SAM Research Group
Internet-Draft
Intended status: Informational
Expires: February 6, 2013
J. Buford
Avaya Labs Research
M. Kolberg, Ed.
University of Stirling
August 05, 2012

Application Layer Multicast Extensions to RELOAD
draft-irtf-samrg-sam-baseline-protocol-00

Abstract

We define a RELOAD Usage for Application Layer Multicast as well as extensions to RELOAD message layer to support ALM. The ALM Usage is intended to support a variety of ALM control algorithms in an overlay-independent way. Scribe is defined as an example algorithm.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction 4
   1.1. Requirements Language 4
2. Definitions 5
   2.1. Overlay Network 5
   2.2. Overlay Multicast 5
   2.3. Peer 5
3. Assumptions 6
   3.1. Overlay 6
   3.2. Overlay Multicast 6
   3.3. RELOAD 6
   3.4. NAT 7
   3.5. Tree Topology 7
4. Architecture Extensions to RELOAD 7
5. RELOAD ALM Usage 9
6. ALM Tree Control Signaling 9
7. ALM Messages Added to RELOAD Protocol 11
   7.1. Introduction 11
   7.2. Tree Lifecycle Messages 12
      7.2.1. Create Tree 12
      7.2.2. Join 13
      7.2.3. Join Accept 13
      7.2.4. Join Confirm 14
      7.2.5. Join Decline 14
      7.2.6. Leave 14
      7.2.7. Re-Form or Optimize Tree 15
      7.2.8. Heartbeat 15
8. Scribe Algorithm 16
   8.1. Overview 16
   8.2. Create 17
   8.3. Join 18
   8.4. Leave 18
   8.5. JoinConfirm 18
   8.6. JoinDecline 19
   8.7. Multicast 19
9. P2PCast Algorithm 19
   9.1. Overview 19
   9.2. Create 20
   9.3. Join 20
   9.4. Leave 21
   9.5. JoinConfirm 21
   9.6. JoinDecline 21
   9.7. Multicast 22
10. Examples 22
   10.1. Create Tree 22
   10.2. Join Tree 23
   10.3. Leave Tree 24

10.4. Add Direct Application Edge 24
10.5. Adjust Tree to Churn 24
10.6. Push Data 24
11. Kind Definitions 24
11.1. ALMTree Kind Definition 24
12. Configuration File Extensions 24
13. Change History 25
14. Open Issues 25
15. IANA Considerations 25
16. Security Considerations 27
17. Acknowledgement 27
18. References 27
18.1. Normative References 27
18.2. Informative References 28
Appendix A. Additional Stuff 30
Authors' Addresses 30

1. Introduction

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

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

Adaptability includes

- o use of different control mechanisms for different multicast trees depending on initial application parameters or application class
- o 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:

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

RELOAD [I-D.ietf-p2psip-base] 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.

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 2 of [I-D.ietf-p2psip-base], specifically the distinction between Node, Peer, and Client.

2.1. Overlay Network

```
                    P    P    P   P     P
                  ..+....+....+...+.....+...
                 .                          +P
              P+                              .
                 .                          +P
                  ..+....+....+...+.....+...
                    P    P    P   P     P
```

Figure 1

Overlay network - An application layer virtual or logical network in which end points are addressable and 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 the above figure, "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.

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. 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 which use proxies connected in an overlay multicast backbone "proxied overlay multicast" or POM.

2.3. 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, 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 out-performed using separate overlays for each multicast session. We use a single overlay rather than separate overlays per multicast sessions.

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 which is closer to some of the entries than the current peer which 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 re-join operations for dislocated children nodes.

