

Support for Multicast over UNI 3.0/3.1 based ATM Networks.

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited

Abstract

Mapping the connectionless IP multicast service over the connection oriented ATM services provided by UNI 3.0/3.1 is a non-trivial task. This memo describes a mechanism to support the multicast needs of Layer 3 protocols in general, and describes its application to IP multicasting in particular.

ATM based IP hosts and routers use a Multicast Address Resolution Server (MARS) to support RFC 1112 style Level 2 IP multicast over the ATM Forum's UNI 3.0/3.1 point to multipoint connection service. Clusters of endpoints share a MARS and use it to track and disseminate information identifying the nodes listed as receivers for given multicast groups. This allows endpoints to establish and manage point to multipoint VCs when transmitting to the group.

The MARS behaviour allows Layer 3 multicasting to be supported using either meshes of VCs or ATM level multicast servers. This choice may be made on a per-group basis, and is transparent to the endpoints.

Table of Contents

| | |
|---|----|
| 1. Introduction..... | 4 |
| 1.1 The Multicast Address Resolution Server (MARS)..... | 5 |
| 1.2 The ATM level multicast Cluster..... | 5 |
| 1.3 Document overview..... | 6 |
| 1.4 Conventions..... | 7 |
| 2. The IP multicast service model..... | 7 |
| 3. UNI 3.0/3.1 support for intra-cluster multicasting..... | 8 |
| 3.1 VC meshes..... | 9 |
| 3.2 Multicast Servers..... | 9 |
| 3.3 Tradeoffs..... | 10 |
| 3.4 Interaction with local UNI 3.0/3.1 signalling entity..... | 11 |
| 4. Overview of the MARS..... | 12 |
| 4.1 Architecture..... | 12 |
| 4.2 Control message format..... | 12 |
| 4.3 Fixed header fields in MARS control messages..... | 13 |
| 4.3.1 Hardware type..... | 14 |
| 4.3.2 Protocol type..... | 14 |
| 4.3.3 Checksum..... | 15 |
| 4.3.4 Extensions Offset..... | 15 |
| 4.3.5 Operation code..... | 16 |
| 4.3.6 Reserved..... | 16 |
| 5. Endpoint (MARS client) interface behaviour..... | 16 |
| 5.1 Transmit side behaviour..... | 17 |
| 5.1.1 Retrieving Group Membership from the MARS..... | 18 |
| 5.1.2 MARS_REQUEST, MARS_MULTI, and MARS_NAK messages..... | 20 |
| 5.1.3 Establishing the outgoing multipoint VC..... | 22 |
| 5.1.4 Monitoring updates on ClusterControlVC..... | 24 |
| 5.1.4.1 Updating the active VCs..... | 24 |
| 5.1.4.2 Tracking the Cluster Sequence Number..... | 25 |
| 5.1.5 Revalidating a VC's leaf nodes..... | 26 |
| 5.1.5.1 When leaf node drops itself..... | 27 |
| 5.1.5.2 When a jump is detected in the CSN..... | 27 |
| 5.1.6 'Migrating' the outgoing multipoint VC..... | 27 |
| 5.2. Receive side behaviour..... | 29 |
| 5.2.1 Format of the MARS_JOIN and MARS_LEAVE Messages..... | 30 |
| 5.2.1.1 Important IPv4 default values..... | 32 |
| 5.2.2 Retransmission of MARS_JOIN and MARS_LEAVE messages.... | 33 |
| 5.2.3 Cluster member registration and deregistration..... | 34 |
| 5.3 Support for Layer 3 group management..... | 34 |
| 5.4 Support for redundant/backup MARS entities..... | 36 |
| 5.4.1 First response to MARS problems..... | 36 |
| 5.4.2 Connecting to a backup MARS..... | 37 |
| 5.4.3 Dynamic backup lists, and soft redirects..... | 37 |
| 5.5 Data path LLC/SNAP encapsulations..... | 40 |
| 5.5.1 Type #1 encapsulation..... | 40 |
| 5.5.2 Type #2 encapsulation..... | 41 |

| | |
|--|----|
| 5.5.3 A Type #1 example..... | 42 |
| 6. The MARS in greater detail..... | 42 |
| 6.1 Basic interface to Cluster members..... | 43 |
| 6.1.1 Response to MARS_REQUEST..... | 43 |
| 6.1.2 Response to MARS_JOIN and MARS_LEAVE..... | 43 |
| 6.1.3 Generating MARS_REDIRECT_MAP..... | 45 |
| 6.1.4 Cluster Sequence Numbers..... | 45 |
| 6.2 MARS interface to Multicast Servers (MCSs)..... | 46 |
| 6.2.1 MARS_REQUESTs for MCS supported groups..... | 47 |
| 6.2.2 MARS_MSERV and MARS_UNSERV messages..... | 47 |
| 6.2.3 Registering a Multicast Server (MCS)..... | 49 |
| 6.2.4 Modified response to MARS_JOIN and MARS_LEAVE..... | 49 |
| 6.2.5 Sequence numbers for ServerControlVC traffic..... | 51 |
| 6.3 Why global sequence numbers?..... | 52 |
| 6.4 Redundant/Backup MARS Architectures..... | 52 |
| 7. How an MCS utilises a MARS..... | 53 |
| 7.1 Association with a particular Layer 3 group..... | 53 |
| 7.2 Termination of incoming VCs..... | 54 |
| 7.3 Management of outgoing VC..... | 54 |
| 7.4 Use of a backup MARS..... | 54 |
| 8. Support for IP multicast routers..... | 54 |
| 8.1 Forwarding into a Cluster..... | 55 |
| 8.2 Joining in 'promiscuous' mode..... | 55 |
| 8.3 Forwarding across the cluster..... | 56 |
| 8.4 Joining in 'semi-promiscuous' mode..... | 56 |
| 8.5 An alternative to IGMP Queries..... | 57 |
| 8.6 CMIs across multiple interfaces..... | 58 |
| 9. Multiprotocol applications of the MARS and MARS clients..... | 59 |
| 10. Supplementary parameter processing..... | 60 |
| 10.1 Interpreting the mar\$extoff field..... | 60 |
| 10.2 The format of TLVs..... | 60 |
| 10.3 Processing MARS messages with TLVs..... | 62 |
| 10.4 Initial set of TLV elements..... | 62 |
| 11. Key Decisions and open issues..... | 62 |
| Security Considerations..... | 65 |
| Acknowledgments..... | 65 |
| Author's Address..... | 65 |
| References..... | 66 |
| Appendix A. Hole punching algorithms..... | 67 |
| Appendix B. Minimising the impact of IGMP in IPv4 environments.. | 69 |
| Appendix C. Further comments on 'Clusters'..... | 71 |
| Appendix D. TLV list parsing algorithm..... | 72 |
| Appendix E. Summary of timer values..... | 73 |
| Appendix F. Pseudo code for MARS operation..... | 74 |

1. Introduction.

Multicasting is the process whereby a source host or protocol entity sends a packet to multiple destinations simultaneously using a single, local 'transmit' operation. The more familiar cases of Unicasting and Broadcasting may be considered to be special cases of Multicasting (with the packet delivered to one destination, or 'all' destinations, respectively).

Most network layer models, like the one described in RFC 1112 [1] for IP multicasting, assume sources may send their packets to abstract 'multicast group addresses'. Link layer support for such an abstraction is assumed to exist, and is provided by technologies such as Ethernet.

ATM is being utilized as a new link layer technology to support a variety of protocols, including IP. With RFC 1483 [2] the IETF defined a multiprotocol mechanism for encapsulating and transmitting packets using AAL5 over ATM Virtual Channels (VCs). However, the ATM Forum's currently published signalling specifications (UNI 3.0 [8] and UNI 3.1 [4]) does not provide the multicast address abstraction. Unicast connections are supported by point to point, bidirectional VCs. Multicasting is supported through point to multipoint unidirectional VCs. The key limitation is that the sender must have prior knowledge of each intended recipient, and explicitly establish a VC with itself as the root node and the recipients as the leaf nodes.

This document has two broad goals:

- Define a group address registration and membership distribution mechanism that allows UNI 3.0/3.1 based networks to support the multicast service of protocols such as IP.

- Define specific endpoint behaviours for managing point to multipoint VCs to achieve multicasting of layer 3 packets.

As the IETF is currently in the forefront of using wide area multicasting this document's descriptions will often focus on IP service model of RFC 1112. A final chapter will note the multiprotocol application of the architecture.

This document avoids discussion of one highly non-trivial aspect of using ATM - the specification of QoS for VCs being established in response to higher layer needs. Research in this area is still very formative [7], and so it is assumed that future documents will clarify the mapping of QoS requirements to VC establishment. The default at this time is that VCs are established with a request for

Unspecified Bit Rate (UBR) service, as typified by the IETF's use of VCs for unicast IP, described in RFC 1755 [6].

1.1 The Multicast Address Resolution Server (MARS).

The Multicast Address Resolution Server (MARS) is an extended analog of the ATM ARP Server introduced in RFC 1577 [3]. It acts as a registry, associating layer 3 multicast group identifiers with the ATM interfaces representing the group's members. MARS messages support the distribution of multicast group membership information between MARS and endpoints (hosts or routers). Endpoint address resolution entities query the MARS when a layer 3 address needs to be resolved to the set of ATM endpoints making up the group at any one time. Endpoints keep the MARS informed when they need to join or leave particular layer 3 groups. To provide for asynchronous notification of group membership changes the MARS manages a point to multipoint VC out to all endpoints desiring multicast support

Valid arguments can be made for two different approaches to ATM level multicasting of layer 3 packets - through meshes of point to multipoint VCs, or ATM level multicast servers (MCS). The MARS architecture allows either VC meshes or MCSs to be used on a per-group basis.

1.2 The ATM level multicast Cluster.

Each MARS manages a 'cluster' of ATM-attached endpoints. A Cluster is defined as

The set of ATM interfaces choosing to participate in direct ATM connections to achieve multicasting of AAL_SDUs between themselves.

In practice, a Cluster is the set of endpoints that choose to use the same MARS to register their memberships and receive their updates from.

By implication of this definition, traffic between interfaces belonging to different Clusters passes through an inter-cluster device. (In the IP world an inter-cluster device would be an IP multicast router with logical interfaces into each Cluster.) This document explicitly avoids specifying the nature of inter-cluster (layer 3) routing protocols.

The mapping of clusters to other constrained sets of endpoints (such as unicast Logical IP Subnets) is left to each network administrator. However, for the purposes of conformance with this document network administrators MUST ensure that each Logical IP Subnet (LIS) is

served by a separate MARS, creating a one-to-one mapping between cluster and unicast LIS. IP multicast routers then interconnect each LIS as they do with conventional subnets. (Relaxation of this restriction MAY only occur after future research on the interaction between existing layer 3 multicast routing protocols and unicast subnet boundaries.)

The term 'Cluster Member' will be used in this document to refer to an endpoint that is currently using a MARS for multicast support. Thus potential scope of a cluster may be the entire membership of a LIS, while the actual scope of a cluster depends on which endpoints are actually cluster members at any given time.

1.3 Document overview.

This document assumes an understanding of concepts explained in greater detail in RFC 1112, RFC 1577, UNI 3.0/3.1, and RFC 1755 [6].

Section 2 provides an overview of IP multicast and what RFC 1112 required from Ethernet.

Section 3 describes in more detail the multicast support services offered by UNI 3.0/3.1, and outlines the differences between VC meshes and multicast servers (MCSs) as mechanisms for distributing packets to multiple destinations.

Section 4 provides an overview of the MARS and its relationship to ATM endpoints. This section also discusses the encapsulation and structure of MARS control messages.

Section 5 substantially defines the entire cluster member endpoint behaviour, on both receive and transmit sides. This includes both normal operation and error recovery.

Section 6 summarises the required behaviour of a MARS.

Section 7 looks at how a multicast server (MCS) interacts with a MARS.

Section 8 discusses how IP multicast routers may make novel use of promiscuous and semi-promiscuous group joins. Also discussed is a mechanism designed to reduce the amount of IGMP traffic issued by routers.

Section 9 discusses how this document applies in the more general (non-IP) case.

Section 10 summarises the key proposals, and identifies areas for future research that are generated by this MARS architecture.

The appendices provide discussion on issues that arise out of the implementation of this document. Appendix A discusses MARS and endpoint algorithms for parsing MARS messages. Appendix B describes the particular problems introduced by the current IGMP paradigms, and possible interim work-arounds. Appendix C discusses the 'cluster' concept in further detail, while Appendix D briefly outlines an algorithm for parsing TLV lists. Appendix E summarises various timer values used in this document, and Appendix F provides example pseudo-code for a MARS entity.

1.4 Conventions.

In this document the following coding and packet representation rules are used:

All multi-octet parameters are encoded in big-endian form (i.e. the most significant octet comes first).

In all multi-bit parameters bit numbering begins at 0 for the least significant bit when stored in memory (i.e. the n 'th bit has weight of 2^n).

A bit that is 'set', 'on', or 'one' holds the value 1.

A bit that is 'reset', 'off', 'clear', or 'zero' holds the value 0.

2. Summary of the IP multicast service model.

Under IP version 4 (IPv4), addresses in the range between 224.0.0.0 and 239.255.255.255 (224.0.0.0/4) are termed 'Class D' or 'multicast group' addresses. These abstractly represent all the IP hosts in the Internet (or some constrained subset of the Internet) who have decided to 'join' the specified group.

RFC1112 requires that a multicast-capable IP interface must support the transmission of IP packets to an IP multicast group address, whether or not the node considers itself a 'member' of that group. Consequently, group membership is effectively irrelevant to the transmit side of the link layer interfaces. When Ethernet is used as the link layer (the example used in RFC1112), no address resolution is required to transmit packets. An algorithmic mapping from IP multicast address to Ethernet multicast address is performed locally before the packet is sent out the local interface in the same 'send and forget' manner as a unicast IP packet.

Joining and Leaving an IP multicast group is more explicit on the receive side - with the primitives JoinLocalGroup and LeaveLocalGroup affecting what groups the local link layer interface should accept packets from. When the IP layer wants to receive packets from a group, it issues JoinLocalGroup. When it no longer wants to receive packets, it issues LeaveLocalGroup. A key point to note is that changing state is a local issue, it has no effect on other hosts attached to the Ethernet.

IGMP is defined in RFC 1112 to support IP multicast routers attached to a given subnet. Hosts issue IGMP Report messages when they perform a JoinLocalGroup, or in response to an IP multicast router sending an IGMP Query. By periodically transmitting queries IP multicast routers are able to identify what IP multicast groups have non-zero membership on a given subnet.

A specific IP multicast address, 224.0.0.1, is allocated for the transmission of IGMP Query messages. Host IP layers issue a JoinLocalGroup for 224.0.0.1 when they intend to participate in IP multicasting, and issue a LeaveLocalGroup for 224.0.0.1 when they've ceased participating in IP multicasting.

Each host keeps a list of IP multicast groups it has been JoinLocalGroup'd to. When a router issues an IGMP Query on 224.0.0.1 each host begins to send IGMP Reports for each group it is a member of. IGMP Reports are sent to the group address, not 224.0.0.1, "so that other members of the same group on the same network can overhear the Report" and not bother sending one of their own. IP multicast routers conclude that a group has no members on the subnet when IGMP Queries no longer elicit associated replies.

3. UNI 3.0/3.1 support for intra-cluster multicasting.

For the purposes of the MARS protocol, both UNI 3.0 and UNI 3.1 provide equivalent support for multicasting. Differences between UNI 3.0 and UNI 3.1 in required signalling elements are covered in RFC 1755.

This document will describe its operation in terms of 'generic' functions that should be available to clients of a UNI 3.0/3.1 signalling entity in a given ATM endpoint. The ATM model broadly describes an 'AAL User' as any entity that establishes and manages VCs and underlying AAL services to exchange data. An IP over ATM interface is a form of 'AAL User' (although the default LLC/SNAP encapsulation mode specified in RFC1755 really requires that an 'LLC entity' is the AAL User, which in turn supports the IP/ATM interface).

The most fundamental limitations of UNI 3.0/3.1's multicast support are:

Only point to multipoint, unidirectional VCs may be established.

Only the root (source) node of a given VC may add or remove leaf nodes.

Leaf nodes are identified by their unicast ATM addresses. UNI 3.0/3.1 defines two ATM address formats - native E.164 and NSAP (although it must be stressed that the NSAP address is so called because it uses the NSAP format - an ATM endpoint is NOT a Network layer termination point). In UNI 3.0/3.1 an 'ATM Number' is the primary identification of an ATM endpoint, and it may use either format. Under some circumstances an ATM endpoint must be identified by both a native E.164 address (identifying the attachment point of a private network to a public network), and an NSAP address ('ATM Subaddress') identifying the final endpoint within the private network. For the rest of this document the term will be used to mean either a single 'ATM Number' or an 'ATM Number' combined with an 'ATM Subaddress'.

3.1 VC meshes.

The most fundamental approach to intra-cluster multicasting is the multicast VC mesh. Each source establishes its own independent point to multipoint VC (a single multicast tree) to the set of leaf nodes (destinations) that it has been told are members of the group it wishes to send packets to.

Interfaces that are both senders and group members (leaf nodes) to a given group will originate one point to multipoint VC, and terminate one VC for every other active sender to the group. This criss-crossing of VCs across the ATM network gives rise to the name 'VC mesh'.

3.2 Multicast Servers.

An alternative model has each source establish a VC to an intermediate node - the multicast server (MCS). The multicast server itself establishes and manages a point to multipoint VC out to the actual desired destinations.

The MCS reassembles AAL_SDUs arriving on all the incoming VCs, and then queues them for transmission on its single outgoing point to multipoint VC. (Reassembly of incoming AAL_SDUs is required at the multicast server as AAL5 does not support cell level multiplexing of different AAL_SDUs on a single outgoing VC.)

