SAM Research Group
Internet-Draft
Intended status: Informational
Expires: August 2, 2013

J. Buford
Avaya Labs Research
M. Kolberg, Ed.
University of Stirling
January 29, 2013

Application Layer Multicast Extensions to RELOAD
draft-irtf-samrg-sam-baseline-protocol-02

Abstract

We define a RELOAD Usage for Application Layer Multicast 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.

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 connect 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 [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

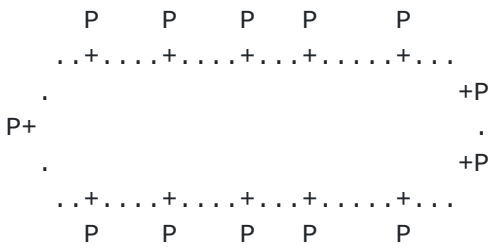

Figure 1: Overlay Network Example

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 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.

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. Source Specific Multicast (SSM)

SSM tree: The creator of the tree is the source. It sends data messages to the tree root which 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 remaining tree edges it is connected to.

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 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 Peer-to-Peer overlay for data storage and 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, 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 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].

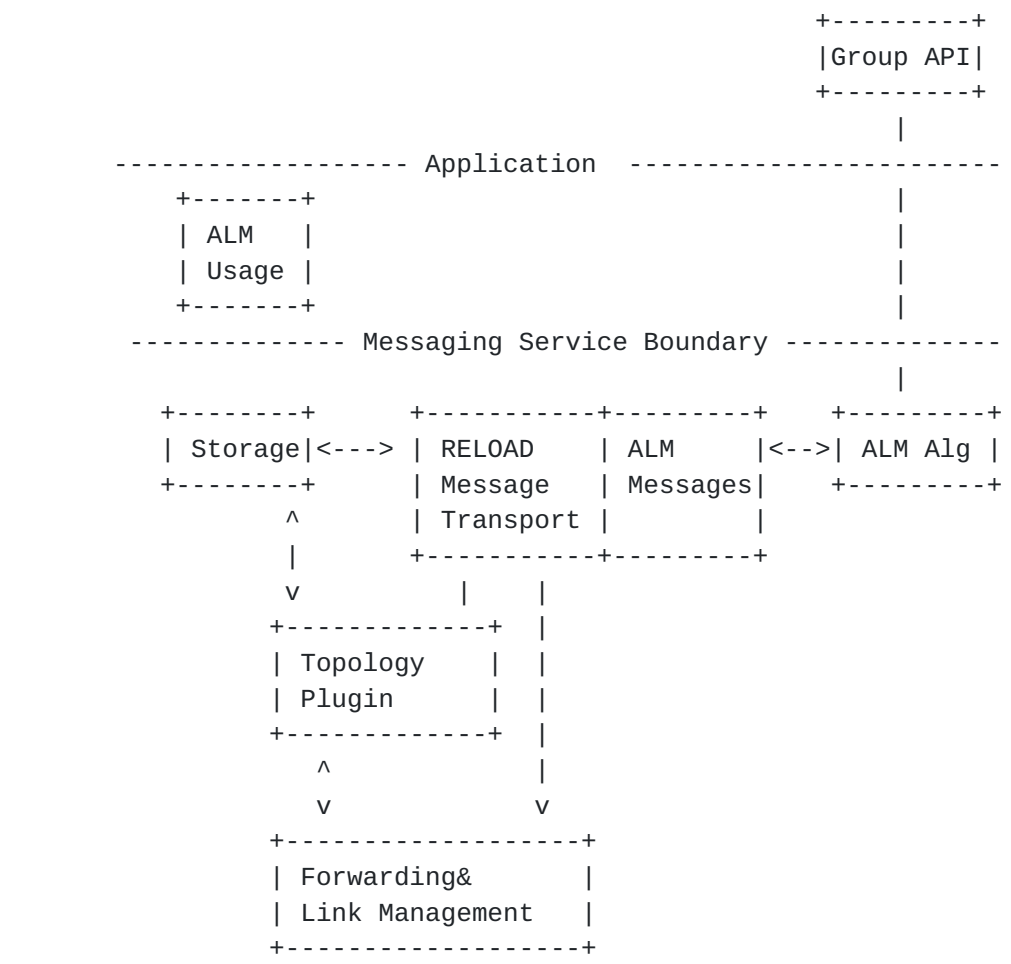

Figure 2: RELOAD Architecture Extensions

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 tree, delay from source to leaf nodes, bandwidth use, 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 which require ALM functionality in the overlay. Figure 2 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:

- Register Kind-Id points
- Define data structures for each kind
- Defines access control rules for each kind
- Defines the Resource Name used to hash to the Resource ID that determines where the kind is stored
- Addresses restoration of values after recovery from a network partition
- Defines the types of connections that can be initiated using AppConnect

an ALM GroupID is a RELOAD Node-ID. The owner of an 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:

- Create tree
- 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 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
```

2. Any joining peer:

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

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 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 [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 defined in this draft can be used with P2P-SIP, and makes the SAM framework is overlay agnostic.

As discussed in the SAM requirements draft [I-D.muramoto-irtf-sam-generic-require], 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 life-cycle (create, join, leave, re-form, heartbeat)

- Peer region and multicast properties

The message codes are defined in Section 11 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 12.

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 draft [I-D.ietf-p2psip-base]. 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 (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 group_id. The common interpretation in a DHT based overlay of group_id is that the peer with 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.

A successful Create Tree 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 Create Tree. If the StoreReq fails, the Create Tree fails too.

After a successful Create Tree, peers can use the RELOAD Fetch method to retrieve the ALMTree struct at address group_id. The ALMTree kind is defined in Section 10.

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

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

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

group_id: 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 a Store operation. The NODE-MATCH access policy defined in section 7.3.2 of RELOAD is used.

7.2.2. CreateTreeResponse

After receiving a CreateTree message from node S, the peer sends a CreateTreeReponse to node S.

```
struct {
  Dictionary options;
} CreateTreeResponse;
```

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

7.2.3. Join

Causes the distributed algorithm for peer join of a specific ALM group to be invoked. The definition of the Join message 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.

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 then 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, then node C confirms with a JoinConfirm request. Node P then responds with a JoinConfirmResponse.

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

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

peer_id: overlay address of joining/leaving peer

group_id: the overlay address of the root of the tree

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

7.2.4. Join Accept (Join Response)

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 which receives neither a JoinConfirm or JoinDecline message MUST expire.

```
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 which accepts the joining peer

child_peer_id: overlay address of joining peer

group_id: the overlay address of the root of the tree

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

7.2.5. Join Reject (Join Response)

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

7.2.6. 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 {
  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 which is a child of the parent peer

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

group_id: the overlay address of the root of the tree

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

7.2.7. Join Confirm Response

A peer receiving a JoinConfirm message responds with a JoinConfirmResponse

7.2.8. Join Decline

A peer receiving a JoinAccept message which does not wish to accept it 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 which declines the JoinAccept

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

group_id: the overlay address of the root of the tree

7.2.9. Join Decline Response

A peer receiving a JoinConfirm message responds with a JoinDeclineResponse

7.2.10. Leave

A peer which is part of an ALM tree identified by group_id 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 {
  node_id peer_id;
  node_id group_id;
  Dictionary options;
} Leave;
```

peer_id: overlay address of leaving peer

group_id: the overlay address of the root of the tree

options: name-value list of options

7.2.11. Leave Response

A peer receiving a Leave message responds with a LeaveResponse

7.2.12. 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 {
  node_id group_id;
  node_id peer_id;
  Dictionary options;
} Reform;
```

group_id: the 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 sub-tree identified by peer_id.

options: name-value list of options

7.2.13. Reform Response

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 message from the unresponsive peer. N is configurable.

```
struct {
  node_id peer_id_1;
  node_id peer_id_2;
  node_id group_id;
  Dictionary   options;
} Heartbeat;
```

peer_id_1: source of heartbeat

peer_id_2: 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 to a Heartbeat message from a child node in a tree 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_1;
  node_id peer_id_2;
} NodeQuery;
```

peer_id_1: source of query

peer_id_2: destination of query

7.2.17. NodeQuery Response

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        active_number_trees;
      TreeData      tree_data<0..2^8-1>;
      ImplementationInfo  imp_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.

active_number_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 the number_active_trees.

impl_info: information about the implementation of this usage