3.2. Overlay Multicast

The overlay supports concurrent multiple multicast trees. The limit on 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 [I-D.ietf-p2psip-base] as the distibuted hash table (DHT) for data storage and overlay 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, 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, shown in the figure below. New ALM messages are added to RELOAD Message Transport. A plug-in for ALM algorithms handles the ALM state and control. The ALM Algorithm is under control of the application via the Group API [I-D.irtf-samrg-common-api].

```
                                                    +---------+
                                                    |Group API|
                                                    +---------+
                                                         |
       ------------------- Application  ------------------------
           +-------+                                     |
           | ALM   |                                     |
           | Usage |                                     |
           +-------+                                     |
        -------------- Messaging Service Boundary --------------
                                                         |
          +--------+      +-----------+---------+    +---------+
          | Storage|<---> | RELOAD    | ALM     |<-->| ALM Alg |
          +--------+      | Message   | Messages|    +---------+
                  ^       | Transport |         |
                  |       +-----------+---------+
                  v          |    |
                 +-------------+  |
                 | Topology    |  |
                 | Plugin      |  |
                 +-------------+  |
                    ^             |
                    v             v
                 +-------------------+
                 | Forwarding&       |
                 | Link Management   |
                 +-------------------+

        ---------- Overlay Link Service Boundary --------------
```

Figure 2

The ALM components interact with RELOAD as follows:

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

- o ALM applications and management tools may use the RELOAD data storage functionality to store diagnostic information about the operation of tree, including average number of tree, delay from source to leaf nodes, bandwidth use, lost packet 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 be can 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 which require ALM functionality in the overlay. The figure below shows the internal structure of the ALM Usage. This contains the Group API ([I-D.irtf-samrg-common-api]) an ALM algorithm plugin (e.g. Scribe) and the ALM messages which are then sent out to the RELOAD network.

A RELOAD Usage is required [I-D.ietf-p2psip-base] to define the following:

o Register Kind-Id points

o Define data structures for each kind

o Defines access control rules for each kind

o Defines the Resource Name used to hash to the Resource ID where the kind is stored

o Addresses restoration of values after recovery from a network partition

o Defines the types of connections that can be initiated using AppConnect

A ALM GroupID is a RELOAD Node-ID. The owner of a ALM group creates a RELOAD Node-ID as specified in [I-D.ietf-p2psip-base]. This means that a GroupID 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:

o Create tree

o Join

- Leave

- Re-Form or optimize tree

There are a variety of algorithms for peers to form multicast trees in the overlay. We permit multiple such algorithms to be supported in the overlay, since different algorithms may be more suitable for certain application requirements, and since we wish to support 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 algorithm for forming a multicast tree in the overlay. Its main advantage is use of the overlay routing mechanism for routing both control and data messages. The group creator doesn't have to be the root of the tree or even in the tree. It doesn't consider per node load, admission control, or alternative paths.

As stated earlier, multiple algorithms will co-exist in the overlay.

1. Peer which initiates multicast group:

```
groupID = create();  // allocate a unique groupId
                     // the root is the nearest
                     // peer in the overlay
                     // out of band advertisement or
                     // distribution of groupID,
                     // perhaps by publishing in DHT
```

Figure 3

2. Any joining peer:

```
// out of band discovery of groupID, perhaps by lookup in DHT
joinTree(groupID); // sends "join groupID" message
```

Figure 4

The overlay routes the join request using the overlay routing

mechanism toward the peer with the nearest id to the groupID. This peer is the root. Peers on the path to the root join the tree as forwarding points.

3. Leave Tree:

   leaveTree(groupID) // removes this node from the tree

   Propagates a leave message 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 message to its parent. A child node receiving a leave message from a parent sends a join message to the groupID.

4. Message forwarding:

   multicastMsg(groupID, msg);

5. For the message forwarding there are two approaches:

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

   * 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 remaining tree edges it is connected to.

7. ALM Messages Added to RELOAD Protocol

7.1. Introduction

In this document we define messages for overlay multicast tree creation, using an existing proposal (RELOAD) in the P2P-SIP WG [I-D.ietf-p2psip-base] 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 [I-D.irtf-sam-hybrid-overlay-framework] (hereafter SAM framework) can be used with P2P-SIP, and that the SAM framework is overlay agnostic.