The leaf nodes of the multicast server's point to multipoint VC must be established prior to packet transmission, and the multicast server requires an external mechanism to identify them. A side-effect of this method is that ATM interfaces that are both sources and group members will receive copies of their own packets back from the MCS (An alternative method is for the multicast server to explicitly retransmit packets on individual VCs between itself and group members. A benefit of this second approach is that the multicast server can ensure that sources do not receive copies of their own packets.)

The simplest MCS pays no attention to the contents of each AAL_SDU. It is purely an AAL/ATM level device. More complex MCS architectures (where a single endpoint serves multiple layer 3 groups) are possible, but are beyond the scope of this document. More detailed discussion is provided in section 7.

3.3 Tradeoffs.

Arguments over the relative merits of VC meshes and multicast servers have raged for some time. Ultimately the choice depends on the relative trade-offs a system administrator must make between throughput, latency, congestion, and resource consumption. Even criteria such as latency can mean different things to different people - is it end to end packet time, or the time it takes for a group to settle after a membership change? The final choice depends on the characteristics of the applications generating the multicast traffic.

If we focussed on the data path we might prefer the VC mesh because it lacks the obvious single congestion point of an MCS. Throughput is likely to be higher, and end to end latency lower, because the mesh lacks the intermediate AAL_SDU reassembly that must occur in MCSs. The underlying ATM signalling system also has greater opportunity to ensure optimal branching points at ATM switches along the multicast trees originating on each source.

However, resource consumption will be higher. Every group member's ATM interface must terminate a VC per sender (consuming on-board memory for state information, instance of an AAL service, and buffering in accordance with the vendors particular architecture). On the contrary, with a multicast server only 2 VCs (one out, one in) are required, independent of the number of senders. The allocation of VC related resources is also lower within the ATM cloud when using a multicast server. These points may be considered to have merit in environments where VCs across the UNI or within the ATM cloud are valuable (e.g. the ATM provider charges on a per VC basis), or AAL contexts are limited in the ATM interfaces of endpoints.

If we focus on the signalling load then MCSs have the advantage when faced with dynamic sets of receivers. Every time the membership of a multicast group changes (a leaf node needs to be added or dropped), only a single point to multipoint VC needs to be modified when using an MCS. This generates a single signalling event across the MCS's UNI. However, when membership change occurs in a VC mesh, signalling events occur at the UNIs of every traffic source - the transient signalling load scales with the number of sources. This has obvious ramifications if you define latency as the time for a group's connectivity to stabilise after change (especially as the number of senders increases).

Finally, as noted above, MCSs introduce a 'reflected packet' problem, which requires additional per-AAL_SDU information to be carried in order for layer 3 sources to detect their own AAL_SDUs coming back.

The MARS architecture allows system administrators to utilize either approach on a group by group basis.

3.4 Interaction with local UNI 3.0/3.1 signalling entity.

The following generic signalling functions are presumed to be available to local AAL Users:

| | |
|--------------|--|
| L_CALL_RQ | - Establish a unicast VC to a specific endpoint. |
| L_MULTI_RQ | - Establish multicast VC to a specific endpoint. |
| L_MULTI_ADD | - Add new leaf node to previously established VC. |
| L_MULTI_DROP | - Remove specific leaf node from established VC. |
| L_RELEASE | - Release unicast VC, or all Leaves of a multicast VC. |

The signalling exchanges and local information passed between AAL User and UNI 3.0/3.1 signalling entity with these functions are outside the scope of this document.

The following indications are assumed to be available to AAL Users, generated by the local UNI 3.0/3.1 signalling entity:

| | |
|----------------|--|
| L_ACK | - Successful completion of a local request. |
| L_REMOTE_CALL | - A new VC has been established to the AAL User. |
| ERR_L_RQFAILED | - A remote ATM endpoint rejected an L_CALL_RQ, L_MULTI_RQ, or L_MULTI_ADD. |
| ERR_L_DROP | - A remote ATM endpoint dropped off an existing VC. |
| ERR_L_RELEASE | - An existing VC was terminated. |

The signalling exchanges and local information passed between AAL User and UNI 3.0/3.1 signalling entity with these functions are outside the scope of this document.

4. Overview of the MARS.

The MARS may reside within any ATM endpoint that is directly addressable by the endpoints it is serving. Endpoints wishing to join a multicast cluster must be configured with the ATM address of the node on which the cluster's MARS resides. (Section 5.4 describes how backup MARSS may be added to support the activities of a cluster. References to 'the MARS' in following sections will be assumed to mean the acting MARS for the cluster.)

4.1 Architecture.

Architecturally the MARS is an evolution of the RFC 1577 ARP Server. Whilst the ARP Server keeps a table of {IP,ATM} address pairs for all IP endpoints in an LIS, the MARS keeps extended tables of {layer 3 address, ATM.1, ATM.2, ATM.n} mappings. It can either be configured with certain mappings, or dynamically 'learn' mappings. The format of the {layer 3 address} field is generally not interpreted by the MARS.

A single ATM node may support multiple logical MARSS, each of which support a separate cluster. The restriction is that each MARS has a unique ATM address (e.g. a different SEL field in the NSAP address of the node on which the multiple MARSS reside). By definition a single instance of a MARS may not support more than one cluster.

The MARS distributes group membership update information to cluster members over a point to multipoint VC known as the ClusterControlVC. Additionally, when Multicast Servers (MCSS) are being used it also establishes a separate point to multipoint VC out to registered MCSSs, known as the ServerControlVC. All cluster members are leaf nodes of ClusterControlVC. All registered multicast servers are leaf nodes of ServerControlVC (described further in section 6).

The MARS does NOT take part in the actual multicasting of layer 3 data packets.

4.2 Control message format.

By default all MARS control messages MUST be LLC/SNAP encapsulated using the following codepoints:

```
[0xAA-AA-03][0x00-00-5E][0x00-03][MARS control message]
      (LLC)           (OUI)           (PID)
```

(This is a PID from the IANA OUI.)

MARS control messages are made up of 4 major components:

[Fixed header][Mandatory fields][Addresses][Supplementary TLVs]

[Fixed header] contains fields indicating the operation being performed and the layer 3 protocol being referred to (e.g. IPv4, IPv6, AppleTalk, etc). The fixed header also carries checksum information, and hooks to allow this basic control message structure to be re-used by other query/response protocols.

The [Mandatory fields] section carries fixed width parameters that depend on the operation type indicated in [Fixed header].

The following [Addresses] area carries variable length fields for source and target addresses - both hardware (e.g. ATM) and layer 3 (e.g. IPv4). These provide the fundamental information that the registrations, queries, and updates use and operate on. For the MARS protocol fields in [Fixed header] indicate how to interpret the contents of [Addresses].

[Supplementary TLVs] represents an optional list of TLV (type, length, value) encoded information elements that may be appended to provide supplementary information. This feature is described in further detail in section 10.

MARS messages contain variable length address fields. In all cases null addresses SHALL be encoded as zero length, and have no space allocated in the message.

(Unique LLC/SNAP encapsulation of MARS control messages means MARS and ARP Server functionality may be implemented within a common entity, and share a client-server VC, if the implementor so chooses. Note that the LLC/SNAP codepoint for MARS is different to the codepoint used for ATMARP.)

4.3 Fixed header fields in MARS control messages.

The [Fixed header] has the following format:

| | | |
|-------------|---------|---|
| Data: | | |
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hrrsv | 24 bits | Reserved. Unused by MARS control protocol. |
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code. |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$sstl | 8 bits | Type & length of source ATM subaddress. (q) |

mar\$sh1 and mar\$st1 provide information regarding the source's hardware (ATM) address. In the MARS protocol these fields are always present, as every MARS message carries a non-null source ATM address. In all cases the source ATM address is the first variable length field in the [Addresses] section.

The other fields in [Fixed header] are described in the following subsections.

4.3.1 Hardware type.

mar\$afn defines the type of link layer addresses being carried. The value of 0x000F SHALL be used by MARS messages generated in accordance with this document. The encoding of ATM addresses and subaddresses when mar\$afn = 0x000F is described in section 5.1.2. Encodings when mar\$afn != 0x000F are outside the scope of this document.

4.3.2 Protocol type.

The mar\$pro field is made up of two subfields:

mar\$pro.type 16 bits Protocol type.
mar\$pro.snap 40 bits Optional SNAP extension to protocol type.

The mar\$pro.type field is a 16 bit unsigned integer representing the following number space:

0x0000 to 0x00FF Protocols defined by the equivalent NLPIDs.
0x0100 to 0x03FF Reserved for future use by the IETF.
0x0400 to 0x04FF Allocated for use by the ATM Forum.
0x0500 to 0x05FF Experimental/Local use.
0x0600 to 0xFFFF Protocols defined by the equivalent Ethertypes.

(based on the observations that valid Ethertypes are never smaller than 0x600, and NLPIDs never larger than 0xFF.)

The NLPID value of 0x80 is used to indicate a SNAP encoded extension is being used to encode the protocol type. When mar\$pro.type == 0x80 the SNAP extension is encoded in the mar\$pro.snap field. This is termed the 'long form' protocol ID.

If mar\$pro.type != 0x80 then the mar\$pro.snap field MUST be zero on transmit and ignored on receive. The mar\$pro.type field itself identifies the protocol being referred to. This is termed the 'short form' protocol ID.

In all cases, where a protocol has an assigned number in the `mar$pro.type` space (excluding 0x80) the short form MUST be used when transmitting MARS messages. Additionally, where a protocol has valid short and long forms of identification, receivers MAY choose to recognise the long form.

`mar$pro.type` values other than 0x80 MAY have 'long forms' defined in future documents.

For the remainder of this document references to `mar$pro` SHALL be interpreted to mean `mar$pro.type`, or `mar$pro.type` in combination with `mar$pro.snap` as appropriate.

The use of different protocol types is described further in section 9.

4.3.3 Checksum.

The `mar$chksum` field carries a standard IP checksum calculated across the entire MARS control message (excluding the LLC/SNAP header). The field is set to zero before performing the checksum calculation.

As the entire LLC/SNAP encapsulated MARS message is protected by the 32 bit CRC of the AAL5 transport, implementors MAY choose to ignore the checksum facility. If no checksum is calculated these bits MUST be reset before transmission. If no checksum is performed on reception, this field MUST be ignored. If a receiver is capable of validating a checksum it MUST only perform the validation when the received `mar$chksum` field is non-zero. Messages arriving with `mar$chksum` of 0 are always considered valid.

4.3.4 Extensions Offset.

The `mar$extoff` field identifies the existence and location of an optional supplementary parameters list. Its use is described in section 10.

4.3.5 Operation code.

The mar\$op field is further subdivided into two 8 bit fields - mar\$op.version (leading octet) and mar\$op.type (trailing octet). Together they indicate the nature of the control message, and the context within which its [Mandatory fields], [Addresses], and [Supplementary TLVs] should be interpreted.

mar\$op.version

| | |
|-------------|---|
| 0 | MARS protocol defined in this document. |
| 0x01 - 0xEF | Reserved for future use by the IETF. |
| 0xF0 - 0xFE | Allocated for use by the ATM Forum. |
| 0xFF | Experimental/Local use. |

mar\$op.type

Value indicates operation being performed, within context of the control protocol version indicated by mar\$op.version.

For the rest of this document references to the mar\$op value SHALL be taken to mean mar\$op.type, with mar\$op.version = 0x00. The values used in this document are summarised in section 11.

(Note this number space is independent of the ATMARP operation code number space.)

4.3.6 Reserved.

mar\$hrrsv may be subdivided and assigned specific meanings for other control protocols indicated by mar\$op.version != 0.

5. Endpoint (MARS client) interface behaviour.

An endpoint is best thought of as a 'shim' or 'convergence' layer, sitting between a layer 3 protocol's link layer interface and the underlying UNI 3.0/3.1 service. An endpoint in this context can exist in a host or a router - any entity that requires a generic 'layer 3 over ATM' interface to support layer 3 multicast. It is broken into two key subsections - one for the transmit side, and one for the receive side.

Multiple logical ATM interfaces may be supported by a single physical ATM interface (for example, using different SEL values in the NSAP formatted address assigned to the physical ATM interface). Therefore implementors MUST allow for multiple independent 'layer 3 over ATM' interfaces too, each with its own configured MARS (or table of MARSs, as discussed in section 5.4), and ability to be attached to the same or different clusters.

The initial signalling path between a MARS client (managing an endpoint) and its associated MARS is a transient point to point, bidirectional VC. This VC is established by the MARS client, and is used to send queries to, and receive replies from, the MARS. It has an associated idle timer, and is dismantled if not used for a configurable period of time. The minimum suggested value for this time is 1 minute, and the RECOMMENDED default is 20 minutes. (Where the MARS and ARP Server are co-resident, this VC may be used for both ATM ARP traffic and MARS control traffic.)

The remaining signalling path is ClusterControlVC, to which the MARS client is added as a leaf node when it registers (described in section 5.2.3).

The majority of this document covers the distribution of information allowing endpoints to establish and manage outgoing point to multipoint VCs - the forwarding paths for multicast traffic to particular multicast groups. The actual format of the AAL_SDUs sent on these VCs is almost completely outside the scope of this specification. However, endpoints are not expected to know whether their forwarding path leads directly to a multicast group's members or to an MCS (described in section 3). This requires additional per-packet encapsulation (described in section 5.5) to aid in the detection of reflected AAL_SDUs.

5.1 Transmit side behaviour.

The following description will often be in terms of an IPv4/ATM interface that is capable of transmitting packets to a Class D address at any time, without prior warning. It should be trivial for an implementor to generalise this behaviour to the requirements of another layer 3 data protocol.

When a local Layer 3 entity passes down a packet for transmission, the endpoint first ascertains whether an outbound path to the destination multicast group already exists. If it does not, the MARS is queried for a set of ATM endpoints that represent an appropriate forwarding path. (The ATM endpoints may represent the actual group members within the cluster, or a set of one or more MCSs. The endpoint does not distinguish between either case. Section 6.2 describes the MARS behaviour that leads to MCSs being supplied as the forwarding path for a multicast group.)

The query is executed by issuing a MARS_REQUEST. The reply from the MARS may take one of two forms:

MARS_MULTI - Sequence of MARS_MULTI messages returning the set of ATM endpoints that are to be leaf nodes of an outgoing point to multipoint VC (the forwarding path).

MARS_NAK - No mapping found, group is empty.

The formats of these messages are described in section 5.1.2.

Outgoing VCs are established with a request for Unspecified Bit Rate (UBR) service, as typified by the IETF's use of VCs for unicast IP, described in RFC 1755 [6]. Future documents may vary this approach and allow the specification of different ATM traffic parameters from locally configured information or parameters obtained through some external means.

5.1.1 Retrieving Group Membership from the MARS.

If the MARS had no mapping for the desired Class D address a MARS_NAK will be returned. In this case the IP packet MUST be discarded silently. If a match is found in the MARS's tables it proceeds to return addresses ATM.1 through ATM.n in a sequence of one or more MARS_MULTIs. A simple mechanism is used to detect and recover from loss of MARS_MULTI messages.

(If the client learns that there is no other group member in the cluster - the MARS returns a MARS_NAK or returns a MARS_MULTI with the client as the only member - it MUST delay sending out a new MARS_REQUEST for that group for a period no less than 5 seconds and no more than 10 seconds.)

Each MARS_MULTI carries a boolean field x, and a 15 bit integer field y - expressed as MARS_MULTI(x,y). Field y acts as a sequence number, starting at 1 and incrementing for each MARS_MULTI sent. Field x acts as an 'end of reply' marker. When x == 1 the MARS response is considered complete.

In addition, each MARS_MULTI may carry multiple ATM addresses from the set {ATM.1, ATM.2, ATM.n}. A MARS MUST minimise the number of MARS_MULTIs transmitted by placing as many group members' addresses in a single MARS_MULTI as possible. The limit on the length of an individual MARS_MULTI message MUST be the MTU of the underlying VC.

For example, assume n ATM addresses must be returned, each MARS_MULTI is limited to only p ATM addresses, and $p \ll n$. This would require a sequence of k MARS_MULTI messages (where $k = (n/p)+1$, using integer arithmetic), transmitted as follows:

```
MARS_MULTI(0,1) carries back {ATM.1 ... ATM.p}
MARS_MULTI(0,2) carries back {ATM.(p+1) ... ATM.(2p)}
[.....]
MARS_MULTI(1,k) carries back { ... ATM.n}
```

If $k == 1$ then only MARS_MULTI(1,1) is sent.

Typical failure mode will be losing one or more of MARS_MULTI(0,1) through MARS_MULTI(0,k-1). This is detected when y jumps by more than one between consecutive MARS_MULTI's. An alternative failure mode is losing MARS_MULTI(1,k). A timer MUST be implemented to flag the failure of the last MARS_MULTI to arrive. A default value of 10 seconds is RECOMMENDED.

If a 'sequence jump' is detected, the host MUST wait for the MARS_MULTI(1,k), discard all results, and repeat the MARS_REQUEST.

If a timeout occurs, the host MUST discard all results, and repeat the MARS_REQUEST.

A final failure mode involves the MARS Sequence Number (described in section 5.1.4.2 and carried in each part of a multi-part MARS_MULTI). If its value changes during the reception of a multi-part MARS_MULTI the host MUST wait for the MARS_MULTI(1,k), discard all results, and repeat the MARS_REQUEST.

(Corruption of cell contents will lead to loss of a MARS_MULTI through AAL5 CPCS_PDU reassembly failure, which will be detected through the mechanisms described above.)

If the MARS is managing a cluster of endpoints spread across different but directly accessible ATM networks it will not be able to return all the group members in a single MARS_MULTI. The MARS_MULTI message format allows for either E.164, ISO NSAP, or (E.164 + NSAP) to be returned as ATM addresses. However, each MARS_MULTI message may only return ATM addresses of the same type and length. The returned addresses MUST be grouped according to type (E.164, ISO NSAP, or both) and returned in a sequence of separate MARS_MULTI parts.

5.1.2 MARS_REQUEST, MARS_MULTII, and MARS_NAK messages.

MARS_REQUEST is shown below. It is indicated by an 'operation type value' (mar\$op) of 1.

