SCTE | STANDARDS

Data Standards Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 137-4 2017 (R2021)

Edge Resource Manager Interface for Modular Cable Modem Termination Systems

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards and Operational Practices (hereafter called "documents") are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interoperability, interchangeability, best practices, and the long term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE members.

SCTE assumes no obligations or liability whatsoever to any party who may adopt the documents. Such adopting party assumes all risks associated with adoption of these documents and accepts full responsibility for any damage and/or claims arising from the adoption of such documents.

NOTE: The user's attention is called to the possibility that compliance with this document may require the use of an invention covered by patent rights. By publication of this document, no position is taken with respect to the validity of any such claim(s) or of any patent rights in connection therewith. If a patent holder has filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license, then details may be obtained from the standards developer. SCTE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this document have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE web site at https://scte.org.

All Rights Reserved
©2021 Society of Cable Telecommunications Engineers, Inc.
140 Philips Road
Exton, PA 19341

DOCSIS® and M-CMTS™ are trademarks of Cable Television Laboratories, Inc. (CableLabs) and used in this document with permission.

Document Types and Tags

Document Type: Specification
Document Tags:

☐ Test or Measurement	☐ Checklist	☐ Facility
X Architecture or Framework	☐ Metric	X Access Network
☐ Procedure, Process or Method	☐ Cloud	☐ Customer Premises

Document Release History

Release	Date
SCTE 137-4 2007	*9/18/2007*
SCTE 137-4 2010	*11/2/2010*
SCTE 137-4 2017	*2/13/2017*

Note: This document is a reaffirmation of SCTE 137-4 2017. No substantive changes have been made to this document. Information components may have been updated such as the title page, NOTICE text, headers, and footers.

Contents

Figures

Tables

1 SCOPE

NOTE: This document is identical to SCTE 137-4 2010 except for informative components which may have been updated such as the title page, NOTICE text, headers and footers. No normative changes have been made to this document.

1.1 Introduction and Overview

This document specifies interfaces that are used by Edge QAM devices (EQAMs), Edge Resource Managers (ERMs) and M-CMTS Cores within the context of a Modular Cable Modem Termination System (M-CMTS). This is one of several specifications that together define and specify a complete M-CMTS system (see Section 1.3). The basic architecture of a complete M-CMTS system is shown in Figure 1–1.

Three interfaces are specified in this document:

ERMI-1: A registration interface between an ERM and an EQAM. This interface is used to register and unregister EQAM resources (i.e., QAM channels) with an ERM.

ERMI-2: A control interface between an EQAM and an ERM. This interface is used by an ERM to request QAM channel resources from an EQAM, and by an EQAM to acknowledge resources to an ERM.

ERMI-3: A control interface between an M-CMTS Core and an ERM. This interface is used by the M-CMTS Core to request specific QAM channel resources from the ERM, and by the ERM to respond to such requests with the location of QAM channel resources.

The interfaces specified in this document are shown in Figure 1–2 in Section 1.2.

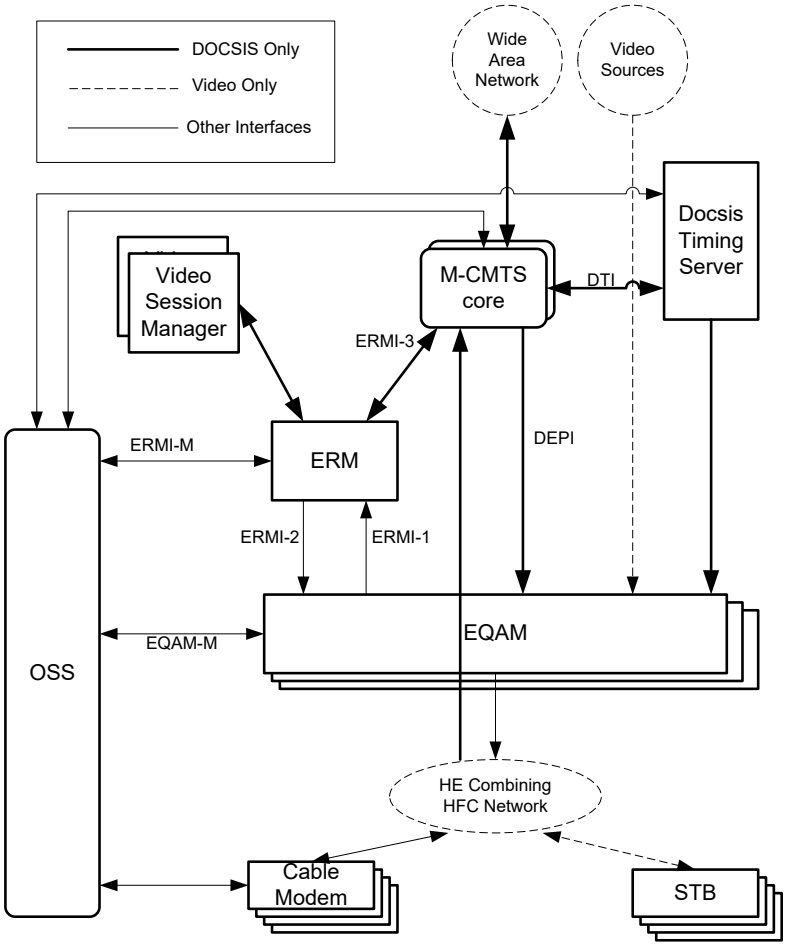


Figure 1–1 - M-CMTS Reference Architecture

1.2 Assumptions

In developing this specification, the following assumptions were made concerning the implementation and deployment of M-CMTS systems:

The system must function in the absence of an ERM.

In the absence of an ERM, all the interfaces to and from the ERM (i.e., the interfaces specified in this document) are necessarily absent. Therefore, we assume that they will be replaced by a static configuration in which:

- The M-CMTS Core is provisioned or otherwise configured with sufficient information to send data to EQAMs, and has sufficient knowledge of the resources available on each EQAM to access those resources without further information;
- The EQAMs are capable of cooperating with an M-CMTS Core, even though they have not registered their resources with an ERM;
- The operator has configured the system in such a way that no unintended conflicts for resources arise.

The system will be compatible with Euro-DOCSIS [RFI2].

QAM channel identifiers (TSIDs) will be unique per head-end. In practice, this means that a TSID must be unique within the administrative domain that includes both the DOCSIS and VOD systems. A QAM channel is also identified by a unique QAM name.

The system will be compatible with the Digital Set-top Gateway (DSG) specification [DSG].

A QAM channel resource will register with, and be controlled by a single ERM at any one time.

A QAM channel is used by one MAC Domain at any one time.

Only one CMTS may use the primary QAM channel (the QAM channel that carries the MAC control messages when DOCSIS channel-bonding is used).

Multiple ERMs may exist in a single system and the following are beyond the scope of this specification:

- Communication between ERMs;
- Resource aggregation using data from multiple ERMs;
- The method by which a session manager chooses the ERM with which to communicate.

A single ERM may manage both DOCSIS and video resources. The interface between Video Session Managers and the ERM is out of scope.

The M-CMTS system must function with DOCSIS 1.x and 2.0 cable modems (which do not provide feedback to the CMTS as to which downstream channels they can receive).

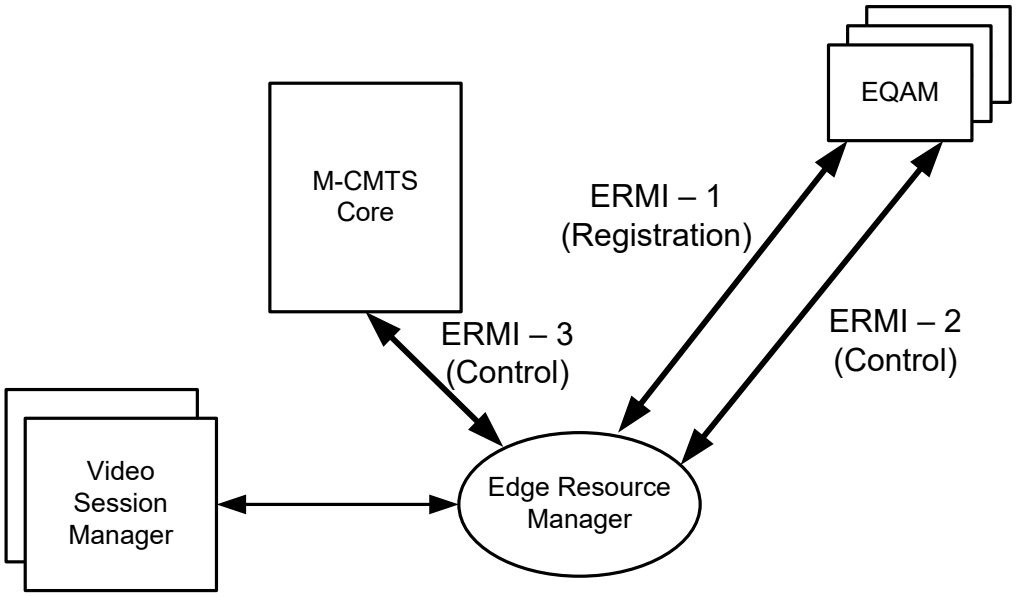


Figure 1–2 - ERMI Interfaces

1.3 EQAM Profiles

An EQAM can be any of three designated profiles:

- Video EQAM Profile

  An EQAM of this profile supports DOCSIS EQAM requirements applicable to delivering digital video and does not support DOCSIS M-CMTS EQAM requirements.

  A video EQAM supporting only static UDP port mapping does not satisfy all necessary requirements to be designated as fitting this profile.

- M-CMTS EQAM Profile

  An EQAM of this profile supports requirements applicable to an M-CMTS EQAM.

- Universal EQAM Profile

  An EQAM of this profile supports requirements applicable to delivering digital video and supporting M-CMTS EQAM requirements. Output QAM channels can be flexibly allocated to digital video delivery or DOCSIS high speed data service. It is not required that the Universal EQAM be able to multiplex both digital video and DOCSIS data on the same QAM channel.

This specification contains requirements in support of all three EQAM profiles. To delineate the application of normative statements for requirements that are specific either to the support of digital video functionality or to the support of M-CMTS functionality, this specification explicitly indicates that such a requirement applies to the video profile or to the M-CMTS profile. The abovementioned indications notwithstanding, all EQAM requirement statements provided in this specification apply to the Universal EQAM profile.

1.4 Requirements and Conventions

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

The following documents contain provisions, which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreement based on this standard are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

2.1 Normative References

[DEPI]	ANSI/SCTE 137-2 2007, DOCSIS Downstream External PHY Interface for Modular Cable Modem Termination Systems.
[J.83]	ITU-T Recommendation. 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]	ANSI/SCTE 137-6 2010, Modular Headend Architecture Part 6: Edge QAM Video Stream Interface

2.2 Informative References

[DSG]	SCTE 106 2010, DOCSIS® Set-top Gateway (DSG) Specification.
[RFC 2131]	IETF RFC 2131, R. Droms, "Dynamic Host Configuration Protocol", March 1997, Internet Engineering Task Force.
[RFI2]	ANSI/SCTE 79-1 2009, DOCSIS 2.0 Part 1: Radio Frequency Interface.

2.3 Reference Acquisition

- Internet Engineering Task Force (IETF), http://www.ietf.org
- International Telecommunication Union – Telecommunication Standardization Sector (ITU-T), http://www.itu.int/itu-t/

3 TERMS AND DEFINITIONS

This specification defines the following terms:

Edge Input Group	A set of EQAM input interfaces that have equivalent connectivity in the operator's Ethernet network.
Edge Resource Manager	A network element that manages the input and output resources of an EQAM via the protocols defined in this specification.
EQAM (or Edge QAM)	A head-end or hub device that receives packets of digital video or data from the operator network. It re-packetizes the video or data into an MPEG transport stream and digitally modulates the transport stream onto a downstream RF carrier using QAM.
ERRP Node	An entity that participates in exchanges of ERRP messages; an ERM or an EQAM.
Input Map	A list of EQAM input interfaces that reach a particular QAM channel within the internal architecture of an EQAM.
MAC domain	A grouping of layer 2 devices that can communicate with each other without using bridging or routing. In DOCSIS, a MAC domain is the group of CMs that are using upstream and downstream channels linked together through a MAC forwarding entity.
QAM Group	A set of QAM channels that reach a common set of set-top boxes.
Route	In the scope of ERRP, a QAM channel.
Service Group	An HFC service group (also known as a service group) is a portion of an HFC access network used to deliver a set of services to a population of cable modems or set-top boxes that share a common spectrum of RF channels**.**
Transport-spec	A comma delimited element in an RTSP transport header.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations and acronyms:

aBNF	augmented Backus-Naur Form
CA	Conditional Access
CAS	Conditional Access System
CM	Cable Modem
CMTS	Cable Modem Termination System
DEPI	DOCSIS External PHY Interface
DOCSIS®	Data-Over-Cable Service Interface Specifications
DOCSIS-MPT	DOCSIS MPEG Transport Stream mode of DEPI
DOCSIS-PSP	DOCSIS-Packet-Streaming-Protocol of DEPI
DS	Downstream
DSM-CC	Digital Storage Media Command and Control
EQAM	Edge QAM
ERM	Edge Resource Manager
ERMI	Edge Resource Manager Interface(s)
ERRP	Edge Resource Registration Protocol
FQDN	Fully Qualified Domain Name
FSM	Finite State Machine
HFC	Hybrid Fiber Coax
IP	Internet Protocol
M-CMTS	Modular Cable Modem Termination System
MAC	Media Access Control. Layer 2 of the ISO seven-layer model.
MPEG	Motion Picture Experts Group
MPEG-TS	Motion Picture Experts Group Transport Stream
MPTS	Multiple Program Transport Stream
PID	Packet Identifier
PSP	Packet Streaming Protocol
QAM	Quadrature Amplitude Modulator (or Modulation)
RF	Radio Frequency
RTSP	Real-Time Streaming Protocol
SDV	Switched Digital Video
SPTS	Single Program Transport Stream
SRM	Session Resource Manager
SSP	Session Setup Protocol
STB	Set top box
TRIP	Telephony Routing over IP
TSID	MPEG2 Transport Stream ID
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VOD	Video On Demand

5 TECHNICAL OVERVIEW

An EQAM is an edge device that receives packets of digital video or data from the IP network. It re-packetizes the video or data and delivers to the HFC network using QAM outputs. ERM is an edge resource manager that manages QAM resources on EQAMs.

Specifically, ERMI defines a mechanism for an ERM to discover EQAM resources via a registration interface. ERMI also defines resource allocation interfaces for an ERM to allocate QAM resources from an EQAM and for a CMTS core to allocate QAM resources from an ERM.

5.1 Edge Architecture Overview

An EQAM can be modeled as a device that has a number of IP input interfaces and a number of QAM channel outputs. The ERM learns the inputs and outputs from the registration interface.