```
public struct {
  uint32       tree_id;
  uint8        algorithm;
  NodeId       tree_root;
  uint8        number_parents;
  NodeId       parent<0..2^8-1>;
  Uint16       number_children_nodes;
  NodeId       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 NodeId of each parent node

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

children: the RELOAD NodeId of each child node

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

path_delay_to_root: RTT in millisec. to root node

path_delay_to_child: last measured RTT in msec to child node with largest RTT.

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

join_confirm_timeout: The default time for join confirm/decline, 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 heartbeat interval is 2000 msec.

heartbeat timeout interval: The 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

7.2.19. PushResponse

After receiving a Push message from node S, the receiving peer sends a PushReponse 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 which 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 draft) and Scribe messages as defined in [CASTRO2002].

Section in Draft	RELOAD ALM Message	Scribe Message
7.2.1	CreateALMTree	Create
7.2.2	Join	Join
7.2.3	JoinAccept	
7.2.4	JoinConfirm	
7.2.5	JoinDecline	
7.2.6	Leave	Leave
7.2.7	Reform	
7.2.8	Heartbeat	
7.2.9	NodeQuery	
7.2.10	Push	Multicast
	Note 1	deliver
	Note 1	forward
	Note 1	route
	Note 1	send

Figure 3: Mapping to Scibe 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 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. Groups may have multiple sources of multicast messages.

```
CREATE : groups.add(msg.group_id)
```

group_id: 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 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 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, msg.group_id)
                  SEND(JOINAccept(node_id, msg.source, msg.group_id))
        }
```

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

8.6. JoinDecline

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

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.group_id].children as node_id)
                  SEND(msg,node_id)
            if memberOf(msg.group_id)
                  invokeMessageHandler(msg.group_id, 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:

- Incomplete Nodes: A node with less than f children in its proper stripe;
- Only-Child Nodes: A node whose parent (in any multicast tree) is an incomplete node;
- Complete Nodes: A node with exactly f children in its proper stripe
- Special Node: A single node which 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 draft) and P2PCast messages as defined in [P2PCAST].

Section in Draft	RELOAD ALM Message	P2PCast Message
7.2.1	CreateALMTree	Create
7.2.2	Join	Join
7.2.3	JoinAccept	
7.2.4	JoinConfirm	
7.2.5	JoinDecline	
7.2.6	Leave	Leave
7.2.7	Reform	Takeon Substitute Search Replace Direct Update
7.2.8	Heartbeat	
7.2.9	NodeQuery	
7.2.10	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 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.

9.4. 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:

- Incomplete Nodes: 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 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.

- Complete Nodes: 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, 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 can adopt the node in one tree and become a child in the other.

P2PCast uses defined messages for communication between nodes during reorganisation. 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 reorganisation 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 would-be 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).

9.5. 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.

9.6. JoinConfirm

This message is not part of the P2PCast protocol, but required by the basic protocol defined 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.

9.7. 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. ALMTree Kind

an ALMTree Kind is defined per section 7.4.5 in 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.

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

Kind-Id: 0xf0000001 (This is a private-use code-point per section 14.6 of RELOAD.

Data model:

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

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

11. Message Codes

All messages are mapped to the RELOAD experimental message type. The mapping is given in the following table. The format of the body of a message is given in Figure 5.

Message	RELOAD Code Point	ALM Message Code
CreateALMTRee	exp_a_req	00
CreateALMTreeResponse	exp_a_ans	01
Join	exp_a_req	02
JoinAccept	exp_a_ans	03
JoinReject	exp_a_ans	04
JoinConfirm	exp_a_req	05
JoinConfirmResponse	exp_a_ans	06
JoinDecline	exp_a_req	07
JoinDeclineResponse	exp_a_ans	08
Leave	exp_a_req	09
LeaveResponse	exp_a_ans	x0A
Reform	exp_a_req	x0B
ReformResponse	exp_a_ans	x0C
Heartbeat	exp_a_req	x0D
HeartbeatResponse	exp_a_ans	x0E
NodeQuery	exp_a_req	x0F
NodeQueryResponse	exp_a_ans	x10
Push	exp_a_req	x11
PushResponse	exp_a_ans	x12

Figure 5: RELOAD Message Code mapping

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.

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. 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

11.1. ALMHeader Definition

