2 REFERENCES

2.1 Normative References

[DEPI] Downstream External PHY Interface, CM-SP-DEPI-I08-100611, June 11, 2010, Cable Television Laboratories, Inc.

[J.83] ITU-T Recommendations, J.83, (04/97) Digital multi-programme systems for television sound and data services for cable distribution.

[RFC 1123] IETF RFC 1123/STD 3, Braden, R., "Requirements for Internet Hosts – Application and Support" October 1989, Internet Engineering Task Force.

[RFC 2068] IETF RFC 2068, R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1", January 1997, Internet Engineering Task Force.

[RFC 2326] IETF RFC 2326, H. Schulzrinne; et al, Real Time Streaming Protocol (RTSP), April 1998, Internet Engineering Task Force.

[RFC 4291] IETF RFC 4291, Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", February 2006, Internet Engineering Task Force.

[RFC 3219] IETF RFC 3219, J. Rosenberg, H. Salama and M. Squire, "Telephony Routing over IP (TRIP)," January 2002, Internet Engineering Task Force.

[VSI] Edge QAM Video Stream Interface Specification, CM-SP-EQAM-VSI-I01-081107, November 07, 2008, Cable Television Laboratories, Inc.

2.2 Informative References

[DSG] DOCSIS Set-top Gateway (DSG) Interface Specification, CM-SP-DSG-I18-110623, June 23, 2011, Cable Television Laboratories, Inc.

[RFC 2131] IETF RFC 2131, R. Droms, "Dynamic Host Configuration Protocol", March 1997, Internet Engineering Task Force.

[RFI2] DOCSIS 2.0 Radio Frequency Interface Specification, CM-SP-RFIv2.0-C02-090422, November 22, 2009, Cable Television Laboratories, Inc.

2.3 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone +1-303-661-9100; Fax +1-303-661-9199; Internet: http://www.cablelabs.com
- Internet Engineering Task Force (IETF), http://www.ietf.org
- International Telecommunication Union – Telecommunication Standardization Sector (ITU-T), http://www.itu.int/itu-t/

5.2.1.1 Registering QAM Channels

Each ERM is responsible for managing the resources of one or more EQAMs. In order for an ERM to manage an EQAM's resources, it must first obtain an inventory of the resources associated with that EQAM. It must also obtain IP and/or RF addressing information associated with the EQAM and those resources in order to determine how to communicate with them. For example, an EQAM contains one or more QAM channels, each of which has associated RF properties (for example, the carrier frequency) and an RF address (the MPEG-2 Transport Stream ID (TSID)).

In order to allocate a particular QAM channel to a DOCSIS MAC domain, an ERM must know the following properties associated with that QAM channel:

- TSID
- QAM configuration, Capability, and QAM grouping
- Fiber nodes
- Total available bandwidth

EQAMs use the Registration Interface to advertise available resources to an ERM. The ERM may use this information to populate a database of available resources.

5.2.2 Overall Architecture

The Registration Interface allows an EQAM to advertise to an ERM, information about the location and properties of resources under its control. The ERM may use this information to populate a database of the resources that have been reported to it by multiple EQAMs. The ERM may use this data to formulate responses to incoming requests for resources.

The Figure 5-3 shows the Registration component architecture. The Registration Interface between an ERM and an EQAM carries messages conforming to the Edge Resource Registration Protocol (ERRP), as specified in Section 6.2.

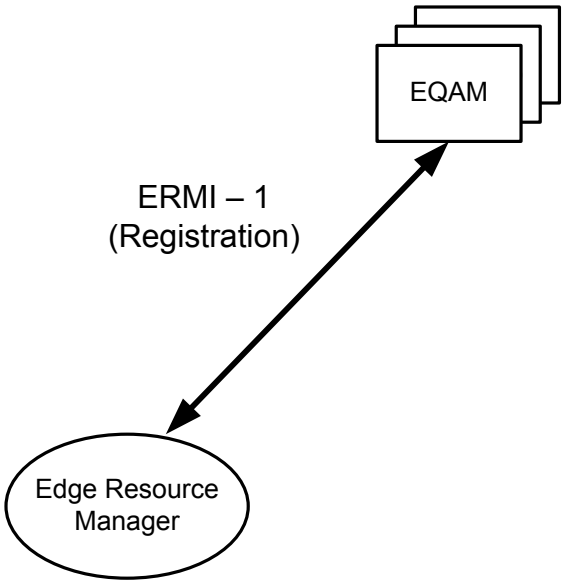


Figure 5-3 - Registration Interface and Components

An ERM normally manages multiple EQAMs, as shown in the Figure 5-3. An EQAM requires an ERM IP address in order to contact the ERM. An EQAM may use a DNS query to obtain an ERM IP address. An EQAM may also

The M-CMTS Core or Video Session Manager initiates a QAM resource transaction with the ERM when it requests or releases a QAM channel resource. When an M-CMTS requests a MAC domain to be created, for example, the M-CMTS Core provides the ERM with details of the desired service group, bandwidth, and QAM channel capability. The ERM then consults its database of available resources, verifies that the resources are available, and returns the contact information for an appropriate QAM channel, if one is available.

An EQAM is a device which has a pool of QAM channels that may be allocated to DOCSIS, video, or both. A particular QAM channel may support only a subset of all possible DOCSIS capabilities. For example, some QAM channels may support only certain interleave settings; or some QAM channels may not support the DOCSIS PSP mode specified in [DEPI]. The capabilities of each individual QAM channel are advertised to the ERM when the EQAM advertises that particular QAM channel.

The ERM may apply operator-dependent policies when selecting a QAM channel. Such policies may take into consideration factors such as: QAM channel load balancing; whether DOCSIS bonded traffic may share a QAM channel with video (i.e., VOD, SDV etc.) traffic; and the existence of QAM channels that have been reserved for future DOCSIS traffic, etc.

In an EQAM, QAM channels are physically present on external RF ports. A single RF port may be associated with multiple QAM channels. The RF port may impose limitations on the configuration of associated QAM channels. For example, all the QAM channels associated with a single RF port may be required to be configured identically. Although a QAM channel used by a video flow and one used by DOCSIS may have the same carrier frequency and modulation type, they may have different interleave settings. To allow an individual QAM channel to switch between operating in a mode compatible with video flows and one compatible with DOCSIS, the EQAM should be able to control the configuration of a single QAM channel without affecting the configuration of any other QAM channels.

This document specifies two resource-allocation interfaces. ERMI-2 is an interface between an ERM and an EQAM, and is used to allocate QAM channel resources selected by the ERM. ERMI-3 is an interface between an M-CMTS Core and an ERM, and is used to request and return QAM channel resources. The interface between the ERM and Video Session Manager is beyond the scope of this document. When the CMTS core allocates QAM channel resources from the ERM, either explicit list of QAMs or fiber node information can be given to the ERM.

5.3.2 Signaling Protocol

The protocol used for the resource allocation interfaces (ERMI-2 and ERMI-3) is the Real Time Streaming Protocol, RTSP [RFC 2326]. RTSP is an application-level client/server protocol designed to control the delivery of real-time data. RTSP provides an extensible framework to enable controlled, on-demand delivery of such data. The protocol is intended to control multiple simultaneous data delivery sessions, to provide a means for choosing delivery channels, and to provide a way to choose among multiple delivery mechanisms.

The client initiates an RTSP session by sending a SETUP message containing a request for resources. The server allocates the resources for the session and replies by identifying a suitable resource that meets the client's request. In ERMI-2, the ERM is an RTSP client and the EQAM is an RTSP server. In ERMI-3, the M-CMTS Core is an RTSP client and the ERM is an RTSP server.

An RTSP message is either a request or a response. A request contains an RTSP method, the object on which the method is operating, and any parameters necessary to further define the operation. Depending on the RTSP method, the direction of an RTSP request can be from client to server or vice-versa.

In video applications, RTSP is extended with new methods and headers to support the video application. In the DOCSIS and video applications specified in this document, additional headers are defined. Naturally, in order for an ERM/EQAM pair to function correctly in both DOCSIS and video applications, all devices must support the (different) RTSP extensions used by the two applications.

RTSP allows considerable variations in the capabilities supported by conformant implementations. This specification indicates the subset of RTSP requirements that must be supported by the RTSP components in the ERMI resource allocation interfaces (ERMI-2 and ERMI-3). In addition, extensions needed for the DOCSIS application are also specified herein.

5.3.3 Selecting an ERM

There may be multiple ERMs in the operator's head-end network. The M-CMTS Core or Video Session Manager selects the correct ERM with which to communicate by means that are outside the scope of this document. For the simplest case, this can be achieved by static configuration. In a more dynamic network, ERMs may report resources to an aggregation point by means that are outside the scope of this document. Such an aggregation point may serve as a proxy to locate the correct ERM.

5.4 Static Partitioning

In current head-end networks, the QAM channel resources used by video applications and those used by DOCSIS applications are separate and distinct. In order to allow resources to be switched easily between these two types of application, common interfaces should be used within the ERM for registration, allocation, and control. In order to provide a relatively simple migration path, QAM channel resources may initially be partitioned by using static configuration.

Some EQAMs may control QAM channels that are capable of supporting both video and DOCSIS applications. It is permissible to provision statically, some of the QAM channels in the EQAM for video service and the rest for DOCSIS data applications.

5.4.1 Simplified Architecture for Static QAM Resource Sharing

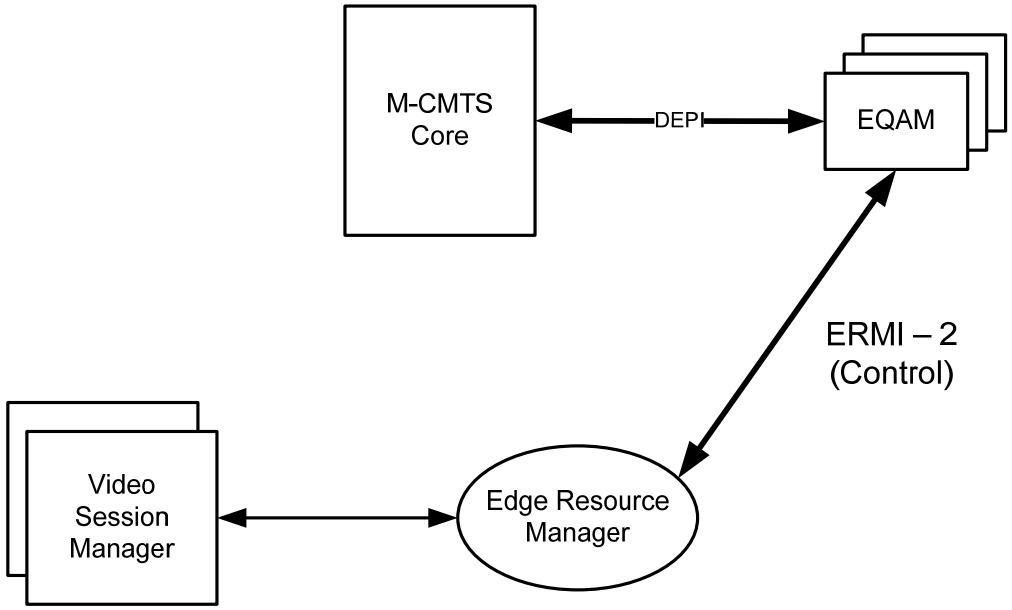


Figure 5-5 - Simplified Framework for Static QAM Partitioning

Figure 5-5 shows an architecture suitable for partitioning QAM channels statically between video and DOCSIS applications. In this simple architecture, the ERM is not used at all by DOCSIS applications.

The EQAM is configured so that only video QAM channels are advertised to the ERM (since the ERM is used only by video applications).