As discussed in the SAM requirements draft, 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.

7.2. Tree Lifecycle Messages

Peers use the overlay to transmit ALM (application layer multicast) operations defined in this section.

7.2.1. Create Tree

A new ALM tree is created in the overlay with the identity specified by GroupId. The usual interpretation of GroupId is that the peer with peer id closest to and less than the GroupId is the root of the tree. The tree has no children at the time it is created.

The GroupId 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 GroupId from the SessionKey MUST be done using the overlay's id generation mechanism.

A successful Create Tree causes an ALMTree structure to be stored in the overlay at the node responsible for NodeID equal to the GroupId.

```
struct {
  NodeID PeerId;
  opaque SessionKey<0..2^32-1>;
  NodeID GroupId;
  Dictionary Options;
} ALMTree;
```

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

SessionKey: a well-known string when hashed using the overlay's id generation algorithm produces the GroupId.

GroupId: the 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, heartbeat interval.

Tree creation is subject to access control since it involves an Store operation. Before the Store of an ALMTree structure is permitted, the storing peer MUST check that:

- o The certificate contains a SessionKey

- o The certificate contains a Node-ID that is the same as GroupID that it is being stored at Node-ID (this is the NODE-MATCH access

policy)

7.2.2. Join

Causes the distributed algorithm for peer join of a specific ALM group to be invoked. If successful, the PeerId 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 {
  NodeID PeerId;
  NodeID GroupId;
  Dictionary Options;
} Join;
```

PeerId: overlay address of joining/leaving peer

GroupId: the overlay address of the root of the tree

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

7.2.3. Join Accept

Tells the requesting joining peer that the indicated peer is available to act as its parent in the ALM tree specified by GroupId, with the corresponding Options specified. A peer MAY receive more than one JoinAccept from different candidate parent peers in the GroupId tree. The peer accepts a peer as parent using a JoinConfirm message. A JoinAccept which receives neither a JoinConfirm or JoinDecline response MUST expire.

```
struct {
  NodeID ParentPeerId;
  NodeID ChildPeerId;
  NodeID GroupId;
  Dictionary Options;
} JoinAccept;
```

ParentPeerId: overlay address of a peer which accepts the joining peer

ChildPeerId: overlay address of joining peer

GroupId: the overlay address of the root of the tree

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

7.2.4. Join Confirm

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

```
struct {
  NodeID ChildPeerId;
  NodeID ParentPeerId;
  NodeID GroupId;
  Dictionary Options;
} JoinConfirm;
```

ChildPeerId: overlay address of joining peer which is a child of the parent peer

ParentPeerId: overlay address of the peer which is the parent of the joining peer

GroupId: the overlay address of the root of the tree

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

7.2.5. Join Decline

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

```
struct {
  NodeID PeerId;
  NodeID ParentPeerId;
  NodeID GroupId;
} JoinDecline;
```

PeerId: overlay address of joining peer which declines the JoinAccept

ParentPeerId: overlay address of the peer which issued a JoinAccept to this peer

GroupId: the overlay address of the root of the tree

7.2.6. Leave

A peer which is part of an ALM tree identified by GroupId which intends to detach from either a child or parent peer SHOULD send a Leave message to the peer it wishes to detach from. A peer receiving

a Leave message from a peer which is neither in its parent or child lists SHOULD ignore the message.

```
struct {
  NodeID PeerId;
  NodeID GroupId;
  Dictionary Options;
} Leave;
```

PeerId: overlay address of leaving peer

GroupId: the overlay address of the root of the tree

Options: name-value list of options

7.2.7. Re-Form or Optimize Tree

This triggers a reorganization of either the entire tree or only a sub-tree. It MAY include hints to specific peers of recommended parent or child peers to reconnect to. 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 {
  NodeID GroupId;
  NodeID PeerId;
  Dictionary Options;
} Reform;
```

GroupId: the overlay address of the root of the tree

PeerId: if omitted, then the tree is reorganized starting from the root, otherwise it is reorganized only at the sub-tree identified by PeerId.