Input interfaces advertised to an ERM are considered by the ERM as IP destinations that can be used to reach QAM channel outputs of the registering EQAM. Input interfaces advertised may be physical interface addresses or virtual interface addresses of the EQAM. An EQAM may support either any-to-any or partial internal connectivity between input interfaces and output QAM channels. When only partial connectivity is supported, the input to output mapping is advertised by the EQAM to the ERM via an "Input Map" for each QAM channel. The ERM might need to further subdivide the inputs of an EQAM (or of an Input Map in the case of partial internal connectivity) based on network connectivity external to the EQAM. A set of EQAM input interfaces that have equivalent connectivity in the operator's Ethernet network (i.e., source to EQAM) are called an Edge Input Group (EIG).

An ERM manages the EQAM output at the granularity of a QAM channel, though several QAM channels can be physically connected to a single RF port on the EQAM. QAM channels are grouped to TSID groups and QAM groups. TSID group is used to manage the configuration of QAM channels when there are hardware limitations on QAM channel independency. For example, QAM channels connected to the same RF port may be restricted to have a contiguous frequency assignment. QAM group is defined as a group of QAM channels that have equivalent connectivity in the RF topology (i.e., they reach the same set of fiber nodes).

The following figure illustrates EQAMs, their input and output relationships, and these network topology concepts. In the diagram, there are two EQAMs. The first EQAM has four input interfaces (G1, G2, G3, and G4) and four output QAM channels (Q1, Q2, Q3, and Q4), with any-to-any internal connectivity between inputs and outputs. This EQAM has its four input interfaces grouped into two Edge Input Groups EIG1 (G1 and G2) and EIG2 (G3 and G4) based on the Ethernet network topology. This EQAM has its four QAM channels grouped into two QAM groups QG1 (Q1 and Q2) and QG2 (Q3 and Q4) based on the RF topology. The second EQAM has four input interfaces (G5, G6, G7, G8) and four output QAM channels (Q5, Q6, Q7, Q8), but does not have any-to-any internal connectivity between inputs and outputs. In this EQAM, Q5 and Q6 are only reachable by G5 and G6, while Q7 and Q8 are only reachable by G7 and G8. This EQAM would advertise these input maps in registration. Even though the Ethernet network topology is common for all four input interfaces, and thus they are configured as a single Edge Input Group, the input maps result in the ERM restricting its input interface selection when assigning a stream to an output QAM channel. Since all four output QAM channels reach the same set of fiber nodes, they are considered to be a single QAM group (QG3).

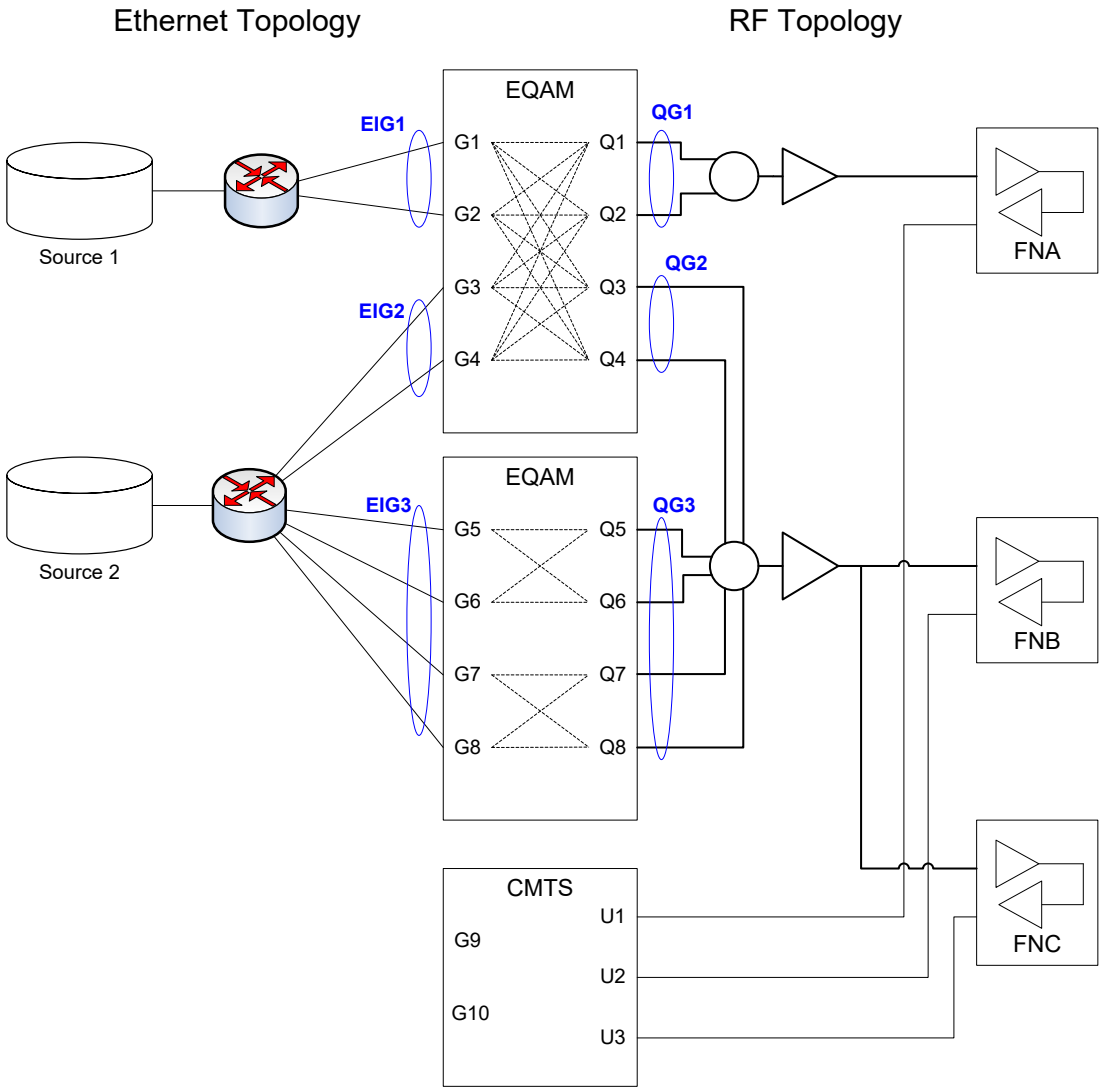


Figure 5–1 - RF Topology

ERMI facilitates the RF topology configuration and discovery feature introduced in DOCSIS3.0. In DOCSIS 3.0, fiber node is configured and used such that the downstream service groups and upstream service groups can be associated in MAC domains. Downstream service group is the complete set of downstream channels that could potentially reach a single Cable Modem or cable STB. Throughout this spec, service group refers to downstream service group.

A service group may span one or multiple fiber nodes. A service group may include one or multiple QAM groups. In the edge architecture shown in Figure 5–1, service group A (SGA) is feeding fiber node A. Service group SGA includes only a single QAM group QG1. Service group B (SGB) is feeding both fiber node B and fiber node C. Service group SGB includes QAM groups QG2 and QG3. In addition, it is also possible for a QAM group to belong to multiple service groups.

RF topology information is configured at EQAMs. QAM channel to QAM group mapping and QAM group to fiber node mapping are configured at EQAMs. An ERM learns the RF topology information via the registration interface. When a CMTS core requests edge resources from an ERM, the fiber node information is communicated between the CMTS core and the ERM. When a Video Session Manager requests edge resources from an ERM, the service group QAM list is communicated between the Video Session manager and the ERM.

Service groups are not directly managed by the EQAM and the ERM. Service groups may be managed directly by higher level entities such as the CMTS core or a video session manager.

5.2 Registration Interface

The registration interface (ERMI–1) allows EQAMs to register their QAM channel resources with an ERM.

5.2.1 Goals, Scope and Constraints

The Registration interface was designed to meet the following goals:

- Enable EQAMs to register their resources with an ERM,
- Help the ERM to detect when failures occur in resources that it manages.

The protocol used on this interface is based on TRIP [RFC 3219]. TRIP was designed to assist networks to locate voice-over-IP gateways and to route voice-over-IP traffic to an appropriate egress gateway. TRIP is a policy-driven protocol for advertising the reachability of telephony destinations between location servers, and for advertising attributes of the routes to those destinations.

TRIP is extended for this application to provide EQAMs with a mechanism to advertise reachability of QAM channels and to advertise capabilities and attributes of those QAM channels.

When a VOD application needs a downstream QAM channel resource, it will request a downstream QAM channel resource from the ERM. When a DOCSIS application needs a downstream QAM channel resource, it will request a downstream QAM channel resource from the ERM.

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 5-2 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.

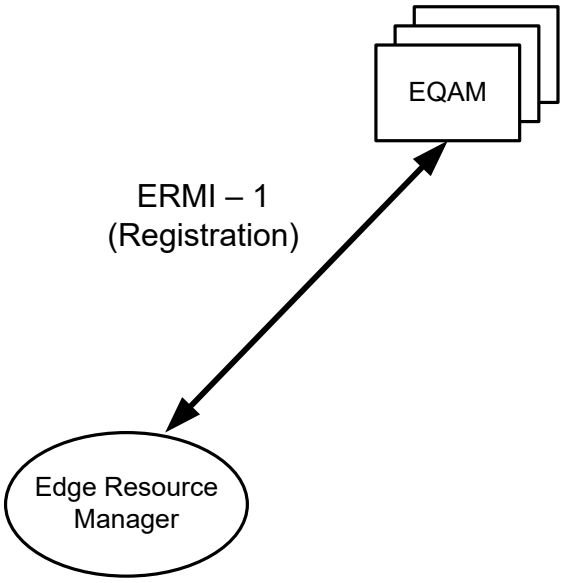


Figure 5-2 - Registration Interface and Components

An ERM normally manages multiple EQAMs, as shown in the Figure 5-2. 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 be configured to communicate with a secondary ERM in the event that it can no longer communicate with the primary ERM. However, a single QAM channel is controlled by a single ERM at a time.

Messages sent using the ERRP protocol are carried over a TCP/IP connection that may be initiated by either the EQAM or the ERM.

In this specification, the term "ERRP Node" (or simply "node") is sometimes used to refer to a generic entity that participates in exchanges of ERRP messages.

Because the device control interface for an EQAM is based on RTSP (see Section 6), the EQAM advertises an RTSP URL, in addition to the IP address. This RTSP URL is used to establish an RTSP transport connection with the EQAM. This allows the ERM to send device control messages to the EQAM to request QAM channel resources.

In normal use, an M-CMTS Core requests a QAM channel resource from an ERM, (i.e., during the process of creating a DOCSIS MAC domain (over ERMI-3)). The resources previously advertised by EQAMs (over ERMI-1) are used by the ERM to select a suitable QAM channel resource to meet the requirements of an incoming request. The ERM checks with the EQAM (over ERMI-2) that the resource is available, and then the ERM returns addressing information for that resource (over ERMI-3) to the requesting M-CMTS Core.

In addition to an initial advertisement, the EQAM sends additional advertisements (over ERMI-1) whenever an attribute of a resource under its control changes. For example, if a QAM channel within an EQAM changes its carrier frequency, the EQAM will transmit the new information to the ERM. Also, if an EQAM detects a fatal failure in one of its QAM channels, it will inform the ERM.

5.2.3 ERRP Operation

This section describes how EQAMs use ERRP to register a QAM channel resource with an ERM and how ERMs use ERRP to obtain information about QAM channel resources, in order to build a database of available resources.

5.2.3.1 ERRP Addressing

An ERRP advertisement provides the mapping from an application-specific address field (in our case, a URL) of an advertised resource to an IP signaling address where control messages for that resource should be sent. The application-specific address identifies the individual resource being advertised.

When it initializes, the EQAM advertises the application-specific addresses (i.e., URLs) of its available QAM channel resources to the ERM. When an M-CMTS Core sends a suitable request to the ERM, the ERM will select a QAM channel resource from its database. The ERM then sends an RTSP request to the EQAM to allocate this QAM channel resource to the DOCSIS session. After the EQAM receives a resource allocation request from the ERM, the EQAM verifies that the resource is available and that the QAM channel can be configured as requested. The EQAM then marks the resource as being in use and responds to the ERM request with an indication of success.

For each QAM channel resource that the EQAM wishes to advertise, it sends a ERRP ReachableRoute attribute to the ERM, embedded in a ERRP UPDATE message. To withdraw a QAM channel from service, the EQAM sends a WithdrawRoute attribute.[1]

ERRP advertisements include the IP address of the EQAM (to which RTSP signaling requests are sent) in a NextHopServer attribute. The syntax and semantics of the ERRP NextHopServer attribute are described in Section 6.2.3.3.

5.2.3.2 RTSP URLs

An RTSP URL identifies the protocol to be used to access the resource as RTSP, and contains two additional fields: hostname and abs-path, separated by a slash: rtsp://hostname[:port]/.

The hostname field may contain either a FQDN or an IP address. In either case, it may include an optional TCP port number. The value identifies the address of the server that receives RTSP requests. In the context of the ERMI-2 interface, the ERM is an RTSP client and the EQAM is an RTSP server.

5.2.3.3 ERRP Timers

ERRP uses two timers, the Hold Timer and ConnectRetry Timer.

5.2.3.3.1 Hold Timer

Whenever an ERRP Node receives a message, it resets and starts the Hold Timer. If the Hold Timer fires before it receives another message, then it sends a NOTIFICATION message which causes the ERRP connection to be closed. The duration of the Hold Timer, which is called the Hold Time, is negotiated by the ERRP Nodes when the ERRP connection is established. KEEPALIVE messages should be sent at an interval of 1/3 of the Hold Time, in order to ensure that the Hold Timer does not fire.

5.2.3.3.2 ConnectRetry Timer

The ConnectRetry Timer is used during the process of establishing a ERRP connection. After the ERRP Node initiates the TCP connection to the remote ERRP Node, it starts the ConnectRetry Timer and waits for a response from the remote node. If the ConnectRetry Timer expires before receiving a response from the remote node, the ERRP Node will retry to establish the TCP connection.

5.2.3.4 ERRP Attributes

In a ERRP UPDATE message, EQAMs may advertise the ERRP attributes (for each QAM channel) to the ERM. Each of these attributes is specified in detail in Section 6.2.3.

[1] The names of the Route, ReachableRoute and WithdrawRoute attributes are drawn from TRIP. Although this document retains these names, it may help the reader to think of them instead as Resource, UsableResource and WithdrawResource.

5.3 Resource Allocation Signaling

5.3.1 Resource Allocation Components and Interface

Figure 5-3 shows the components involved in the resource allocation interfaces.