5.4.2 Operation

When the M-CMTS Core needs a QAM channel resource, such as in the case of creating a new MAC domain, it selects a QAM channel from the appropriate service group. The M-CMTS Core obtains the IP address of the EQAM from its configuration database. The M-CMTS Core then uses the DEPI protocol to establish a control channel to this IP address. Once this channel is in place, the M-CMTS Core negotiates the physical settings for the QAM channel, as specified in [DEPI].

5.5 Device Configuration

The ERMI components, EQAM and M-CMTS Core, must be properly configured for ERMI to function as intended. The configuration for dynamic signaling and static portioning is slightly different. The differences are noted below.

Each QAM channel in the EQAM is configured with:

- QAM TSID and QAM name;
- Input port IP addresses;
- Fiber Node list;
- Modulation type;
- Channel bandwidth;
- Interleaver setting (I, J);
- J83 annex;
- Carrier frequency;
- Power;
- QAM capability;
- Resource allocation mode (ERM, or non-ERM);
- ERM IP address (only needed for dynamic signaling).

In the M-CMTS Core, configuration depends on whether static partitioning or the dynamic signaling is used for QAM sharing. When static partitioning is used, QAM TSID, QAM IP address and QAM RF topology are configured. When dynamic signaling is used, ERM IP is configured. In addition, the M-CMTS Core may choose to either directly configure QAM RF topology or just configure the fiber node for RF topology.

6 Edge Resource Registration Protocol (ERRP)

Several requirements in this section are written for an "ERRP Speaker". The EQAM MUST implement the ERRP Speaker functionality. Some requirements are written for an "ERRP Listener". The ERM MUST implement the ERRP Listener functionality. Many of the requirements in this section apply to both ERRP Speakers and ERRP Listeners, and are written for an "ERRP Node". The EQAM MUST implement the ERRP Node functionality. The ERM MUST implement the ERRP Node functionality.

6.1 Relationship with TRIP [RFC 3219]

ERRP shares many similarities with the protocol described in TRIP, [RFC 3219]. While TRIP was originally developed for telephony services, a subset of the protocol can be used to deal with the resource manager selection problem addressed in this specification, specifically, TRIP has similar procedures and a similar Finite State Machine for connection establishment. TRIP also shares the same format for messages. ERRP Nodes are also conformant TRIP nodes, although they perform only a subset of the messaging described in [RFC 3219].

When TRIP is adapted to ERRP in a cable environment, data plane devices will be TRIP speakers, and resource managers will act as TRIP listeners.

ERRP supports four messages that may be exchanged between ERRP Nodes: OPEN, UPDATE, NOTIFICATION, and KEEPALIVE:

- The OPEN message is used to initiate a ERRP connection between ERRP Nodes. The OPEN message is used exactly as specified in [RFC 3219].
- The UPDATE message is used by an EQAM to advertise resources under its control to an ERM.
- The NOTIFICATION message is used by an ERRP Node to inform the far end that an error has occurred. ERRP connections are closed after a NOTIFICATION message is sent or received. The NOTIFICATION message is used exactly as specified in [RFC 3219].
- ERRP includes a periodic bidirectional KEEPALIVE message whose frequency is negotiated by the two sides when the ERRP connection is established. The KEEPALIVE message is used as specified in [RFC 3219].

If the ERM does not receive a KEEPALIVE message within the agreed-upon period, it assumes that the EQAM has failed and updates its database accordingly, by marking the associated resources as no longer available. The ERM MAY try to re-establish the ERRP connection to that EQAM before removing that EQAM from its resource pool. The ERM can also discover a change in a QAM channel resource through receipt of an UPDATE message. Failures of QAM resources and indications of the availability of new QAM resources are conveyed to an ERM through the WithdrawnRoutes and ReachableRoutes attributes of the UPDATE message.

6.2 ERRP

This section documents the subset of [RFC 3219] used in this architecture, as well as the additional attributes beyond [RFC 3219] that are used to advertise QAM channel resources. Instead of referring to the applicable sections of [RFC 3219], this section includes normatively the relevant sections of [RFC 3219]. The resultant protocol is known as ERRP, the Edge Resource Registration Protocol.

6.2.1 Establishing a ERRP Connection

ERRP Nodes MUST use TCP/IP connections to carry ERRP messages. An ERRP Node MUST listen on TCP port 6069 for incoming connections that will be used to carry ERRP traffic.

The ERRP connection MAY be initiated by either of the ERRP Nodes (speaker or listener).

An ERRP Node MUST conform to the ERRP state machine specified in Section 6.2.7. An ERRP Node begins in the [Idle] state and goes through several state transitions before reaching the [Established] state. At that point the EQAM SHOULD advertise its operational resources to the ERM.

Typically, the EQAM (acting as a speaker) will be configured to establish the connection with ERMs. The ERMs, as listeners, will have no prior knowledge of the EQAM and therefore cannot establish the transport connection. KEEPALIVE messages are sent periodically to ensure adjacent peer ERRP Nodes are operational. Notification messages are sent in response to errors or special conditions. If a connection encounters an error condition, a Notification message is sent and the connection is closed.

6.2.2 Message Formats

ERRP Nodes MUST comply with the syntax and format of messages as outlined in the subsections below. Incomplete received ERRP messages MUST be ignored. An ERRP Node MUST not process a message until it is entirely received. An ERRP Node MUST NOT transmit ERRP messages that exceed 4096 octets. The recipient ERRP Node MUST be able to process ERRP messages up to 4096 octets in size. ERRP Nodes must transmit messages in "network order" so octets are transmitted most significant octet first and, within an octet, most significant bit first.

The smallest message that may be sent consists of an ERRP header without a data portion or 3 octets.

6.2.2.1 Message Header

Every ERRP message begins with a 3-octet header, specified below. There may or may not be a data portion following the header, depending on the message type. The layout of the header fields is shown in Figure 6–1:

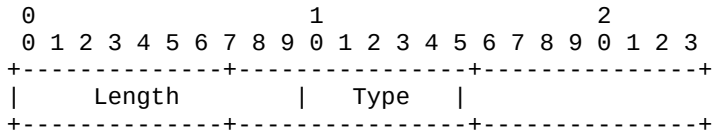


Figure 6–1 - DDRP Header Format

Length

This 2-octet unsigned integer indicates the total length of the message, including the header, in octets. Thus, it allows the recipient to locate the beginning of the next message in the message stream. The Length field is mandatory. The value of the Length field is in the range $3 \leq \text{Length} \leq 4096$. The value may be further constrained by the message type. The stream of ERRP messages does not contain padding between messages.

Type

This 1-octet unsigned integer encodes the type of the message. The Type field is mandatory. The value of the field is one of the values identified in Table 6–1:

Table 6–1 - ERRP Message Types

Type	ERRP message
1	OPEN
2	UPDATE
3	NOTIFICATION
4	KEEPALIVE

6.2.2.2 OPEN Message

The first ERRP message sent by an ERRP Node MUST be either an OPEN message or a NOTIFICATION message sent in response to a received OPEN message.

If the OPEN message is acceptable to the recipient ERRP Node, a KEEPALIVE message confirming the OPEN MUST be returned.

The minimum length of the OPEN message is 17 octets (including the message header). OPEN messages not meeting this minimum requirement are handled as defined in Section 6.2.4.2.

Following the fixed-size ERRP header, the OPEN message contains the following fields:

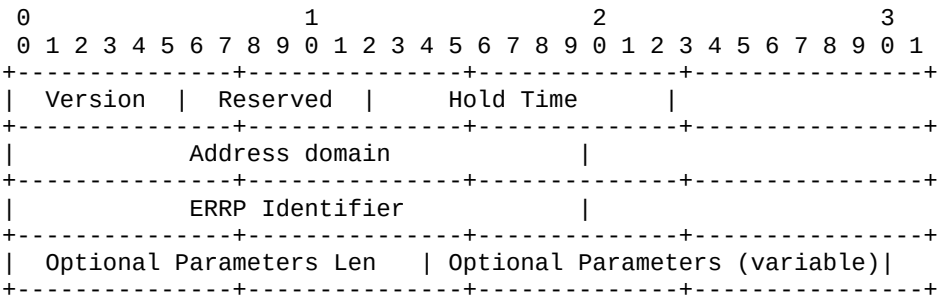


Figure 6–2 - ERRP OPEN Header

Version

This 1-octet unsigned integer indicates the protocol version of the message. The Version field is mandatory. The Version field contains the value 3.

Hold Time

This 2-octet unsigned integer indicates the number of seconds that the sender proposes for the value of the Hold Timer. The Hold Time field is mandatory. Upon receipt of an OPEN message, an ERRP Node MUST calculate the value of the Hold Timer, T_H, by using the smaller of its configured Hold Time (if any), and the value of the Hold Time field in the OPEN message. The value of the Hold Time field in an OPEN message is either 0 or greater than two; see Section 6.2.4.2 for error conditions. The configured Hold Time (if any) is either 0 or at least three seconds. A recipient SHOULD reject connections if the value of the Hold Time field does not meet local policy. The calculated value of T_H is the maximum number of seconds that may elapse between the receipt of successive KEEPALIVE and/or UPDATE messages.

Address domain

This 4-octet unsigned integer indicates the address domain number of the sender. The Address domain field is mandatory. Two ERRP Nodes MUST have the same address domain in order to establish an ERRP connection, unless one has the reserved address domain number of zero.

The value of the Address domain field is less than or equal to 255.

An Address domain field that contains the value 0 is interpreted by the recipient ERRP Node to mean that the advertised address can be reached from any address domain.

ERRP Identifier

This 4-octet unsigned integer indicates the ERRP Identifier of the sender. The ERRP Identifier field is mandatory. The value of this field uniquely identifies this ERRP Node within its address domain. An ERRP Node MAY set the value of its ERRP Identifier to an IPv4 address assigned to that ERRP Node. The value of the ERRP Identifier of an

ERRP Node is configured on the ERRP Node. The ERRP Node MUST use the same value for the ERRP Identifier field in all OPEN messages sent to other ERRP Nodes.

When comparing two ERRP identifiers, the ERRP Identifier is to be interpreted as a numerical 4-octet unsigned integer. It is recommended that the ERRP identifier is set to its IPv4 address as the default.

Optional Parameters Len

This 2-octet unsigned integer indicates the total length of the Optional Parameters field in octets. The Optional Parameters Len field is mandatory. If the value of this field is zero, the Optional Parameters field is absent. Note, this is not the same integer as 'Parameter Length' defined below.

Parameters

This field contains a list of parameters where each is encoded as a <Parameter Type, Parameter Length, Parameter Value> triplet as described in Figure 6–3. There are two mandatory parameters, Component Name and Streaming Zone, the rest are optional.

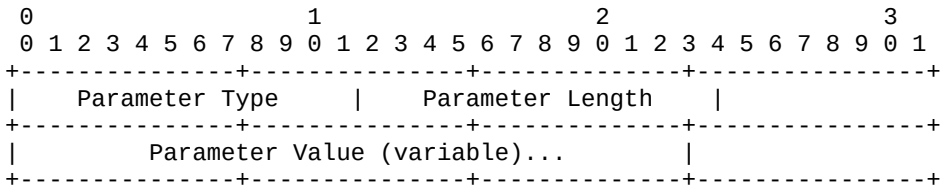


Figure 6–3 - Optional Parameter Encoding

Parameter Type

This is a 2-octet field that identifies the type of this parameter. The Parameter Type field is mandatory.

Parameter Length

This 2-octet unsigned integer contains the length of the Parameter Value field in octets. The Parameter Length field is mandatory.

Parameter Value

This is a variable length field that is interpreted according to the value of the Parameter Type field. The Parameter Value field is optional; its presence depends on the parameter being passed.

6.2.2.2.1 Open Message Parameters

6.2.2.2.1.1 Capability Information

If present, the Capability Information parameter identifies its presence by setting the value of the Parameter Type field to 1.

The Capability Information parameter is optional and is used by an ERRP Node to indicate to its peer ERRP Node the capabilities that it supports. Capability negotiation is specified in Section 6.2.6.