Options: name-value list of options

7.2.8. Heartbeat

A node signals to its adjacent nodes in the tree that it is alive. If a peer does not receive a Heartbeat message within N heartbeat time intervals, it MUST treat this as an explicit Leave message from the unresponsive peer. N is configurable.

```
struct {
  NodeID PeerId1;
  NodeID PeerId2;
  NodeID GroupId;
} Heartbeat;
```

PeerId1: source of heartbeat

PeerId2: destination of heartbeat

GroupId: overlay address of the root of the tree

8. Scribe Algorithm

8.1. Overview

The following table shows a mapping between RELOAD ALM messages (as defined in Section 5 of this draft) and Scribe messages as defined in [CASTRO2002].

Section in Draft	RELOAD ALM Message	Scribe Message
5.2.1	CreateALMTree	Create
5.2.2	Join	Join
5.2.3	JoinAccept	
5.2.4	JoinConfirm	
5.2.5	JoinDecline	
5.2.8	Leave	Leave
5.2.10	Reform	
5.2.11	Heartbeat	
new	Push/Deliver/Send	Multicast
	Note 1	deliver
	Note 1	forward
	Note 1	route
	Note 1	send

Figure 5

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 GroupId. This message will be delivered to the node whose NodeId is closest to the GroupId. This node becomes the rendezvous point and root for the new multicast tree. Groups may have multiple sources of multicast messages.

CREATE : groups.add(msg.GroupId)

GroupId: the overlay address of the root of the tree

8.3. Join

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

To adapt the Scribe algorithm into the ALM Usage proposed here, after a JOIN request is accepted, a JOINAccept message is returned to the joining node.

```
JOIN : if(checkAccept(msg)) {
                    recvJoins.add(msg.source, msgGroupId)
                    SEND(JOINAccept(nodeID, msg.source, msg.GroupId))
                }
```

8.4. Leave

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

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

8.5. JoinConfirm

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

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

8.6. JoinDecline

```
JOINDecline: if(recvJoins.contains(msg.source,msg.GroupId))
                         recvJoins.del(msg.source, msgGroupId)
```

8.7. Multicast

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

```
MULTICAST : foreach(groups[msg.GroupId].children as NodeId)
                   SEND(msg,NodeId)
            if memberOf(msg.GroupId)
                   invokeMessageHandler(msg.GroupId, msg)
```

9. P2PCast Algorithm

9.1. Overview

P2PCast [P2PCAST]creates a forest of related trees to increase load balancing. P2PCast is independent on 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). The remainder of this section will assume f=2 for the discussion. This is to keep the complexity for the description down. However, the algorithm scales for any number f.

P2PCast distinguishes the following types of nodes:

- o Incomplete Nodes: A node with less than f children in its proper stripe;

- o Only-Child Nodes: A node whose parent (in any multicast tree) is an incomplete node;

- o Complete Nodes: A node with exactly f children in its proper stripe

- o Special Node: A single node which is a leaf in all multicast trees of the forest

9.2. Create

This message will create a group with group_id. This message will 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.

9.3. Join

To join a multicast tree a joining node N sends a JOIN request to a random node A already part of the tree. Depending of the type of A the joining algorithm continues as follows:

- o Incomplete Nodes: 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 N will adopt A as a child (taking A's place in the tree) thus becoming an internal node in the stripe that A didn't choose.

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

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

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

P2PCast uses defined messages for communication between nodes during reorganisation. Here these messages are encapsulated by the message type REFORM is used. The P2PCast message is included in the Options parameter of REFORM. The following 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 node which makes us its child sheds off a random child

DIRECT: To direct a node to its wouldbe parent

UPDATE: A node sends its updated state to its children

To adapt the P2PCast algorithm into the ALM Usage proposed here, after a JOIN request is accepted, a JOINAccept message is returned to the joining node (one for every subtree).