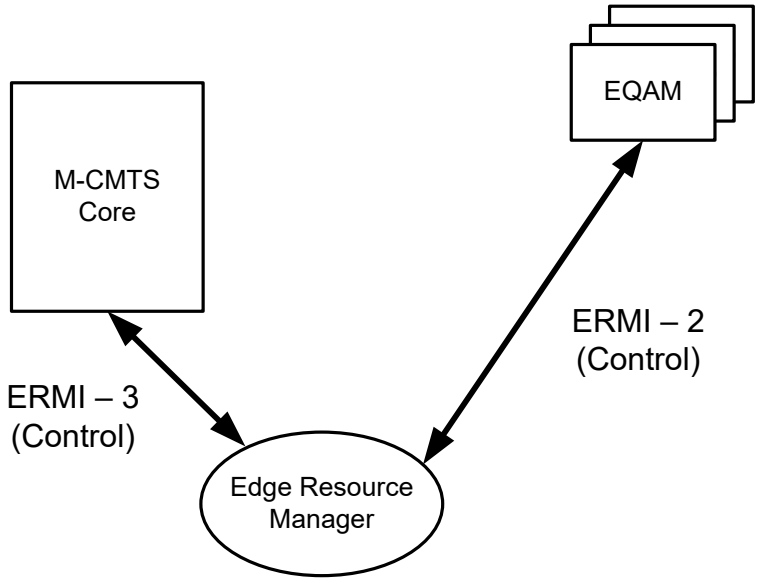


Figure 5-3 - Resource Allocation Interfaces and Components

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

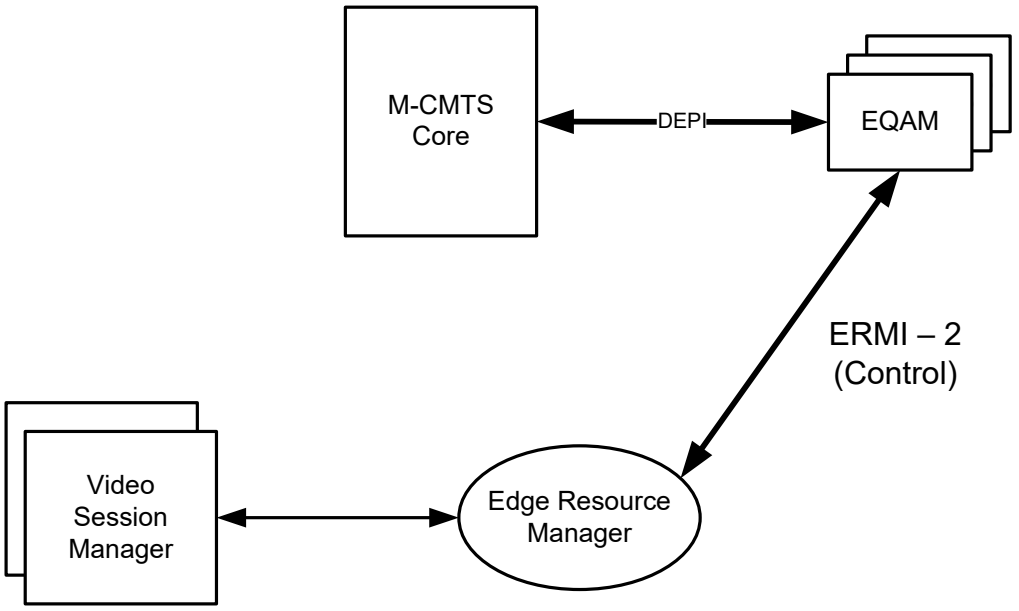


Figure 5-4 - Simplified Framework for Static QAM Partitioning

Figure 5-4 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 partioning 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:

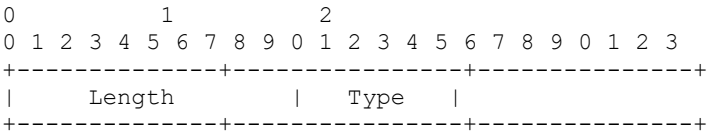


Figure 6–1 - DDRP Header Format

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:

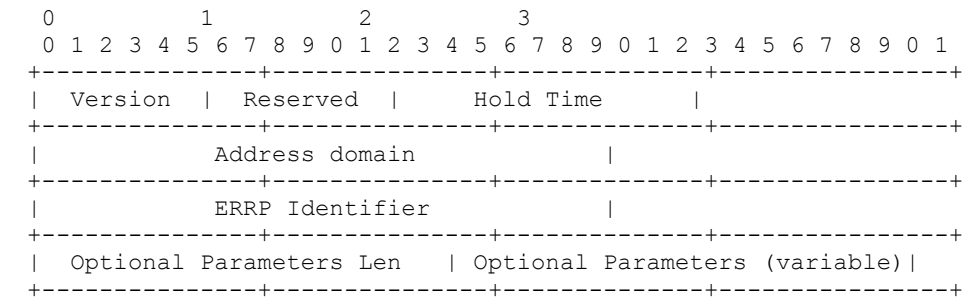


Figure 6–2 - ERRP OPEN Header

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.

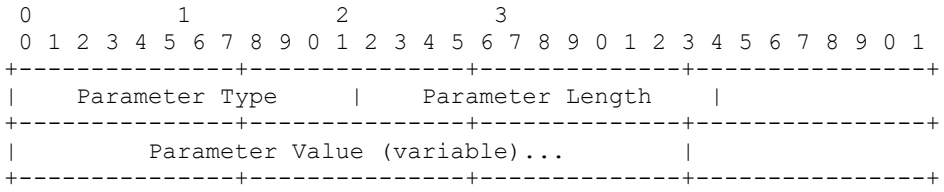


Figure 6–3 - Optional Parameter Encoding

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:

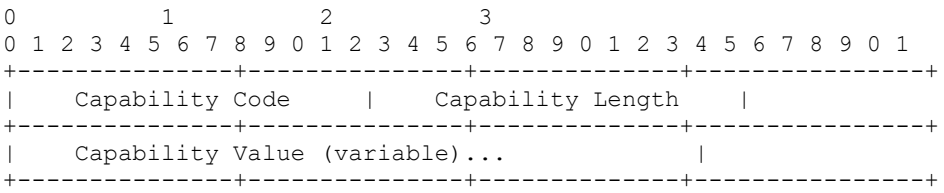


Figure 6–4 - Capability Optional Parameter

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:

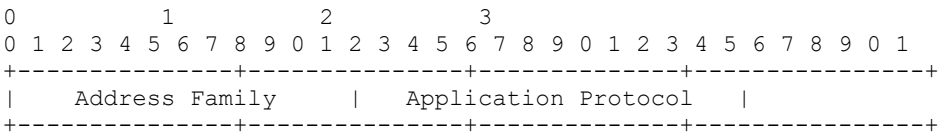


Figure 6–5 - Route Type Format

The Address Family and Application Protocol fields are specified in Section 6.2.3.1. The Address Family field identifies the address family of the resource within the ReachableRoutes attribute. The Address Family field is mandatory. The Application Protocol field identifies the application for which the resource may be used. The Application Protocol field is mandatory.

For example, a Route Type for ERRP could be <URL, ERMI>, indicating a URL for the ERMI resource signaling interface to the EQAM.

The Route Types Supported capability may contain multiple Route Types in the capability. The number of Route Types listed in the capability is limited only by the value of the Capability Length field.

6.2.2.2.1.1.2 Send Receive

This capability specifies the mode in which the ERRP Node will communicate with the remote ERRP Node. The possible modes are: Send Only, Receive Only, and Send Receive. The default mode is Send Receive.

In Send Only mode, an ERRP Node speaker transmits UPDATE messages to its ERRP Node peer, but the ERRP Node peer MUST NOT transmit UPDATE messages to that ERRP Node speaker. If an ERRP Node speaker in Send Only mode receives an UPDATE message from its ERRP Node peer, it MUST discard that message, but no further action should be taken.

In Receive Only mode, the ERRP Node listener acts as a passive TRIP listener. In Receive Only mode, an ERRP Node MUST NOT transmit an UPDATE message though it receives and processes UPDATE messages from its ERRP Node peer. In Receive Only mode, an ERRP Node still transmits OPEN, KEEPALIVE, and NOTIFICATION messages. In Send Only mode, an ERRP Node still receives OPEN, KEEPALIVE, and NOTIFICATION messages. The Send Receive Capability field is a 4-octet unsigned integer. The value of the Send Receive Capability field is one of the values identified in Table 6–3.

Table 6–3 - Send Receive Capability

Send Receive Capability	Meaning
1	Send Receive
2	Send Only
3	Receive Only

If an ERRP Node discovers while processing an OPEN message that both it and its remote ERRP Node are in Send Only mode or in Receive Only mode, it MUST send a NOTIFICATION message to the remote ERRP Node to close the session. The error code in this NOTIFICATION message is set to "Capability Mismatch" (see the ERRP error handling Section 6.2.4).

An ERRP Node MUST send the same value of Send Receive Capability to all remote ERRP Nodes. An ERM SHOULD advertise Send Receive mode if it supports both VOD and DOCSIS applications. An EQAM MUST advertise Send Only mode. ERRP component send receive capability is summarized in

Table 6–4.

Table 6–4 - ERRP Component Send Receive Capability

Component	ERRP Send Receive Capability
EQAM	Send Only mode
ERM	Send Receive mode or Receive Only mode

6.2.2.2.1.1.3 ERRP Version

This is a 4-octet unsigned integer, representing the version of ERRP supported by the ERRP Node. If an ERRP Node receives an OPEN message containing a value for the ERRP Version field that it does not support, it MUST respond with a NOTIFICATION message. The error code in the NOTIFICATION message is set to "Capability Mismatch" (see the ERRP error handling Section 6.2.4).

The value of the ERRP Version Capability field is 1.

6.2.2.2.1.2 Streaming Zone

StreamingZone Name is a mandatory parameter when supporting video applications. It identifies its presence by setting the value of the Parameter Type field to 2. The capability is optional when signaling DOCSIS only resources. The value is to be set to the string that represents the StreamingZone Name i.e., <region>.<local name>. The length should be set to the length of the string and as such is variable. The Streaming Zone name represents the Streaming Zone within which the component operates.

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.

6.2.2.2.1.3 Component Name

Component Name is a mandatory parameter when supporting video applications. It identifies its presence by setting the value of the Parameter Type field to 3. The capability is optional when signaling DOCSIS only resources. The value is to be set to the string that represents the Component Name i.e., <region>.<local name>. The length should be set to the length of the string and as such is variable. The Component Name is the name of the component for which the data in the update message applies. For example, the Component Name would be the EQAM name in the case of the EQAM Registration Interface.

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.

6.2.2.2.1.4 Vendor Specific String

Vendor Specific String is an optional parameter. It identifies its presence by setting the value of the Parameter Type field to 4. The length should be set to the length of the string and as such is variable. The value could be anything the vendor of the component wishes to use to pass "innovation" hints to resource managers. Examples could include vendor and model name and number or enumerated list of functions.

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.

6.2.2.3 UPDATE Message Format

UPDATE messages are used to transfer information about resources between ERRP Nodes. Information received in UPDATE messages is used to populate a database of available resources.

In particular, UPDATE messages are used to advertise and to withdraw resources. A single UPDATE message may simultaneously advertise and withdraw ERRP resources.

In addition to the ERRP header, the ERRP UPDATE may contain a list of Routing Attributes, which are formatted as shown in Figure 6–6. Routing Attributes are contiguous (no padding) in the message.

Figure 6–6 - ERRP UPDATE Format

The minimum length of an UPDATE message is 3 octets (i.e., there are no mandatory attributes).

6.2.2.3.1 Routing Attributes

A variable length sequence of Routing Attributes is present in every UPDATE message. Each Routing Attribute is formatted as shown in Figure 6–7 and Figure 6–8.

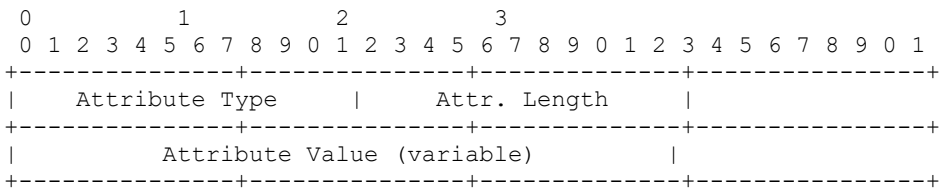


Figure 6–7 - Routing Attribute Format

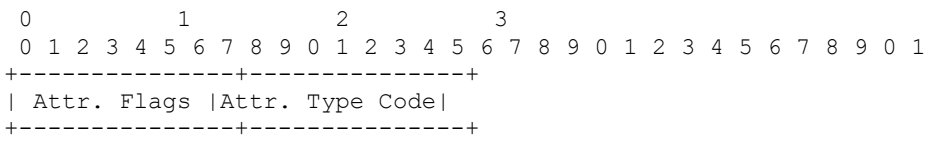


Figure 6–8 - Attribute Type Format

Attribute Type

A two-octet field that consists of two sub-fields: a one-octet Attribute Flags sub-field, followed by a one-octet Attribute Type Code sub-field. The Attribute Type field is mandatory. The Attribute Flags sub-field is mandatory. The Attribute Type Code sub-field is mandatory.

The value of the Attribute Type Code field identifies the type of the attribute. The allowable values are specified later in this section. If multiple attributes are present in an UPDATE message, they are in increasing numerical order of the Attribute Type Code field. A particular value of this field is not to appear more than once in a single UPDATE message. Attribute flags are used to control attribute processing when the attribute type is unknown, and are specified in Section 6.2.2.3.2.

ERRP uses TRIP as base protocol. ERRP Nodes MUST conform to [RFC 3219] in the use of the Attribute Type Code. [RFC 3219] reserves the value of zero and defines attributes with codes between 1 and 11 for this field. [RFC 3219] allows new attributes to be defined using vendor-specific codes 224 to 255 (these are the codes with the first three bits of the code equal to 1). Attribute Type Codes (other than those reserved for vendor specific use) are controlled by IANA. ERRP supports some of the attributes (ReachableRoutes, WithdrawnRoutes, and NextHopServer) defined in [RFC 3219]. ERRP defines more ERRP specific attributes using Attributes Type Codes between 224 and 255. These ERRP-supported attributes are further specified in Section 6.2.3.

Attributes not supported by ERRP MUST NOT be used by ERRP Nodes.

Attribute Length

The Attribute Length is a two-octet unsigned integer that contains the length of the Attribute Value field in octets.

Attribute Value

The remaining octets of the attribute represent the Attribute Value and are interpreted in accordance with the values of the Attribute Flags, Attribute Type Code, and Attribute Length fields. The supported attribute types, their values, and uses are defined in Section 6.2.3.

6.2.2.3.2 Attribute Flags

The Attribute Flags field is a one-octet field with the following structure:

Table 6–5 - Attribute Flag Field Bit Definition

Bit	Flag name
0 (most significant)	Well-known Flag
1 through 7	Reserved

The high-order bit (bit 0) of the Attribute Flags octet is the Well-Known Bit. It defines whether the attribute is not well-known (if set to 1) or well-known (if set to 0). Messages received with bit 0 unset are processed. Messages received with bit 0 set may be processed.