A valid Capability Information parameter contains one or more triplets <Capability Code, Capability Length, Capability Value>, where each triplet is encoded as shown in Figure 6–4:

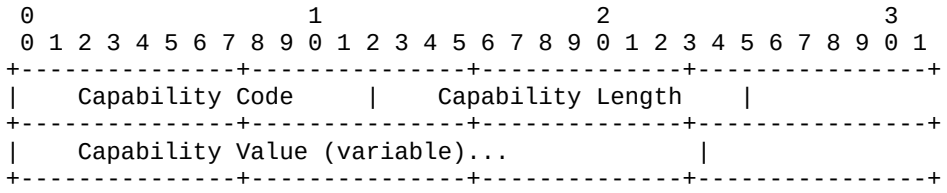


Figure 6–4 - Capability Optional Parameter

Capability Code

This is a 2-octet field, used to identify individual capabilities. The Capability Code field is mandatory.

Capability Length

This 2-octet unsigned integer contains the length of the Capability Value field in octets. The Capability Length field is mandatory.

Capability Value

This is a variable-length field that is interpreted according to the value of the Capability Code field. A single capability, as identified by the value of its Capability Code field, may appear more than once in the Optional Parameters field.

The value of the Capability Code field is one of the values identified in Table 6–2.

Table 6–2 - Capability Codes

Capability Code	Capability
1	Route Types Supported
2	Send Receive
32768	ERRP Version

This specification reserves Capability Codes 32769-65535 for vendor-specific applications (these are the codes with the first bit of the code value equal to 1). This specification reserves value 0. Capability Codes (other than those reserved for vendor specific use) are controlled by IANA.

6.2.2.2.1.1.1 Route Types Supported

The Route Types Supported Capability Code lists the route types supported in this peering session by the transmitting ERRP Node. An ERRP Node speaker MUST NOT use route types that are not supported by its ERRP Node peer in any particular peering session. If the route types supported by an ERRP Node peer are not satisfactory, an ERRP Node speaker SHOULD terminate the peering session.

The format for a Route Type is shown in Figure 6–5:

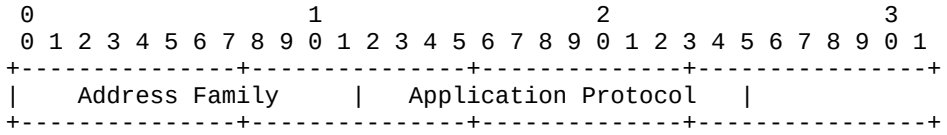


Figure 6–5 - Route Type Format

Required Flags: Well-known

ERRP Type Code: 250

The Encryptor Capability attribute is used to advertise an encryption triplet to the ERM that fully identifies one content encryption algorithm supported by the encryptor on a given QAM output. The encryptor may support more than one type of encryption algorithm at the same time. The embedded encryptor within an EQAM that supports multiple types of encryption algorithms will use multiple UPDATE messages to advertise these capabilities. Each UPDATE message will carry one Encryptor Capability attribute for a single type of encryption algorithm.

The Encryptor Capability attribute consists of multiple parameters, including CA Encryption Type, Encryption Scheme, Key Length and one or more CAS Identifiers. The CA Encryption Type indicates the conditional access technology used by the encryptor. The Encryption Scheme defines the encryption algorithm and the fixed key length associated with the algorithm or a default key length to be used with the algorithm. The key length field contains the length in bits of the encryption key and is used to override the default key length.

The Num CAS Identifier parameter indicates the number of CAS Identifiers that will follow. In the case of Simulcrypt or when CA-Encryption-Type specifies Simulcrypt, Num-CAS-Identifier may have a value greater than one; else its value is limited to one. The CAS Identifier 1 to CAS Identifier n specify the Conditional Access system identifiers associated with the ECMGs employed by the embedded encryptor.

The Key Length parameter, when non-zero, specifies the key length value (in bits) of the encryption key for the reported Encryption Scheme, overriding the default key length. A value of zero (0) is used to indicate the default key length as specified by the reported Encryption Scheme. Key Length is a 16-bit unsigned integer.

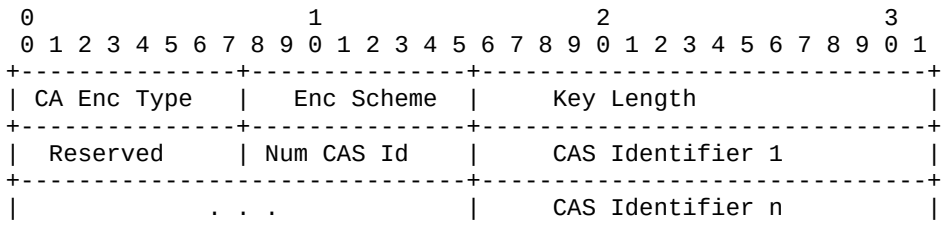


Figure 6–24 - Encryptor Capability Attribute Syntax

Table 6–26 - Values of the CA Encryption Type Parameter

Code	Keyword	Description
1	Motorola	The encryptor uses Motorola CA encryption.
2	Cisco	The encryptor uses Cisco CA encryption.
3	Simulcrypt	The encryptor uses Simulcrypt-based CA encryption.

Table 6–27 - Values of the Encryption Scheme Parameter

Code	Keyword	Description
1	DES	DES encryption; key length is 56 bits
2	CSA	DVB-CSA encryption; key length is fixed and specified under license
3	CSA3	DVB-CSA3 encryption; key length is fixed and specified under license
4	AES	AES encryption; default key length is 128 bits
5	3DES	Triple DES encryption; key length is 112 bits

6.2.3.11 Cost

The Cost attribute represents the relative cost associated with the resources from that device. A device that advertises a lower value of the cost attribute should be preferred over a device that advertises a higher value of the cost attribute. This attribute may appear within any ERRP UPDATE. In other words, this attribute is associated with the data plane component and not the particular route. The Cost attribute values are determined by means that are out of scope.

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 236

The Cost attribute has an 8-bit numeric value.

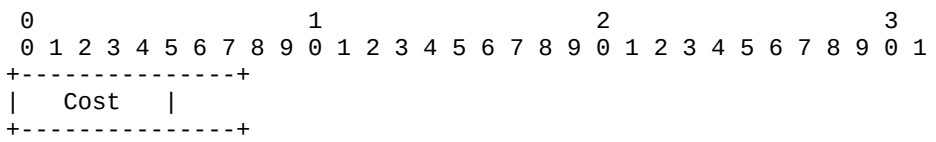


Figure 6–25 - Cost Attribute Syntax

6.2.3.12 Edge Input

The Edge Input attribute does not describe a reachable route, but instead describes the data input interfaces of the EQAM device. The data interfaces described may be physical network interfaces, loopback interfaces, or an aggregation interface (meaning a collection of network interfaces operating as an aggregated group (i.e., link aggregation, ECMP, etc.)). Therefore, it only needs to appear in one ERRP UPDATE message, not in every Reachable Route UPDATE message. It should only be sent after a connection is opened and when the value changes. This attribute describes the data plane inputs to an EQAM.

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 237

The Edge Input attribute is used by EQAMs to specify the data plane addresses to which data streams should be sent. With the data plane input address, a resource manager can determine whether the data plane components are reachable within a partially connected IP topology. In the case of an aggregation interface, a single Edge Input attribute is sufficient to describe this entry, with a max bandwidth value equal to the sum of aggregated input interface link speeds.

The Edge Input attribute describes the inputs to the "logical" EQAM component. Therefore, the Edge Input is not related to a route and can appear in a UPDATE message by itself or with a Reachable or Withdrawn Route. The syntax for the Edge Input is described as follows:

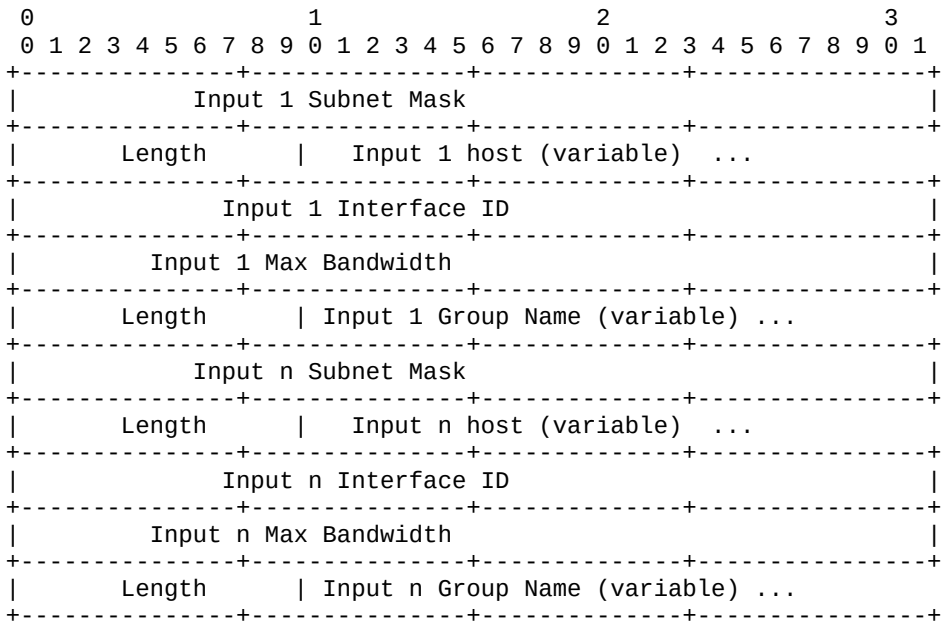


Figure 6–26 - Edge Input Attribute Syntax

Subnet Mask

The Subnet Mask field gives the IP subnet mask of the host. This information can be used by an edge resource manager to determine if multiple hosts are on the same subnet.

Length

The Length field gives the number of octets in the host field, and the host field contains the name or data plane address of the advertising component. The host field is represented as a string of ASCII characters. It is defined as follows.

Host

A legal Internet host domain name or an IPv4 address using the textual representation defined in section 2.1 of [RFC 1123] or an IPv6 address using the textual representation defined in section 2.2 of [RFC 4291]. The IPv6 address is enclosed in "[" and "]" characters. Note that the Host value may map to an individual interface, a loopback interface, or an aggregation interface on the EQAM. A Host value of 0.0.0.0 indicates that the EQAM is responsible for managing the load balancing of input video streams associated with the advertised EIG. This mode of operation is critical for EQAMs with NSI input interfaces shared with other non-ERM managed devices, such as a CMTS in a CCAP design.

Interface ID

The Interface ID consists of a binary encoded 32 bit value that is guaranteed to be unique within the context of an EQAM. The Interface id is used to determine which interface on the chassis this input represents. In the case where the Host value is set to 0.0.0.0, the Interface ID value is set to 0.

Max Bandwidth

This max bandwidth is a binary encoded 32 bit value with units of kilobits per second (kbps). This value is the maximum bandwidth that the edge input can carry. Note that an aggregation interface reports the total aggregated bandwidth of its component links as its “Max Bandwidth” value. If multiple inputs are part of the same Edge Input Group, the corresponding Max Group Bandwidth is derived from the max bandwidth of individual input interfaces by summation. When the EQAM performs load balancing of input video streams, the Max Bandwidth value indicates the total bandwidth available.

Group Name

The group name consists of a 2-byte word containing the length of the name followed by Length ASCII bytes. The field specifies the name of the Edge Input Group associated with this input.

The characters comprising the string are in the set within TEXT defined in section 15.1 of [RFC 2326]. Implementations must support minimum string lengths of 64; however, the composition of the string used is defined by implementation agreements specified by the service provider.

6.2.3.13 Input Map

The Input Map attribute identifies the internal connectivity limitation between input interfaces and output QAM channels. When this attribute is used, the QAM channel is reachable only through the listed input interfaces. Otherwise, the QAM channel is reachable through any input interface of an EQAM.