```
 struct {
    uint32           sam_token;
    uint32           algorithm;
    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.

algorithm: The code of the multicast algorithm being used. Each multicast tree uses only one algorithm. Trees with different multicast algorithm can co-exist, and can share the same nodes.

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.

11.2. 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 11.

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 Section 8 and Section 9 describing the various ALM methods for the definitions of the payload contents.

11.3. Response Codes

Response codes are defined in section 6.3.3.1 in RELOAD. This experimental 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 X.

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

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

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

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

Error_Link_Capacity_Limit_Reached: The capacity of a link has been reached

Error_Node_Memory_Capacity_Limit_Reached: An internal memory capacity of the node has been reached

Error_Node_CPU_Capacity_Limit_Reached: An internal processing capacity of the node has been reached

Error_Path_Limit_Reached: The maximum path length in hopcount 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.

11.4. Algorithm Codes

ALM Algorithm Types: There are currently two types: SCRIBE and P2PCAST.

0001 - SCRIBE

0002 - P2PCAST

0003 .. 0xFFFF undefined

12. 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.

12.1. Create Tree

A node with "NODE-MATCH" rights sends a request CreateTree 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 CreateTree succeeds, the ALMTree instance can be retrieved using Fetch. An example message flow for ceating a tree is depicted in Figure 6.

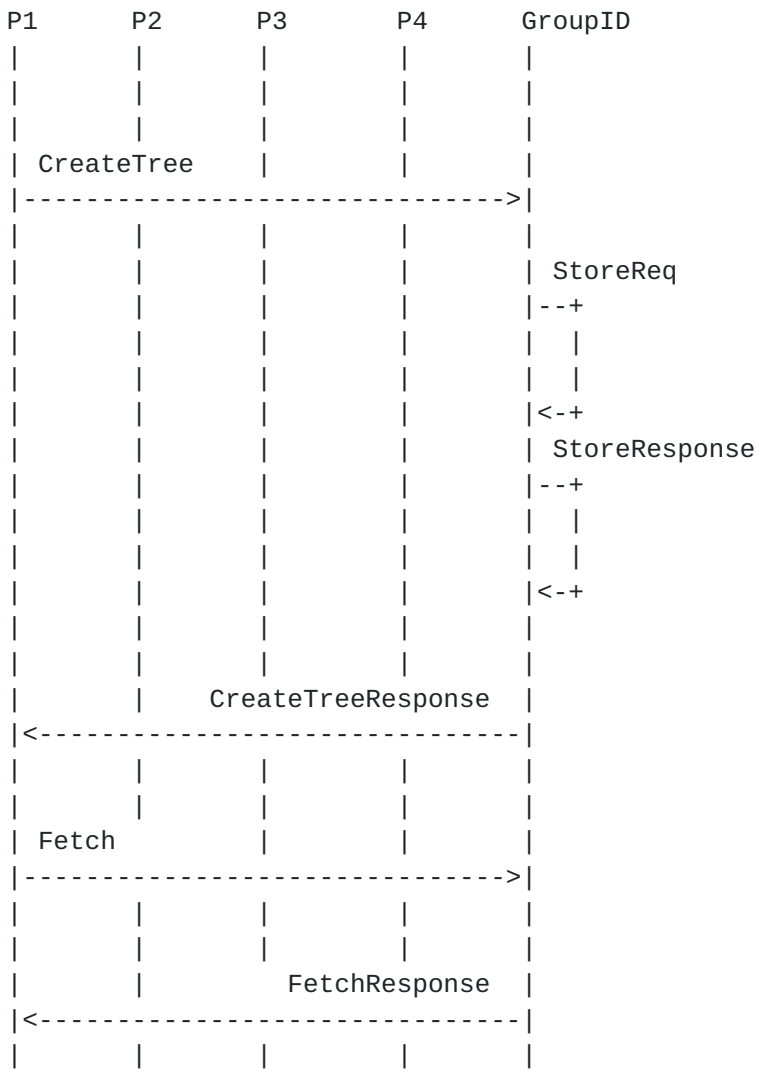

Figure 6: Message flow example for CreateTree.