9.4. Leave

When leaving a multicast group a node will change its local state to indicate that it left the group. Distregarding 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 bubble is created which 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 reorganisation of the tree, the REFORM message type is used as defined in the previous section.

9.5. JoinConfirm

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

9.6. JoinDecline

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

9.7. Multicast

A message to be multicast to a group is 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 will pass the multicast data up to the application.

10. Examples

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

10.1. Create Tree

```
     P1      P2      P3       P4      GroupID
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
     | CreateTree    |        |       |
     |------------------------------->|
     |       |       |        |       |
     |       |       |        |       |
     |       |    CreateTreeResponse  |
     |<-------------------------------|
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
     |       |       |        |       |
```

Figure 6

10.2. Join Tree

```
        P1      P2      P3       P4      GroupID
        |       |       |        |       |
        |       |       |        |       |
        | Join                           |
        |------------------------------->|
        |       |       |        |       |
        | JoinAccept                     |
        |<-------------------------------|
        |       |       |        |       |
        |       |       |        |       |
        |       |Join                    |
        |       |----------------------->|
        |       |       |        |       |
        |                            Join|
        |<-------------------------------|
        |       |       |        |       |
        |JoinAccept     |        |       |
        |------>|       |        |       |
        |       |       |        |       |
        |JoinConfirm    |        |       |
        |<------|       |        |       |
        |       |       |        |       |
        |       |       |        |Join   |
        |       |       |        |------>|
        |       |       |        |  Join |
        |<-------------------------------|
        |       |       |        |       |
        | Join  |       |        |       |
        |------>|       |        |       |
        |       |       |        |       |
        | JoinAccept    |        |       |
        |----------------------->|       |
        |       |       |        |       |
        |       | JoinAccept     |       |
        |       |--------------->|       |
        |       |       |        |       |
        |       |       |        |       |
        |       |   Join Confirm |       |
        |<-----------------------|       |
        |       |       |        |       |
        |       |   Join Decline |       |
        |       |<---------------|       |
        |       |       |        |       |
        |       |       |        |       |
```

Figure 7

10.3. Leave Tree

```
        P1      P2      P3       P4      GroupID
        |       |       |        |       |
        |       |       |        |       |
        |       |       |  Leave |       |
        |<-----------------------|       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
        |       |       |        |       |
```

Figure 8

10.4. Add Direct Application Edge

10.5. Adjust Tree to Churn

10.6. Push Data

11. Kind Definitions

11.1. ALMTree Kind Definition

This section defines the ALMTree kind.

Kind IDs The Resource Name for the ALMTree Kind-ID is the SessionKey used to identify the ALM tree

Data Model The data model is the ALMTree structure.

Access Control NODE-MATCH

12. Configuration File Extensions

In RELOAD, peers receive a configuration document at bootstrap time. ALM parameter definitions for the configuration file will be defined in a later version.

13. Change History

- Version 02: Remove Hybrid ALM material. Define ALMTree kind. Define new RELOAD messages. Define RELOAD architecture extensions. Add Scribe as base algorithm for ALM usage. Define code points. Define preliminary ALM-specific security issues.

- Version 03: Add P2Pcast Algorithm.

14. Open Issues

- The specific capabilities of clients in terms of tree creation and being parents of other nodes will be described in subsequent versions.

- ALM parameter definitions for the RELOAD configuration file will be defined in a later version.

- Should any other ALM algorithms be mapped

-

15. IANA Considerations

This memo includes no request to IANA.