The syntax for the Input Map attribute is as shown in Figure 6–27.

Conditional Mandatory: False (may be present to modify a Reachable Route advertised)

Required Flags: Well-known

ERRP Type Code: 249

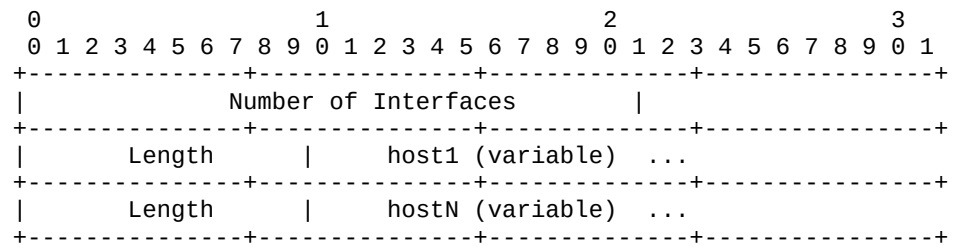


Figure 6–27 - Input Map Attribute

Number of Interfaces

The Number of Interfaces field identifies the number of input interfaces listed in this Input Map attribute.

Length

The Length field is an unsigned two-octet integer that contains the length of the host field, in octets. The Length field is mandatory.

Host

The Host field is mandatory. The Host field contains a string that represents:

- an FQDN, or
- an IPv4 address using the textual representation defined in section 2.1 of [RFC 1123], or
- an IPv6 address using the textual representation defined in section 2.2 of [RFC 4291] and enclosed in "[" and "]" characters.

6.2.3.14 UDP Map

This attribute is used to specify the UDP/IP port number that is associated with an MPEG program number in the MPTS carried within a QAM. It also may optionally show which ports can be assigned by the ERM if the provisioning interface is supported.

Conditional Mandatory: True (is mandatory to modify a Reachable Route advertised)

Required Flags: Well-known

ERRP Type Code: 239

This attribute defines the statically mapped UDP ports and the ports/programs available for dynamic provisioning. The UDP port determines the port number to which data should be sent so that the EQAM can multiplex it into the MPTS and do proper PID remapping.

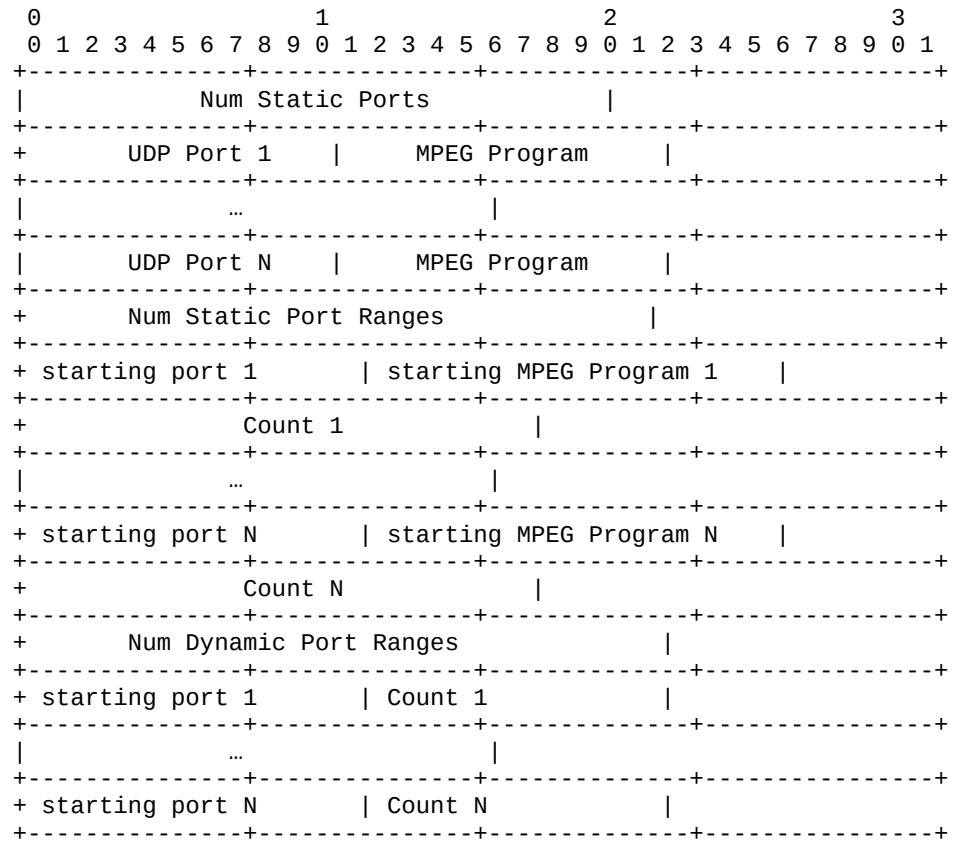


Figure 6–28 - UDP Map Attribute Syntax

Num Static ports: This is the number of static ports being described. This value will be zero for QAM channels that have no static port maps defined. If this value is non-zero, and the QAM Capability attribute does not indicate support for Mixed Static/Dynamic mode, then this QAM channel is not available for dynamic session setup.

UDP Port (n): This is the UDP/IP port number that data should be sent to in order to be multiplexed into the MPTS. These UDP ports will exist on all edge inputs reported in the Edge input attribute.

MPEG Program: This is the MPEG program number to which the data being sent to the port will be mapped.

Num Static port Ranges: This field describes the number of tuples to follow that describe the ranges of static ports/programs. This value could be zero for EQAMs that do not support static port range in which case no tuples will follow.

Starting Port(n): This is the port number that the dynamic port range starts at.

Starting Program Number(n): This is the MPEG program number associated with the Starting Port.

Count(n): The range of dynamic ports will consist of count elements, each port and program will be incremented by a value of one.

Num Dynamic port Ranges: This field describes the number of tuples to follow that describe the ranges of ports for dynamical provisioning of ports within the edge. If presented, the ERM must select UDP port from this range when provisioning a dynamic session for this QAM channel. This value should be zero if the EQAM allows the ERM to use any UDP ports in which case no tuples will follow. In dynamic provisioning, the ERM always controls the program number.

Starting Port(n): This is the port number that the dynamic port range starts at.

Count(n): The range of dynamic ports will consist of count elements, each port will be incremented by a value of one.

6.2.3.15 Max MPEG flows

This indicates the max number of MPEG flows a resource can contain.

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 242

The max MPEG flows attribute is used by data plane devices to specify the maximum number of MPEG flows that can be supported by that device. This attribute is used by data plane devices that have limitations on the number of simultaneous MPEG streams they support.

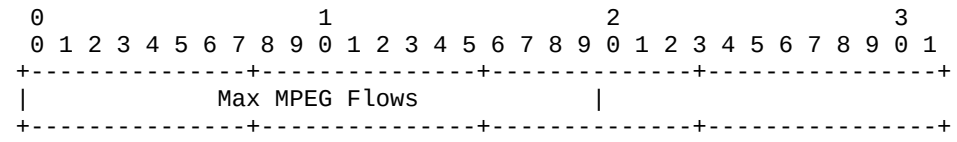


Figure 6–29 - Max MPEG Flows Attribute Syntax

The service status is a binary encoded value that specifies the maximum number of MPEG flows that a data plane component supports. It may be used as part of the resource allocation logic implemented by resource managers.

6.2.4 ERRP Error Detection and Handling

This section and its subsections specify errors to be detected and the actions to be taken while processing ERRP messages. ERRP Nodes MUST process message in the manner specified per Section 6.2.4. If any of the conditions described in subsections below are detected:

- A NOTIFICATION message with the indicated Error Code, Error Subcode, and Data fields is sent (If no Error Subcode is specified, then a zero Subcode is used);
- The TCP connection carrying the ERRP messages is torn down;
- The ERM MUST treat any QAM channel resources previously advertised through this ERRP connection as being unavailable;
- The EQAM SHOULD NOT disrupt active QAM channel resource reservations (QAM channel resources with active inbound data traffic). See Section 7.9.1.2. as well. These requirements must be applied when considering any reference to ‘releasing resources’ mentioned in Section 6.2.7.

Unless specified otherwise, the Data field of the NOTIFICATION message that is sent to indicate an error is to be empty.

6.2.4.1 Errors in Message Headers

Errors detected while processing a Message Header are indicated by sending a NOTIFICATION message with the Error Code field set to the value "Message Header Error". The Error Subcode elaborates on the specific nature of the error. The checks in this section are performed upon receipt of every ERRP message.

If the Length field of the message violates any of the requirements summarized in this section, then:

- the Error Subcode field is set to "Bad Message Length",

- the Data field contains the value from the erroneous Length field.

The Length field requirements are:

- The Length field of the message header contains a value between 3 and 4096;
- The Length field of an OPEN message contains a value bigger than 16;
- The Length field of an UPDATE message contains a value bigger than 2;
- The Length field of a KEEPALIVE message contains the value 3;
- The Length field of a NOTIFICATION message contains a value bigger than 4.

If the Type field of the message header is not recognized, then:

- the Error Subcode field is set to "Bad Message Type",
- the Data field contains the value from the erroneous Type field.

6.2.4.2 Errors in OPEN Messages

Errors detected while processing an OPEN message s indicated by sending a NOTIFICATION message with the Error Code field set to the value "OPEN Message Error". The Error Subcode elaborates on the specific nature of the error. The checks in this section are performed upon receipt of every OPEN message.

If the version number contained in the Version field of the received OPEN message is not supported, then:

- the Error Subcode field is set to "Unsupported Version Number";
- the value of the Data field is set to a 1-octet unsigned integer, indicating the largest supported version number.

If the value contained in the Address Domain field is unacceptable, then the Error Subcode field is set to "Bad (Peer) Address Domain".

If the value contained in the Hold Time field is unacceptable, then the Error Subcode field is set to "Unacceptable Hold Time". Hold Time values of one and two seconds are to be rejected. An implementation may reject any proposed Hold Time. An implementation that accepts a Hold Time uses the negotiated value for the Hold Time. If the value contained in the ERRP Identifier field is invalid, then the Error Subcode field is set to "Bad ERRP Identifier". An ERRP identifier is four octets in length and can take any value. The recipient of an OPEN message treats the contents of the ERRP Identifier field as invalid if it already has an ERRP connection in place with the same address domain and ERRP identifier.

If one of the Optional Parameters in the OPEN message is not recognized, the Error Subcode field is set to "Unsupported Optional Parameter". If the Optional Parameters of the OPEN message include Capability Information with any capability that has an unsupported capability type, or an unsupported capability value, the Error Subcode field is set to "Unsupported Capability". In this case, the unsupported capability (type and value) is listed in the Data field of the NOTIFICATION message. If the Optional Parameters of the OPEN message include Capability Information that does not match the recipient's capabilities, the Error Subcode field is set to "Capability Mismatch". In this case, all the mismatched capabilities are listed in the Data field of the NOTIFICATION message.

6.2.4.3 Errors in UPDATE Messages

Errors detected while processing an UPDATE message are indicated by sending a NOTIFICATION message with the Error Code field set to the value "UPDATE Message Error". The Error Subcode elaborates on the specific nature of the error. The checks in this section are performed upon receipt of every UPDATE message.

If any recognized attribute has Attribute Flags that conflict with the Attribute Type Code, then:

- the Error Subcode field is set to "Attribute Flags Error",
- the Data field contains the erroneous attribute (type, length, and value).

If any recognized attribute has an Attribute Length that conflicts with the expected length (based on the Attribute Type Code), then:

- the Error Subcode field is set to "Attribute Length Error",
- the Data field contains the erroneous attribute (type, length, and value).

If a required attribute is not present, then:

- the Error Subcode field is set to "Missing Well-known Mandatory Attribute",
- the Data field contains the Attribute Type Code of the missing attribute.