12.2. Join Tree

P1 joins node GroupID 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

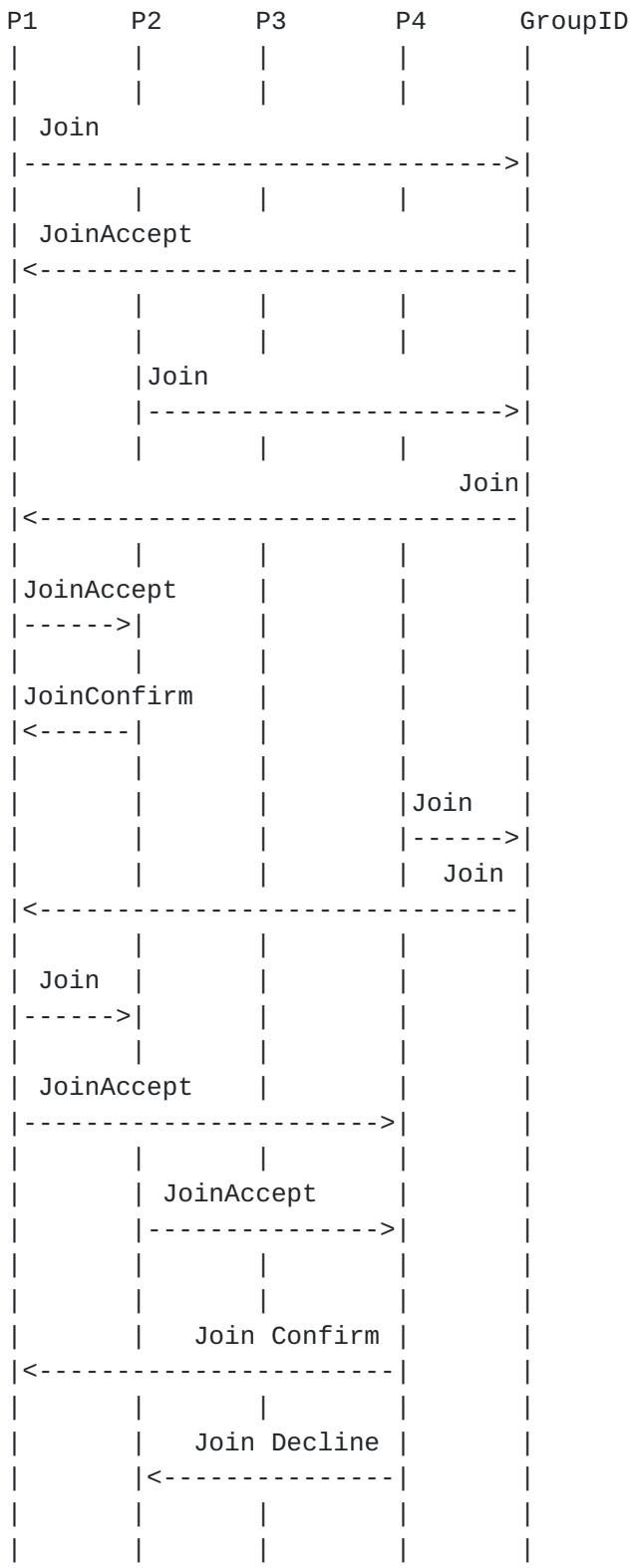

Figure 7: Message flow example for tree Join.