The multicast address being resolved is placed into the the target protocol address field (mar\$tpa), and the target hardware address is set to null (mar\$thtl and mar\$stsl both zero).

In IPv4 environments the protocol type (mar\$pro) is 0x800 and the target protocol address length (mar\$tpln) MUST be set to 4. The source fields MUST contain the ATM number and subaddress of the client issuing the MARS_REQUEST (the subaddress MAY be null).

Data:

| | | |
|-------------|---------|---|
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hdrsv | 24 bits | Reserved. Unused by MARS control protocol. |
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code (MARS_REQUEST = 1) |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$sstl | 8 bits | Type & length of source ATM subaddress. (q) |
| mar\$spln | 8 bits | Length of source protocol address (s) |
| mar\$thtl | 8 bits | Type & length of target ATM number (x) |
| mar\$stsl | 8 bits | Type & length of target ATM subaddress (y) |
| mar\$tpln | 8 bits | Length of target group address (z) |
| mar\$pad | 64 bits | Padding (aligns mar\$sha with MARS_MULTII). |
| mar\$sha | roctets | source ATM number |
| mar\$ssa | qoctets | source ATM subaddress |
| mar\$spa | soctets | source protocol address |
| mar\$tpa | zoctets | target multicast group address |
| mar\$tha | xoctets | target ATM number |
| mar\$tsa | yoctets | target ATM subaddress |

Following the RFC1577 approach, the mar\$shtl, mar\$sstl, mar\$thtl and mar\$stsl fields are coded as follows:

```

  7 6 5 4 3 2 1 0
+---+---+---+---+
|0|x|  length  |
+---+---+---+---+
```

The most significant bit is reserved and MUST be set to zero. The second most significant bit (x) is a flag indicating whether the ATM address being referred to is in:

- ATM Forum NSAPA format (x = 0).
- Native E.164 format (x = 1).

The bottom 6 bits is an unsigned integer value indicating the length of the associated ATM address in octets. If this value is zero the flag x is ignored.

The mar\$spln and mar\$tpln fields are unsigned 8 bit integers, giving the length in octets of the source and target protocol address fields respectively.

MARS packets use true variable length fields. A null (non-existent) address MUST be coded as zero length, and no space allocated for it in the message body.

MARS_NAK is the MARS_REQUEST returned with operation type value of 6. All other fields are left unchanged from the MARS_REQUEST (e.g. do not transpose the source and target information. In all cases MARS clients use the source address fields to identify their own messages coming back).

The MARS_MULTI message is identified by an mar\$op value of 2. The message format is:

| | | | |
|-------------|---------|---|--|
| Data: | | | |
| mar\$afn | 16 bits | Address Family (0x000F). | |
| mar\$pro | 56 bits | Protocol Identification. | |
| mar\$hdrsv | 24 bits | Reserved. Unused by MARS control protocol. | |
| mar\$chksum | 16 bits | Checksum across entire MARS message. | |
| mar\$extoff | 16 bits | Extensions Offset. | |
| mar\$op | 16 bits | Operation code (MARS_MULTI = 2). | |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) | |
| mar\$sstl | 8 bits | Type & length of source ATM subaddress. (q) | |
| mar\$spln | 8 bits | Length of source protocol address (s) | |
| mar\$thtl | 8 bits | Type & length of target ATM number (x) | |
| mar\$ttstl | 8 bits | Type & length of target ATM subaddress (y) | |
| mar\$tpln | 8 bits | Length of target group address (z) | |
| mar\$tnum | 16 bits | Number of target ATM addresses returned (N) | |
| mar\$seqxy | 16 bits | Boolean flag x and sequence number y. | |
| mar\$msn | 32 bits | MARS Sequence Number. | |
| mar\$sha | roctets | source ATM number | |
| mar\$ssa | qoctets | source ATM subaddress | |
| mar\$spa | soctets | source protocol address | |
| mar\$tpa | zoctets | target multicast group address | |

```

mar$tha.1      xoctets  target ATM number 1
mar$tsa.1      yoctets  target ATM subaddress 1
mar$tha.2      xoctets  target ATM number 2
mar$tsa.2      yoctets  target ATM subaddress 2
               [.....]
mar$tha.N      xoctets  target ATM number N
mar$tsa.N      yoctets  target ATM subaddress N

```

The source protocol and ATM address fields are copied directly from the MARS_REQUEST that this MARS_MULTI is in response to (not the MARS itself).

mar\$seqxy is coded with flag x in the leading bit, and sequence number y coded as an unsigned integer in the remaining 15 bits.

```

| 1st octet | 2nd octet |
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|x|                                     y                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

mar\$num is an unsigned integer indicating how many pairs of {mar\$tha,mar\$tsa} (i.e. how many group member's ATM addresses) are present in the message. mar\$msn is an unsigned 32 bit number filled in by the MARS before transmitting each MARS_MULTI. Its use is described further in section 5.1.4.

As an example, assume we have a multicast cluster using 4 byte protocol addresses, 20 byte ATM numbers, and 0 byte ATM subaddresses. For n group members in a single MARS_MULTI we require a (60 + 20n) byte message. If we assume the default MTU of 9180 bytes, we can return a maximum of 456 group member's addresses in a single MARS_MULTI.

5.1.3 Establishing the outgoing multipoint VC.

Following the completion of the MARS_MULTI reply the endpoint may establish a new point to multipoint VC, or reuse an existing one.

If establishing a new VC, an L_MULTI_RQ is issued for ATM.1, followed by an L_MULTI_ADD for every member of the set {ATM.2, ..., ATM.n} (assuming the set is non-null). The packet is then transmitted over the newly created VC just as it would be for a unicast VC.

After transmitting the packet, the local interface holds the VC open and marks it as the active path out of the host for any subsequent IP packets being sent to that Class D address.

When establishing a new multicast VC it is possible that one or more L_MULTI_RQ or L_MULTI_ADD may fail. The UNI 3.0/3.1 failure cause must be returned in the ERR_L_RQFAILED signal from the local signalling entity to the AAL User. If the failure cause is not 49 (Quality of Service unavailable), 51 (user cell rate not available - UNI 3.0), 37 (user cell rate not available - UNI 3.1), or 41 (Temporary failure), the endpoint's ATM address is dropped from the set {ATM.1, ATM.2, ..., ATM.n} returned by the MARS. Otherwise, the L_MULTI_RQ or L_MULTI_ADD should be reissued after a random delay of 5 to 10 seconds. If the request fails again, another request should be issued after twice the previous delay has elapsed. This process should be continued until the call succeeds or the multipoint VC gets released.

If the initial L_MULTI_RQ fails for ATM.1, and n is greater than 1 (i.e. the returned set of ATM addresses contains 2 or more addresses) a new L_MULTI_RQ should be immediately issued for the next ATM address in the set. This procedure is repeated until an L_MULTI_RQ succeeds, as no L_MULTI_ADDs may be issued until an initial outgoing VC is established.

Each ATM address for which an L_MULTI_RQ failed with cause 49, 51, 37, or 41 MUST be tagged rather than deleted. An L_MULTI_ADD is issued for these tagged addresses using the random delay procedure outlined above.

The VC MAY be considered 'up' before failed L_MULTI_ADDs have been successfully re-issued. An endpoint MAY implement a concurrent mechanism that allows data to start flowing out the new VC even while failed L_MULTI_ADDs are being re-tried. (The alternative of waiting for each leaf node to accept the connection could lead to significant delays in transmitting the first packet.)

Each VC MUST have a configurable inactivity timer associated with it. If the timer expires, an L_RELEASE is issued for that VC, and the Class D address is no longer considered to have an active path out of the local host. The timer SHOULD be no less than 1 minute, and a default of 20 minutes is RECOMMENDED. Choice of specific timer periods is beyond the scope of this document.

VC consumption may also be reduced by endpoints noting when a new group's set of {ATM.1, ..., ATM.n} matches that of a pre-existing VC out to another group. With careful local management, and assuming the QoS of the existing VC is sufficient for both groups, a new pt to mpt VC may not be necessary. Under certain circumstances endpoints may decide that it is sufficient to re-use an existing VC whose set of leaf nodes is a superset of the new group's membership (in which case some endpoints will receive multicast traffic for a layer 3 group

they haven't joined, and must filter them above the ATM interface). Algorithms for performing this type of optimization are not discussed here, and are not required for conformance with this document.

5.1.4 Tracking subsequent group updates.

Once a new VC has been established, the transmit side of the cluster member's interface needs to monitor subsequent group changes - adding or dropping leaf nodes as appropriate. This is achieved by watching for MARS_JOIN and MARS_LEAVE messages from the MARS itself. These messages are described in detail in section 5.2 - at this point it is sufficient to note that they carry:

- The ATM address of a node joining or leaving a group.
- The layer 3 address of the group(s) being joined or left.
- A Cluster Sequence Number (CSN) from the MARS.

MARS_JOIN and MARS_LEAVE messages arrive at each cluster member across ClusterControlVC. MARS_JOIN or MARS_LEAVE messages that simply confirm information already held by the cluster member are used to track the Cluster Sequence Number, but are otherwise ignored.

5.1.4.1 Updating the active VCs.

If a MARS_JOIN is seen that refers to (or encompasses) a group for which the transmit side already has a VC open, the new member's ATM address is extracted and an L_MULTI_ADD issued locally. This ensures that endpoints already sending to a given group will immediately add the new member to their list of recipients.

If a MARS_LEAVE is seen that refers to (or encompasses) a group for which the transmit side already has a VC open, the old member's ATM address is extracted and an L_MULTI_DROP issued locally. This ensures that endpoints already sending to a given group will immediately drop the old member from their list of recipients. When the last leaf of a VC is dropped, the VC is closed completely and the affected group no longer has a path out of the local endpoint (the next outbound packet to that group's address will trigger the creation of a new VC, as described in sections 5.1.1 to 5.1.3).

The transmit side of the interface MUST NOT shut down an active VC to a group for which the receive side has just executed a LeaveLocalGroup. (This behaviour is consistent with the model of hosts transmitting to groups regardless of their own membership status.)

If a MARS_JOIN or MARS_LEAVE arrives with mar\$pnum == 0 it carries no <min,max> pairs, and is only used for tracking the CSN.

5.1.4.2 Tracking the Cluster Sequence Number.

It is important that endpoints do not miss group membership updates issued by the MARS over ClusterControlVC. However, this will happen from time to time. The Cluster Sequence Number is carried as an unsigned 32 bit value in the mar\$msn field of many MARS messages (except for MARS_REQUEST and MARS_NAK). It increments once for every transmission the MARS makes on ClusterControlVC, regardless of whether the transmission represents a change in the MARS database or not. By tracking this counter, cluster members can determine whether they have missed a previous message on ClusterControlVC, and possibly a membership change. This is then used to trigger revalidation (described in section 5.1.5).

The current CSN is copied into the mar\$msn field of MARS messages being sent to cluster members, whether out ClusterControlVC or on a point to point VC.

Calculations on the sequence numbers MUST be performed as unsigned 32 bit arithmetic.

Every cluster member keeps its own 32 bit Host Sequence Number (HSN) to track the MARS's sequence number. Whenever a message is received that carries an mar\$msn field the following processing is performed:

```
Seq.diff = mar$msn - HSN

mar$msn -> HSN
{...process MARS message as appropriate...}

if ((Seq.diff != 1) && (Seq.diff != 0))
    then {...revalidate group membership information...}
```

The basic result is that the cluster member attempts to keep locked in step with membership changes noted by the MARS. If it ever detects that a membership change occurred (in any group) without it noticing, it re-validates the membership of all groups it currently has multicast VCs open to.

The mar\$msn value in an individual MARS_MULTI is not used to update the HSN until all parts of the MARS_MULTI (if more than 1) have arrived. (If the mar\$msn changes the MARS_MULTI is discarded, as described in section 5.1.1.)

The MARS is free to choose an initial value of CSN. When a new cluster member starts up it should initialise HSN to zero. When the cluster member sends the MARS_JOIN to register (described later), the HSN will be correctly updated to the current CSN value when the

endpoint receives the copy of its MARS_JOIN back from the MARS.

5.1.5 Revalidating a VC's leaf nodes.

Certain events may inform a cluster member that it has incorrect information about the sets of leaf nodes it should be sending to. If an error occurs on a VC associated with a particular group, the cluster member initiates revalidation procedures for that specific group. If a jump is detected in the Cluster Sequence Number, this initiates revalidation of all groups to which the cluster member currently has open point to multipoint VCs.

Each open and active multipoint VC has a flag associated with it called 'VC_revalidate'. This flag is checked everytime a packet is queued for transmission on that VC. If the flag is false, the packet is transmitted and no further action is required.

However, if the VC_revalidate flag is true then the packet is transmitted and a new sequence of events is started locally.

Revalidation begins with re-issuing a MARS_REQUEST for the group being revalidated. The returned set of members {NewATM.1, NewATM.2, NewATM.n} is compared with the set already held locally. L_MULTII_DROPS are issued on the group's VC for each node that appears in the original set of members but not in the revalidated set of members. L_MULTII_ADDS are issued on the group's VC for each node that appears in the revalidated set of members but not in the original set of members. The VC_revalidate flag is reset when revalidation concludes for the given group. Implementation specific mechanisms will be needed to flag the 'revalidation in progress' state.

The key difference between constructing a VC (section 5.1.3) and revalidating a VC is that packet transmission continues on the open VC while it is being revalidated. This minimises the disruption to existing traffic.

The algorithm for initiating revalidation is:

- When a packet arrives for transmission on a given group, the groups membership is revalidated if VC_revalidate == TRUE. Revalidation resets VC_revalidate.
- When an event occurs that demands revalidation, every group has its VC_revalidate flag set TRUE at a random time between 1 and 10 seconds.

Benefit: Revalidation of active groups occurs quickly, and essentially idle groups are revalidated as needed. Randomly distributed setting of VC_revalidate flag improves chances of

staggered revalidation requests from senders when a sequence number jump is detected.

5.1.5.1 When leaf node drops itself.

During the life of a multipoint VC an ERR_L_DROP may be received indicating that a leaf node has terminated its participation at the ATM level. The ATM endpoint associated with the ERR_L_DROP MUST be removed from the locally held set {ATM.1, ATM.2, ATM.n} associated with the VC.

After a random period of time between 1 and 10 seconds the VC_revalidate flag associated with that VC MUST be set true.

If an ERR_L_RELEASE is received then the entire set {ATM.1, ATM.2, ATM.n} is cleared and the VC is considered to be completely shut down. Further packet transmission to the group served by this VC will result in a new VC being established as described in section 5.1.3.

5.1.5.2 When a jump is detected in the CSN.

Section 5.1.4.2 describes how a CSN jump is detected. If a CSN jump is detected upon receipt of a MARS_JOIN or a MARS_LEAVE then every outgoing multicast VC MUST have its VC_revalidate flag set true at some random interval between 1 and 10 seconds from when the CSN jump was detected.

The only exception to this rule is if a sequence number jump is detected during the establishment of a new group's VC (i.e. a MARS_MULTI reply was correctly received, but its mar\$msn indicated that some previous MARS traffic had been missed on ClusterControlVC). In this case every open VC, EXCEPT the one just established, MUST have its VC_revalidate flag set true at some random interval between 1 and 10 seconds from when the CSN jump was detected. (The VC being established at the time is considered already validated.)

5.1.6 'Migrating' the outgoing multipoint VC

In addition to the group tracking described in section 5.1.4, the transmit side of a cluster member must respond to 'migration' requests by the MARS. This is triggered by the reception of a MARS_MIGRATE message from ClusterControlVC. The MARS_MIGRATE message is shown below, with an mar\$op code of 13.

Data:

| | | |
|------------|---------|--|
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hdrsv | 24 bits | Reserved. Unused by MARS control protocol. |

| | | |
|-------------|---------|---|
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code (MARS_MIGRATE = 13). |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$sstl | 8 bits | Type & length of source ATM subaddress. (q) |
| mar\$spln | 8 bits | Length of source protocol address (s) |
| mar\$thtl | 8 bits | Type & length of target ATM number (x) |
| mar\$stsl | 8 bits | Type & length of target ATM subaddress (y) |
| mar\$tpln | 8 bits | Length of target group address (z) |
| mar\$num | 16 bits | Number of target ATM addresses returned (N) |
| mar\$resv | 16 bits | Reserved. |
| mar\$msn | 32 bits | MARS Sequence Number. |
| mar\$sha | roctets | source ATM number |
| mar\$ssa | qoctets | source ATM subaddress |
| mar\$spa | soctets | source protocol address |
| mar\$tpa | zoctets | target multicast group address |
| mar\$tha.1 | xoctets | target ATM number 1 |
| mar\$tsta.1 | yoctets | target ATM subaddress 1 |
| mar\$tha.2 | xoctets | target ATM number 2 |
| mar\$tsta.2 | yoctets | target ATM subaddress 2 |
| | | [.....] |
| mar\$tha.N | xoctets | target ATM number N |
| mar\$tsta.N | yoctets | target ATM subaddress N |

A migration is requested when the MARS determines that it no longer wants cluster members forwarding their packets directly to the ATM addresses it had previously specified (through MARS_REQUESTs or MARS_JOINS). When a MARS_MIGRATE is received each cluster member MUST perform the following steps:

Close down any existing outgoing VC associated with the group carried in the mar\$tpa field (L_RELEASE), or dissociate the group from any outgoing VC it may have been sharing (as described in section 5.1.3).

Establish a new outgoing VC for the specified group, using the algorithm described in section 5.1.3 and taking the set of ATM addresses supplied in the MARS_MIGRATE as the group's new set of members {ATM.1, ATM.n}.

The MARS_MIGRATE carries the new set of members {ATM.1, ATM.n} in a single message, in similar manner to a single part MARS_MULTIPART. As with other messages from the MARS, the Cluster Sequence Number carried in mar\$msn is checked as described in section 5.1.4.2.

5.2. Receive side behaviour.

A cluster member is a 'group member' (in the sense that it receives packets directed at a given multicast group) when its ATM address appears in the MARS's table entry for the group's multicast address. A key function within each cluster is the distribution of group membership information from the MARS to cluster members.

An endpoint may wish to 'join a group' in response to a local, higher level request for membership of a group, or because the endpoint supports a layer 3 multicast forwarding engine that requires the ability to 'see' intra-cluster traffic in order to forward it.

Two messages support these requirements - MARS_JOIN and MARS_LEAVE. These are sent to the MARS by endpoints when the local layer 3/ATM interface is requested to join or leave a multicast group. The MARS propagates these messages back out over ClusterControlVC, to ensure the knowledge of the group's membership change is distributed in a timely fashion to other cluster members.

Certain models of layer 3 endpoints (e.g. IP multicast routers) expect to be able to receive packet traffic 'promiscuously' across all groups. This functionality may be emulated by allowing routers to request that the MARS returns them as 'wild card' members of all Class D addresses. However, a problem inherent in the current ATM model is that a completely promiscuous router may exhaust the local reassembly resources in its ATM interface. MARS_JOIN supports a generalisation to the notion of 'wild card' entries, enabling routers to limit themselves to 'blocks' of the Class D address space. Use of this facility is described in greater detail in Section 8.

A block can be as small as 1 (a single group) or as large as the entire multicast address space (e.g. default IPv4 'promiscuous' behaviour). A block is defined as all addresses between, and inclusive of, a <min,max> address pair. A MARS_JOIN or MARS_LEAVE may carry multiple <min,max> pairs.

Cluster members MUST provide ONLY a single <min,max> pair in each JOIN/LEAVE message they issue. However, they MUST be able to process multiple <min,max> pairs in JOIN/LEAVE messages when performing VC management as described in section 5.1.4 (the interpretation being that the join/leave operation applies to all addresses in the range from <min> to <max> inclusive, for every <min,max> pair).

In RFC1112 environments a MARS_JOIN for a single group is triggered by a JoinLocalGroup signal from the IP layer. A MARS_LEAVE for a single group is triggered by a LeaveLocalGroup signal from the IP layer.

Cluster members with special requirements (e.g. multicast routers) may issue MARS_JOINS and MARS_LEAVES specifying a single block of 2 or more multicast group addresses. However, a cluster member SHALL NOT issue such a multi-group block join for an address range fully or partially overlapped by multi-group block join(s) that the cluster member has previously issued and not yet retracted. A cluster member MAY issue combinations of single group MARS_JOINS that overlap with a multi-group block MARS_JOIN.

An endpoint MUST register with a MARS in order to become a member of a cluster and be added as a leaf to ClusterControlVC. Registration is covered in section 5.2.3.

Finally, the endpoint MUST be capable of terminating unidirectional VCs (i.e. act as a leaf node of a UNI 3.0/3.1 point to multipoint VC, with zero bandwidth assigned on the return path). RFC 1755 describes the signalling information required to terminate VCs carrying LLC/SNAP encapsulated traffic (discussed further in section 5.5).

5.2.1 Format of the MARS_JOIN and MARS_LEAVE Messages.

The MARS_JOIN message is indicated by an operation type value of 4. MARS_LEAVE has the same format and operation type value of 5. The message format is:

| | | |
|-------------|---------|---|
| Data: | | |
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hdrsv | 24 bits | Reserved. Unused by MARS control protocol. |
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code (MARS_JOIN or MARS_LEAVE). |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$sstl | 8 bits | Type & length of source ATM subaddress. (q) |
| mar\$spln | 8 bits | Length of source protocol address (s) |
| mar\$tpln | 8 bits | Length of group address (z) |
| mar\$pnum | 16 bits | Number of group address pairs (N) |
| mar\$flags | 16 bits | layer3grp, copy, and register flags. |
| mar\$cmi | 16 bits | Cluster Member ID |
| mar\$msn | 32 bits | MARS Sequence Number. |
| mar\$sha | roctets | source ATM number. |
| mar\$ssa | qoctets | source ATM subaddress. |
| mar\$spa | soctets | source protocol address |
| mar\$min.1 | zoctets | Minimum multicast group address - pair.1 |
| mar\$max.1 | zoctets | Maximum multicast group address - pair.1 |
| [.....] | | |
| mar\$min.N | zoctets | Minimum multicast group address - pair.N |
| mar\$max.N | zoctets | Maximum multicast group address - pair.N |

mar\$spln indicates the number of bytes in the source endpoint's protocol address, and is interpreted in the context of the protocol indicated by the mar\$pro field. (e.g. in IPv4 environments mar\$pro will be 0x800, mar\$spln is 4, and mar\$tpln is 4.)

The mar\$flags field contains three flags:

- Bit 15 - mar\$flags.layer3grp.
- Bit 14 - mar\$flags.copy.
- Bit 13 - mar\$flags.register.
- Bit 12 - mar\$flags.punched.
- Bit 0-7 - mar\$flags.sequence.

Bits 8 to 11 are reserved and MUST be zero.

mar\$flags.sequence is set by cluster members, and MUST always be passed on unmodified by the MARS when retransmitting MARS_JOIN or MARS_LEAVE messages. It is source specific, and MUST be ignored by other cluster members. Its use is described in section 5.2.2.

mar\$flags.punched MUST be zero when the MARS_JOIN or MARS_LEAVE is transmitted to the MARS. Its use is described in section 5.2.2 and section 6.2.4.

mar\$flags.copy MUST be set to 0 when the message is being sent from a MARS client, and MUST be set to 1 when the message is being sent from a MARS. (This flag is intended to support integrating the MARS function with one of the MARS clients in your cluster. The destination of an incoming MARS_JOIN can be determined from its value.)

mar\$flags.layer3grp allows the MARS to provide the group membership information described further in section 5.3. The rules for its use are:

mar\$flags.layer3grp MUST be set when the cluster member is issuing the MARS_JOIN as the result of a layer 3 multicast group being explicitly joined. (e.g. as a result of a JoinHostGroup operation in an RFC1112 compliant host).

mar\$flags.layer3grp MUST be reset in each MARS_JOIN if the MARS_JOIN is simply the local ip/atm interface registering to receive traffic on that group for its own reasons.

mar\$flags.layer3grp is ignored and MUST be treated as reset by the MARS for any MARS_JOIN that specifies a block covering more than a single group (e.g. a block join from a router ensuring their forwarding engines 'see' all traffic).

mar\$flags.register indicates whether the MARS_JOIN or MARS_LEAVE is being used to register or deregister a cluster member (described in section 5.2.3). When used to join or leave specific groups the mar\$register flag MUST be zero.

mar\$pnum indicates how many <min,max> pairs are included in the message. This field MUST be 1 when the message is sent from a cluster member. A MARS MAY return a MARS_JOIN or MARS_LEAVE with any mar\$pnum value, including zero. This will be explained further in section 6.2.4.

The mar\$cmi field MUST be zeroed by cluster members, and is used by the MARS during cluster member registration, described in section 5.2.3.

mar\$msn MUST be zero when transmitted by an endpoint. It is set to the current value of the Cluster Sequence Number by the MARS when the MARS_JOIN or MARS_LEAVE is retransmitted. Its use has been described in section 5.1.4.

To simplify construction and parsing of MARS_JOIN and MARS_LEAVE messages, the following restrictions are imposed on the <min,max> pairs:

Assume max(N) is the <max> field from the Nth <min,max> pair.
Assume min(N) is the <min> field from the Nth <min,max> pair.
Assume a join/leave message arrives with K <min,max> pairs.

The following must hold:

max(N) < min(N+1) for 1 <= N < K
max(N) >= min(N) for 1 <= N <= K

In plain language, the set must specify an ascending sequence of address blocks. The definition of "greater" or "less than" may be protocol specific. In IPv4 environments the addresses are treated as 32 bit, unsigned binary values (most significant byte first).

5.2.1.1 Important IPv4 default values.

The JoinLocalGroup and LeaveLocalGroup operations are only valid for a single group. For any arbitrary group address X the associated MARS_JOIN or MARS_LEAVE MUST specify a single pair <X, X>. mar\$flags.layer3grp MUST be set under these circumstances.

A router choosing to behave strictly in accordance with RFC1112 MUST specify the entire Class D space. The associated MARS_JOIN or MARS_LEAVE MUST specify a single pair <224.0.0.0, 239.255.255.255>. Whenever a router issues a MARS_JOIN only in order to forward IP traffic it MUST reset mar\$flags.layer3grp.

The use of alternative <min, max> values by multicast routers is discussed in Section 8.

5.2.2 Retransmission of MARS_JOIN and MARS_LEAVE messages.

Transient problems may result in the loss of messages between the MARS and cluster members

A simple algorithm is used to solve this problem. Cluster members retransmit each MARS_JOIN and MARS_LEAVE message at regular intervals until they receive a copy back again, either on ClusterControlVC or the VC on which they are sending the message. At this point the local endpoint can be certain that the MARS received and processed it.

The interval should be no shorter than 5 seconds, and a default value of 10 seconds is recommended. After 5 retransmissions the attempt should be flagged locally as a failure. This MUST be considered as a MARS failure, and triggers the MARS reconnection described in section 5.4.

A 'copy' is defined as a received message with the following fields matching a previously transmitted MARS_JOIN/LEAVE:

- mar\$op
- mar\$flags.register
- mar\$flags.sequence
- mar\$pnum
- Source ATM address
- First <min,max> pair

In addition, a valid copy MUST have the following field values:

- mar\$flags.punched = 0
- mar\$flags.copy = 1

The mar\$flags.sequence field is never modified or checked by a MARS. Implementors MAY choose to utilize locally significant sequence number schemes, which MAY differ from one cluster member to the next. In the absence of such schemes the default value for mar\$flags.sequence MUST be zero.

Careful implementations MAY have more than one unacknowledged MARS_JOIN/LEAVE outstanding at a time.

5.2.3 Cluster member registration and deregistration.

To become a cluster member an endpoint must register with the MARS. This achieves two things - the endpoint is added as a leaf node of ClusterControlVC, and the endpoint is assigned a 16 bit Cluster Member Identifier (CMI). The CMI uniquely identifies each endpoint that is attached to the cluster.

Registration with the MARS occurs when an endpoint issues a MARS_JOIN with the mar\$flags.register flag set to one (bit 13 of the mar\$flags field).

The cluster member MUST include its source ATM address, and MAY choose to specify a null source protocol address when registering.

No protocol specific group addresses are included in a registration MARS_JOIN.

The cluster member retransmits this MARS_JOIN in accordance with section 5.2.2 until it confirms that the MARS has received it.

When the registration MARS_JOIN is returned it contains a non-zero value in mar\$cmi. This value MUST be noted by the cluster member, and used whenever circumstances require the cluster member's CMI.

An endpoint may also choose to de-register, using a MARS_LEAVE with mar\$flags.register set. This would result in the MARS dropping the endpoint from ClusterControlVC, removing all references to the member in the mapping database, and freeing up its CMI.

As for registration, a deregistration request MUST include the correct source ATM address for the cluster member, but MAY choose to specify a null source protocol address.

The cluster member retransmits this MARS_LEAVE in accordance with section 5.2.2 until it confirms that the MARS has received it.

5.3 Support for Layer 3 group management.

Whilst the intention of this specification is to be independent of layer 3 issues, an attempt is being made to assist the operation of layer 3 multicast routing protocols that need to ascertain if any groups have members within a cluster.

One example is IP, where IGMP is used (as described in section 2) simply to determine whether any other cluster members are listening to a group because they have higher layer applications that want to receive a group's traffic.

Routers may choose to query the MARS for this information, rather than multicasting IGMP queries to 224.0.0.1 and incurring the associated cost of setting up a VC to all systems in the cluster.

The query is issued by sending a MARS_GROUPLIST_REQUEST to the MARS. MARS_GROUPLIST_REQUEST is built from a MARS_JOIN, but it has an operation code of 10. The first <min,max> pair will be used by the MARS to identify the range of groups in which the querying cluster member is interested. Any additional <min,max> pairs will be ignored. A request with mar\$pnum = 0 will be ignored.

The response from the MARS is a MARS_GROUPLIST_REPLY, carrying a list of the multicast groups within the specified <min,max> block that have Layer 3 members. A group is noted in this list if one or more of the MARS_JOINS that generated its mapping entry in the MARS contained a set mar\$flags.layer3grp flag.

MARS_GROUPLIST_REPLYs are transmitted back to the querying cluster member on the VC used to send the MARS_GROUPLIST_REQUEST.

MARS_GROUPLIST_REPLY is derived from the MARS_MULTI but with mar\$op = 11. It may have multiple parts if needed, and is received in a similar manner to a MARS_MULTI.

Data:

| | | |
|-------------|---------|---|
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hrrsv | 24 bits | Reserved. Unused by MARS control protocol. |
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code (MARS_GROUPLIST_REPLY). |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$stl | 8 bits | Type & length of source ATM subaddress. (q) |
| mar\$spln | 8 bits | Length of source protocol address (s) |
| mar\$thtl | 8 bits | Unused - set to zero. |
| mar\$stl | 8 bits | Unused - set to zero. |
| mar\$tpln | 8 bits | Length of target group address (z) |
| mar\$tnum | 16 bits | Number of group addresses returned (N). |
| mar\$seqxy | 16 bits | Boolean flag x and sequence number y. |
| mar\$msn | 32 bits | MARS Sequence Number. |
| mar\$sha | roctets | source ATM number. |
| mar\$ssa | qoctets | source ATM subaddress. |
| mar\$spa | soctets | source protocol address |
| mar\$mgrp.1 | zoctets | Group address 1 |
| | [.....] | |
| mar\$mgrp.N | zoctets | Group address N |

mar\$seqxy is coded as for the MARS_MULTI - multiple

MARS_GROUPLIST_REPLY components are transmitted and received using the same algorithm as described in section 5.1.1 for MARS_MULTII. The only difference is that protocol addresses are being returned rather than ATM addresses.

As for MARS_MULTIIs, if an error occurs in the reception of a multi part MARS_GROUPLIST_REPLY the whole thing MUST be discarded and the MARS_GROUPLIST_REQUEST re-issued. (This includes the mar\$msn value being constant.)

Note that the ability to generate MARS_GROUPLIST_REQUEST messages, and receive MARS_GROUPLIST_REPLY messages, is not required for general host interface implementations. It is optional for interfaces being implemented to support layer 3 multicast forwarding engines. However, this functionality MUST be supported by the MARS.

5.4 Support for redundant/backup MARS entities.

Endpoints are assumed to have been configured with the ATM address of at least one MARS. Endpoints MAY choose to maintain a table of ATM addresses, representing alternative MARSS that will be contacted in the event that normal operation with the original MARS is deemed to have failed. It is assumed that this table orders the ATM addresses in descending order of preference.

An endpoint will typically decide there are problems with the MARS when:

- It fails to establish a point to point VC to the MARS.
- MARS_REQUESTs fail (section 5.1.1).
- MARS_JOIN/MARS_LEAVES fail (section 5.2.2).
- It has not received a MARS_REDIRECT_MAP in the last 4 minutes (section 5.4.3).

(If it is able to discern which connection represents ClusterControlVC, it may also use connection failures on this VC to indicate problems with the MARS).

5.4.1 First response to MARS problems.

The first response is to assume a transient problem with the MARS being used at the time. The cluster member should wait a random period of time between 1 and 10 seconds before attempting to re-connect and re-register with the MARS. If the registration MARS_JOIN is successful then:

The cluster member MUST then proceed to rejoin every group that its local higher layer protocol(s) have joined. It is

recommended that a random delay between 1 and 10 seconds be inserted before attempting each MARS_JOIN.

The cluster member MUST initiate the revalidation of every multicast group it was sending to (as though a sequence number jump had been detected, section 5.1.5).

The rejoin and revalidation procedure must not disrupt the cluster member's use of multipoint VCs that were already open at the time of the MARS failure.

If re-registration with the current MARS fails, and there are no backup MARS addresses configured, the cluster member MUST wait for at least 1 minute before repeating the re-registration procedure. It is RECOMMENDED that the cluster member signals an error condition in some locally significant fashion.

This procedure may repeat until network administrators manually intervene or the current MARS returns to normal operation.

5.4.2 Connecting to a backup MARS.

If the re-registration with the current MARS fails, and other MARS addresses have been configured, the next MARS address on the list is chosen to be the current MARS, and the cluster member immediately restarts the re-registration procedure described in section 5.4.1. If this is successful the cluster member will resume normal operation using the new MARS. It is RECOMMENDED that the cluster member signals a warning of this condition in some locally significant fashion.

If the attempt at re-registration with the new MARS fails, the cluster member MUST wait for at least 1 minute before choosing the next MARS address in the table and repeating the procedure. If the end of the table has been reached, the cluster member starts again at the top of the table (which should be the original MARS that the cluster member started with).

In the worst case scenario this will result in cluster members looping through their table of possible MARS addresses until network administrators manually intervene.

5.4.3 Dynamic backup lists, and soft redirects.

To support some level of autoconfiguration, a MARS message is defined that allows the current MARS to broadcast on ClusterControlVC a table of backup MARS addresses. When this message is received, cluster members that maintain a list of backup MARS addresses MUST insert this information at the top of their locally held list (i.e. the

information provided by the MARS has a higher preference than addresses that may have been manually configured into the cluster member).

The message is MARS_REDIRECT_MAP. It is based on the MARS_MULTI message, with the following changes:

- mar\$tpln field replaced by mar\$redirf.
- mar\$spln field reserved.
- mar\$tpa and mar\$spa eliminated.

MARS_REDIRECT_MAP has an operation type code of 12 decimal.

Data:

| | | |
|-------------|---------|---|
| mar\$afn | 16 bits | Address Family (0x000F). |
| mar\$pro | 56 bits | Protocol Identification. |
| mar\$hrrsv | 24 bits | Reserved. Unused by MARS control protocol. |
| mar\$chksum | 16 bits | Checksum across entire MARS message. |
| mar\$extoff | 16 bits | Extensions Offset. |
| mar\$op | 16 bits | Operation code (MARS_REDIRECT_MAP). |
| mar\$shtl | 8 bits | Type & length of source ATM number. (r) |
| mar\$stl | 8 bits | Type & length of source ATM subaddress. (q) |
| mar\$spln | 8 bits | Length of source protocol address (s) |
| mar\$thtl | 8 bits | Type & length of target ATM number (x) |
| mar\$stl | 8 bits | Type & length of target ATM subaddress (y) |
| mar\$redirf | 8 bits | Flag controlling client redirect behaviour. |
| mar\$num | 16 bits | Number of MARS addresses returned (N). |
| mar\$seqxy | 16 bits | Boolean flag x and sequence number y. |
| mar\$msn | 32 bits | MARS Sequence Number. |
| mar\$sha | roctets | source ATM number |
| mar\$ssa | qoctets | source ATM subaddress |
| mar\$tha.1 | xoctets | ATM number for MARS 1 |
| mar\$tsa.1 | yoctets | ATM subaddress for MARS 1 |
| mar\$tha.2 | xoctets | ATM number for MARS 2 |
| mar\$tsa.2 | yoctets | ATM subaddress for MARS 2 |
| [.....] | | |
| mar\$tha.N | xoctets | ATM number for MARS N |
| mar\$tsa.N | yoctets | ATM subaddress for MARS N |

The source ATM address field(s) MUST identify the originating MARS. A multi-part MARS_REDIRECT_MAP may be transmitted and reassembled using the mar\$seqxy field in the same manner as a multi-part MARS_MULTI (section 5.1.1). If a failure occurs during the reassembly of a multi-part MARS_REDIRECT_MAP (a part lost, reassembly timeout, or illegal MARS Sequence Number jump) the entire message MUST be discarded.

This message is transmitted regularly by the MARS (it MUST be transmitted at least every 2 minutes, it is RECOMMENDED that it is transmitted every 1 minute).

The MARS_REDIRECT_MAP is also used to force cluster members to shift from one MARS to another. If the ATM address of the first MARS contained in a MARS_REDIRECT_MAP table is not the address of cluster member's current MARS the client MUST 'redirect' to the new MARS. The mar\$redirf field controls how the redirection occurs.

mar\$redirf has the following format:

```

      7 6 5 4 3 2 1 0
+---+---+---+---+---+
|x|               |
+---+---+---+---+---+
```

If Bit 7 (the most significant bit) of mar\$redirf is 1 then the cluster member MUST perform a 'hard' redirect. Having installed the new table of MARS addresses carried by the MARS_REDIRECT_MAP, the cluster member re-registers with the MARS now at the top of the table using the mechanism described in sections 5.4.1 and 5.4.2.

If Bit 7 of mar\$redirf is 0 then the cluster member MUST perform a 'soft' redirect, beginning with the following actions:

- open a point to point VC to the first ATM address.
- attempt a registration (section 5.2.3).

If the registration succeeds, the cluster member shuts down its point to point VC to the current MARS (if it had one open), and then proceeds to use the newly opened point to point VC as its connection to the 'current MARS'. The cluster member does NOT attempt to rejoin the groups it is a member of, or revalidate groups it is currently sending to.

This is termed a 'soft redirect' because it avoids the extra rejoining and revalidation processing that occurs when a MARS failure is being recovered from. It assumes some external synchronisation mechanisms exist between the old and new MARS - mechanisms that are outside the scope of this specification.

Some level of trust is required before initiating a soft redirect. A cluster member MUST check that the calling party at the other end of the VC on which the MARS_REDIRECT_MAP arrived (supposedly ClusterControlVC) is in fact the node it trusts as the current MARS.

Additional applications of this function are for further study.

5.5 Data path LLC/SNAP encapsulations.

An extended encapsulation scheme is required to support the filtering of possible reflected packets (section 3.3).

Two LLC/SNAP codepoints are allocated from the IANA OUI space. These support two different mechanisms for detecting reflected packets. They are called Type #1 and Type #2 multicast encapsulations.

Type #1

```
[0xAA-AA-03][0x00-00-5E][0x00-01][Type #1 Extended Layer 3 packet]
      LLC           OUI           PID
```

Type #2

```
[0xAA-AA-03][0x00-00-5E][0x00-04][Type #2 Extended Layer 3 packet]
      LLC           OUI           PID
```

For conformance with this document MARS clients:

MUST transmit data using Type #1 encapsulation.

MUST be able to correctly receive traffic using Type #1 OR Type #2 encapsulation.

MUST NOT transmit using Type #2 encapsulation.

5.5.1 Type #1 encapsulation.

The Type #1 Extended layer 3 packet carries within it a copy of the source's Cluster Member ID (CMI) and either the 'short form' or 'long form' of the protocol type as appropriate (section 4.3).

When carrying packets belonging to protocols with valid short form representations the [Type #1 Extended Layer 3 packet] is encoded as:

```
[pkt$cmi][pkt$pro][Original Layer 3 packet]
  2octet  2octet      N octet
```

The first 2 octets (pkt\$cmi) carry the CMI assigned when an endpoint registers with the MARS (section 5.2.3). The second 2 octets (pkt\$pro) indicate the protocol type of the packet carried in the remainder of the payload. This is copied from the mar\$pro field used in the MARS control messages.

When carrying packets belonging to protocols that only have a long form representation (pkt\$pro = 0x80) the overhead SHALL be further

extended to carry the 5 byte `mar$pro.snap` field (with padding for 32 bit alignment). The encoded form SHALL be:

```
[pkt$cmi][0x00-80][mar$pro.snap][padding][Original Layer 3 packet]
  2octet   2octet   5 octets   3 octets           N octet
```

The CMI is copied into the `pkt$cmi` field of every outgoing Type #1 packet. When an endpoint interface receives an AAL_SDU with the LLC/SNAP codepoint indicating Type #1 encapsulation it compares the CMI field with its own Cluster Member ID for the indicated protocol. The packet is discarded silently if they match. Otherwise the packet is accepted for processing by the local protocol entity identified by the `pkt$pro` (and possibly SNAP) field(s).

Where a protocol has valid short and long forms of identification, receivers MAY choose to additionally recognise the long form.

5.5.2 Type #2 encapsulation.

Future developments may enable direct multicasting of AAL_SDUs beyond cluster boundaries. Expanding the set of possible sources in this way may cause the CMI to become an inadequate parameter with which to detect reflected packets. A larger source identification field may be required.

The Type #2 Extended layer 3 packet carries within it an 8 octet source ID field and either the 'short form' or 'long form' of the protocol type as appropriate (section 4.3). The form and content of the source ID field is currently unspecified, and is not relevant to any MARS client built in conformance with this document. Received Type #2 encapsulated packets MUST always be accepted and passed up to the higher layer indicated by the protocol identifier.

When carrying packets belonging to protocols with valid short form representations the [Type #2 Extended Layer 3 packet] is encoded as:

```
[8 octet sourceID][mar$pro.type][Null pad][Original Layer 3
packet]
                2octets      2octets
```

When carrying packets belonging to protocols that only have a long form representation (`pkt$pro = 0x80`) the overhead SHALL be further extended to carry the 5 byte `mar$pro.snap` field (with padding for 32 bit alignment). The encoded form SHALL be:

```
[8 octet sourceID][mar$pro.type][mar$pro.snap][Null pad][Layer 3
packet]
```

2octets 5octets 1octet

(Note that in this case the padding after the SNAP field is 1 octet rather than the 3 octets used in Type #1.)

Where a protocol has valid short and long forms of identification, receivers MAY choose to additionally recognise the long form.

(Future documents may specify the contents of the source ID field. This will only be relevant to implementations sending Type #2 encapsulated packets, as they are the only entities that need to be concerned about detecting reflected Type #2 packets.)

5.5.3 A Type #1 example.

An IPv4 packet (fully identified by an Ethertype of 0x800, therefore requiring 'short form' protocol type encoding) would be transmitted as:

```
[0xAA-AA-03][0x00-00-5E][0x00-01][pkt$cmi][0x800][IPv4 packet]
```

The different LLC/SNAP codepoints for unicast and multicast packet transmission allows a single IPv4/ATM interface to support both by demuxing on the LLC/SNAP header.

6. The MARS in greater detail.

Section 5 implies a lot about the MARS's basic behaviour as observed by cluster members. This section summarises the behaviour of the MARS for groups that are VC mesh based, and describes how a MARSs behaviour changes when an MCS is registered to support a group.

The MARS is intended to be a multiprotocol entity - all its mapping tables, CMIs, and control VCs MUST be managed within the context of the mar\$pro field in incoming MARS messages. For example, a MARS supports completely separate ClusterControlVCs for each layer 3 protocol that it is registering members for. If a MARS receives messages with a mar\$pro that it does not support, the message is dropped.

In general the MARS treats protocol addresses as arbitrary byte strings. For example, the MARS will not apply IPv4 specific 'class' checks to addresses supplied under mar\$pro = 0x800. It is sufficient for the MARS to simply assume that endpoints know how to interpret the protocol addresses that they are establishing and releasing mappings for.

The MARS requires control messages to carry the originator's identity in the source ATM address field(s). Messages that arrive with an empty ATM Number field are silently discarded prior to any other processing by the MARS. (Only the ATM Number field needs to be checked. An empty ATM Number field combined with a non-empty ATM Subaddress field does not represent a valid ATM address.)

(Some example pseudo-code for a MARS can be found in Appendix F.)

6.1 Basic interface to Cluster members.

The following MARS messages are used or required by cluster members:

- 1 MARS_REQUEST
- 2 MARS_MULTI
- 4 MARS_JOIN
- 5 MARS_LEAVE
- 6 MARS_NAK
- 10 MARS_GROUPLIST_REQUEST
- 11 MARS_GROUPLIST_REPLY
- 12 MARS_REDIRECT_MAP

6.1.1 Response to MARS_REQUEST.

Except as described in section 6.2, if a MARS_REQUEST arrives whose source ATM address does not match that of any registered Cluster member the message MUST be dropped and ignored.

6.1.2 Response to MARS_JOIN and MARS_LEAVE.

When a registration MARS_JOIN arrives (described in section 5.2.3) the MARS performs the following actions:

- Adds the node to ClusterControlVC.
- Allocates a new Cluster Member ID (CMI).
- Inserts the new CMI into the mar\$cmi field of the MARS_JOIN.
- Retransmits the MARS_JOIN back privately.

If the node is already a registered member of the cluster associated with the specified protocol type then its existing CMI is simply copied into the MARS_JOIN, and the MARS_JOIN retransmitted back to the node. A single node may register multiple times if it supports multiple layer 3 protocols. The CMIs allocated by the MARS for each such registration may or may not be the same.

The retransmitted registration MARS_JOIN must NOT be sent on ClusterControlVC. If a cluster member issues a deregistration MARS_LEAVE it too is retransmitted privately.

Non-registration MARS_JOIN and MARS_LEAVE messages are ignored if they arrive from a node that is not registered as a cluster member.

MARS_JOIN or MARS_LEAVE messages MUST arrive at the MARS with mar\$flags.copy set to 0, otherwise the message is silently ignored.

All outgoing MARS_JOIN or MARS_LEAVE messages SHALL have mar\$flags.copy set to 1, and mar\$msn set to the current Cluster Sequence Number for ClusterControlVC (Section 5.1.4.2).

mar\$flags.layer3grp (section 5.3) MUST be treated as reset for MARS_JOINS specifying a single <min,max> pair covering more than a single group. If a MARS_JOIN/LEAVE is received that contains more than one <min,max> pair, the MARS MUST silently drop the message.

If one or more MCSs have registered with the MARS, message processing continues as described in section 6.2.4.

The MARS database is updated to add the node to any indicated group(s) that it was not already considered a member of, and message processing continues as follows:

If a single group was being joined or left:

mar\$flags.punched is set to 0.

If the joining (leaving) node was already (is still) considered a member of the specified group, the message is retransmitted privately back to the cluster member. Otherwise the message is retransmitted on ClusterControlVC.

If a single block covering 2 or more groups was being joined or left:

A copy of the original MARS_JOIN/LEAVE is made. This copy then has its <min,max> block replaced with a 'hole punched' set of zero or more <min,max> pairs. The 'hole punched' set of <min,max> pairs covers the entire address range specified by the original <min,max> pair, but excludes those addresses/groups which the joining (leaving) node is already (still) a member of due to a previous single group join.

If no 'holes' were punched in the specified block, the original MARS_JOIN/LEAVE is retransmitted out on ClusterControlVC. Otherwise the following occurs:

The original MARS_JOIN/LEAVE is transmitted back to the source cluster member unchanged, using the VC it arrived on. The mar\$flags.punched field MUST be reset to 0 in this message.

If the hole-punched set contains 1 or more <min,max> pair, the copy of the original MARS_JOIN/LEAVE is transmitted on ClusterControlVC, carrying the new <min,max> list. The mar\$flags.punched field MUST be set to 1 in this message. (The mar\$flags.punched field is set to ensure the hole-punched copy is ignored by the message's source when trying to match received MARS_JOIN/LEAVE messages with ones previously sent (section 5.2.2)).

If the MARS receives a deregistration MARS_LEAVE (described in section 5.2.3) that member's ATM address MUST be removed from all groups for which it may have joined, dropped from ClusterControlVC, and the CMI released.

If the MARS receives an ERR_L_RELEASE on ClusterControlVC indicating that a cluster member has disconnected, that member's ATM address MUST be removed from all groups for which it may have joined, and the CMI released.

6.1.3 Generating MARS_REDIRECT_MAP.

A MARS_REDIRECT_MAP message (described in section 5.4.3) MUST be regularly transmitted on ClusterControlVC. It is RECOMMENDED that this occur every 1 minute, and it MUST occur at least every 2 minutes. If the MARS has no knowledge of other backup MARSs serving the cluster, it MUST include its own address as the only entry in the MARS_REDIRECT_MAP message (in addition to filling in the source address fields).

The design and use of backup MARS entities is beyond the scope of this document, and will be covered in future work.

6.1.4 Cluster Sequence Numbers.

The Cluster Sequence Number (CSN) is described in section 5.1.4, and is carried in the mar\$msn field of MARS messages being sent to cluster members (either out ClusterControlVC or on an individual VC). The MARS increments the CSN after every transmission of a message on ClusterControlVC. The current CSN is copied into the mar\$msn field of MARS messages being sent to cluster members, whether out ClusterControlVC or on a private VC.

A MARS should be carefully designed to minimise the possibility of the CSN jumping unnecessarily. Under normal operation only cluster members affected by transient link problems will miss CSN updates and be forced to revalidate. If the MARS itself glitches, it will be innundated with requests for a period as every cluster member attempts to revalidate.

Calculations on the CSN MUST be performed as unsigned 32 bit arithmetic.

One implication of this mechanism is that the MARS should serialize its processing of 'simultaneous' MARS_REQUEST, MARS_JOIN and MARS_LEAVE messages. Join and Leave operations should be queued within the MARS along with MARS_REQUESTS, and not processed until all the reply packets of a preceeding MARS_REQUEST have been transmitted. The transmission of MARS_REDIRECT_MAP should also be similarly queued.

(The regular transmission of MARS_REDIRECT_MAP serves a secondary purpose of allowing cluster members to track the CSN, even if they miss an earlier MARS_JOIN or MARS_LEAVE.)

6.2 MARS interface to Multicast Servers (MCS).

When the MARS returns the actual addresses of group members, the endpoint behaviour described in section 5 results in all groups being supported by meshes of point to multipoint VCs. However, when MCSs register to support particular layer 3 multicast groups the MARS modifies its use of various MARS messages to fool endpoints into using the MCS instead.

The following MARS messages are associated with interaction between the MARS and MCSs.

- 3 MARS_MSERV
- 7 MARS_UNSERV
- 8 MARS_SJOIN
- 9 MARS_SLEAVE

The following MARS messages are treated in a slightly different manner when MCSs have registered to support certain group addresses:

- 1 MARS_REQUEST
- 4 MARS_JOIN
- 5 MARS_LEAVE

A MARS must keep two sets of mappings for each layer 3 group using MCS support. The original {layer 3 address, ATM.1, ATM.2, ... ATM.n} mapping (now termed the 'host map', although it includes routers) is augmented by a parallel {layer 3 address, server.1, server.2, server.K} mapping (the 'server map'). It is assumed that no ATM addresses appear in both the server and host maps for the same multicast group. Typically K will be 1, but it will be larger if multiple MCSs are configured to support a given group.

The MARS also maintains a point to multipoint VC out to any MCSs registered with it, called `ServerControlVC` (section 6.2.3). This serves an analogous role to `ClusterControlVC`, allowing the MARS to update the MCSs with group membership changes as they occur. A MARS MUST also send its regular `MARS_REDIRECT_MAP` transmissions on both `ServerControlVC` and `ClusterControlVC`.

6.2.1 Response to a `MARS_REQUEST` if MCS is registered.

When the MARS receives a `MARS_REQUEST` for an address that has both host and server maps it generates a response based on the identity of the request's source. If the requestor is a member of the server map for the requested group then the MARS returns the contents of the host map in a sequence of one or more `MARS_MULTIs`. Otherwise, if the source is a valid cluster member, the MARS returns the contents of the server map in a sequence of one or more `MARS_MULTIs`. If the source is neither a cluster member, nor a member of the server map for the group, the request is dropped and ignored.

Servers use the host map to establish a basic distribution VC for the group. Cluster members will establish outgoing multipoint VCs to members of the group's server map, without being aware that their packets will not be going directly to the multicast group's members.

6.2.2 `MARS_MSERV` and `MARS_UNSERV` messages.

`MARS_MSERV` and `MARS_UNSERV` are identical to the `MARS_JOIN` message. An MCS uses a `MARS_MSERV` with a `<min,max>` pair of `<X,X>` to specify the multicast group X that it is willing to support. A single group `MARS_UNSERV` indicates the group that the MCS is no longer willing to support. The operation code for `MARS_MSERV` is 3 (decimal), and `MARS_UNSERV` is 7 (decimal).

Both of these messages are sent to the MARS over a point to point VC (between MCS and MARS). After processing, they are retransmitted on `ServerControlVC` to allow other MCSs to note the new node.

When registering or deregistering support for specific groups the `mar$flags.register` flag MUST be zero. (This flag is only one when the MCS is registering as a member of `ServerControlVC`, as described in section 6.2.3.)

When an MCS issues a `MARS_MSERV` for a specific group the message MUST be dropped and ignored if the source has not already registered with the MARS as a multicast server (section 6.2.3). Otherwise, the MARS adds the new ATM address to the server map for the specified group, possibly constructing a new server map if this is the first MCS for the group.

If a MARS_MSERV represents the first MCS to register for a particular group, and there exists a non null host map serving that particular group, the MARS issues a MARS_MIGRATE (section 5.1.6) on ClusterControlVC. The MARS's own identity is placed in the source protocol and hardware address fields of the MARS_MIGRATE. The ATM address of the MCS is placed as the first and only target ATM address. The address of the affected group is placed in the target multicast group address field.

If a MARS_MSERV is not the first MCS to register for a particular group the MARS simply changes its operation code to MARS_JOIN, and sends a copy of the message on ClusterControlVC. This fools the cluster members into thinking a new leaf node has been added to the group specified. In the retransmitted MARS_JOIN mar\$flags.layer3grp MUST be zero, mar\$flags.copy MUST be one, and mar\$flags.register MUST be zero.

When an MCS issues a MARS_UNSERV the MARS removes its ATM address from the server maps for each specified group, deleting any server maps that end up being null after the operation.

The operation code is then changed to MARS_LEAVE and the MARS sends a copy of the message on ClusterControlVC. This fools the cluster members into thinking a leaf node has been dropped from the group specified. In the retransmitted MARS_LEAVE mar\$flags.layer3grp MUST be zero, mar\$flags.copy MUST be one, and mar\$flags.register MUST be zero.

The MARS retransmits redundant MARS_MSERV and MARS_UNSERV messages directly back to the MCS generating them. MARS_MIGRATE messages are never repeated in response to redundant MARS_MSERVs.

The last or only MCS for a group MAY choose to issue a MARS_UNSERV while the group still has members. When the MARS_UNSERV is processed by the MARS the 'server map' will be deleted. When the associated MARS_LEAVE is issued on ClusterControlVC, all cluster members with a VC open to the MCS for that group will close down the VC (in accordance with section 5.1.4, since the MCS was their only leaf node). When cluster members subsequently find they need to transmit packets to the group, they will begin again with the MARS_REQUEST/MARS_MULTI sequence to establish a new VC. Since the MARS will have deleted the server map, this will result in the host map being returned, and the group reverts to being supported by a VC mesh.

The reverse process is achieved through the MARS_MIGRATE message when the first MCS registers to support a group. This ensures that cluster members explicitly dismantle any VC mesh they may have had

up, and re-establish their multicast forwarding path with the MCS as its termination point.

6.2.3 Registering a Multicast Server (MCS).

Section 5.2.3 describes how endpoints register as cluster members, and hence get added as leaf nodes to ClusterControlVC. The same approach is used to register endpoints that intend to provide MCS support.

Registration with the MARS occurs when an endpoint issues a MARS_MSERV with `mar$flags.register` set to one. Upon registration the endpoint is added as a leaf node to ServerControlVC, and the MARS_MSERV is returned to the MCS privately.

The MCS retransmits this MARS_MSERV until it confirms that the MARS has received it (by receiving a copy back, in an analogous way to the mechanism described in section 5.2.2 for reliably transmitting MARS_JOINS).

The `mar$cmi` field in MARS_MSERVs MUST be set to zero by both MCS and MARS.

An MCS may also choose to de-register, using a MARS_UNSERV with `mar$flags.register` set to one. When this occurs the MARS MUST remove all references to that MCS in all servermaps associated with the protocol (`mar$pro`) specified in the MARS_UNSERV, and drop the MCS from ServerControlVC.

Note that multiple logical MCSs may share the same physical ATM interface, provided that each MCS uses a separate ATM address (e.g. a different SEL field in the NSAP format address). In fact, an MCS may share the ATM interface of a node that is also a cluster member (either host or router), provided each logical entity has a different ATM address.

A MARS MUST be capable of handling a multi-entry servermap. However, the possible use of multiple MCSs registering to support the same group is a subject for further study. In the absence of an MCS synchronisation protocol a system administrator MUST NOT allow more than one logical MCS to register for a given group.

6.2.4 Modified response to MARS_JOIN and MARS_LEAVE.

The existence of MCSs supporting some groups but not others requires the MARS to modify its distribution of single and block join/leave updates to cluster members. The MARS also adds two new messages - MARS_SJOIN and MARS_SLEAVE - for communicating group changes to MCSs

over ServerControlVC.

The MARS_SJOIN and MARS_SLEAVE messages are identical to MARS_JOIN, with operation codes 18 and 19 (decimal) respectively.

When a cluster member issues MARS_JOIN or MARS_LEAVE for a single group, the MARS checks to see if the group has an associated server map. If the specified group does not have a server map processing continues as described in section 6.1.2.

However, if a server map exists for the group a new set of actions are taken.

If the joining (leaving) node was not already (is no longer) considered a member of the specified group, a copy of the MARS_JOIN/LEAVE is made with type MARS_SJOIN or MARS_SLEAVE as appropriate, and transmitted on ServerControlVC. This allows the MCS(s) supporting the group to note the new member and update their data VCs.

The original message is transmitted back to the source cluster member unchanged, using the VC it arrived on rather than ClusterControlVC. The `mar$flags.punched` field MUST be reset to 0 in this message.

(Section 5.2.2 requires cluster members have a mechanism to confirm the reception of their message by the MARS. For mesh supported groups, using ClusterControlVC serves dual purpose of providing this confirmation and distributing group update information. When a group is MCS supported, there is no reason for all cluster members to process null join/leave messages on ClusterControlVC, so they are sent back on the private VC between cluster member and MARS.)

Receipt of a block MARS_JOIN (e.g. from a router coming on-line) or MARS_LEAVE requires a more complex response. The single `<min,max>` block may simultaneously cover mesh supported and MCS supported groups. However, cluster members only need to be informed of the mesh supported groups that the endpoint has joined. Only the MCSs need to know if the endpoint is joining any MCS supported groups.

The solution is to modify the MARS_JOIN or MARS_LEAVE that is retransmitted on ClusterControlVC. The following action is taken:

A copy of the MARS_JOIN/LEAVE is made with type MARS_SJOIN or MARS_SLEAVE as appropriate, with its `<min,max>` block replaced with a 'hole punched' set of zero or more `<min,max>` pairs. The 'hole punched' set of `<min,max>` pairs covers the entire address range specified by the original `<min,max>` pair, but excludes those

addresses/groups which the joining (leaving) node is already (still) a member of due to a previous single group join.

Before transmission on the ClusterControlVC, the original MARS_JOIN/LEAVE then has its <min,max> block replaced with a 'hole punched' set of zero or more <min,max> pairs. The 'hole punched' set of <min,max> pairs covers the entire address range specified by the original <min,max> pair, but excludes those addresses/groups supported by MCSs or which the joining (leaving) node is already (still) a member of due to a previous single group join.

If no 'holes' were punched in the specified block, the original MARS_JOIN/LEAVE is re-transmitted out on ClusterControlVC unchanged. Otherwise the following occurs:

The original MARS_JOIN/LEAVE is transmitted back to the source cluster member unchanged, using the VC it arrived on. The `mar$flags.punched` field MUST be reset to 0 in this message.

If the hole-punched set contains 1 or more <min,max> pair, a copy of the original MARS_JOIN/LEAVE is transmitted on ClusterControlVC, carrying the new <min,max> list. The `mar$flags.punched` field MUST be set to 1 in this message.

The `mar$flags.punched` field is set to ensure the hole-punched copy is ignored by the message's source when trying to match received MARS_JOIN/LEAVE messages with ones previously sent (section 5.2.2).

(Appendix A discusses some algorithms for 'hole punching'.)

It is assumed that MCSs use the MARS_SJOINS and MARS_SLEAVES to update their own VCs out to the actual group's members.

`mar$flags.layer3grp` is copied over into the messages transmitted by the MARS. `mar$flags.copy` MUST be set to one.

6.2.5 Sequence numbers for ServerControlVC traffic.

In an analogous fashion to the Cluster Sequence Number, the MARS keeps a Server Sequence Number (SSN) that is incremented after every transmission on ServerControlVC. The current value of the SSN is inserted into the `mar$msn` field of every message the MARS issues that it believes is destined for an MCS. This includes MARS_MULTIs that are being returned in response to a MARS_REQUEST from an MCS, and MARS_REDIRECT_MAP being sent on ServerControlVC. The MARS must check the MARS_REQUESTs source, and if it is a registered MCS the SSN is

copied into the `mar$msn` field, otherwise the CSN is copied into the `mar$msn` field.

MCSs are expected to track and use the SSNs in an analogous manner to the way endpoints use the CSN in section 5.1 (to trigger revalidation of group membership information).

A MARS should be carefully designed to minimise the possibility of the SSN jumping unnecessarily. Under normal operation only MCSs that are affected by transient link problems will miss `mar$msn` updates and be forced to revalidate. If the MARS itself glitches it will be innundated with requests for a period as every MCS attempts to revalidate.

6.3 Why global sequence numbers?

The CSN and SSN are global within the context of a given protocol (e.g. IPv4, `mar$pro` = 0x800). They count `ClusterControlVC` and `ServerControlVC` activity without reference to the multicast group(s) involved. This may be perceived as a limitation, because there is no way for cluster members or multicast servers to isolate exactly which multicast group they may have missed an update for. An alternative was to try and provide a per-group sequence number.

Unfortunately per-group sequence numbers are not practical. The current mechanism allows sequence information to be piggy-backed onto MARS messages already in transit for other reasons. The ability to specify blocks of multicast addresses with a single `MARS_JOIN` or `MARS_LEAVE` means that a single message can refer to membership change for multiple groups simultaneously. A single `mar$msn` field cannot provide meaningful information about each group's sequence. Multiple `mar$msn` fields would have been unwieldy.

Any MARS or cluster member that supports different protocols **MUST** keep separate mapping tables and sequence numbers for each protocol.

6.4 Redundant/Backup MARS Architectures.

If backup MARSS exist for a given cluster then mechanisms are needed to ensure consistency between their mapping tables and those of the active, current MARS.

(Cluster members will consider backup MARSS to exist if they have been configured with a table of MARS addresses, or the regular `MARS_REDIRECT_MAP` messages contain a list of 2 or more addresses.)

The definition of an MARS-synchronization protocol is beyond the current scope of this document, and is expected to be the subject of

further research work. However, the following observations may be made:

MARS_REDIRECT_MAP messages exist, enabling one MARS to force endpoints to move to another MARS (e.g. in the aftermath of a MARS failure, the chosen backup MARS will eventually wish to hand control of the cluster over to the main MARS when it is functioning properly again).

Cluster members and MCSs do not need to start up with knowledge of more than one MARS, provided that MARS correctly issues MARS_REDIRECT_MAP messages with the full list of MARSs for that cluster.

Any mechanism for synchronising backup MARSs (and coping with the aftermath of MARS failures) should be compatible with the cluster member behaviour described in this document.

7. How an MCS utilises a MARS.

When an MCS supports a multicast group it acts as a proxy cluster endpoint for the senders to the group. It also behaves in an analogous manner to a sender, managing a single outgoing point to multipoint VC to the real group members.

Detailed description of possible MCS architectures are beyond the scope of this document. This section will outline the main issues.

7.1 Association with a particular Layer 3 group.

When an MCS issues a MARS_MSERV it forces all senders to the specified layer 3 group to terminate their VCs on the supplied source ATM address.

The simplest MCS architecture involves taking incoming AAL_SDUs and simply flipping them back out a single point to multipoint VC. Such an MCS cannot support more than one group at once, as it has no way to differentiate between traffic destined for different groups. Using this architecture, a physical node would provide MCS support for multiple groups by creating multiple logical instances of the MCS, each with different ATM Addresses (e.g. a different SEL value in the node's NSAPA).

A slightly more complex approach would be to add minimal layer 3 specific processing into the MCS. This would look inside the received AAL_SDUs and determine which layer 3 group they are destined for. A single instance of such an MCS might register its ATM Address with the MARS for multiple layer 3 groups, and manage multiple independent

outgoing point to multipoint VCs (one for each group).

When an MCS starts up it MUST register with the MARS as described in section 6.2.3, identifying the protocol it supports with the `mar$pro` field of the `MARS_MSERV`. This also applies to logical MCSs, even if they share the same physical ATM interface. This is important so that the MARS can react to the loss of an MCS when it drops off `ServerControlVC`. (One consequence is that 'simple' MCS architectures end up with one `ServerControlVC` member per group. MCSs with layer 3 specific processing may support multiple groups while still only registering as one member of `ServerControlVC`.)

An MCS MUST NOT share the same ATM address as a cluster member, although it may share the same physical ATM interface.

7.2 Termination of incoming VCs.

An MCS MUST terminate unidirectional VCs in the same manner as a cluster member. (e.g. terminate on an LLC entity when LLC/SNAP encapsulation is used, as described in RFC 1755 for unicast endpoints.)

7.3 Management of outgoing VC.

An MCS MUST establish and manage its outgoing point to multipoint VC as a cluster member does (section 5.1).

`MARS_REQUEST` is used by the MCS to establish the initial leaf nodes for the MCS's outgoing point to multipoint VC. After the VC is established, the MCS reacts to `MARS_SJOINS` and `MARS_SLEAVES` in the same way a cluster member reacts to `MARS_JOINS` and `MARS_LEAVES`.

The MCS tracks the Server Sequence Number from the `mar$msn` fields of messages from the MARS, and revalidates its outgoing point to multipoint VC(s) when a sequence number jump occurs.

7.4 Use of a backup MARS.

The MCS uses the same approach to backup MARSs as a cluster member (section 5.4), tracking `MARS_REDIRECT_MAP` messages on `ServerControlVC`.

8. Support for IP multicast routers.

Multicast routers are required for the propagation of multicast traffic beyond the constraints of a single cluster (inter-cluster traffic). (In a sense, they are multicast servers acting at the next higher layer, with clusters, rather than individual endpoints, as

their abstract sources and destinations.)

Multicast routers typically participate in higher layer multicast routing algorithms and policies that are beyond the scope of this memo (e.g. DVMRP [5] in the IPv4 environment).

It is assumed that the multicast routers will be implemented over the same sort of IP/ATM interface that a multicast host would use. Their IP/ATM interfaces will register with the MARS as cluster members, joining and leaving multicast groups as necessary. As noted in section 5, multiple logical 'endpoints' may be implemented over a single physical ATM interface. Routers use this approach to provide interfaces into each of the clusters they will be routing between.

The rest of this section will assume a simple IPv4 scenario where the scope of a cluster has been limited to a particular LIS that is part of an overlaid IP network. Not all members of the LIS are necessarily registered cluster members (you may have unicast-only hosts in the LIS).

8.1 Forwarding into a Cluster.

If the multicast router needs to transmit a packet to a group within the cluster its IP/ATM interface opens a VC in the same manner as a normal host would. Once a VC is open, the router watches for MARS_JOIN and MARS_LEAVE messages and responds to them as a normal host would.

The multicast router's transmit side MUST implement inactivity timers to shut down idle outgoing VCs, as for normal hosts.

As with normal host, the multicast router does not need to be a member of a group it is sending to.

8.2 Joining in 'promiscuous' mode.

Once registered and initialised, the simplest model of IPv4 multicast router operation is for it to issue a MARS_JOIN encompassing the entire Class D address space. In effect it becomes 'promiscuous', as it will be a leaf node to all present and future multipoint VCs established to IPv4 groups on the cluster.

How a router chooses which groups to propagate outside the cluster is beyond the scope of this document.

Consistent with RFC 1112, IP multicast routers may retain the use of IGMP Query and IGMP Report messages to ascertain group membership. However, certain optimisations are possible, and are described in

section 8.5.

8.3 Forwarding across the cluster.

Under some circumstances the cluster may simply be another hop between IP subnets that have participants in a multicast group.