If an attribute with the Well Known flag set to zero is not recognized, then:

- the Error Subcode field is set to "Unrecognized Well-known Attribute",
- the Data field contains the unrecognized attribute (type, length, and value).

If any attribute has a value that is syntactically incorrect, undefined, or is an invalid value, then:

- the Error Subcode field is set to "Invalid Attribute",
- the Data field contains the incorrect attribute (type, length, and value).

If any attribute appears more than once in the UPDATE message, the Error Subcode field is set to "Malformed Attribute List".

6.2.4.4 Errors in NOTIFICATION Messages

Errors detected when processing NOTIFICATION messages should be logged to some error reporting and recording facility, as there is unfortunately no means of reporting this error via a subsequent NOTIFICATION message.

6.2.4.5 Hold Timer Expiration

If a system does not receive successive messages within the period required by the negotiated Hold Time, a NOTIFICATION message is sent with the Error Code field set to the value "Hold Timer Expired" and the ERRP connection is closed.

6.2.4.6 Errors in the Finite State Machine

Errors detected by the ERRP Finite State Machine (see Section 6.2.7) (e.g., receipt of an unexpected event) causes a NOTIFICATION message to be sent with the Error Code field set to the value "Finite State Machine Error" and the ERRP connection is closed.

6.2.4.7 Cease

In the absence of any fatal errors (defined in Section 6.2.4), an ERRP Node SHOULD close its ERRP connection by sending a NOTIFICATION message with the Error Code field set to the value "Cease". The Cease NOTIFICATION message is not to be used when a fatal error has been detected.

6.2.4.8 Connection Collision Detection

If two ERRP Nodes try simultaneously to establish an ERRP connection to each other, a race condition exists, with the possibility that two parallel connections between these ERRP Nodes might be created. Upon receipt of an OPEN message, the recipient ERRP Node examines all its connections that are in the [OpenConfirm] state (see Section 6.2.7.5). If it finds a connection in the [OpenConfirm] state with the remote ERRP Node that was the source of the OPEN message, it cleanly tears down (i.e., by transmitting a Cease NOTIFICATION) the connection with the lower numerical value of ERRP Identifier.

Upon receipt of an OPEN message, the recipient may examine connections in the [OpenSent] state if it knows the ERRP Identifier of the other ERRP Node by means outside the protocol. If it finds a connection in the [OpenSent] state with the entity that was the source of the OPEN message, it cleanly tears down (i.e., by transmitting a Cease NOTIFICATION) the connection with the lower numerical value of ERRP Identifier.

6.2.5 Negotiating the ERRP Version

ERRP Nodes may negotiate the version of ERRP by making multiple attempts to open an ERRP connection, starting with the highest version number each supports. If an attempt to open a connection fails with the Error Code "OPEN Message Error" and the Error Subcode "Unsupported Version Number", then the ERRP Node has available: the

version number it tried; the version number that the remote ERRP Node tried; the version number passed by the remote ERRP Node in the NOTIFICATION message; and the version numbers that it supports. If the two ERRP Nodes support one or more common versions, this will allow them to determine the highest version they have in common.

6.2.6 ERRP Capability Negotiation

An ERRP Node MAY include the Capabilities Option in its OPEN message. A remote ERRP Node that receives an OPEN message MUST NOT use any capabilities that were not included in the OPEN message when communicating with that ERRP Node.

6.2.7 ERRP Finite State Machine

This section specifies ERRP operation in terms of a Finite State Machine (FSM). Implementations of ERRP Nodes that conform to this specification MUST operate in accordance with the FSM described in this section. An ERRP Node MUST implement an independent FSM for every ERRP connection.

The FSM contains six states:

Table 6–28 - ERRP FSM States

State Name	Brief State Description
[Idle]	Initial state
[Connect]	TCP connection pending or open
[Active]	Listening for connection from remote node
[OpenSent]	An OPEN has been sent
[OpenConfirm]	Waiting for a KEEPALIVE or NOTIFICATION response to an OPEN
[Established]	ERRP connection ready for use

The FSM contains 13 events:

Table 6–29 - ERRP FSM Events

Event Name	Brief Event Description
{ERRP Start}	The node is instructed to open a connection to a remote node
{ERRP Stop}	The node is instructed to end the ERRP session
{ERRP TCP connection open}	A TCP connection has been successfully created
{ERRP TCP connection closed}	The TCP connection has been closed
{ERRP TCP connection open failed}	An attempt to establish a TCP connection has failed
{ERRP TCP fatal error}	The established TCP connection has terminated unexpectedly
{ConnectRetry timer expired}	The ConnectRetry timer has fired
{Hold timer expired}	The Hold timer has fired

Event Name	Brief Event Description
{Keepalive timer expired}	The Keepalive timer has fired
{Receive OPEN message}	An error-free OPEN message has been received
{Receive KEEPALIVE message}	An error-free KEEPALIVE message has been received
{Receive UPDATE message}	An error-free UPDATE message has been received
{Receive NOTIFICATION message}	An error-free NOTIFICATION message has been received

ERRP Nodes implementing state transitions in the FSM MUST conform to Table 6–30 through Table 6–35. Following each table is text that specifies the details of each table.

The FSM begins in the [Idle] state.

6.2.7.1 [Idle] State

Table 6–30 - ERRP FSM Transitions from [Idle]

Initial State	Event	Final State
[Idle]	{ERRP Start}	[Connect]
[Idle]	{ERRP Stop}	[Idle]
[Idle]	{ERRP TCP connection open}	[Idle]
[Idle]	{ERRP TCP connection closed}	[Idle]
[Idle]	{ERRP TCP connection open failed}	[Idle]
[Idle]	{ERRP TCP fatal error}	[Idle]
[Idle]	{ConnectRetry timer expired}	[Idle]
[Idle]	{Hold timer expired}	[Idle]
[Idle]	{Keepalive timer expired}	[Idle]
[Idle]	{Receive OPEN message}	[Idle]
[Idle]	{Receive KEEPALIVE message}	[Idle]
[Idle]	{Receive UPDATE message}	[Idle]
[Idle]	{Receive NOTIFICATION message}	[Idle]

Initially, the ERRP Node is in the [Idle] state. In this state, the ERRP Node ignores all incoming connections.

In response to the {ERRP Start} event (initiated by either the system or the operator), the ERRP Node FSMs:

- initialize ERRP resources;
- start the ConnectRetry timer;
- attempt to establish a TCP connection to the remote ERRP Node;
- listen for a TCP connection from the remote ERRP Node;
- enter the [Connect] state.

The value of the ConnectRetry timer has to be sufficiently large to allow TCP initialization. The ConnectRetry timer value should be 120 seconds. If an ERRP Node detects an error when in some other state, it closes the TCP connection and changes its state to [Idle]. As Table 6–31 shows, transitioning from the [Idle] state requires receipt of the {ERRP Start} event. If such an event is generated automatically, persistent errors may result in flapping of the ERRP Node. To avoid flapping when {ERRP Start} events are created automatically, whenever an ERRP Node has transitioned to [Idle] because of an error, the time between automatically generated {ERRP Start} event increases exponentially up to some maximum value. The initial value of the timer that generates the {ERRP Start} events should be 60 seconds. The value of the exponent is at least two. The maximum value of the retry timer should be at least 900 seconds.

6.2.7.2 [Connect] State

Table 6–31 - ERRP FSM Transitions from [Connect]

Initial State	Event	Final State
[Connect]	{ERRP Start}	[Connect]
[Connect]	{ERRP Stop}	[Idle]
[Connect]	{ERRP TCP connection open}	[OpenSent]
[Connect]	{ERRP TCP connection closed}	[Idle]
[Connect]	{ERRP TCP connection open failed}	[Active]
[Connect]	{ERRP TCP fatal error}	[Idle]
[Connect]	{ConnectRetry timer expired}	[Connect]
[Connect]	{Hold timer expired}	[Idle]
[Connect]	{Keepalive timer expired}	[Idle]
[Connect]	{Receive OPEN message}	[Idle]
[Connect]	{Receive KEEPALIVE message}	[Idle]
[Connect]	{Receive UPDATE message}	[Idle]
[Connect]	{Receive NOTIFICATION message}	[Idle]

In this state, an ERRP Node is waiting for a TCP protocol connection to be completed to a remote ERRP Node, and is listening for inbound TCP connections from that ERRP Node.

If the TCP connection succeeds, the ERRP Node FSMs:

- clear the ConnectRetry timer;
- send an OPEN message to the remote ERRP Node;
- set its Hold Timer to a value of at least 240 seconds;
- start the Hold Timer;
- enter the [OpenSent] state.

If the attempt to open a TCP connection fails, (e.g., because of a retransmission timeout), the ERRP Node FSMs:

- restart the ConnectRetry timer;
- continue to listen for a connection from the remote ERRP Node;
- enter the [Active] state.

If the ConnectRetry timer expires, the ERRP Node FSMs:

- cancel any ERRP TCP connection to the remote ERRP Node;
- restart the ConnectRetry timer;
- initiates a TCP connection to the remote ERRP Node;
- continue to listen for a TCP connection from the remote ERRP Node;
- stays in the [Connect] state.

If an inbound TCP connection succeeds, the ERRP Node FSMs:

- clear the ConnectRetry timer;
- complete any necessary internal initialization;
- send an OPEN message to the remote ERRP Node;
- set its Hold Timer to a value of at least 240 seconds;
- start the Hold Timer;
- enter the [OpenSent] state.

The {ERRP Start} event is ignored.

In response to any other event (initiated by either the system or the operator), the ERRP Node FSMs:

- release all ERRP resources associated with the connection;
- enter the [Idle] state.

If the local ERRP speaker detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local ERRP speaker rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

6.2.7.3 [Active] State

Table 6–32 - ERRP FSM Transitions from [Active]

Initial State	Event	Final State
[Active]	{ERRP Start}	[Active]
[Active]	{ERRP Stop}	[Idle]

Initial State	Event	Final State
[Active]	{ERRP TCP connection open}	[OpenSent]
[Active]	{ERRP TCP connection closed}	[Idle]
[Active]	{ERRP TCP connection open failed}	[Active]
[Active]	{ERRP TCP fatal error}	[Idle]
[Active]	{ConnectRetry timer expired}	[Connect]
[Active]	{Hold timer expired}	[Idle]
[Active]	{Keepalive timer expired}	[Idle]
[Active]	{Receive OPEN message}	[Idle]
[Active]	{Receive KEEPALIVE message}	[Idle]
[Active]	{Receive UPDATE message}	[Idle]
[Active]	{Receive NOTIFICATION message}	[Idle]

In this state, an ERRP Node is listening for an inbound connection from the remote ERRP Node, but is not in the process of initiating a connection to the remote ERRP Node.

If an inbound attempt to create a TCP connection succeeds, the ERRP Node FSMs:

- clear the ConnectRetry timer;
- complete initialization;
- send an OPEN message to the remote ERRP Node;
- set its Hold Timer to at least 240 seconds;
- start the Hold Timer;
- enter the [OpenSent] state.

If the ConnectRetry timer expires, the ERRP Node FSMs:

- restart the ConnectRetry timer;
- initiate a TCP connection to the remote ERRP Node;
- continue to listen for a connection from the remote ERRP Node;
- enter the [Connect] state.

In response to any other event (initiated by either the system or the operator), the ERRP Node FSMs:

- release all ERRP resources associated with the connection;
- enter the [Idle] state.