Bits 1 through 7 are zero on transmit. Bits 1 through 7 are ignored on receipt.

6.2.2.4 KEEPALIVE Message Format

ERRP does not depend on any lower-level, keepalive mechanism to determine whether remote ERRP Nodes remain reachable. Instead, KEEPALIVE messages are exchanged sufficiently often to ensure that the Hold Timer does not expire. The maximum time between the transmissions of successive KEEPALIVE messages by an ERRP Node SHOULD NOT be more than one third of the negotiated Hold Time. An ERRP Node MUST NOT send KEEPALIVE messages more than once every 3 seconds.

If the negotiated Hold Time is zero, then KEEPALIVE messages MUST NOT be sent by an ERRP Node.

The KEEPALIVE message contains only a message header, so it has a length of 3 octets.

6.2.2.5 NOTIFICATION Message Format

A NOTIFICATION message MUST NOT be sent by an ERRP Node except in response to detection of an error. The TCP connection carrying the ERRP traffic are torn down immediately following transmission or reception of a NOTIFICATION message.

In addition to the fixed-size ERRP header, the NOTIFICATION message contains Error Code and Error Subcode fields. The NOTIFICATION message contains a Data field if the Data field is specified for the specific values of Error Code and Error Subcode in the ERRP error handling (Section 6.2.4). The error report has the following format:

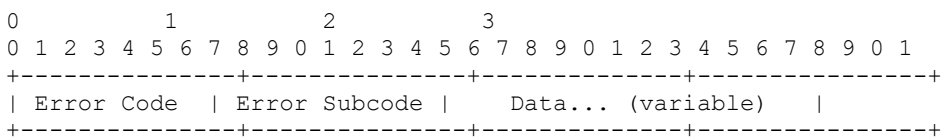


Figure 6–9 - ERRP NOTIFICATION Format

Error Code

This 1-octet unsigned integer indicates the type of the NOTIFICATION. This field is mandatory.

The value of the Error Code field is one of the values identified in Table 6–6:

Table 6–6 - ERRP Error Code

Error Code	Name	Reference
1	Message Header Error	Section 6.2.4.1
2	OPEN Message Error	Section 6.2.4.2
3	UPDATE Message Error	Section 6.2.4.3
4	Hold Timer Expired	Section 6.2.4.5
5	Finite State Machine Error	Section 6.2.4.6
6	Cease	Section 6.2.4.7

Error Subcode:

This 1-octet unsigned integer provides more specific information about the nature of the reported error. The Error Subcode field is mandatory. Each Error Code may have one or more Error Subcodes associated with it. If no appropriate Error Subcode is defined, then a zero (Unspecific) value is used for the Error Subcode field.

In the case of a Message Header Error, the value of the Error Subcode field is one of the values identified in Table 6–7:

Table 6–7 - Message Header Error Subcodes

Error Subcode	Meaning
0	Unspecified error
1	Bad Message Length
2	Bad Message Type

In the case of an OPEN Message Error, the value of the Error Subcode field is one of the values identified in Table 6–8.

Table 6–8 - OPEN Message Error Subcodes

Error Subcode	Meaning
0	Unspecified error
1	Unsupported Version Number
2	Bad Peer Address Domain
3	Bad ERRP Identifier
4	Unsupported Optional Parameter

Error Subcode	Meaning
5	Unacceptable Hold Time
6	Unsupported Capability
7	Capability Mismatch

In the case of an UPDATE Message Error, the value of the Error Subcode field is one of the values identified in Table 6–9.

Table 6–9 - UPDATE Message Error Subcodes

Error Subcode	Meaning
0	Unspecified error
1	Malformed Attribute List
2	Unrecognized Well-known Attribute
3	Missing Well-known Mandatory Attribute
4	Attribute Flags error
5	Attribute Length error
6	Invalid Attribute

Data

This variable-length field is used to provide the reason for the NOTIFICATION. The contents of the Data field depend upon the Error Code and Error Subcode fields. The Data field is mandatory if the Data field format is specified for the particular values of the Error Code and Error Subcode fields (see the ERRP error handling Section 6.2.4).

The length of the Data field can be simply determined from the message length field:

Data Length = Message Length - 5

The minimum length of a NOTIFICATION message is 5 octets (including the message header).

6.2.3 ERRP Attributes

This section specifies the syntax and semantics of each ERRP UPDATE attribute listed in Table 6–10. ERRP Nodes using Attributes in message MUST use the syntax and format of the attributes listed in the following subsections.

Table 6–10 - ERRP Attribute Type Codes

Advertised Attributes	Type Code	Supported by ERM/EQAM
Withdrawn Route	1	Must
Reachable Route	2	Must
NextHopServer	3	Must
QAM Names	232	Must
CAS Capability	233	May

Advertised Attributes	Type Code	Supported by ERM/EQAM
Total Bandwidth	234	Must
Available Bandwidth	235	May
Cost	236	Should
Edge Input	237	Must
QAM Channel Configuration	238	Must
UDP Map	239	Must
Service Status	241	Should
Max MPEG Flows	242	May
NextHopAlternate	243	May
Output Port/Port ID	244	Must
Fibre Node	245	May
QAM Capability	247	Must
Input Map	249	May

There are no mandatory attributes in an ERRP message. However, there are conditional mandatory attributes. A conditional mandatory attribute is an attribute that is included in an UPDATE message if another attribute is included in that message. The EQAM implements the "Support" column of Table 6–10 as follows:

- The EQAM MUST support the ‘Must’ ERRP Attributes from Table 6–10.
- The EQAM SHOULD support the ‘Should’ ERRP Attributes from Table 6–10.
- The EQAM MAY support the ‘May’ ERRP Attributes from Table 6–10.

The two base attributes in ERRP are WithdrawnRoutes and ReachableRoutes. Their presence in an UPDATE message is entirely optional and independent of any other attributes. Attributes appear in the UPDATE message in increasing order of the Attribute Type Code.

6.2.3.1 WithdrawnRoutes

The WithdrawnRoutes attribute identifies zero or more routes that have been removed from service. When a QAM channel is out of service, the EQAM uses a WithdrawnRoutes attribute to inform the ERM that it must remove the resource from its database of available resources.

6.2.3.1.1 Syntax of WithdrawnRoutes

The WithdrawnRoutes attribute encodes a number of routes (which may be zero) in its value field. The format for individual routes is specified in Section 6.2.3.1.1.1. The WithdrawnRoutes attribute lists the individual routes sequentially and with no padding, as shown in Figure 6–10.

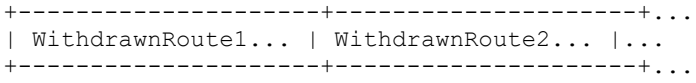


Figure 6–10 - WithdrawRoutes Format

Conditional Mandatory: False

Required Flags: Well-known

Attribute Type Code: 1

6.2.3.1.1.1 Generic ERRP Route Format

The WithdrawRoutes and ReachableRoutes attributes share the same generic Route format. A ERRP route is formatted as shown in Figure 6–11.

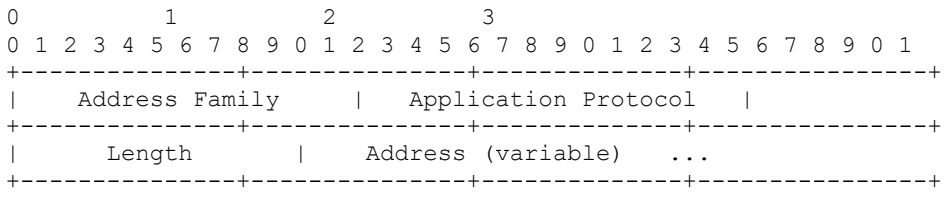


Figure 6–11 - Route Format for WithdrawRoutes and ReachableRoutes

Address Family

The Address Family field gives the type of address for the resource (route). The Address Family field is mandatory. The Address Family field contains the value given in Table 6–11.

Table 6–11 - Values for Address Family

Value of Address Family field	Meaning
32769	Video Name

The address family of Video Name in ERRP will contain a specific resource name such as QAM Group Name.

This specification reserves address family code 0. According to [RFC 3219], address family codes 1 – 32767 are administered by IANA. This specification reserves address family codes 32768, and 32770-65535 for vendor-specific applications (these are the codes with the first bit of the code value equal to 1. The values from Table 6–11 are purposefully in the vendor-specific range and can not be reused by a vendor in the context of using this specification.

Application Protocol

The Application Protocol field identifies the protocol for which the resource database is maintained. The Application Protocol field is mandatory. The Application Protocol field contains the value given in Table 6–12:

Table 6–12 - Application Protocols Supported in ERRP

Component	Application Protocol Code	Session Layer Interface	Protocol
Edge QAM	32766	pre-provisioned (static portmap)	None
	32768	Session Parameter Only (dynamic session)	RTSP
	32770	Session Parameter and provisioning (dynamic session)	RTSP

This specification reserves application protocol code 0. According to [RFC 3219], application protocol codes 1 – 32767 are administered by IANA. This specification reserves application protocol codes 32771-65535 for vendor specific applications. The values from Table 6–12 are purposefully in the vendor-specific range and can not be reused by a vendor in the context of using this specification.

Length

This field is a 2-octet unsigned integer containing the length of the Address field, in octets. The Length field is mandatory.

Address

This is an address (prefix) of the family type given by the Address Family field. The Address field is mandatory. The length of the Address field is variable and is given by the value contained in the Length field.

If Address Family field contains 32769, the Address field is a Route Name. The Route Name field will contain an ASCII text string representing an identifier for a route. This is typically made up of an Output Name or Group Name. For EQAM, QAM Group Name is used (see Table 6–13).

The characters comprising the value 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.

Table 6–13 - Route Name

Component	Route Name
Edge QAM	<QAM Group Name> (e.g., Detroit.GrossePointe.ERM1.10)

6.2.3.2 ReachableRoutes

The ReachableRoutes attribute identifies zero or more routes that have been placed in service.

6.2.3.2.1 Syntax of ReachableRoutes

The ReachableRoutes attribute encodes a number of routes (which may be zero) in its value field. Different QAM channel resources are represented using different routes. The format for individual routes is specified in Section 6.2.3.1.1.1. The ReachableRoutes attribute lists the individual routes sequentially and with no padding, as shown in Figure 6–12.

```
+--------------------+--------------------+...
| ReachableRoute1... | ReachableRoute2... |...
+--------------------+--------------------+...
```

Figure 6–12 - ReachableRoutes Format

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 2

6.2.3.2.2 Resource Selection and ReachableRoutes

An EQAM advertises its QAM channel resource(s) using one or more ReachableRoutes attributes. When a QAM channel resource becomes available for use, the ReachableRoutes attribute is used to advertise the availability of that resource. The ERM uses the resources advertised in the ReachableRoutes attributes to populate its database of resources.

6.2.3.3 NextHopServer

The NextHopServer attribute may identify the TCP port number for the next-hop signaling server. If the TCP port number is not identified, then the default port (listed in Section 6.2.1) of the signaling protocol is used.

The address identified by the NextHopServer attribute is specific to the set of destinations and application protocol identified in the ReachableRoutes attribute. This is not necessarily the address to which media (voice, video, *etc*.) will be transmitted.

A NextHopServer Attribute is mandatory if a ReachableRoutes or a WithdrawnRoutes Attribute is present.

6.2.3.3.1 NextHopServer Syntax

In order to support a variety of protocols, the identity of the next-hop server may be provided in any one of several formats (such as FQDN, IPv4, IPv6). The NextHopServer attribute includes the address domain number of the next-hop server, a Component Address field, and a Streaming Zone field.

The syntax for the NextHopServer attribute is shown in Figure 6–13.

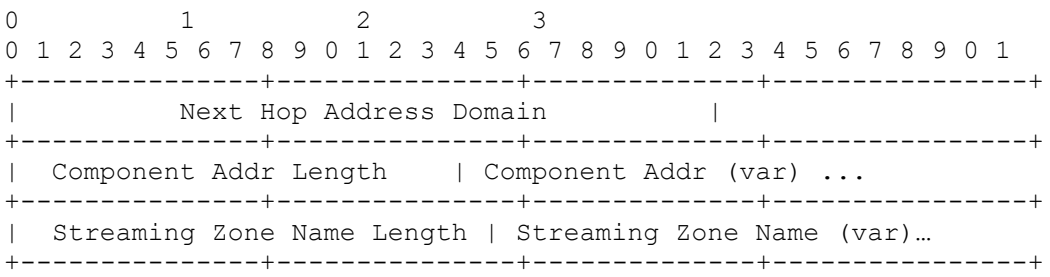


Figure 6–13 - NextHopServer Syntax

Next Hop Address Domain

The Next-Hop Address Domain field is mandatory. This 4-octet unsigned integer indicates the address domain number of the next-hop server.

Component Address

The Component Address is composed of two fields; the Component Addr Length field and the Component Addr field. Both fields are mandatory.

Component Addr Length

This is a two-octet unsigned integer containing the length of the Component Addr field, in octets.

Component Addr

This field identifies the next-hop server. This field contains a string that conforms to the following syntax:

Component Addr = host[:port]

where

host = 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.

port = numerical value (1-65535)

If the port is empty or not given, the default port (listed in Section 6.2.1) is assumed.

Streaming Zone

The Streaming Zone is composed of two fields; the Streaming Zone Name Length field and the Streaming Zone Name. The Streaming zone information is mandatory for video application, and is optional for DOCSIS application.

Streaming Zone Name Length

The length in octets of the Streaming Zone Name. The length is set to zero for DOCSIS application.

Streaming Zone Name

The ASCII string that represents the StreamingZone with which the NextHopServer is associated.

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.

6.2.3.3.1.1 NextHopServerAlternate

This attribute gives the identities of alternate hosts to which signaling messages may be sent. These hosts may be used in place of the host identified by the NextHopServer attribute.

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 243

The type of component, of the NextHopServerAlternate is identical to the NextHopServer. All values in the list must reference the same type of component. In fact they should specify different addresses for the same component as the NextHopServer.

For generality, the addresses of the next-hop servers may be of various types (domain name, IPv4, IPv6, etc). The NextHopServerAlternate attribute the number of alternate next-hop servers in this attribute, plus the length, and the next-hop name or address for each of the next-hop servers. The syntax for the NextHopServer is given in the following:

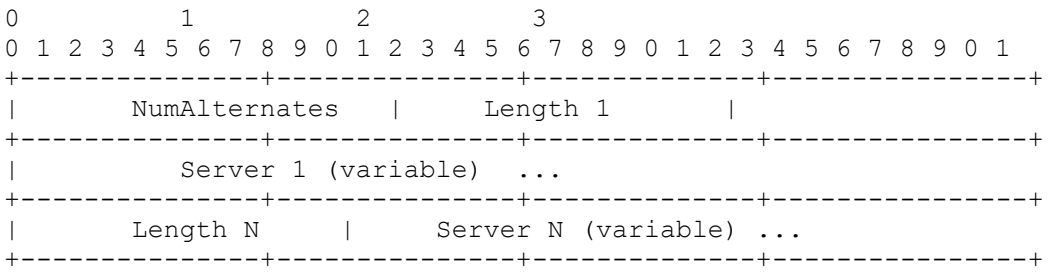


Figure 6–14 - NextHopServerAlternates Syntax

NumAlternates: This field gives the number of alternate servers identified in this attribute. In the syntax given above, NumAlternates equals N.