```
[LAN.1] ----- IPmcR.1 -- [cluster/LIS] -- IPmcR.2 ----- [LAN.2]
```

LAN.1 and LAN.2 are subnets (such as Ethernet) with attached hosts that are members of group X.

IPmcR.1 and IPmcR.2 are multicast routers with interfaces to the LIS.

A traditional solution would be to treat the LIS as a unicast subnet, and use tunneling routers. However, this would not allow hosts on the LIS to participate in the cross-LIS traffic.

Assume IPmcR.1 is receiving packets promiscuously on its LAN.1 interface. Assume further it is configured to propagate multicast traffic to all attached interfaces. In this case that means the LIS.

When a packet for group X arrives on its LAN.1 interface, IPmcR.1 simply sends the packet to group X on the LIS interface as a normal host would (Issuing MARS_REQUEST for group X, creating the VC, sending the packet).

Assuming IPmcR.2 initialised itself with the MARS as a member of the entire Class D space, it will have been returned as a member of X even if no other nodes on the LIS were members. All packets for group X received on IPmcR.2's LIS interface may be retransmitted on LAN.2.

If IPmcR.1 is similarly initialised the reverse process will apply for multicast traffic from LAN.2 to LAN.1, for any multicast group. The benefit of this scenario is that cluster members within the LIS may also join and leave group X at anytime.

8.4 Joining in 'semi-promiscuous' mode.

Both unicast and multicast IP routers have a common problem - limitations on the number of AAL contexts available at their ATM interfaces. Being 'promiscuous' in the RFC 1112 sense means that for every M hosts sending to N groups, a multicast router's ATM interface will have M*N incoming reassembly engines tied up.

It is not hard to envisage situations where a number of multicast groups are active within the LIS but are not required to be propagated beyond the LIS itself. An example might be a distributed

simulation system specifically designed to use the high speed IP/ATM environment. There may be no practical way its traffic could be utilised on 'the other side' of the multicast router, yet under the conventional scheme the router would have to be a leaf to each participating host anyway.

As this problem occurs below the IP layer, it is worth noting that 'scoping' mechanisms at the IP multicast routing level do not provide a solution. An IP level scope would still result in the router's ATM interface receiving traffic on the scoped groups, only to drop it.

In this situation the network administrator might configure their multicast routers to exclude sections of the Class D address space when issuing MARS_JOIN(s). Multicast groups that will never be propagated beyond the cluster will not have the router listed as a member, and the router will never have to receive (and simply ignore) traffic from those groups.

Another scenario involves the product $M \times N$ exceeding the capacity of a single router's interface (especially if the same interface must also support a unicast IP router service).

A network administrator may choose to add a second node, to function as a parallel IP multicast router. Each router would be configured to be 'promiscuous' over separate parts of the Class D address space, thus exposing themselves to only part of the VC load. This sharing would be completely transparent to IP hosts within the LIS.

Restricted promiscuous mode does not break RFC 1112's use of IGMP Report messages. If the router is configured to serve a given block of Class D addresses, it will receive the IGMP Report. If the router is not configured to support a given block, then the existence of an IGMP Report for a group in that block is irrelevant to the router. All routers are able to track membership changes through the MARS_JOIN and MARS_LEAVE traffic anyway. (Section 8.5 discusses a better alternative to IGMP within a cluster.)

Mechanisms and reasons for establishing these modes of operation are beyond the scope of this document.

8.5 An alternative to IGMP Queries.

An unfortunate aspect of IGMP is that it assumes multicasting of IP packets is a cheap and trivial event at the link layer. As a consequence, regular IGMP Queries are multicasted by routers to group 224.0.0.1. These queries are intended to trigger IGMP Replies by cluster members that have layer 3 members of particular groups.

The MARS_GROUPLIST_REQUEST and MARS_GROUPLIST_REPLY messages were designed to allow routers to avoid actually transmitting IGMP Queries out into a cluster.

Whenever the router's forwarding engine wishes to transmit an IGMP query, a MARS_GROUPLIST_REQUEST can be sent to the MARS instead. The resulting MARS_GROUPLIST_REPLY(s) (described in section 5.3) from the MARS carry all the information that the router would have ascertained from IGMP replies.

It is RECOMMENDED that multicast routers utilise this MARS service to minimise IGMP traffic within the cluster.

By default a MARS_GROUPLIST_REQUEST SHOULD specify the entire address space (e.g. <224.0.0.0, 239.255.255.255> in an IPv4 environment). However, routers serving part of the address space (as described in section 8.4) MAY choose to issue MARS_GROUPLIST_REQUESTs that specify only the subset of the address space they are serving.

(On the surface it would also seem useful for multicast routers to track MARS_JOINS and MARS_LEAVES that arrive with mar\$flags.layer3grp set. These might be used in lieu of IGMP Reports, to provide the router with timely indication that a new layer 3 group member exists within the cluster. However, this only works on VC mesh supported groups, and is therefore NOT recommended).

Appendix B discusses less elegant mechanisms for reducing the impact of IGMP traffic within a cluster, on the assumption that the IP/ATM interfaces to the cluster are being used by un-optimised IP multicasting code.

8.6 CMIs across multiple interfaces.

The Cluster Member ID is only unique within the Cluster managed by a given MARS. On the surface this might appear to leave us with a problem when a multicast router is routing between two or more Clusters using a single physical ATM interface. The router will register with two or more MARSS, and thereby acquire two or more independent CMI's. Given that each MARS has no reason to synchronise their CMI allocations, it is possible for a host in one cluster to have the same CMI as the router's interface to another Cluster. How does the router distinguish between its own reflected packets, and packets from that other host?

The answer lies in the fact that routers (and hosts) actually implement logical IP/ATM interfaces over a single physical ATM interface. Each logical interface will have a unique ATM Address (eg. an NSAP with different SElector fields, one for each logical

interface).

Each logical IP/ATM interface is configured with the address of a single MARS, attaches to only one cluster, and so has only one CMI to worry about. Each of the MARSs that the router is registered with will have been given a different ATM Address (corresponding to the different logical IP/ATM interfaces) in each registration MARS_JOIN.

When hosts in a cluster add the router as a leaf node, they'll specify the ATM Address of the appropriate logical IP/ATM interface on the router in the L_MULTI_ADD message. Thus, each logical IP/ATM interface will only have to check and filter on CMIs assigned by its own MARS.

In essence the cluster differentiation is achieved by ensuring that logical IP/ATM interfaces are assigned different ATM Addresses.

9. Multiprotocol applications of the MARS and MARS clients.

A deliberate attempt has been made to describe the MARS and associated mechanisms in a manner independent of a specific higher layer protocol being run over the ATM cloud. The immediate application of this document will be in an IPv4 environment, and this is reflected by the focus of key examples. However, the `mar$pro.type` and `mar$pro.snap` fields in every MARS control message allow any higher layer protocol that has a 'short form' or 'long form' of protocol identification (section 4.3) to be supported by a MARS.

Every MARS MUST implement entirely separate logical mapping tables and support. Every cluster member must interpret messages from the MARS in the context of the protocol type that the MARS message refers to.

Every MARS and MARS client MUST treat Cluster Member IDs in the context of the protocol type carried in the MARS message or data packet containing the CMI.

For example, IPv6 has been allocated an Ethertype of 0x86DD. This means the 'short form' of protocol identification must be used in the MARS control messages and the data path encapsulation (section 5.5). An IPv6 multicasting client sets the `mar$pro.type` field of every MARS message to 0x86DD. When carrying IPv6 addresses the `mar$spln` and `mar$tpln` fields are either 0 (for null or non-existent information) or 16 (for the full IPv6 address).

Following the rules in section 5.5, an IPv6 data packet is encapsulated as:

[0xAA-AA-03][0x00-00-5E][0x00-01][pkt\$cmi][0x86DD][IPv6 packet]

A host or endpoint interface that is using the same MARS to support multicasting needs of multiple protocols MUST not assume their CMI will be the same for each protocol.

10. Supplementary parameter processing.

The `mar$extoff` field in the [Fixed header] indicates whether supplementary parameters are being carried by a MARS control message. This mechanism is intended to enable the addition of new functionality to the MARS protocol in later documents.

Supplementary parameters are conveyed as a list of TLV (type, length, value) encoded information elements. The TLV(s) begin on the first 32 bit boundary following the [Addresses] field in the MARS control message (e.g. after `mar$tsa.N` in a `MARS_MULTII`, after `mar$max.N` in a `MARS_JOIN`, etc).

10.1 Interpreting the `mar$extoff` field.

If the `mar$extoff` field is non-zero it indicates that a list of one or more TLVs have been appended to the MARS message. The first TLV is found by treating `mar$extoff` as an unsigned integer representing an offset (in octets) from the beginning of the MARS message (the MSB of the `mar$afn` field).

As TLVs are 32 bit aligned the bottom 2 bits of `mar$extoff` are also reserved. A receiver MUST mask off these two bits before calculating the octet offset to the TLV list. A sender MUST set these two bits to zero.

If `mar$extoff` is zero no TLVs have been appended.

10.2 The format of TLVs.

When they exist, TLVs begin on 32 bit boundaries, are multiples of 32 bits in length, and form a sequential list terminated by a NULL TLV.

The TLV structure is:

[Type - 2 octets][Length - 2 octets][Value - n*4 octets]

The Type subfield indicates how the contents of the Value subfield are to be interpreted.

The Length subfield indicates the number of VALID octets in the Value subfield. Valid octets in the Value subfield start immediately after

the Length subfield. The offset (in octets) from the start of this TLV to the start of the next TLV in the list is given by the following formula:

$$\text{offset} = (\text{length} + 4 + ((4 - (\text{length} \& 3)) \% 4))$$

(where % is the modulus operator)

The Value subfield is padded with 0, 1, 2, or 3 octets to ensure the next TLV is 32 bit aligned. The padded locations MUST be set to zero.

(For example, a TLV that needed only 5 valid octets of information would be 12 octets long. The Length subfield would hold the value 5, and the Value subfield would be padded out to 8 bytes. The 5 valid octets of information begin at the first octet of the Value subfield.)

The Type subfield is formatted in the following way:

```

      | 1st octet | 2nd octet |
      7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| x |                                     y |
+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

The most significant 2 bits (Type.x) determine how a recipient should behave when it doesn't recognise the TLV type indicated by the lower 14 bits (Type.y). The required behaviours are:

Type.x = 0 Skip the TLV, continue processing the list.
 Type.x = 1 Stop processing, silently drop the MARS message.
 Type.x = 2 Stop processing, drop message, give error indication.
 Type.x = 3 Reserved. (currently treat as x = 0)

(The error indication generated when Type.x = 2 SHOULD be logged in some locally significant fashion. Consequential MARS message activity in response to such an error condition will be defined in future documents.)

The TLV type space (Type.y) is further subdivided to encourage use outside the IETF.

| | |
|-----------------|-----------------------------|
| 0 | Null TLV. |
| 0x0001 - 0x0FFF | Reserved for the IETF. |
| 0x1000 - 0x11FF | Allocated to the ATM Forum. |
| 0x1200 - 0x37FF | Reserved for the IETF. |
| 0x3800 - 0x3FFF | Experimental use. |

10.3 Processing MARS messages with TLVs.

Supplementary parameters act as modifiers to the basic behaviour specified by the `mar$op` field of any given MARS message.

If a MARS message arrives with a non-zero `mar$extoff` field its TLV list **MUST** be parsed before handling the MARS message in accordance with the `mar$op` value. Unrecognised TLVs **MUST** be handled as required by their `Type.x` value.

How TLVs modify basic MARS operations will be `mar$op` and TLV specific.

10.4 Initial set of TLV elements.

Conformance with this document only **REQUIRES** the recognition of one TLV, the Null TLV. This terminates a list of TLVs, and **MUST** be present if `mar$extoff` is non-zero in a MARS message. It **MAY** be the only TLV present.

The Null TLV is coded as:

```
[0x00-00][0x00-00]
```

Future documents will describe the formats, contents, and interpretations of additional TLVs. The minimal parsing requirements imposed by this document are intended to allow conformant MARS and MARS client implementations to deal gracefully and predictably with future TLV developments.

11. Key Decisions and open issues.

The key decisions this document proposes:

A Multicast Address Resolution Server (MARS) is proposed to co-ordinate and distribute mappings of ATM endpoint addresses to arbitrary higher layer 'multicast group addresses'. The specific case of IPv4 multicast is used as the example.

The concept of 'clusters' is introduced to define the scope of a MARS's responsibility, and the set of ATM endpoints willing to participate in link level multicasting.

A MARS is described with the functionality required to support intra-cluster multicasting using either VC meshes or ATM level multicast servers (MCSs).

LLC/SNAP encapsulation of MARS control messages allows MARS and ATMARP traffic to share VCs, and allows partially co-resident MARS and ATMARP entities.

New message types:

MARS_JOIN, MARS_LEAVE, MARS_REQUEST. Allow endpoints to join, leave, and request the current membership list of multicast groups.

MARS_MULTII. Allows multiple ATM addresses to be returned by the MARS in response to a MARS_REQUEST.

MARS_MSERV, MARS_UNSERV. Allow multicast servers to register and deregister themselves with the MARS.

MARS_SJOIN, MARS_SLEAVE. Allow MARS to pass on group membership changes to multicast servers.

MARS_GROUPLIST_REQUEST, MARS_GROUPLIST_REPLY. Allow MARS to indicate which groups have actual layer 3 members. May be used to support IGMP in IPv4 environments, and similar functions in other environments.

MARS_REDIRECT_MAP. Allow MARS to specify a set of backup MARS addresses.

MARS_MIGRATE. Allows MARS to force cluster members to shift from VC mesh to MCS based forwarding tree in single operation.

'wild card' MARS mapping table entries are possible, where a single ATM address is simultaneously associated with blocks of multicast group addresses.

For the MARS protocol `mar$op.version = 0`. The complete set of MARS control messages and `mar$op.type` values is:

- 1 MARS_REQUEST
- 2 MARS_MULTII
- 3 MARS_MSERV
- 4 MARS_JOIN
- 5 MARS_LEAVE
- 6 MARS_NAK
- 7 MARS_UNSERV
- 8 MARS_SJOIN
- 9 MARS_SLEAVE
- 10 MARS_GROUPLIST_REQUEST
- 11 MARS_GROUPLIST_REPLY

- 12 MARS_REDIRECT_MAP
- 13 MARS_MIGRATE

A number of issues are left open at this stage, and are likely to be the subject of on-going research and additional documents that build upon this one.

The specified endpoint behaviour allows the use of redundant/backup MARSS within a cluster. However, no specifications yet exist on how these MARSS co-ordinate amongst themselves. (The default is to only have one MARS per cluster.)

The specified endpoint behaviour and MARS service allows the use of multiple MCSs per group. However, no specifications yet exist on how this may be used, or how these MCSs co-ordinate amongst themselves. Until further work is done on MCS co-ordination protocols the default is to only have one MCS per group.

The MARS relies on the cluster member dropping off ClusterControlVC if the cluster member dies. It is not clear if additional mechanisms are needed to detect and delete 'dead' cluster members.

Supporting layer 3 'broadcast' as a special case of multicasting (where the 'group' encompasses all cluster members) has not been explicitly discussed.

Supporting layer 3 'unicast' as a special case of multicasting (where the 'group' is a single cluster member, identified by the cluster member's unicast protocol address) has not been explicitly discussed.

The future development of ATM Group Addresses and Leaf Initiated Join to ATM Forum's UNI specification has not been addressed. (However, the problems identified in this document with respect to VC scarcity and impact on AAL contexts will not be fixed by such developments in the signalling protocol.)

Possible modifications to the interpretation of the mar\$hrdrsv and mar\$afn fields in the Fixed header, based on different values for mar\$op.version, are for further study.

Security Considerations

Security issues are not addressed in this document.

Acknowledgments

The discussions within the IP over ATM Working Group have helped clarify the ideas expressed in this document. John Moy (Cascade Communications Corp.) initially suggested the idea of wild-card entries in the ARP Server. Drew Perkins (Fore Systems) provided rigorous and useful critique of early proposed mechanisms for distributing and validating group membership information. Susan Symington (and co-workers at MITRE Corp., Don Chirieleison, and Bill Barns) clearly articulated the need for multicast server support, proposed a solution, and challenged earlier block join/leave mechanisms. John Shirron (Fore Systems) provided useful improvements on my original revalidation procedures.

Susan Symington and Bryan Gleeson (Adaptec) independently championed the need for the service provided by MARS_GROUPLIST_REQUEST/REPLY. The new encapsulation scheme arose from WG discussions, captured by Bryan Gleeson in an interim Work in Progress (with Keith McCloghrie (Cisco), Andy Malis (Ascom Nexion), and Andrew Smith (Bay Networks) as key contributors). James Watt (Newbridge) and Joel Halpern (Newbridge) motivated the development of a more multiprotocol MARS control message format, evolving it away from its original ATMARP roots. They also motivated the development of Type #1 and Type #2 data path encapsulations. Rajesh Talpade (Georgia Tech) helped clarify the need for the MARS_MIGRATE function.

Maryann Maher (ISI) provided valuable sanity and implementation checking during the latter stages of the document's development. Finally, Jim Rubas (IBM) supplied the MARS pseudo-code in Appendix F and also provided detailed proof-reading in the latter stages of the document's development.

Author's Address

Grenville Armitage
Bellcore, 445 South Street
Morristown, NJ, 07960
USA

EMail: gja@thumper.bellcore.com
Phone: +1 201 829 2635

References

- [1] Deering, S., "Host Extensions for IP Multicasting", STD 3, RFC 1112, Stanford University, August 1989.
- [2] Heinanen, J., "Multiprotocol Encapsulation over ATM Adaption Layer 5", RFC 1483, Telecom Finland, July 1993.
- [3] Laubach, M., "Classical IP and ARP over ATM", RFC 1577, Hewlett-Packard Laboratories, December 1993.
- [4] ATM Forum, "ATM User Network Interface (UNI) Specification Version 3.1", ISBN 0-13-393828-X, Prentice Hall, Englewood Cliffs, NJ, June 1995.
- [5] Waitzman, D., Partridge, C., and S. Deering, "Distance Vector Multicast Routing Protocol", RFC 1075, November 1988.
- [6] Perez, M., Liaw, F., Grossman, D., Mankin, A., Hoffman, E., and A. Malis, "ATM Signaling Support for IP over ATM", RFC 1755, February 1995.
- [7] Borden, M., Crawley, E., Davie, B., and S. Batsell, "Integration of Real-time Services in an IP-ATM Network Architecture.", RFC 1821, August 1995.
- [8] ATM Forum, "ATM User-Network Interface Specification Version 3.0", Englewood Cliffs, NJ: Prentice Hall, September 1993.

Appendix A. Hole punching algorithms.

Implementations are entirely free to comply with the body of this memo in any way they see fit. This appendix is purely for clarification.

A MARS implementation might pre-construct a set of <min,max> pairs (P) that reflects the entire Class D space, excluding any addresses currently supported by multicast servers. The <min> field of the first pair MUST be 224.0.0.0, and the <max> field of the last pair must be 239.255.255.255. The first and last pair may be the same. This set is updated whenever a multicast server registers or deregisters.

When the MARS must perform 'hole punching' it might consider the following algorithm:

Assume the MARS_JOIN/LEAVE received by the MARS from the cluster member specified the block <Emin, Emax>.

Assume Pmin(N) and Pmax(N) are the <min> and <max> fields from the Nth pair in the MARS's current set P.

Assume set P has K pairs. Pmin(1) MUST equal 224.0.0.0, and Pmax(M) MUST equal 239.255.255.255. (If K == 1 then no hole punching is required).

