registry Allocations

These values have been made available for the purposes of experimentation. These values are not meant for vendor-specific use of any sort and MUST NOT be used for operational deployments.

14.2. Message Code Registration

IANA has created the "SAM ALM Message Codes" registry. Entries in this registry are 16-bit integers denoting message codes as described in Section 10.2 of this document. Code points in the range 0x0014 to 0x7FFF SHALL be registered via RFC 5226 [RFC5226] Expert Review. Code points in the range 0x8000 to 0xFFFF are reserved for private use. The initial contents of this registry are:

Message Code Name	Message Code Value	RFC
InvalidMessageCode	0x0000	RFC 7019
CreateALMTree	0x0001	RFC 7019
CreateALMTreeResponse	0x0002	RFC 7019
Join	0x0003	RFC 7019
JoinAccept	0x0004	RFC 7019
JoinReject	0x0005	RFC 7019
JoinConfirm	0x0006	RFC 7019
JoinConfirmResponse	0x0007	RFC 7019
JoinDecline	0x0008	RFC 7019
JoinDeclineResponse	0x0009	RFC 7019
Leave	0x000A	RFC 7019
LeaveResponse	0x000B	RFC 7019
Reform	0x000C	RFC 7019
ReformResponse	0x000D	RFC 7019
Heartbeat	0x000E	RFC 7019
HeartbeatResponse	0x000F	RFC 7019
NodeQuery	0x0010	RFC 7019
NodeQueryResponse	0x0011	RFC 7019
Push	0x0012	RFC 7019
PushResponse	0x0013	RFC 7019
Reserved	0x8000-0xFFFF	RFC 7019

Figure 11: "SAM ALM Message Codes" Registry Allocations

These values have been made available for the purposes of experimentation. These values are not meant for vendor-specific use of any sort and MUST NOT be used for operational deployments.

14.3. Error Code Registration

IANA has created the "SAM ALM Error Codes" registry. Entries in this registry are 16-bit integers denoting error codes as described in Section 10.3 of this document. Code points in the range 0x000D to

0x7FFF SHALL be registered via RFC 5226 [RFC5226] Expert Review. Code points in the range 0x8000 to 0xFFFF are reserved for private use. The initial contents of this registry are:

Error Code Name	Code Value	RFC
InvalidErrorCode	0x0000	RFC 7019
Error_Unknown_Algorithm	0x0001	RFC 7019
Error_Child_Limit_Reached	0x0002	RFC 7019
Error_Node_Bandwidth_Reached	0x0003	RFC 7019
Error_Node_Conn_Limit_Reached	0x0004	RFC 7019
Error_Link_Cap_Limit_Reached	0x0005	RFC 7019
Error_Node_Mem_Limit_Reached	0x0006	RFC 7019
Error_Node_CPU_Cap_Limit_Reached	0x0007	RFC 7019
Error_Path_Limit_Reached	0x0008	RFC 7019
Error_Path_Delay_Limit_Reached	0x0009	RFC 7019
Error_Tree_Fanout_Limit_Reached	0x000A	RFC 7019
Error_Tree_Depth_Limit_Reached	0x000B	RFC 7019
Error_Other	0x000C	RFC 7019
Reserved	0x8000-0xFFFF	RFC 7019

Figure 12: "SAM ALM Error Codes" Registry Allocations

These values have been made available for the purposes of experimentation. These values are not meant for vendor-specific use of any sort and MUST NOT be used for operational deployments.

15. Security Considerations

Overlays are vulnerable to DoS and collusion attacks. We are not solving overlay security issues. We assume that the node authentication model as defined in [RELOAD] will be used.

Security issues specific to ALM Usage include the following:

- The right to create group_id at some node_id
- The right to store Tree info at some location in the DHT
- A limit on number of messages per second and bandwidth use
- The right to join an ALM tree

16. Acknowledgements

Marc Petit-Huguenin, Michael Welzl, Joerg Ott, and Lars Eggert provided important comments on earlier versions of this document.

17. References

17.1. Normative Reference

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

17.2. Informative References

[BUFORD2008] Buford, J. and H. Yu, "P2P: Overlay Multicast", Encyclopedia of Wireless and Mobile Communications, 2008, <http://www.tandfonline.com/doi/abs/10.1081/E-EWMC-120043583>.

[BUFORD2009] Buford, J., Yu, H., and E. Lua, "P2P Networking and Applications (Chapter 9)", Morgan Kaufman, 2009, <http://www.sciencedirect.com/science/book/9780123742148>.

[CASTRO2002] Castro, M., Druschel, P., Kermarrec, A., and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications, Vol. 20, No. 8, October 2002, <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1038579>.

[CASTRO2003] Castro, M., Jones, M., Kermarrec, A., Rowstron, A., Theimer, M., Wang, H., and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays", Proceedings of IEEE INFOCOM 2003, April 2003, <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1208986>.

[COMMON-API] Waehlisch, M., Schmidt, T., and S. Venaas, "A Common API for Transparent Hybrid Multicast", Work in Progress, April 2013.

[KOLBERG2010] Kolberg, M., "Employing Multicast in P2P Overlay Networks", Handbook of Peer-to-Peer Networking, 2010, <http://link.springer.com/content/pdf/10.1007%2F978-0-387-09751-0_30.pdf>.

[P2PCAST] Nicolosi, A. and S. Annapureddy, "P2PCast: A Peer-to-Peer Multicast Scheme for Streaming Data", Stanford Secure Computer Systems Group Report, May 2003, <http://www.scs.stanford.edu/~reddy/research/p2pcast/report.pdf>.

[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.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[SAM-GENERIC] Muramoto, E., Imai, Y., and N. Kawaguchi, "Requirements for Scalable Adaptive Multicast Framework in Non-GIG Networks", Work in Progress, November 2006.

[SPLITSTREAM] Castro, M., Druschel, P., Nandi, A., Kermarrec, A., Rowstron, A., and A. Singh, "SplitStream: High-Bandwidth Multicast in a Cooperative Environment", SOSP '03, Lake Bolton, New York, October 2003, <http://dl.acm.org/citation.cfm?id=945474>.

Authors' Addresses

John Buford
Avaya Labs Research
211 Mt. Airy Rd.
Basking Ridge, New Jersey 07920
USA

Phone: +1 908 848 5675
EMail: buford@avaya.com

Mario Kolberg (editor)
University of Stirling
Dept. of Computing Science and Mathematics
Stirling FK9 4LA
UK

Phone: +44 1786 46 7440
EMail: mkolberg@ieee.org
URI: http://www.cs.stir.ac.uk/~mko