Length X: This field gives the number of octets in the Server X field, and the Server X field contains the name or address of the Xth next-hop server specified in this attribute.

Server X: This field is represented as a string of ASCII characters. It is defined as follows:

Server = host [":" port]

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.>

port = 1-65535

If the port is empty or not given, the default port is assumed.

6.2.3.3.2 QAM Names

This attribute specifies the QAM Name for the ReachableRoute being updated. This attribute only includes one QAM Name when an update is sent between an EQAM and an ERM.

Conditional Mandatory: true (only included in an UPDATE messages that contain a Reachable or Withdrawn Route)

Required Flags: Well-known

ERRP Type Code: 232

The QAM Name is a variable length field. When an EQAM sends a Reachable Route to an ERM, only one QAM Name value will be included in the QAM Names attribute.

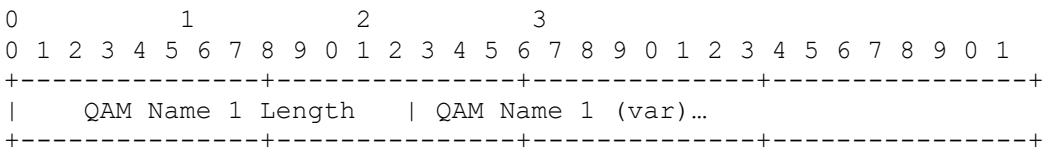


Figure 6–15 - QAM Names Attribute Syntax

QAM Name Length: This is the string length in octets of the QAM Name.

QAM Name: This is an ASCII string that represents the QAM Name.

The characters comprising the QAM Name 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.

6.2.3.3.3 Fiber Node

Fiber Node attributes specify the list of fiber node names a QAM is connected to. The Fiber Node is a variable length field.

Conditional Mandatory: true (only included in an UPDATE messages that contain a Reachable or Withdrawn Route)

Required Flags: Well-known

ERRP Type Code: 244

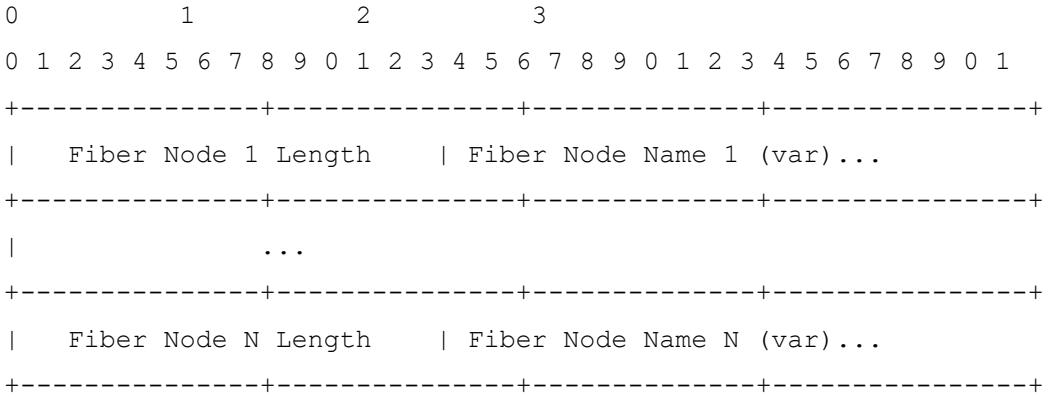


Figure 6–16 - Fiber Nodes Attribute Syntax

The characters comprising the Fiber Node string are in the set within qdtext 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.

6.2.3.4 QAM Capability

The QAM Capability attribute is used to advertise the ability of a QAM channel to support different types of operation. The QAM Capability attribute is mandatory if a ReachableRoutes attribute is present.

Conditional Mandatory: true (only included in an UPDATE messages that contain a Reachable or Withdrawn Route)

Required Flags: Well-known

ERRP Type Code: 247

The syntax for the QAM Capability attribute is as shown in Figure 6–17:

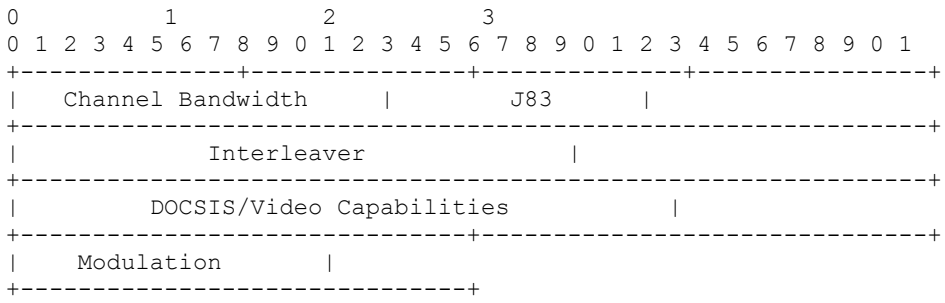


Figure 6–17 - QAM Capability Format

The fields of the QAM Capability attribute are further defined in Table 6–14 through Table 6–18. A bit value of 1 means that the corresponding capability is supported and a bit value of 0 means that the corresponding capability is not supported. Bit 0 is the most significant bit. All reserved bits are set to 0.

The first bit in each field is the Lock bit. If the Lock bit is set, the referenced parameter is not to be changed. If the Lock bit is not set, the referenced parameter may be changed. The next 7 bits comprise the TSID Group ID. If the referenced parameter is common to a group of QAM channels, the TSID Group ID is a non-zero number that identifies the QAM group in an EQAM. Otherwise, the TSID Group ID is set to 0.

Channel Bandwidth

The Channel Bandwidth field is used to describe channel bandwidth capabilities for the QAM channel.

Table 6–14 - QAM Channel Bandwidth Capability Bits

Channel Bandwidth Capability Bit	Description
0	Lock
1-7	TSID Group ID
8	6 MHz channel bandwidth
9	7 MHz channel bandwidth
10	8 MHz channel bandwidth
11-15	Reserved

J83

The J83 field is used to describe which of the operational modes, defined in [J.83], are supported by the QAM channel.

Table 6–15 - J83 Capability Bits

J.83 Capability Bit	Description
0	Lock
1-7	TSID Group ID
8	Annex A

J.83 Capability Bit	Description
9	Annex B
10	Annex C
11-15	Reserved

Interleaver

The Interleaver field is used to identify the combinations of interleaver filter tap (I) and interleaver increment (J), as defined by [J.83], that are supported by the QAM channel.

Table 6–16 - QAM Interleaver Capability Bits

Interleaver Capability Bit	Description
0	Lock
1-7	TSID Group ID
8	I=8, J=16
9	I=16, J=8
10	I=32, J=4
11	I=64, J=2
12	I=128, J=1
13	I=128, J=2
14	I=128, J=3
15	I=128, J=4
16	I=128, J=5
17	I=128, J=6
18	I=128, J=7
19	I=128, J=8
20	I=12, J=7
21-31	Reserved

DOCSIS/VIDEO Capabilities

The DOCSIS/Video Capabilities field is used to describe capabilities that are supported by the QAM channel.

Table 6–17 - DOCSIS/Video Capabilities - Bit map

DOCSIS/Video Capability Bit	Description
0	Lock
1-7	TSID Group ID
8	Mixed Video/Data
9	Mixed DOCSIS mode
10-15	Reserved
16	Video
17	DOCSIS MPT

DOCSIS/Video Capability Bit	Description
18	DOCSIS PSP
19	Mixed Static/Dynamic mode
20	Stream Redundancy
21-31	Reserved

Video mode (bit 16) is the operation mode for MPEG-2 transport video over QAM. DEPI modes, such as DOCSIS MPT and DOCSIS PSP are further defined in [DEPI]. The Mixed DOCSIS mode bit is set only when DOCSIS MPT and DOCSIS PSP data can be freely mixed in a single QAM channel. The supported combination of DOCSIS modes is determined jointly by bits 9, 17, and 18. The Mixed Video/Data bit is set only when the supported combination of DOCSIS modes can be mixed with video in the same QAM channel. Mixed Static/Dynamic mode (bit 19) is set only when static (UDP port map) sessions and dynamic (RTSP) sessions can be freely mixed in a single QAM channel.

The Stream Redundancy bit is set when an EQAM supports redundant input streams. If redundant input streams are signaled to the EQAM, the EQAM selects an input stream to deliver to the QAM output. If the selected stream fails, the EQAM switches over to the redundant stream. The input bandwidth for redundant streams is managed by the ERM.

Modulation

The Modulation field is used to describe QAM modes that are supported by the QAM channel.

Table 6–18 - Modulation Capability Bits

Modulation Capability Bit	Description
0	Lock
1-7	TSID Group ID
8	64 QAM
9	256 QAM
10-15	Reserved

6.2.3.5 Total Bandwidth

The Total Bandwidth attribute is used to define the total amount of bandwidth that the downstream resource is capable of receiving (from the network), processing, and transmitting. The Total Bandwidth attribute is mandatory if a ReachableRoutes attribute is present.

The syntax for the Total Bandwidth attribute is as in Figure 6–18.

```
0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|         Total Bandwidth Capacity (Kbps)         |
+---------------+---------------+---------------+---------------+
```

Figure 6–18 - Total Bandwidth Syntax

Conditional Mandatory: true (is mandatory for Reachable Route UPDATES)

Required Flags: Well-known

VREP Type Code: 234

Total Bandwidth Capacity

This is an unsigned 32-bit integer that contains the bandwidth of the resource in units of kilobits per second.

6.2.3.5.1 Available Bandwidth

The available bandwidth attribute is used to specify the average amount of bandwidth that the resources controlled by a resource manager have at their disposal over a given period of time. The available bandwidth is computed by calculating a running average of the difference between the total bandwidth and the amount of bandwidth that is in use over the averaging period.

This attribute may be updated each time there is a significant change in average available bandwidth over a minimum averaging period or when available bandwidth drops below a critical threshold. The intent of this attribute is to reflect changes in resource usage over periods long enough to reflect the behavior of a population of subscribers such as busy hour periods. Because of this, the typical averaging / transmission period for this attribute would be approximately 30 minutes. It has the same format as the total bandwidth attribute.

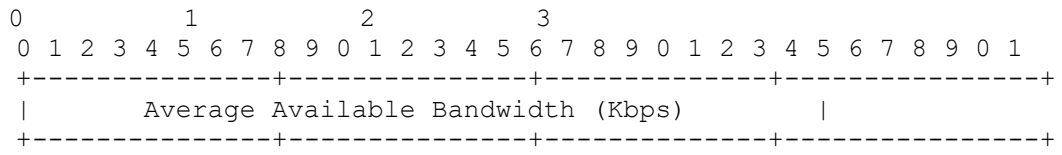


Figure 6–19 - Available Bandwidth Attribute Syntax

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 235

6.2.3.6 QAM Channel Configuration

The QAM Channel Configuration attribute is used by EQAMs to inform the ERM of the modulation frequency, modulation mode, TSID, ITU-T J.83 operation mode, channel bandwidth, and interleaver parameters configured for an advertised QAM channel. The modulation mode may be required if a set-top can only support certain QAM modulations. The ERM uses information on QAM modulation frequencies as part of its resource allocation algorithm for EQAMs.

The QAM Channel Configuration attribute is mandatory if a ReachableRoutes attribute is present.

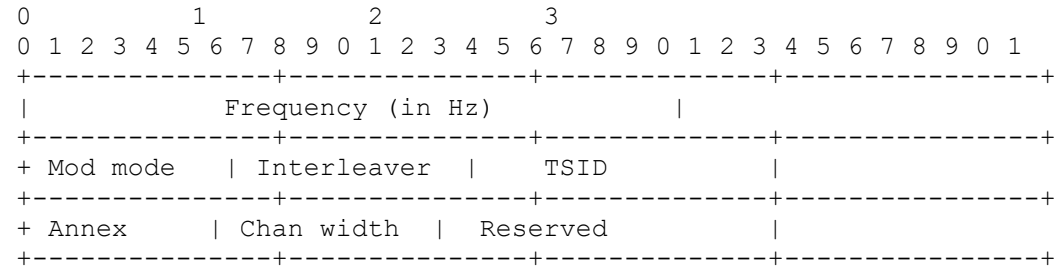


Figure 6–20 - QAM Configuration Attribute

Conditional Mandatory: true (is mandatory if Reachable Route is a QAM)

Required Flags: Well-known

ERRP Type Code: 238

Frequency

The Frequency field is a 32-bit unsigned integer containing the carrier frequency of the QAM channel, in hertz The Frequency field is mandatory.

Modulation Mode

The Modulation field is an 8-bit field that encodes the modulation type of the QAM channel. The Modulation Mode field is mandatory. The Modulation field contains a value from Table 6–19 (synchronized with DOCSIS OSSI document).

Table 6–19 - QAM Types

Modulation Mode Value	Description
3	64-QAM
4	256-QAM
5	128-QAM
6	512-QAM
7	1024-QAM

Interleaver

The interleaver value is one of the pairs as defined in Table 6–16. The I and J fields are mandatory. This is based on the enumeration constant that describes the interleaver (synchronized with DOCSIS OSSI document).

Table 6–20 - Interleaver Settings

Interleaver	Description
3	I=8, J=16
4	I=16, J=8
5	I=32, J=4
6	I=64, J=2
7	I=128, J=1
8	I=12, J=7
9	I=128, J=2
10	I=128, J=3
11	I=128, J=4
12	I=128, J=5
13	I=128, J=6
14	I=128, J=7
15	I=128, J=8

TSID

The TSID field is a 16-bit field that represents the TSID carried in the PAT of the QAM. This field is mandatory.

Annex

The Annex field is a 8-bit field that encodes the J83 mode (see [J.83]) that the QAM channel is using. The Annex field is mandatory. The Annex field contains a value from Table 6–21 (synchronized with DOCSIS OSSI document).

Table 6–21 - QAM Annex Modes

J83	Description
0	Annex A
1	Annex B
2	Annex C

Chan Width

The Chan Width field is a 8-bit field that encodes the channel bandwidth supported by the QAM channel. The Chan Width field is mandatory. The Chan Width field contains a value from Table 6–22 (synchronized with DOCSIS OSSI document).