If the local LS detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local LS rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

6.2.7.4 [OpenSent] State

Table 6–33 - ERRP FSM Transitions from [OpenSent]

Initial State	Event	Final State
[OpenSent]	{ERRP Start}	[OpenSent]
[OpenSent]	{ERRP Stop}	[Idle]
[OpenSent]	{ERRP TCP connection open}	[Idle]
[OpenSent]	{ERRP TCP connection closed}	[Active]
[OpenSent]	{ERRP TCP connection open failed}	[Idle]
[OpenSent]	{ERRP TCP fatal error}	[Idle]
[OpenSent]	{ConnectRetry timer expired}	[Idle]
[OpenSent]	{Hold timer expired}	[Idle]
[OpenSent]	{Keepalive timer expired}	[Idle]
[OpenSent]	{Receive OPEN message}	[Idle] or [OpenConfirm] (see text)
[OpenSent]	{Receive KEEPALIVE message}	[Idle]
[OpenSent]	{Receive UPDATE message}	[Idle]
[OpenSent]	{Receive NOTIFICATION message}	[Idle]

In this state, an ERRP Node has sent an OPEN message to a remote ERRP Node and is waiting for an OPEN message from that ERRP Node.

When an OPEN message is received, the ERRP Node FSMs check all fields for conformance with this specification.

If the message header checking (see Section 6.2.4.1) or the OPEN message checking (see Section 6.2.4.2) detects an error or a connection collision (see Section 6.2.4.8), the ERRP Node FSMs:

- send a NOTIFICATION message;
- enter the [Idle] state.

If there are no errors in the OPEN message, the ERRP Node FSMs:

- send a KEEPALIVE message;
- set the KeepAlive timer, unless the Hold Time value is zero;
- set the Hold Timer to the negotiated Hold Time value, unless the Hold Time value is zero (see Section 6.2.2.2);
- enter the [OpenConfirm] state.

If the {ERRP TCP connection closed} event occurs, the ERRP Node FSMs:

- start the ConnectRetry timer;

- continue to listen for a connection from the remote ERRP Node;
- enter the [Active] state.

If the Hold Timer expires, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Hold Timer Expired";
- enter the [Idle] state.

If the ERRP Node receives a {ERRP Stop} event (initiated by either system or operator) the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Cease";
- enter the [Idle] state;

The {ERRP Start} event is ignored

In response to any other event, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Finite State Machine Error";
- enter the [Idle] state.

Whenever ERRP Node changes state from [OpenSent] to [Idle], it

- closes the TCP connection, and
- releases all resources associated with the connection after keepalive timeout.

If the local ERRP speaker detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local LS rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

6.2.7.5 [OpenConfirm] State

Table 6–34 - ERRP FSM Transitions from [OpenConfirm]

Initial State	Event	Final State
[OpenConfirm]	{ERRP Start}	[OpenConfirm]
[OpenConfirm]	{ERRP Stop}	[Idle]
[OpenConfirm]	{ERRP TCP connection open}	[Idle]
[OpenConfirm]	{ERRP TCP connection closed}	[Idle]
[OpenConfirm]	{ERRP TCP connection open failed}	[Idle]
[OpenConfirm]	{ERRP TCP fatal error}	[Active]
[OpenConfirm]	{ConnectRetry timer expired}	[Idle]
[OpenConfirm]	{Hold timer expired}	[Idle]
[OpenConfirm]	{Keepalive timer expired}	[OpenConfirm]
[OpenConfirm]	{Receive OPEN message}	[Idle]
[OpenConfirm]	{Receive KEEPALIVE message}	[Established]

Initial State	Event	Final State
[OpenConfirm]	{Receive UPDATE message}	[Idle]
[OpenConfirm]	{Receive NOTIFICATION message}	[Idle]

In this state, an ERRP Node has sent an OPEN message to the remote ERRP Node, received an OPEN message from that ERRP Node, and sent a KEEPALIVE message in response to the OPEN message. The ERRP Node is now waiting for a KEEPALIVE or a NOTIFICATION message in response to its OPEN message.

If the ERRP Node receives a KEEPALIVE message, it enters the [Established] state.

If the Hold Timer expires before a KEEPALIVE message is received, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Hold Timer Expired";
- enter the [Idle] state.

If the ERRP Node receives a NOTIFICATION message, it enters the [Idle] state.

If the Keepalive timer expires, the ERRP Node FSMs:

- send a KEEPALIVE message;
- restart the Keepalive timer.

If a disconnect notification is received from the underlying transport protocol (i.e., TCP), the ERRP Node FSMs:

- tear down the TCP connection;
- restart the ConnectRetry timer;
- continue to listen for a connection from the remote ERRP Node;
- enter the [Active] state.

If the ERRP Node receives a {ERRP Stop} event (initiated by either system or operator) it:

- sends a NOTIFICATION message with the Error Code "Cease";
- enters the [Idle] state.

The {ERRP Start} event is ignored.

In response to any other event, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Finite State Machine Error";
- enter the [Idle] state.

Whenever the FSM changes state from [OpenSent] to [Idle], the ERRP Node:

- closes the TCP connection and
- should release all resources associated with the connection after keepalive timeout.

6.2.7.6 [Established] State

Table 6–35 - ERRP FSM Transitions from [Established]

Initial State	Event	Final State
[Established]	{ERRP Start}	[Established]
[Established]	{ERRP Stop}	[Idle]
[Established]	{ERRP TCP connection open}	[Idle]
[Established]	{ERRP TCP connection closed}	[Idle]
[Established]	{ERRP TCP connection open failed}	[Idle]
[Established]	{ERRP TCP fatal error}	[Idle]
[Established]	{ConnectRetry timer expired}	[Idle]
[Established]	{Hold timer expired}	[Idle]
[Established]	{Keepalive timer expired}	[Established]
[Established]	{Receive OPEN message}	[Idle]
[Established]	{Receive KEEPALIVE message}	[Established]
[Established]	{Receive UPDATE message}	[Idle] or [Established] (see text)
[Established]	{Receive NOTIFICATION message}	[Idle]

In this state, an ERRP Node can exchange UPDATE, NOTIFICATION, and KEEPALIVE messages with the remote ERRP Node.

If the negotiated Hold Timer is zero, no procedures are needed or used to keep alive a session with a remote ERRP Node.

If the Hold Timer expires, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Hold Timer Expired";
- enter the [Idle] state.

If the Keepalive Timer expires, the ERRP Node FSMs:

- send a KEEPALIVE message;
- restart the Keepalive Timer.

If the ERRP Node Hold Timer is nonzero and the ERRP Node receives an UPDATE or a KEEPALIVE message, it restarts its Hold Timer.

If the Hold Timer is nonzero, then every time that the ERRP Node transmits an UPDATE message or a KEEPALIVE message, it restarts its Keepalive Timer.

If the ERRP Node receives a NOTIFICATION message, it enters the [Idle] state.

If the ERRP Node receives an UPDATE message and the UPDATE message error handling procedure (see Section 6.2.4.3) detects an error, it:

- sends a NOTIFICATION message;
- enters the [Idle] state.

If a {ERRP TCP fatal error} event occurs, the ERRP Node FSMs enter the [Idle] state.

If the ERRP Node receives a {ERRP Stop} event (initiated by either system or operator) it:

- sends a NOTIFICATION message with the Error Code "Cease";
- enters the [Idle] state.

The {ERRP Start} event is ignored.

In response to any other event, the ERRP Node FSMs:

- send a NOTIFICATION message with the Error Code "Finite State Machine Error";
- enter the [Idle] state.

Whenever the FSM changes state from [Established] to [Idle], the ERRP Node:

- closes the TCP connection and
- releases all resources associated with the connection after keepalive timeout.

6.3 ERRP Message Examples

This section provides an example of an ERRP message exchange between an EQAM and an ERM.

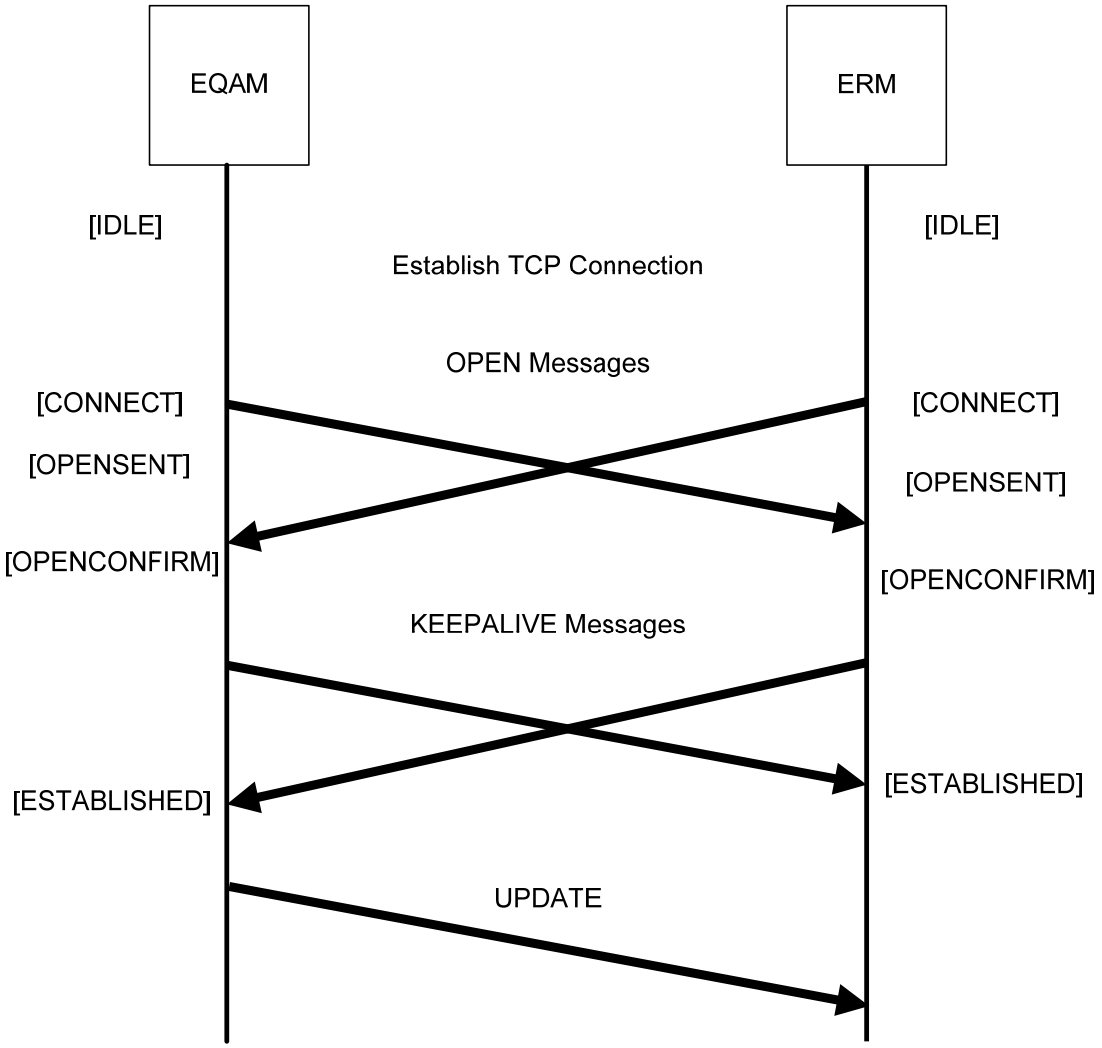


Figure 6–30 - Example ERRP Connection Establishment

6.3.1 OPEN message

The first ERRP messages sent between the EQAM and the ERM are a pair of OPEN messages.

An example message from the EQAM to the ERM is shown in Table 6–36.