12.3. Leave Tree

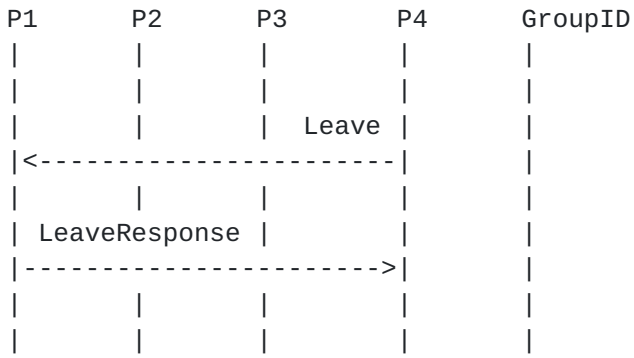

Figure 8: Message flow example for Leave tree.

12.4. Push Data

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

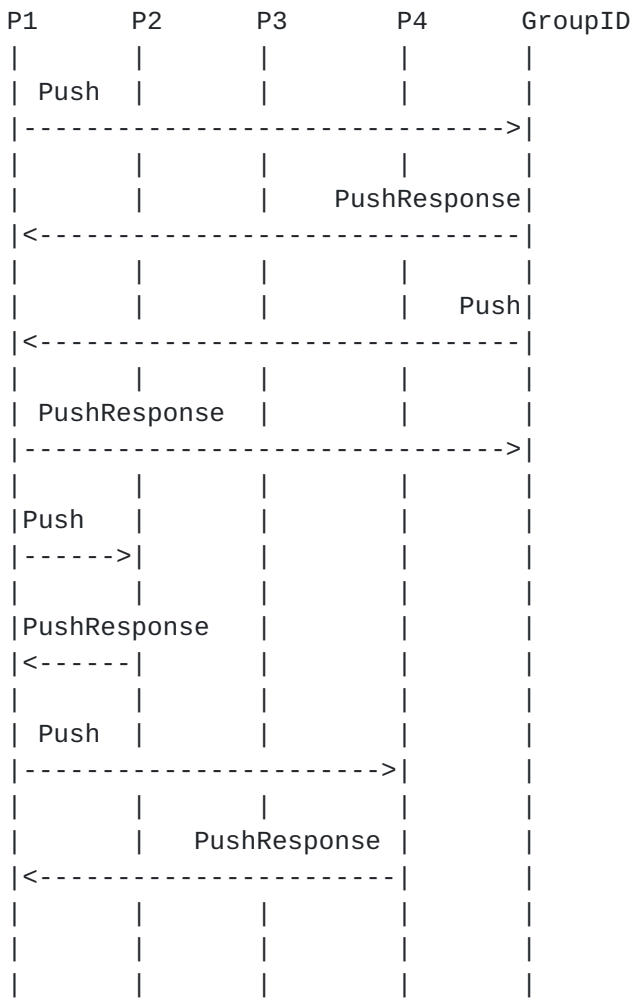

Figure 9: Message flow example for pushing data.

13. Kind Definitions

13.1. ALMTree Kind Definition

This section defines the ALMTree kind.

Kind IDs 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

14. RELOAD Configuration File Extensions

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

15. 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.

- Version 04: Add mapping to RELOAD experimental message. Modified IANA considerations setion. New algorithm identification coding. New message coding. Added push message. Create Tree access policy changed to use NODE-MATCH. Create Tree StoreReq clarified. Updated the diagrams in the Examples section. Added a Push data example. Defined the ALMTree kind.

16. IANA Considerations

This memo includes no request to IANA.

17. 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 node_id

- Right to store Tree info at some Location in the DHT

- Limit on # messages / sec and bandwidth use

- Right to join an ALM tree

18. Acknowledgement

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

19. 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-24 (work in progress), January 2013.

[I-D.ietf-p2psip-sip] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S.,

Schulzrinne, H., and T. Schmidt, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-08 (work in progress), December 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.

[I-D.muramoto-irtf-sam-generic-require]
Muramoto, E., "Requirements for Scalable Adaptive Multicast Framework in Non-GIG Networks", draft-muramoto-irtf-sam-generic-require-01 (work in progress), November 2006.

[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>.

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

[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.

[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.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

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

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

[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.

[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