Table 6–22 - Channel Bandwidth Types

Channel Bandwidth	Description
0	6 MHz channel
2	7 MHz channel
1	8 MHz channel

Reserved

This field should be set to 0.

6.2.3.7 Port ID

The Port ID attribute identifies the RF port to which the QAM channel is attached. The Port ID attribute is mandatory if a ReachableRoutes attribute is present. The port ID is unique only with in the EQAM.

Conditional Mandatory: True (required if UPDATE contains Reachable Routes)

Required Flags: Well-known

ERRP Type Code: 244

The Port ID attribute is a 32-bit value that has the format in Figure 6–21:

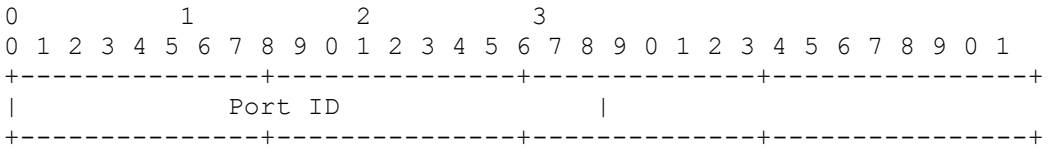


Figure 6–21 - Port ID Format

6.2.3.8 Service Status

The Service Status attribute identifies the operational status of a QAM channel. The Service Status attribute is mandatory if a ReachableRoutes attribute is present. It can also identify the operational state of a device such as an

EQAM if sent without a route. This attribute does not modify a Route and can be sent in an UPDATE message (that contains routes) or by itself.

Conditional Mandatory: False

Required Flags: Well-known

ERRP Type Code: 241

The syntax for the Service Status attribute is as shown in Figure 6–22.

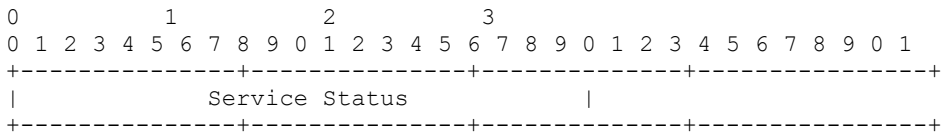


Figure 6–22 - Service Status Format

The Service Status field contains a 32-bit unsigned integer that encodes the status of a QAM channel or an EQAM. The value of the Service Status field is taken from Table 6–23:

Table 6–23 - Service Status Values

Value	Meaning of Service Status
1	Operational – The component is operational. The resource manager is permitted to allocate resources from this component.
2	Shutting Down – The component is being shut down properly. The ERM SHOULD NOT allocate resources from this component for new sessions. When all sessions on this component have been torn down, the EQAM MUST advertise that its resources have been withdrawn from service (by using the ERRP Withdrawn-Routes attribute).
3	Standby – This is a redundant component that may be used by the resource manager to replace a failed component. Details of the operation of components during failover are not specified in this document.

6.2.3.9 CAS Capability

This attribute will be used by encryption devices to specify the type of conditional access that is supported if the Video EQAM contains an embedded encryptor.

Conditional Mandatory: True (If a Video EQAM contains an embedded encryptor, this attribute is sent to the ERM.)

Required Flags: Well-known

ERRP Type Code: 233

The CAS Capability attribute is used to advertise the potential CAS encoding methods that an Encryption Engine is capable of supporting, as well as its identifier. The CAS attribute includes a multi-field parameter that specifies an Encryption Type, an Encryption Scheme and Key Length. The Encryption Type generally describes whether an embedded encryptor is present and if so, whether it supports per session provisioning. The encryption Scheme describes the encryption algorithm supported by the encryptor. The key length field contains the length in bits of the encryption key. Vendor specific keywords may be used in addition to the keywords specified in the table. The fourth field is the 2-byte CAS identifier. The CAS Capability attribute is relevant to both external Encryption Engines and EQAMs that have integrated CAS Scrambling capabilities. For EQAM that does not have an embedded encryptor, this attribute is not expected to be reported.

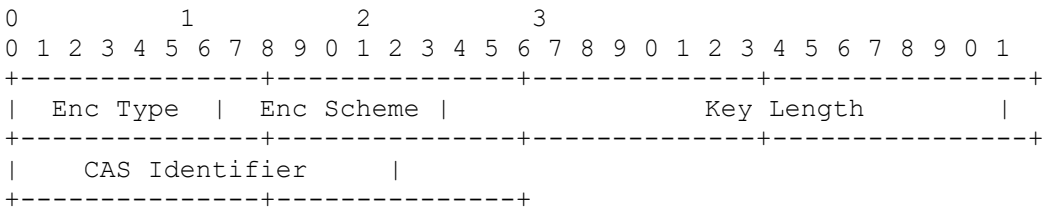


Figure 6–23 - CAS Capability Attribute Syntax

Table 6–24 - Potential values of the Encryption Type Parameter

Code	Keyword	Description
0	NONE	No encryption – content in the clear
1	NON-SESSION	Tier-based or fixed key Real Time encryption
2	SESSION	Session -based Real Time encryption

Table 6–25 - Potential values of the Encryption Scheme Parameter

Code	Keyword	Description
0	DES	DES encryption
1	3DES	Triple DES encryption
2	AES	AES encryption
3	DVB-CSA	DVB-CSA encryption
4	PKEY	SA PowerKEY Encryption
5	MEDIAC	Motorola MediaCipher Encryption
6	DVS042	SCTE DVS-042 Encryption

Some encryption engines will support more than one type of encryption at the same time. The embedded encryptor within an EQAM that supports multiple encryption types will use multiple UPDATE messages to register these capabilities. Each UPDATE message will carry one CAS Capability attribute for a single encryption type.

6.2.3.10 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.

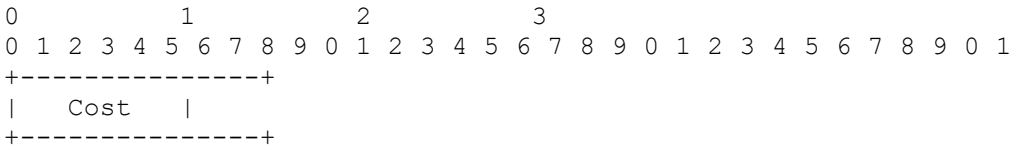


Figure 6–24 - Cost Attribute Syntax

6.2.3.11 Edge Input

The Edge Input attribute does not describe a reachable route, but instead describes the data input interfaces of the EQAM device 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.

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:

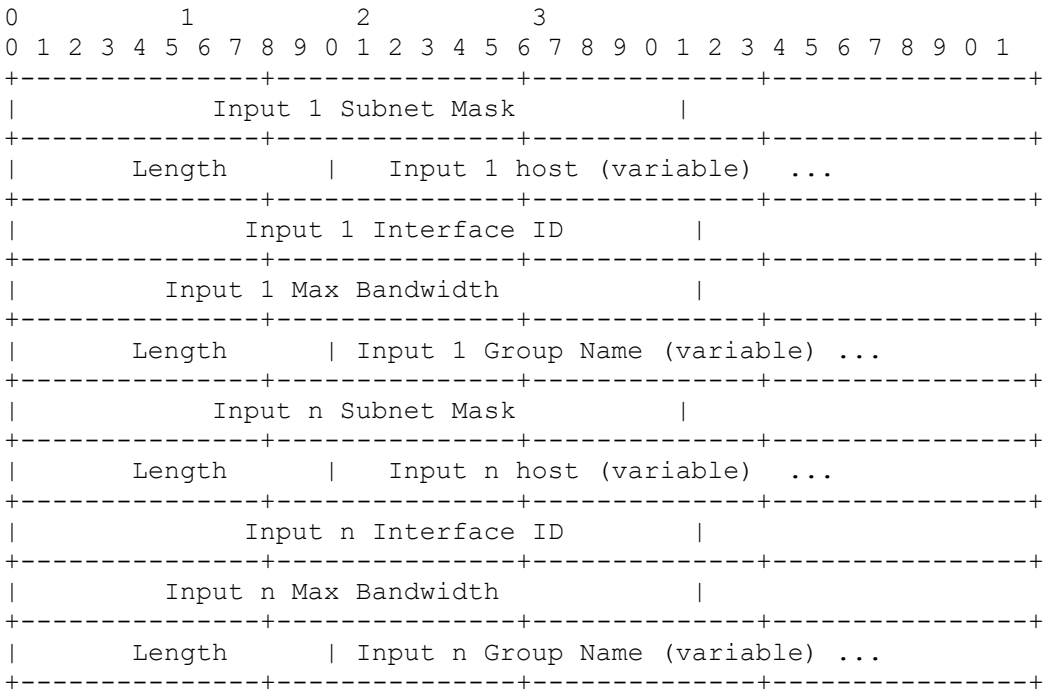


Figure 6–25 - Edge Input Attribute Syntax

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.

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 physical interface on the chassis, this input represents.

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. 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.

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.

6.2.3.12 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 6–26.

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

Required Flags: Well-known

ERRP Type Code: 249

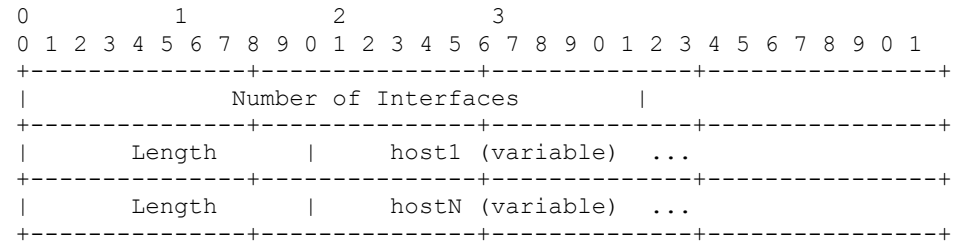


Figure 6–26 - Input Map Attribute

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.

6.2.3.13 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.

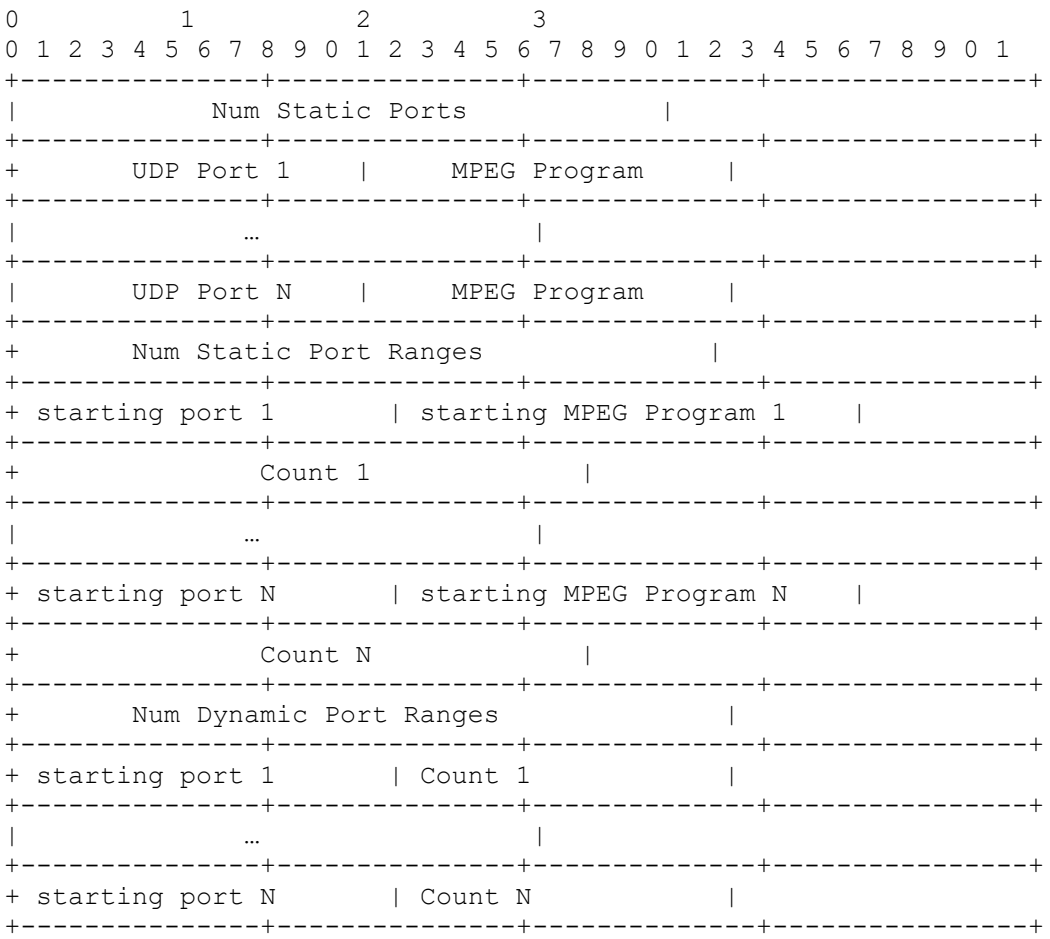


Figure 6–27 - UDP Map Attribute Syntax

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.

6.2.3.14 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.

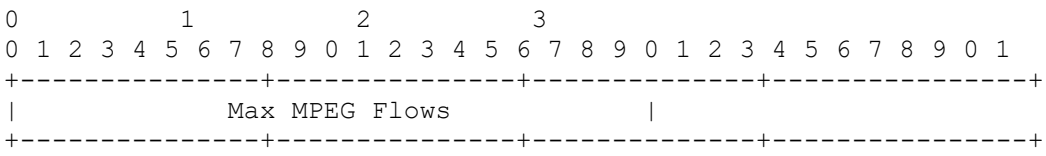


Figure 6–28 - Max MPEG Flows Attribute Syntax

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 contain 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 6–26 - 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 6–27 - 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

Event Name	Brief Event Description
{ConnectRetry timer expired}	The ConnectRetry timer has fired
{Hold timer expired}	The Hold timer has fired
{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 6–28 through Table 6–33. 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 6–28 - 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 a ERRP Node detects an error when in some other state, it closes the TCP connection and changes its state to [Idle]. As Table 6–29 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 a 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 6–29 - 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 6–30 - 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 6–31 - 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 6–32 - 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]
[OpenConfirm]	{Receive UPDATE message}	[Idle]
[OpenConfirm]	{Receive NOTIFICATION message}	[Idle]

In this state, a 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 6–33 - 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]

Initial State	Event	Final State
[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.