Table 6–36 - Example OPEN Message

	Field	Field Length	Field Value	Comment
Header	Length	2	45	Total length of this message
	Type	1	1	OPEN
OPEN	Version	1	1	
	Reserved	1	0	
	Hold Time	2	0	No Hold Time
	Address Domain	4	1	
	ERRP Identifier	4	1234	
	Optional Parameters Len	2	28	Length of optional parameters
	Parameter Type	2	1	Capability Information
	Parameter Length	2	24	Length of the parameter value
	Capability Code	2	1	Route Types Supported
	Capability Length	2	4	Length of capability
	Address Family	2	32769	Indicates that the address is an RTSP URL
	Application Protocol	2	32768	Indicates that the application protocol is RTSP
	Capability Code	2	2	Send Receive Capability
	Capability Length	2	4	Length of capability
	Send Receive Capability	4	2	Send Only mode
	Capability Code	2	32768	ERRP version
	Capability Length	2	4	Length of capability
	ERRP version	4	1	

7.7.3.2 Extension: clab-Notice

The Notice extension header provides information sent from an RTSP Server to an RTSP Client in an ANNOUNCE request.

The syntax of the Notice header is:

clab-Notice = "clab-Notice" ":" notice-code [description] "event-date" "=" date-time " npt" ["=" npt-value]

notice-code = 4DIGIT

description = quoted-string

The Normal Play Time value (npt-value) format is defined in section 3.6 of [RFC 2326]. When the clab-Notice header is used but the npt value is not known, the npt attribute is present but the npt-value is omitted. The date-time format is defined in section 3.7 of [RFC 2326].

An example of the Notice header with known npt is:

clab-Notice:5401 "Downstream failure"

event-date=19930316T064707.735Z npt=2314223

Another example of Notice header with unknown npt is:

clab-Notice:5602 "Bandwidth Exceeded Limit"

event-date=19930316T064707.735Z npt

The clab-Notice codes are defined in Table 7–9. An RTSP Server SHOULD NOT use notice codes in an ANNOUNCE request that are not in Table 7–9. This entire table has meaning "per session".

Table 7–9 - Supported clab-Notice Codes

Code	Message	Description
2104	Delivery succeeded (start of stream reached)	For both unicast and multicast sessions, the 2104 notice code MUST be sent by the RTSP Server when MPEG data is detected as indicated in the notes below.
4400	Error Reading Content Data - PID Conflict	In some applications the ERM can request no PID remapping. If this mode is used, PID conflicts are not known to the EQAM until it sees the stream from the data plane. The EQAM MUST send an ANNOUNCE request with a 4400 notice code to the ERM when a PID conflict is detected.
4401	Input TS invalid	PAT and PMT not found in input stream.
4402	Program number conflict	If the ERM instructs the EQAM to use a Program Number already in use, the EQAM must signal this to the ERM.
5200	Server Resource Unavailable	Send when the multicast or unicast input stream is lost.
5401	Downstream Failure	For usage, section 7.11.5.3 Also, this can be used in the case of QAM failure.
5402	Downstream Destination Unreachable (i.e., ICMP Host/Port Unreachable, ARP Request Failure, etc.)	Applicable to standalone encryptor only.
5404	Unable to Join	All multicast transports were attempted.
5405	Input Interface Failure	Signaled when a stream fails due to input interface failure.

```
                    |       ";" "qam_tsid" "=" qam-tsid
                    |       ";" "fiber_node" "=" fiber-node
                    |       ";" "frequency_range" "=" frequency-range
                    |       ";" "qam_name" "=" qam-name
                    |       ";" "qam_destination" "=" qam-destination
                    |       ";" "modulation" "=" modulation-value
                    |       ";" "j83_annex" "=" j83-annex
                    |       ";" "taps" "=" taps-value ";" "increment" "=" incr-value
                    |       ";" "channel_width" "=" channel-width
                    |       ";" "symbol_rate" "=" symbol-rate
                    |       ";" "pid_input" "=" pid-input
                    |       ";" "pid_output" "=" pid-output
                    |       ";" "pid_mode" "=" pid-mode
                    |       ";" "cas_id" "=" cas-id
bit-rate            =       1*8(DIGIT)
depi-mode           =       "docsis_mpt" | "docsis_psp"
source-ip           =       host
source-port         =       1*5(DIGIT)
destination-ip      =       host
destination-port    =       1*5(DIGIT)
multicast-address   =       host
rank-value          =       1*4(DIGIT)
mpts-program        =       1*5(DIGIT)
qam-tsid            =       TSID
fiber-node          =       node-name [": " fiber-node]
node-name           =       quoted-string
frequency-range     =       frequency_low " - " frequency_high
frequency_low       =       1*10(DIGIT)
frequency_high      =       1*10(DIGIT)
qam-name            =       service-group "." TSID
service-group       =       *<any CHAR except: CTL | "," | ";" | "." >
TSID                =       1*8(DIGIT)
qam-destination     =       frequency "." program-number
frequency           =       1*10(DIGIT)
program-number      =       1*5(DIGIT)
modulation-value    =       "64" | "256"
j83-annex           =       "Annex_A" | "Annex_B" | "Annex_C"
taps-value          =       1*3(DIGIT)
incr-value          =       1*2(DIGIT)
channel-width       =       "6" | "7" | "8"
symbol-rate         =       *DIGIT
pid-input           =       pid
pid-output          =       pid
pid                 =       1*4(DIGIT)
pid-mode            =       "fixed" | "eas" | "nit" | "emm"
cas-id              =       1*4(HEX)
```

7.7.4.1.2 EQAM Input Parameters

The Transport header parameters for the EQAM Input are described below:

<bit-rate> is the data rate in bits per second of the received IP stream. The <bit-rate> defines the maximum bit-rate for the received IP stream (the sum of bandwidth of all contained PIDs).

<depi-mode> is one of the two modes defined in [DEPI].

headers: Session and CSeq. The Session header and the CSeq header are standard RTSP headers defined in [RFC 2326].

A session keepalive request from the RTSP Client to the RTSP Server is as follows:

```
SET_PARAMETER rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Session:12345678
Require: com.cablelabs.ermi
```

A corresponding response from the RTSP Server to the RTSP Client:

```
RTSP/1.0 200 OK
CSeq: 314
Session: 12345678
```

7.9.1.1 Session Timeout Value

A keepalive request SHOULD be sent periodically from the RTSP Client to the RTSP Server. The session timeout value SHOULD be sent by the RTSP Server to RTSP Client as specified as in [RFC 2326], section 12.37. The default session timeout is to be 3 hours in order to minimize the load on the RTSP client that could be managing many thousands of sessions. The keepalive request interval is less than the selected session timeout.

7.9.1.2 Session Timeout Behavior

If the RTSP Server does not receive any RTSP request in a period equal to the session timeout, the RTSP Server MAY tear down the RTSP session and release the resources associated with the RTSP session. For ERMI-2, the QAM channel resource MUST also be released if the EQAM tears down the RTSP session. The EQAM SHOULD NOT tear down the RTSP session and release the resource if it knows that the data path corresponding to this RTSP session is still receiving data traffic.

For EQAMs supporting the video profile, the RTSP Server SHOULD send an ANNOUNCE request with a clab-Notice code of 5700 "Session In Progress" upon detection of a session timeout.

The RTSP Client MUST send an ANNOUNCE response indicating either the session is still in progress (200 OK) or 454 "Session not found". The RTSP Client uses the keepalive mechanism to renew the RTSP Server session timer.

The RTSP Client MAY send a TEARDOWN request to the server to terminate the session.

If the RTSP Server does not receive an ANNOUNCE response or TEARDOWN from the RTSP Client, and it knows that the data path corresponding to this RTSP session is no longer receiving data traffic the RTSP Server MAY tear the session down.

7.9.2 Message Timeout

If an RTSP Client or RTSP Server receives an RTSP request, it MUST transmit a response. If the recipient RTSP Client or RTSP Server has issues with the format or content of the RTSP request or its parameters, it MUST respond to the sender with a RTSP response with the appropriate "Response Code". If an RTSP Client or RTSP Server sends an RTSP request, it MUST start an RTSP response timer immediately after sending the request. If the RTSP response timer expires before reception of the response, the RTSP Client or RTSP Server sender SHOULD consider the request a failure (i.e., it should act as if it had received an error response). The response timer should be set to 10 seconds. If a TEARDOWN request fails to receive a timely response, the RTSP Client SHOULD release any resources associated with the session. If the message sender receives a RTSP response for the message after its timer has expired, it executes appropriate cleanup business logic.

7.10 RTSP Response Code

An RTSP Client MUST accept the response codes defined in Table 7-13. RTSP Servers SHOULD use only the response codes defined in Table 7-13. The response code appears in the first line of RTSP response message as defined in section 7.1 of [RFC 2326]. When sending a response code other than 200, the RTSP Server SHOULD

ERMI. The ERM sends the RTSP SETUP Response back to the VOD Session Manager with the EQAM IP and QAM transport information.

During session teardown the ERM releases the stream resources. In this case, the ERM sends an RTSP TEARDOWN request to the EQAM. The EQAM sends an RTSP TEARDOWN response back via ERMI.

The following diagrams show the use case flows for session lifecycle in dynamic session case (default):

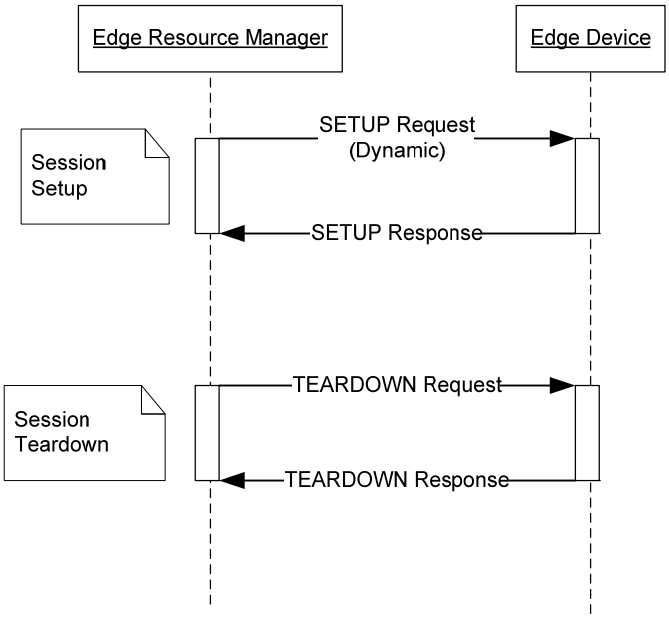


Figure I–2 - Dynamic Session Lifecycle

I.8 Switched Digital Video

To an EQAM, setting up SDV sessions looks no different than unicast on-demand sessions. The only difference is that SDV sessions will have a common source IP/UDP tuple, and one or more TSID destinations.

I.8.1 Synchronous and Asynchronous Modes

A SDV Manager will issue requests to the ERM to open SDV sessions. To the ERM and EQAM, it would appear as a group of unassociated sessions. The SDV Manager will maintain the association and add or remove sessions based on the dynamics of SDV viewership.

When the ERM directs an EQAM to SETUP a video channel, the EQAM allocates resources for the stream and issues a Multicast join request for the multicast that carries the desired channel.

The EQAM must return a SETUP response as soon as it has allocated resources for the stream. Therefore, an EQAM does not wait until the Multicast join succeeds or fails. However, when the Multicast join succeeds or fails, the EQAM sends an ANNOUNCE message to the ERM in the manner documented in Section 7.7.3.2. The EQAM sends an ANNOUNCE message when it begins outputting the first MPEG packet that carries the requested SDV channel.

The ERM is responsible for applying the synchronous / asynchronous mode policy. This is accomplished by using an ANNOUNCE message from the EQAM to indicate when the EQAM is outputting a stream or when a Multicast join failed. If operating in synchronous mode, the ERM will delay delivery of the SETUP response until it receives

an ANNOUNCE from the EQAM indicating the status of the multicast stream. When operating in asynchronous mode, the ERM will immediately return the SETUP response to the SDV Manager when it receives the SETUP response from the EQAM.

The figure below displays the message flow for synchronous mode.