Execute pseudo-code:

```
create copy of set P, call it set C.

index1 = 1;
while (Pmax(index1) <= Emin)
    index1++;

index2 = K;
while (Pmin(index2) >= Emax)
    index2--;

if (index1 > index2)
    Exit, as the hole-punched set is null.

if (Pmin(index1) < Emin)
    Cmin(index1) = Emin;

if (Pmax(index2) > Emax)
    Cmax(index2) = Emax;
```

Set C is the required 'hole punched' set of address blocks.

The resulting set C retains all the MARS's pre-constructed 'holes' covering the multicast servers, but will have been pruned to cover the section of the Class D space specified by the originating host's <Emin,Emax> values.

The host end should keep a table, H, of open VCs in ascending order of Class D address.

Assume $H(x).addr$ is the Class address associated with VC.x.
Assume $H(x).addr < H(x+1).addr$.

The pseudo code for updating VCs based on an incoming JOIN/LEAVE might be:

```
x = 1;
N = 1;

while (x < no.of VCs open)
{
    while (H(x).addr > max(N))
    {
        N++;
        if (N > no. of pairs in JOIN/LEAVE)
            return(0);
    }

    if ((H(x).addr <= max(N) &&
        (H(x).addr >= min(N))
        perform_VC_update();

    x++;
}
```

Appendix B. Minimising the impact of IGMP in IPv4 environments.

Implementing any part of this appendix is not required for conformance with this document. It is provided solely to document issues that have been identified.

The intent of section 5.1 is for cluster members to only have outgoing point to multipoint VCs when they are actually sending data to a particular multicast group. However, in most IPv4 environments the multicast routers attached to a cluster will periodically issue IGMP Queries to ascertain if particular groups have members. The current IGMP specification attempts to avoid having every group member respond by insisting that each group member wait a random period, and responding if no other member has responded before them. The IGMP reply is sent to the multicast address of the group being queried.

Unfortunately, as it stands the IGMP algorithm will be a nuisance for cluster members that are essentially passive receivers within a given multicast group. It is just as likely that a passive member, with no outgoing VC already established to the group, will decide to send an IGMP reply - causing a VC to be established where there was no need for one. This is not a fatal problem for small clusters, but will seriously impact on the ability of a cluster to scale.

The most obvious solution is for routers to use the MARS_GROUPLIST_REQUEST and MARS_GROUPLIST_REPLY messages, as described in section 8.5. This would remove the regular IGMP Queries, resulting in cluster members only sending an IGMP Report when they first join a group.

Alternative solutions do exist. One would be to modify the IGMP reply algorithm, for example:

If the group member has VC open to the group proceed as per RFC 1112 (picking a random reply delay between 0 and 10 seconds).

If the group member does not have VC already open to the group, pick random reply delay between 10 and 20 seconds instead, and then proceed as per RFC 1112.

If even one group member is sending to the group at the time the IGMP Query is issued then all the passive receivers will find the IGMP Reply has been transmitted before their delay expires, so no new VC is required. If all group members are passive at the time of the IGMP Query then a response will eventually arrive, but 10 seconds later than under conventional circumstances.

The preceding solution requires re-writing existing IGMP code, and implies the ability of the IGMP entity to ascertain the status of VCs on the underlying ATM interface. This is not likely to be available in the short term.

One short term solution is to provide something like the preceding functionality with a 'hack' at the IP/ATM driver level within cluster members. Arrange for the IP/ATM driver to snoop inside IP packets looking for IGMP traffic. If an IGMP packet is accepted for transmission, the IP/ATM driver can buffer it locally if there is no VC already active to that group. A 10 second timer is started, and if an IGMP Reply for that group is received from elsewhere on the cluster the timer is reset. If the timer expires, the IP/ATM driver then establishes a VC to the group as it would for a normal IP multicast packet.

Some network implementors may find it advantageous to configure a multicast server to support the group 224.0.0.1, rather than rely on a mesh. Given that IP multicast routers regularly send IGMP queries to this address, a mesh will mean that each router will permanently consume an AAL context within each cluster member. In clusters served by multiple routers the VC load within switches in the underlying ATM network will become a scaling problem.

Finally, if a multicast server is used to support 224.0.0.1, another ATM driver level hack becomes a possible solution to IGMP Reply traffic. The ATM driver may choose to grab all outgoing IGMP packets and send them out on the VC established for sending to 224.0.0.1, regardless of the Class D address the IGMP message was actually for. Given that all hosts and routers must be members of 224.0.0.1, the intended recipients will still receive the IGMP Replies. The negative impact is that all cluster members will receive the IGMP Replies.

Appendix C. Further comments on 'Clusters'.

The cluster concept was introduced in section 1 for two reasons. The more well known term of Logical IP Subnet is both very IP specific, and constrained to unicast routing boundaries. As the architecture described in this document may be re-used in non-IP environments a more neutral term was needed. As the needs of multicasting are not always bound by the same scopes as unicasting, it was not immediately obvious that apriori limiting ourselves to LISSs was beneficial in the long term.

It must be stressed that Clusters are purely an administrative being. You choose their size (i.e. the number of endpoints that register with the same MARS) based on your multicasting needs, and the resource consumption you are willing to put up with. The larger the number of ATM attached hosts you require multicast support for, the more individual clusters you might choose to establish (along with multicast routers to provide inter-cluster traffic paths).

Given that not all the hosts in any given LIS may require multicast support, it becomes conceivable that you might assign a single MARS to support hosts from across multiple LISSs. In effect you have a cluster covering multiple LISSs, and have achieved 'cut through' routing for multicast traffic. Under these circumstances increasing the geographical size of a cluster might be considered a good thing.

However, practical considerations limit the size of clusters. Having a cluster span multiple LISSs may not always be a particular 'win' situation. As the number of multicast capable hosts in your LISSs increases it becomes more likely that you'll want to constrain a cluster's size and force multicast traffic to aggregate at multicast routers scattered across your ATM cloud.

Finally, multi-LIS clusters require a degree of care when deploying IP multicast routers. Under the Classical IP model you need unicast routers on the edges of LISSs. Under the MARS architecture you only need multicast routers at the edges of clusters. If your cluster spans multiple LISSs, then the multicast routers will perceive themselves to have a single interface that is simultaneously attached to multiple unicast subnets. Whether this situation will work depends on the inter-domain multicast routing protocols you use, and your multicast router's ability to understand the new relationship between unicast and multicast topologies.

In the absence of further research in this area, networks deployed in conformance to this document MUST make their IP cluster and IP LIS coincide, so as to avoid these complications.

Appendix D. TLV list parsing algorithm.

The following pseudo-code represents how the TLV list format described in section 10 could be handled by a MARS or MARS client.

```
list = (mar$extoff & 0xFFFFC);

if (list == 0) exit;

list = list + message_base;

while (list->Type.y != 0)
{
    switch (list->Type.y)
    {
        default:
        {
            if (list->Type.x == 0) break;

            if (list->Type.x == 1) exit;

            if (list->Type.x == 2) log-error-and-exit;
        }

        [...other handling goes here...]
    }

    list += (list->Length + 4 + ((4-(list->Length & 3)) % 4));
}

return;
```


Appendix E. Summary of timer values.

This appendix summarises various timers or limits mentioned in the main body of the document. Values are specified in the following format: [x, y, z] indicating a minimum value of x, a recommended value of y, and a maximum value of z. A '-' will indicate that a category has no value specified. Values in minutes are followed by 'min', values in seconds are followed by 'sec'.

Idle time for MARS - MARS client pt to pt VC:
[1 min, 20 min, -]

Idle time for multipoint VCs from client.
[1 min, 20 min, -]

Allowed time between MARS_MULTI components.
[-, -, 10 sec]

Initial random L_MULTI_RQ/ADD retransmit timer range.
[5 sec, -, 10 sec]

Random time to set VC_revalidate flag.
[1 sec, -, 10 sec]

MARS_JOIN/LEAVE retransmit interval.
[5 sec, 10 sec, -]

MARS_JOIN/LEAVE retransmit limit.
[-, -, 5]

Random time to re-register with MARS.
[1 sec, -, 10 sec]

Force wait if MARS re-registration is looping.
[1 min, -, -]

Transmission interval for MARS_REDIRECT_MAP.
[1 min, 1 min, 2 min]

Limit for client to miss MARS_REDIRECT_MAPs.
[-, -, 4 min]

Appendix F. Pseudo code for MARS operation.

Implementations are entirely free to comply with the body of this memo in any way they see fit. This appendix is purely for possible clarification.

A MARS implementation might be built along the lines suggested in this pseudo-code.

1. Main

1.1 Initilization

```

Define a server list as the list of leaf nodes
                                on ServerControlVC.
Define a cluster list as the list of leaf nodes
                                on ClusterControlVC.
Define a host map as the list of hosts that are
                                members of a group.
Define a server map as the list of hosts (MCSs)
                                that are serving a group.

Read config file.
Allocate message queues.
Allocate internal tables.
Set up passive open VC connection.
Set up redirect_map timer.
Establish logging.

```

1.2 Message Processing

```

Forever {
  If the message has a TLV then {
    If TLV is unsupported then {
      process as defined in TLV type field.
    } /* unknown TLV */
  } /* TLV present */
  Place incoming message in the queue.
  For (all messages in the queue) {
    If the message is not a JOIN/LEAVE/MSERV/UNSERV with
      mar$flags.register == 1 then {
      If the message source is (not a member of server list) &&
        (not a member of cluster list) then {
        Drop the message silently.
      }
    }
  }
  If (mar$pro.type is not supported) or
    (the ATM source address is missing) then {
    Continue.
  }
}

```

```

    }
    Determine type of message.
    If an ERR_L_RELEASE arrives on ClusterControlVC then {
        Remove the endpoints ATM address from all groups
        for which it has joined.
        Release the CMI.
        Continue.
    } /* error on CCVC */
    Call specific message handling routine.
    If redirect_map timer pops {
        Call MARS_REDIRECT_MAP message handling routine.
    } /* redirect timer pop */
    } /* all msgs in the queue */
} /* forever loop */

```

2. Message Handler

2.1 Messages:

- MARS_REQUEST

```

Indicate no MARS_MULTI support of TLV.
If the supported TLV is not NULL then {
    Indicate MARS_MULTI support of TLV.
    Process as required.
} else { /* TLV NULL */
    Indicate message to be sent on Private VC.
    If the message source is a member of server list then {
        If the group has a non-null host map then {
            Call MARS_MULTI with the host map for the group.
        } else { /* no group */
            Call MARS_NAK message routine.
        } /* no group */
    } else { /* source is cluster list */
        If the group has a non-null server map then {
            Call MARS_MULTI with the server map for the group.
        } else { /* cluster member but no server map */
            If the group has a non-null host map then {
                Call MARS_MULTI with the host map for the group.
            } else { /* no group */
                Call MARS_NAK message routine.
            } /* no group */
        } /* cluster member but no server map */
    } /* source is a cluster list */
} /* TLV NULL */
If a message exists then {
    Send message as indicated.
}

```



```

If a message exists then {
    mar$flags.copy = 1.
    Send message as indicated.
}
Return.

```

- MARS_LEAVE

```

If (mar$flags.copy != 0) silently ignore the message.
If more than a single <min,max> pair is specified then
silently ignore the message.
Indicate message to be sent on ClusterControlVC.
If (mar$flags.register == 1) then { /* deregistration */
    Update internal tables to remove the member's ATM addr
    from all groups it has joined.
    Drop the endpoint from ClusterControlVC.
    Drop the endpoint from cluster list.
    Release the CMI.
    Indicate message to be sent on Private VC.
} else { /* not a deregistration */
    If the group is a duplicate of a previous MARS_LEAVE then {
        mar$msn = current csn.
        Indicate message to be sent on Private VC.
    } else {
        Indicate no message to be sent.
        If the first <min,max> encompasses any group with
                                a server map then {
            Call the Modified JOIN/LEAVE Processing routine.
        } else {
            If the MARS_LEAVE is for a multi group then {
                Call the MultiGroup JOIN/LEAVE Processing Routine.
            } else {
                Indicate message to be sent on ClusterControlVC.
            }
        }
    }
    Update internal tables.
} /* not a duplicate */
} /* not a deregistration */
If a message exists then {
    mar$flags.copy = 1.
    Send message as indicated.
}
Return.

```

- MARS_MSERV

```

If (mar$flags.register == 1) then { /* server register */
    Add the endpoint as a leaf node to ServerControlVC.
}

```

```

    Add the endpoint to the server list.
    Indicate the message to be sent on Private VC.
    mar$cmi = 0.
  } else { /* not a register */
    If the source has not registered then {
      Drop and ignore the message.
      Indicate no message to be sent.
    } else { /* source is registered */
      If MCS is already member of indicated server map {
        Indicate message to be sent on Private VC.
        mar$flags.layer3grp = 0;
        mar$flags.copy = 1.
      } else { /* New MCS to add. */
        Add the server ATM addr to server map for group.
        Indicate message to be sent on ServerControlVC.
        Send message as indicated.
        Make a copy of the message.
        Indicate message to be sent on ClusterControlVC.
        If new server map was just created {
          Construct MARS_MIGRATE, with MCS as target.
        } else {
          Change the op code to MARS_JOIN.
          mar$flags.layer3grp = 0.
          mar$flags.copy = 1.
        } /* new server map */
      } /* New MCS to add. */
    } /* source is registered */
  } /* not a register */

  If a message exists then {
    Send message as indicated.
  }
  Return.

```

- MARS_UNSERV

```

If (mar$flags.register == 1) then { /* deregister */
  Remove the ATM addr of the MCS from all server maps.
  If a server map becomes null then delete it.
  Remove the endpoint as a leaf of ServerControlVC.
  Remove the endpoint from server list.
  Indicate the message to be sent on Private VC.
} else { /* not a deregister */
  If the source is not a member of server list then {
    Drop and ignore the message.
    Indicate no message to be sent.
  } else { /* source is registered */

```

```

    If MCS is not member of indicated server map {
        Indicate message to be sent on Private VC.
        mar$flags.layer3grp = 0;
        mar$flags.copy = 1.
    } else { /* MCS existed, must be removed. */
        Remove ATM addr of the MCS from indicated server map.
        If a server map is null then delete it.
        Indicate the message to be sent on ServerControlVC.
        Send message as indicated.
        Make a copy of the message.
        Change the op code to MARS_LEAVE.
        Indicate message (copy) to be sent on ClusterControlVC.
        mar$flags.layer3grp = 0;
        mar$flags.copy = 1.
    } /* MCS existed, must be removed. */
} /* source is registered */
} /* not a deregister */
If a message exists then {
    Send message as indicated.
}
Return.

- MARS_NAK

    Build command.
    Return.

- MARS_GROUPLIST_REQUEST

    If (mar$pnum != 1) then Return.
    Call MARS_GROUPLIST_REPLY with the range and output VC.
    Return.

- MARS_GROUPLIST_REPLY

    Build command for specified range.
    Indicate message to be sent on specified VC.
    Send message as indicated.
    Return.

- MARS_REDIRECT_MAP

    Include the MARSs own address in the message.
    If there are backup MARSs then include their addresses.
    Indicate MARS_REDIRECT_MAP is to be sent on ClusterControlVC.
    Send message back as indicated.
    Return.

```

3. Send Message Handler

```
If (the message is going out ClusterControlVC) &&
    (a new csn is required) then {
    mar$msn = obtain a CSN
}
If (the message is going out ServerControlVC) &&
    (a new ssu is required) then {
    mar$msn = obtain a SSN
}
Return.
```

4. Number Generator

4.1 Cluster Sequence Number

```
Generate the next sequence number.
Return.
```

4.2 Server Sequence Number

```
Generate the next sequence number.
Return.
```

4.3 CMI

CMIs are allocated uniquely per registered cluster member within the context of a particular layer 3 protocol type. A single node may register multiple times if it supports multiple layer 3 protocols. The CMIs allocated for each such registration may or may not be the same. Generate a CMI for this protocol. Return.

5. Modified JOIN/LEAVE Processing

This routine processes JOIN/LEAVE when a server map exists.

```
Make a copy of the message.
Change the type of the copy to MARS_SJOIN.
If the message is a MARS_LEAVE then {
    Change the type of the copy to MARS_SLEAVE.
}
mar$flags.copy = 1 (copy).
Hole punch the <min,max> group by excluding
    from the range those groups which the joining
    (leaving) node is already (still) a member of
```



```

    due to it having previously issued a single group
    join.
    Indicate the message to be sent on ServerControlVC.
    If the message (copy) contains one or more <min,max> pair {
        Send message (copy) as indicated.
    }
    mar$flags.punched = 0 in the original message.
    Indicate the message to be sent on Private VC.
    Send message (original) as indicated.
    Hole punch the <min,max> group by excluding
        from the range those groups that are served by MCSs
        or which the joining (leaving) node is already
        (still) a member of due to it having previously
        issued a single group join.
    Indicate the (original) message to be sent on ClusterControlVC.
    If (number of holes punched > 0) then { /* punched holes */
        In original message do {
            mar$flags.punched = 1.
            old punched list <- new punched list.
        }
    } /* punched holes */
    mar$flags.copy = 1.
    Send message as indicated.
    Return.

```

5.1 MultiGroup JOIN/LEAVE Processing

This routine processes JOIN/LEAVE when a multi group exists.

```

    If (mar$flags.layer3grp) {
        Ignore this setting, consider it reset.
    }
    mar$flags.copy = 1.
    Make a copy of the message.
    From the copy hole punch the <min,max> group by
        excluding from the range those groups that this
        node has already joined or left.
    If (number of holes punched > 0) then {
        mar$flags.punch = 0 in original message.
        Indicate original message to be sent on Private VC.
        Send original message as indicated.
        mar$flags.punch = 1 in copy message.
        old group range <- new punched list.
        Indicate message to be sent on ClusterControlVC.
        Send copy of message as indicated.
    } else {
        Indicate message to be sent on ClusterControlVC.
        Send original message as indicated.
    }

```

```
} /* no holes punched */  
Return.
```