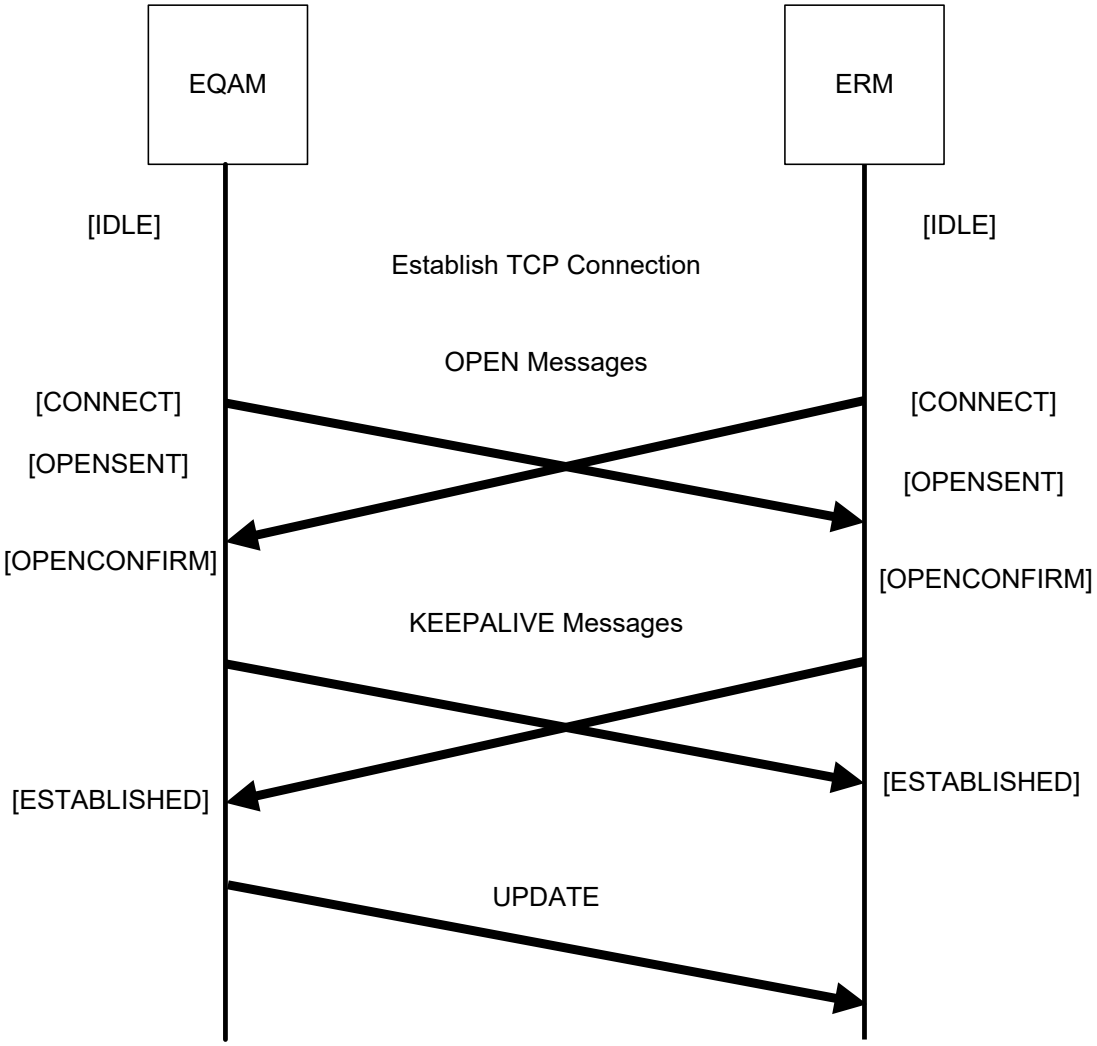


Figure 6–29 - Example ERRP Connection Establishment

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 6–34.

Table 6–34 - 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	

6.3.2 KEEPALIVE message

When each device receives the OPEN, it responds with a KEEPALIVE.

The KEEPALIVE message comprises simply a ERRP header as shown in Table 6–35.

Table 6–35 - Example KEEPALIVE Message

Field	Field Length	Field Value	Comment
Length	2	3	Total length of this message
Type	1	4	KEEPALIVE

6.3.3 UPDATE message

After the ERRP connection has been established, the EQAM reports its available resources to the ERM. To advertise available resources, the EQAM sends a separate UPDATE message for each QAM channel. After the initial UPDATE for a QAM channel, the EQAM sends further UPDATEs for that QAM channel only when the service status or configuration of that QAM channel changes.

In an UPDATE message, either the ReachableRoutes or the WithdrawnRoutes attribute is present. The ReachableRoutes attribute is used to advertise an operational QAM channel; the WithdrawnRoutes attribute is used to remove a QAM channel from service.

An example UPDATE message is shown in Table 6–36.

Table 6–36 - Example UPDATE Message

	Field	Field Length	Field Value	Comment
Header	Length	2	135	Total length of this message
	Type	1	2	UPDATE
UPDATE	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	2	ReachableRoutes
	Attr Length	2	29	Length of the attribute
	Address Family	2	32769	RTSP URL
	Application Protocol	2	32768	ERMI
	Length	2	23	Length of address
	Address	23	rtsp://192.0.2.2/	URL of the QAM channel
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	3	NextHopServer
	Attr Length	2	17	Length of the attribute
	Next Hop Address Domain	4	1	
	Length	2	11	Length of address
	Server	11	192.0.2.2	IPv4 address
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	234	Total Bandwidth

	Field	Field Length	Field Value	Comment
	Attr Length	2	4	Length of the attribute
	Total Bandwidth	8	388000	Bandwidth
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	244	Output/Port ID
	Attr Length	2	4	Length of the attribute
	Port ID	2	0x1234	RF port ID = 0x1234
	Reserved	1	0	
	Channel ID	1	0	Channel ID = 0
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	241	Service status
	Attr Length	2	4	Length of the attribute
	Service Status	4	1	Operational
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	232	QAM Name
	Attr Length	2	12	Length of the attribute
	QAM Name1 Length	2	10	
	QAM Name 1	10	Division.1	
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	247	QAM Capability
	Attr Length	2	14	Length of the attribute
	Channel BW	2	0x0080	Lock=0, Group ID = 0, 6MHz only
	J83	2	0x00C0	Lock=0, Group ID = 0, Annex A and B
	Interleaver	4	0x00F00000	Lock=0, Group ID=0, I=8, J=16, I=16, J=8, I=32, J=4, I=64, J=2
	DOCSIS/Video Capabilities	4	0x00406000	Lock=0, Group ID=0, Mixed DOCSIS mode=1, DOCSIS MPT, DOCSIS PSP

	Field	Field Length	Field Value	Comment
	Modulation	2	0x00C0	Lock=0, Group ID=0, 64QAM, 256QAM
	Attr Flags	1	0x00	Well-known
	Attr Type Code	1	238	QAM channel configuration
	Attr Length	2	12	Length of the attribute
	Frequency	4	550000000	550 MHz
	Modulation Mode	1	4	256QAM
	Interleaver	1	3	I=8, J=16
	TSID	2	6677	TSID
	Annex	1	1	Annex B
	Channel Width	1	0	6Mhz
	Reserved	2	0	

6.3.4 NOTIFICATION message

The EQAM should send a NOTIFICATION to the ERM when it shuts down.

An example NOTIFICATION message is shown in Table 6–37.

Table 6–37 - Example NOTIFICATION Message

	Field	Field Length	Field Value	Comment
Header	Length	2	5	Total length of this message
	Type	1	3	NOTIFICATION
NOTIFICATION	Error Code	1	6	Cease
	Error Subcode	1	0	

7 RESOURCE CONFIGURATION AND PROVISIONING

RTSP [RFC 2326] is an application level protocol to control the delivery of data with real-time properties. In this specification, RTSP is used by the M-CMTS Core, the ERM and EQAM elements to request and allocate QAM channel resources.

The terminology, message syntax, method definitions, and state machine definitions of RTSP are used in accordance with RTSP [RFC 2326] and HTTP [RFC 2068].

While it is recommended for implementations compliant with this specification to comply with [RFC 2326], only a subset of all the methods defined in [RFC 2326] is required by this specification.

In addition to the standard header definition specified in [RFC 2326], RTSP extensions specified in this document MUST be supported by an RTSP Server in order to claim compliance with this specification. In addition to the standard header definition specified in [RFC 2326], RTSP extensions specified in this document MUST be supported by an RTSP Client in order to claim compliance with this specification.

RTSP is a text-based protocol. The definitions in this specification are case-sensitive unless otherwise noted. RTSP methods and headers are terminated by a carriage return and a linefeed. White space can be inserted between tokens and between tokens and delimiters. RTSP messages conform to the syntax defined in [RFC 2068], section 4. In particular, the header fields are separated from the optional message-body by a blank line; i.e., a line with nothing preceding the CRLF.

The RTSP Server MUST use TCP as the transport protocol. An RTSP Server SHOULD listen on TCP port 554 for incoming RTSP connections. The RTSP Client MUST use TCP as the transport protocol. The TCP connection SHOULD be initiated by the RTSP Client. The RTSP Client SHOULD use a persistent TCP connection as defined in section 9 of [RFC 2326].

An RTSP Server MUST support request pipelining as specified in section 9.1 of [RFC 2326]. A RTSP Client MAY send multiple requests without waiting for each response. An RSTP Server MUST send its response (for each session) to those requests in the same order that the requests were received for the session.

Figure 7–1 shows the RTSP Client – Server relationships among the M-CMTS, EQAM, ERM and Video Session Managers.

The M-CMTS Core MUST be capable of acting as an RTSP Client.

The ERM MUST be capable of acting as both an RTSP Client and an RTSP Server.

The EQAM MUST be capable of acting as an RTSP Server.

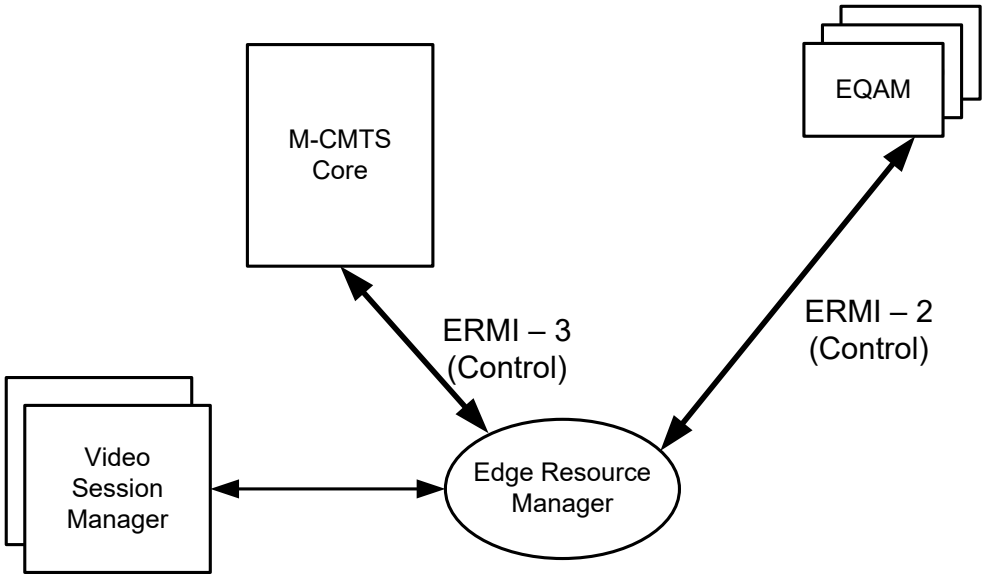


Figure 7–1 - RTSP Client - RTSP Server Relationships

7.1 TCP Connection Behavior for RTSP

This section defines the required behavior of components with respect to TCP connections.

7.1.1 Establishing the TCP socket

The ERMI RTSP Client SHOULD establish the TCP socket and re-establish it if the socket is disconnected. The RTSP Client and RTSP Server should use that socket for messages regarding multiple sessions with no need to establish a separate socket per session. RTSP Clients SHOULD implement a backoff algorithm in case a TCP reconnect attempt fails. While a specific algorithm is not mandated in this document, an initial delay of one second with a randomized exponential increase is recommended.

7.1.2 Connection timeout

The ERMI RTSP Client and RTSP Server maintain the TCP connection persistently. In order to maintain the connection and to report connection issues in a timely manner, the RTSP Server keeps track of the time of last communication with the RTSP Client. The RTSP Server will start a timer that expires at <timeout> seconds from the last communications time. Messages received from the RTSP Client will update the last communications time with that RTSP Client, effectively restarting the connection timer. If the connection timer expires, the RTSP Server MUST close the TCP connection.

Once the TCP connection is established, an RTSP Client MUST send a GET_PARAMETER request with the clab-connection-timeout parameter to understand the connection timeout value. See Section 7.8.2.2.

The RTSP Client SHOULD send an RTSP SET_PARAMETER request as a keepalive message to the RTSP Server if <timeout>/3 seconds have elapsed since the last time it sent a message.

This keepalive and timeout mechanism is independent from the session timeout and keepalive mechanism defined in [RFC 2326] (and described in more detail in Section 7.9). This keepalive and timeout serve solely as an application-layer TCP connection timeout and keepalive, and have no direct effect on the state of individual RTSP sessions.

7.2 RTSP URL

M-CMTS Core, the ERM, and the EQAM MUST use the format of Section 7.2.

Each RTSP request requires a URL. The URL has the format:

rtsp://host[:port]/

The port field is included if its value is other than 554.

For requests passed onto the resource allocation interface ERMI-3:

- the host field should contain the IP address or host name of the ERM;

For requests passed onto the resource allocation interface ERMI-2:

- the host field should contain the IP address or host name of the EQAM.

The ERM SHOULD use the URL when sending an RTSP request to the EQAM over ERMI-2. For example, the ERM could use the URL "rtsp://192.0.2.3/", where 192.0.2.3 is the EQAM's signaling IP address.

7.3 RTSP Methods

The RTSP Server and RTSP Client MUST support the following RTSP methods:

- SETUP
- TEARDOWN
- SET PARAMETER
- GET PARAMETER
- ANNOUNCE

The SETUP method is used to initiate sessions. This method includes headers to specify the resource requested for the session. A SETUP request is sent from an RTSP Client to an RTSP Server. If the resource allocation is successful, the RTSP Server MUST send a response containing a session identifier and Transport headers (see Section 7.7.4) that identify the allocated resource. An EQAM MUST respond to a session setup request within 0.5 seconds. RTSP uses a Session header (see section 12.37 of [RFC 2326]) to identify an RTSP session uniquely within an RTSP Server. If the RTSP Server fails to allocate the requested resources, the RTSP Server MUST send a SETUP response with an appropriate error code.

The TEARDOWN method is used to request session teardown. The session identifier is given in the TEARDOWN request. The TEARDOWN request is sent from an RTSP Client to an RTSP Server. Upon receipt of a TEARDOWN request, the RTSP Server MUST release all the resources associated with the identified RTSP session and send an appropriate response to the RTSP Client.

The GET_PARAMETER method is used by an RTSP Client to get a parameter value from an RTSP Server. Upon receipt of a GET_PARAMETER method, the RTSP Server MUST find the parameter value and send the parameter value in the response. This specification extended the GET_PARAMETER method by defining new parameters such as clab-session-list and clab-connection-timeout. These parameters are specified in Section 7.8.2.