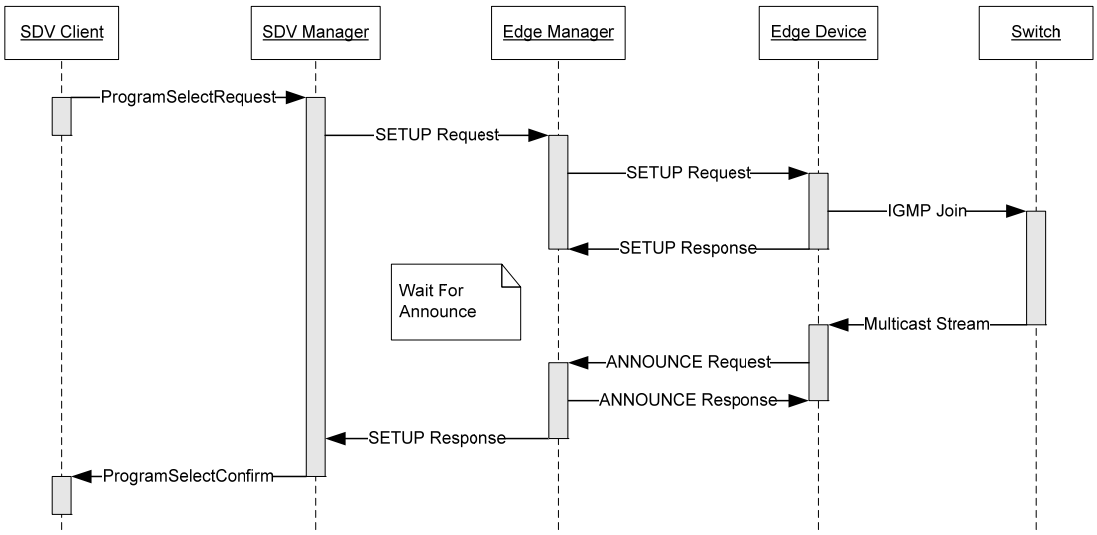


Figure I–3 - Message Flow for Synchronous Mode

The figure below displays the message flow for asynchronous mode.

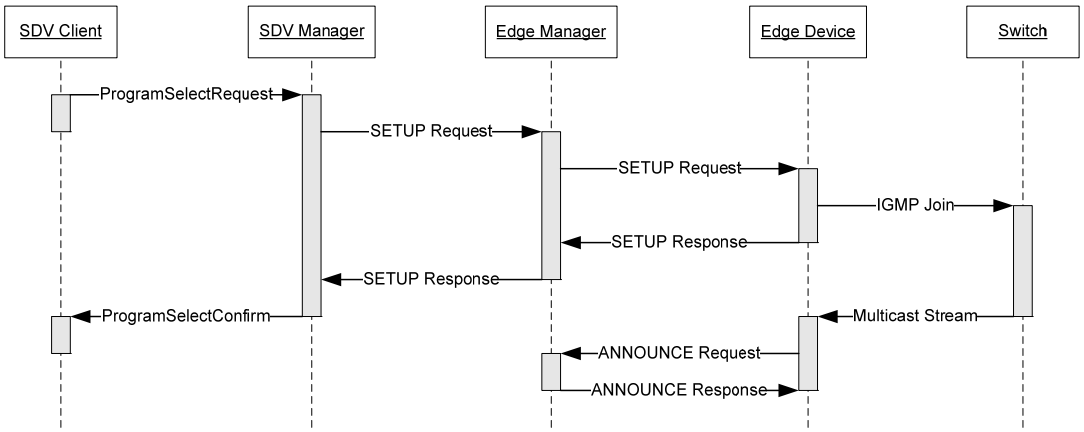


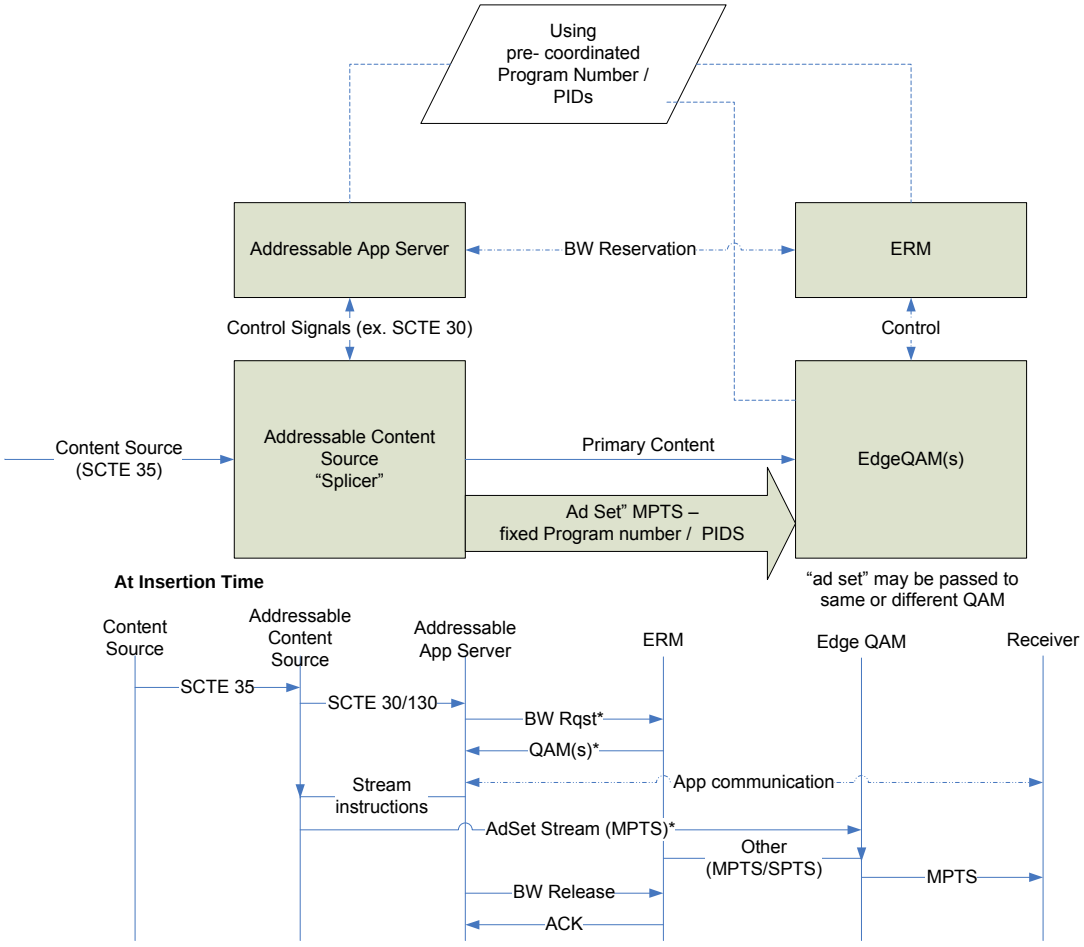
Figure I–4 - Message Flow for Asynchronous Mode

Appendix II Digital Program Insertion

This section contains requirements germane to the insertion of digital programs used for ad insertion.

II.1 Background

Ad insertion functionality, described by (draft) SCTE DVS 766 and other specifications, provides the ability to insert video ads ("Ads") into an existing program video stream ("Prog") on a per-subscriber STB basis. All the elements in the video delivery system must interact to make this possible including the Addressable App Server, the ERM, the EQAM, and each participating subscriber STB. The overall architecture is shown in the following diagram.



* Request specific bandwidth within one or more of the pre-coordinated Program Numbers/PIDs
1. Will be requesting for multiple qam groups
2. options:
A) ad set sources as a multi-cast that each QAM stream input joins
B) ad set sources as a uni-cast to a specific QAM stream input

Figure II–1 - Digital Program Insertion Diagram

The Addressable App Server will request the ERM to set up sessions for Ads. The ERM in turn will send a session setup request to the EQAM.

The sessions containing these Ads may be only be set up seconds before an ad insertion. The tight timing requirements engender the need for pipelining of Prog and Ad video data, minimal signaling overhead, and pre-

arranged coordination of Program numbers and PIDs as outlined in Section II.2.1. This in turn gives rise to the informative text detailed in Section II.2 which is needed to implement ad insertion functionality. Requirements for Addressable App Server and STBs are out of scope.

II.2 Informative text

II.2.1 Treatment of Program numbers and PIDs

The Application governing addressable advertising needs to tell (by methods out of scope) each STB the Program number and PIDs of the ad to tune to. It would not be enough to rely on the STB to parse the PMT to determine the PIDs, due to the additional latency introduced by PMT acquisition and parsing. Therefore, Program numbers/PID data is reserved so that the Program numbers/PID mapping can be performed in advanced by the Addressable App Server and communicated to the STB for loading into the MCARD CA_PMT. The method for accomplishing this is as follows:

- The provisioning configuration file provides reserved PIDs for EQAMs. Also, in the config there is enough information so the EQAM knows which PIDs to use for CA. Note: There may be other applications beyond Ad Insertion that will also need reserved PIDs. The method for sharing this reserved PID space is out of scope. Program numbers are coordinated by means out of scope between the Addressable App Server and the ERM.
- During a session setup, the Ad insertion sessions should be set up with no PID or program remap.
- ERM behavior.

For sessions carrying Ad insertion video streams, the ERM should use the async mode documented in Section I.8.1.

Appendix III Acknowledgements

On behalf of the cable industry and our member companies, CableLabs would like to thank the following individuals for their contributions to the development of this specification.

Contributor	Company Affiliation
Andrew Poole	ARRIS
Ian Wheelock	ARRIS
Doug Jones	BigBand Networks
Victor Hou	Broadcom
Charlie Bergren	CableLabs
Deepak Kharbanda	CableLabs
Greg White	CableLabs
Jean-Francois Mule	CableLabs
Don Dewar	Camiant
Neil Buchen	Cisco
Xiaomei Liu	Cisco
Sangeeta Ramakrishnan	Cisco
Ray Killick	Cisco
Weidong Mao	Comcast
Walt Michel	Comcast
Joon-Young Jung	ETRI
Woongshik You	ETRI
Daniel Cohen	Harmonic Video Systems
Robert Smith	LiquidxStream
John Schlack	Motorola
Steven Battle	Motorola
Ian Macfarlane	Pace Networks
Xuduan Lin	RGB Networks
Chuck Hasek	Time Warner Cable
Scott Davis	Vecima Networks

We would particularly like to thank the following individuals for their significant contributions:

Scott Davis, for his work as team lead for the Resource Configuration and Provisioning section,

Ian Wheelock, for his work as team lead for the Edge Resource Registration Protocol section,

Xiaomei Liu, for her extensive drafting and review throughout the specification,

Robert Smith, Neil Buchen, and Don Dewar for significant contributions and review,

Victor Hou, Sangeeta Ramakrishnan, Andrew Poole, Daniel Cohen, Steve Battle, Ian Macfarlane, Jean-Francois Mule, and Greg White for their extensive review and comments.

We would like to thank Weidong Mao and Chuck Hasek for their guidance during the development of this specification.

Finally, we would like to thank Charles Bergren for his work in leading the development of this specification.

Appendix IV Revision History (Informative)

IV.1 Engineering Change for CM-SP-ERMI-I02-051209

The following Engineering Change was incorporated into CM-SP-ERMI-I02-051209:

ECN	ECN Date	Summary
ERMI-N-05.0244-1	9/28/2005	Editorial changes, including making the figures editable.

IV.2 Engineering Change for CM-SP-ERMI-I03-081107

The following Engineering Change was incorporated into CM-SP-ERMI-I03-081107:

ECN	ECN Date	Summary
ERMI-N-08.0688-7	11/05/2008	Updates to ERMI to include video requirements

IV.3 Engineering Change for CM-SP-ERMI-I04-110623

The following Engineering Change was incorporated into CM-SP-ERMI-I04-110623:

ECN	ECN Date	Summary
ERMI-N-11.0995-3	5/25/2011	CMAP Encryption and Decryption Changes