Message codes

Message	RELOAD Code Point	ALM Message Code
CreateALMTRee	35	00
CreateALMTreeResponse	36	01
Join	36	02
JoinAccept	36	03
JoinReject	36	04
JoinConfirm	36	05
JoinDecline	36	06
Leave	36	07
LeaveResponse	36	08
Reform	36	09
ReformResponse	36	x0A
Heartbeat	36	x0B
Push	36	x0C
PushResponse	36	x0D

Figure 9

Code points for the kinds defined in this document MUST not conflict with any defined code points for RELOAD. RELOAD defines exp_a_req, exp_a_ans for experimental purposes. This specification uses only these message types for all ALM messages, with a sub-type to distinguish the specific ALM message For Data Kind-IDs, the RELOAD specification states: "Code points in the range 0xf0000001 to 0xfffffffe are reserved for private use". ALM Usage Kind-IDs will be defined in the private use range.

All ALM Usage messages support the RELOAD Message Extension mechanism.

No new Error Codes are defined. RELOAD defines Error_Exp_A and Error_Exp_B. This will be used if new error codes are needed.

Application-ID: The ALM Usage Application-IDs must not conflict with other applications of reload. Additionally if AppAttach is used, the port number must be selected to avoid conflicts.

Access Control Policies: No new policies.

ALM Algorithm Types: There are currently two types: SCRIBE-RELOAD, P2PCAST-RELOAD.

16. Security Considerations

Overlays are vulnerable to DOS and collusion attacks. We are not solving overlay security issues. We assume the node authentication model as defined in [I-D.ietf-p2psip-base].

ALM Usage specific security issues:

- Right to create GroupID at some NodeId
- Right to store Tree info at some Location in the DHT
- Limit on # messages / sec and bandwidth use
- Right to join an ALM tree
-

17. Acknowledgement

Marc Petit-Huguenin provided important comments on earlier version of this draft.

18. References

18.1. Normative References

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

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

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

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

[RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.

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

18.2. Informative References

[AGU1984] Aguilar, L., "Datagram Routing for Internet Multicasting", ACM Sigcomm 84 1984, March 1984, <http://dl.acm.org/citation.cfm?id=802060>.

[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://research.microsoft.com/en-us/um/people/antr/past/jsac.pdf>.

[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://research.microsoft.com/en-us/um/people/mcastro/publications/infocom-compare.pdf>.

[HE2005] He, Q. and M. Ammar, "Dynamic Host-Group/Multi-Destination Routing for Multicast Sessions", J. Telecommunication Systems vol. 28, pp. 409-433, 2005, <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1284204&abstractAccess=no&userType=inst>.

[I-D.ietf-mboned-auto-multicast] Bumgardner, G., "Automatic Multicast Tunneling", draft-ietf-mboned-auto-multicast-14 (work in progress), June 2012.

[I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-22 (work in progress), July 2012.

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

[I-D.irtf-p2prg-rtc-security]
Schulzrinne, H., Marocco, E., and E. Ivov, "Security Issues and Solutions in Peer-to-peer Systems for Realtime Communications", draft-irtf-p2prg-rtc-security-05 (work in progress), September 2009.

[I-D.irtf-sam-hybrid-overlay-framework]
Buford, J., "Hybrid Overlay Multicast Framework", draft-irtf-sam-hybrid-overlay-framework-02 (work in progress), February 2008.

[I-D.irtf-samrg-common-api]
Waehlisch, M., Schmidt, T., and S. Venaas, "A Common API for Transparent Hybrid Multicast", draft-irtf-samrg-common-api-06 (work in progress), August 2012.

[I-D.matuszewski-p2psip-security-overview]
Yongchao, S., Matuszewski, M., and D. York, "P2PSIP Security Overview and Risk Analysis", draft-matuszewski-p2psip-security-overview-01 (work in progress), October 2009.

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

[RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, March 1996.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552,

July 2003.

[RFC4286] Haberman, B. and J. Martin, "Multicast Router Discovery", RFC 4286, December 2005.

[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 2003, October 2003, <http://research.microsoft.com/en-us/um/people/antr/PAST/SplitStream-sosp.pdf>.

Appendix A. Additional Stuff

This becomes an Appendix.

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. 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