The SET_PARAMETER method is used by an RTSP Client to set a parameter value in an RTSP Server. A request should only contain a single parameter to allow the RTSP Client to determine why a particular request failed. If the request contains several parameters, the RTSP Server MUST only act on the request if all of the parameters can be set successfully. An RTSP Server MUST allow a parameter to be set repeatedly to the same value. The RTSP Server MAY disallow changing parameter values. A SET_PARAMETER message sent with no body is used as a keepalive mechanism, described in Section 7.9.

The ANNOUNCE method is used by an RTSP Server to transfer information about the status of a session to an RTSP Client in real time. An ANNOUNCE request is sent from RTSP Server to RTSP Client on an open TCP connection. An ANNOUNCE request includes a clab-Notice header per Section 7.7.3.2.

7.4 RTSP Finite State Machine (FSM)

Since support for only a limited set of methods is required by this specification, the RTSP Client and RTSP Server state machines are simple. In general, RTSP is used to create a session to which resources are allocated for streams to flow and be controlled. In the context of this specification, RTSP is being used to create sessions to allocate resources for intermediate devices over which streams will be carried. The streams are controlled by the application server and application client using the intermediate devices. Since no stream control is required for the intermediate devices, the PLAY method is not relevant.

This section specifies the required RTSP operation in terms of a Finite State Machine (FSM). Implementations of RTSP Clients and RTSP Servers that conform to this specification MUST operate in accordance with the FSM described in this section. An RTSP Server MUST implement an independent FSM for every RTSP session. An RTSP Client MUST implement an independent FSM for every RTSP session.

7.4.1 RTSP Server Finite State Machine

The RTSP Server FSM MUST conform to the states, events, and diagrams of Section 7.4.1.

The RTSP Server FSM contains two states: [Init] and [Ready]. See Table 7–1 for further details.

Table 7–1 - RTSP Server FSM States

State Name	Brief State Description
[Init]	Initial state for an RTSP session
[Ready]	The RTSP session is active

The RTSP Server FSM contains two events:

Table 7–2 - RTSP Server FSM Events

Event Name	Brief Event Description
{SETUP}	SETUP request received and processed successfully
{TEARDOWN}	TEARDOWN request received and processed successfully

State transitions in the FSM conform to Table 7–3.

The FSM begins in the [Init] state.

Table 7–3 - RTSP Server State Machine

Initial State	Event	Final State
[Init]	{SETUP}	[Ready]
[Ready]	{TEARDOWN}	[Init]

When the RTSP Server is in the [Init] state, if a SETUP request received by the RTSP Server is processed successfully, the RTSP Server enters the [Ready] state. Otherwise, the RTSP Server remains in the [Init] state.

When the RTSP Server is in the [Ready] state, if a TEARDOWN request received by the RTSP Server is processed successfully, the RTSP Server enters the [Init] state.

When the RTSP Server is in the [Ready] state, it may enter the [Init] state after detecting a session timeout. Note that releasing resources based on a session timeout may not be desirable. An alternate method to resynchronize is found in Section 7.11.4.

7.4.2 RTSP Client State Machine

The RTSP Client FSM MUST conform to the states, events, and diagrams of Section 7.4.2.

The RTSP Client FSM contains two states:

Table 7–4 - RTSP Client FSM States

State Name	Brief State Description
[Idle]	Initial state for an RTSP session
[Ready]	RTSP session is active

The RTSP Client FSM contains two events:

Table 7–5 - RTSP Client FSM Events

Event Name	Brief Event Description
{SETUP}	SETUP request sent and successful response (response code is 200) received
{TEARDOWN}	TEARDOWN request sent and successful response (response code is 200) received (or response timeout)

State transitions in the FSM conform to Table 7–6.

The FSM begins in the [init] state.

Table 7–6 - RTSP Client State Machine

Initial State	Event	Final State
[Init]	{SETUP}	[Ready]
[Ready]	{TEARDOWN}	[Init]

When the RTSP Client is in the [Init] state, if instructed to establish a session, it will send out SETUP request with a Transport header that describes the resource requested. After sending a SETUP request and receiving a successful SETUP response with response code of 200, the RTSP Client enters the [Ready] state. Otherwise, it remains in the [Init] state.

When the RTSP Client is in the [Ready] state, after sending a TEARDOWN request and receiving a successful TEARDOWN response with response code of 200, the RTSP Client enters the [Init] state. If the response times out, the RTSP Client should release the resources prior to entering the [Init] state.

7.5 Session Identifiers

RTSP defines a Session header that identifies a specific RTSP session. Within the video network architecture, a video session can be established using multiple RTSP connections, for example: one between the set-top box and the video session manager, one between the video session manager and the ERM, and one between the ERM and the EQAM. A new RTSP header is therefore defined to identify a specific video session across the relevant RTSP connections.

The video session identifiers are intended to be implemented as follows:

- The RTSP Client receives a ClientSessionId from the video session manager and inserts the clab-ClientSessionId header in the SETUP request, per Section 7.7.3.1.
- The video session manager typically uses the ClientSessionId for every communication that relates to this session, therefore allowing the RTSP session to be associated with the relevant session instances in other components of the video network architecture.
- Each individual RTSP Server (e.g., EQAM) creates its own specific RTSP session ID for its session per [RFC 2326].
- In interactions relating to an existing session (e.g., TEARDOWN request) both the RTSP session ID and the clab-ClientSessionId are sent.

7.6 RTSP Headers

RTSP headers follow the RTSP base syntax specified in section 15.1 of RTSP [RFC 2326]. In [RFC 2326], each header concludes with a carriage return and a line feed. The headers are separated from a (possibly empty) body with a carriage return and a linefeed.

The M-CMTS Core, the ERM, and the EQAM MUST support the requirements and formats defined in Sections 7.6 to 7.9 inclusive.

Table 7–7 - Supported RTSP Headers

Header	Direction	RTSP Method
CSeq	Request & Response	SETUP, TEARDOWN, ANNOUNCE, GET_PARAMETER, SET_PARAMETER
Session	Request & Response	SETUP(response), TEARDOWN, ANNOUNCE
Require	Request	SETUP, TEARDOWN, ANNOUNCE, GET_PARAMETER, SET_PARAMETER
Transport	Request & Response	SETUP ANNOUNCE(response)
Content-Type	Request & Response	SETUP GET_PARAMETER SET_PARAMETER
Content-Length	Request & Response	SETUP GET_PARAMETER SET_PARAMETER

The value of the CSeq header is an RTSP request sequence number. Use of this header conforms to the requirements in section 12.17 of [RFC 2326].

The value of the Session header is an RTSP session identifier. Use of this header conforms to the requirements in section 12.37 of [RFC 2326].

The Require header is used to ensure that all the options passed in the RTSP message are understood by both RTSP Client and RTSP Server. The Require header includes an option tag that represents the feature set supported. In ERMI, a new option tag "com.cablelabs.ermi" is used to represent the requirements specified in this document. Use of this header conforms to the requirements in section 12.32 of [RFC 2326]. An RTSP Server MUST include a Require header with the option tag value "com.cablelabs.ermi" in an RTSP request. An RTSP Client MUST include a Require header with the option tag value "com.cablelabs.ermi" in an RTSP request.

Three Transport header formats are specified in 7.7.4: clab-DOCSIS/QAM (for DOCSIS messages), clab-MP2T/DVBC/QAM and clab-MP2T/DVBC/UDP (for Video messages).

The Content-Type header is defined in section 12.16 of [RFC 2326]. The Content-Type header MUST be included with an entity body. This specification covers two content types for entity bodies:

- text/parameters

  A Content-Type header must be included in both the request and response of a GET_PARAMETER and SET_PARAMETER method when dealing with parameter values and the Content-Type header MUST indicate: text/parameters.

  Note that the SET_PARAMETER method is used without any entity body when used as a keepalive mechanism.

- text/xml

  A Content-Type header indicating text/xml MUST be included in a SETUP message with an XML entity body as defined in Section 7.8.1.

The Content-Length header is defined in section 12.14 of [RFC 2326]. Per [RFC 2326], the Content-Length header is required to be included in all messages that carry an entity body. The Content-Length header indicates the size of the message-body not including any headers, in decimal number of octets, sent to the recipient. The format of Content-Length is:

Content-Length: <digits>
where <digits> is the ASCII representation of decimal number.

7.7 RTSP Extensions

This section specifies the extensions to [RFC 2326] needed to support both data and video EQAM profiles.

7.7.1 Data Representation

The data format for the following fields used in RTSP messages is as follows:

- Client IDs – where the client ID is a MAC address, it will be represented as a 12-digit hex ASCII value, no "0x" prefix, no ":", and with leading zeros. Where there is no valid client ID available, the value "FFFFFFFFFFFF" is used.
- Bandwidth – decimal ASCII representation of an integer in bits/second.
- RTSP session IDs – decimal ASCII representation of an integer.
- IPv4 addresses are in the form n.n.n.n, where n is a decimal ASCII representation of an integer.
- IPv6 addresses use the textual representation defined in section 2.2 of [RFC 4291] and enclosed in "[" and "]" characters.
- Ports are decimal ASCII representation of an integer.

7.7.2 Base RTSP Syntax

Unless otherwise specified, the syntax for the base types is defined in [RFC 2326], section 15.1, titled Base Syntax, describes the syntax in this section. Further definitions and syntax are provided by [RFC 2068]. This syntax expression language is known as an augmented Backus-Naur Form (aBNF).

7.7.3 RTSP Header Extensions

The table below describes additional requirements beyond [RFC 2326] for the inclusion of headers within RTSP messages.

Type "Request/Response" designates general request headers to be found in both requests and responses. Type "Request" designates request headers, and type "Response" designates response headers. Fields marked with "Required" in the column labeled "support" are implemented by the device for a particular method. Note that not all fields marked "Required" will be sent in every request of this type. The word "Required" means that the RTSP Client MUST implement the fields for response headers. The word "Required" also means that the RTSP Server supporting the indicated profile MUST implement the fields for request headers. The last column of Table 7–8 lists the method(s) for which the header field is meaningful.

Table 7–8 - RTSP Header Extensions

Header	Type	Support	Methods
clab-ClientSessionId	Request/Response	Required: Video EQAM	SETUP, TEARDOWN, ANNOUNCE
clab-Notice	Request	Required: Video EQAM	ANNOUNCE
clab-Reason	Request	Required: Video EQAM	TEARDOWN
clab-SessionGroup	Request	Optional : Video EQAM	SETUP, GET_PARAMETER, SET_PARAMETER
clab-Priority	Request	Optional: Video EQAM	SETUP
clab-SetupType	Request	Optional: Video EQAM	SETUP
clab-PidRemap	Request	Optional: Video EQAM	SETUP
clab-MPTSMode	Request	Required: Video EQAM	SETUP
clab-StatmuxGroup	Request	Optional: Video EQAM	SETUP

7.7.3.1 Extension: clab-ClientSessionId

The ClientSessionId extension header defines a unique session identifier that is provided to the RTSP Client by a mechanism that is out of scope for this document. ClientSessionId is used in the session setup message as the RTSP Client's identifier of the video session. It is analogous to the DSM-CC SessionID field, and is typically generated by the Set-top Box.

The syntax of a clab-ClientSessionId is a 20 character ASCII representation of a 10-byte hexadecimal value. The most significant 6 bytes are the Client-ID, the least significant 4 bytes are a Session-ID unique to the RTSP Client. The combination of the two provides a globally unique identifier.

clab-ClientSessionId = "clab-ClientSessionId" ":" "Client-ID" "Session-ID"

Client-ID = 12HEX

Session-ID = 8HEX

Example for an RTSP Client with a MAC address of 00:AF:12:34:56:DE and a Session-ID of 00000001:

clab-ClientSessionId:00AF123456DE00000001

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
5401	Downstream Failure	For usage, Section 7.11.5.3 Also, this can be used in the case of QAM failure
5200	Server Resource Unavailable	Send when the multicast or unicast input stream is lost
5404	Unable to Join	All multicast transports were attempted.
5405	Input Interface Failure	Signaled when a stream fails due to input interface failure
5406	Failover to Redundant Source	Sent when multicast redundancy is in use, and the EQAM is switching to a backup source. See below for detailed requirements.
5502	Internal Server Error	Sent when no other Code applies.
5602	Bandwidth Exceeded Limit	Sent when the data plane exceeds the limit
5700	Session in Progress	For usage, see Section 7.9.1.2
5701	Reclaim Session	Sent by the ERM to the M-CMTS Core to reclaim a QAM channel resource that was previously allocated. See below for detailed requirements.

Notes about Notice codes

- Reclaim Session - When an ERM requests an M-CMTS Core to release a QAM channel resource previously allocated by the M-CMTS Core, the ERM MUST send an RTSP ANNOUNCE request to the M-CMTS Core with a notice-code of 5701. After the M-CMTS Core receives the ANNOUNCE request with a notice-code of 5701, if the M-CMTS Core intends to release the QAM channel resource by sending a TEARDOWN request to the ERM, the M-CMTS Core SHOULD send an ANNOUNCE response with a response code of 200 OK. If the M-CMTS Core does not intend to release the QAM channel resource at the current time, the M-CMTS Core SHOULD send the ANNOUNCE response with a response code of 503 Service Unavailable.
- Joining a stream - The Video EQAM MUST send an ANNOUNCE message with a notice-code of 2104 (including a MulticastTransport header that specifies the source address along with the other multicast information) when it begins outputting the first MPEG packet that carries the requested SDV channel. The Video EQAM MUST repeat this ANNOUNCE message after any reconnection event with an ERM that subsequently issues a GET_PARAMETER request per Sections 7.8.2.1 and 7.8.2.3. The Video EQAM MUST return a SETUP response via ERMI as soon as it has allocated resources for the stream.
- Fail Over to Redundant video source - The Video EQAM MUST send an RTSP ANNOUNCE message to the ERM when it detects a loss of input of the video stream from the current multicast source and it fails over to a redundant multicast source. The Video EQAM MUST use Notice code 5406 "Switch over to redundant multicast source" and include a MulticastTransport header with the ANNOUNCE, a header that specifies the new source address along with the other multicast information.
- Video Stream Loss - The Video EQAM MUST send an RTSP ANNOUNCE message to the ERM when a video stream is lost and the Edge QAM cannot fail over to an alternate stream source. The Video EQAM MUST use Notice code 5200 "Server Resource Unavailable" and may include a MulticastTransport header.
- If the VideoEQAM cannot find a PAT or PMT in an input stream, it MUST send an RTSP ANNOUNCE message to the ERM using Notice code 4401.
- If the Video EQAM finds a program number conflict, it MUST send an RTSP ANNOUNCE message to the ERM using Notice code 4402. A particular example is when a pre-conditioned "ad set" MPTS is being multiplexed with some existing SPTS sessions. The pre-conditioned MPTS will be transported by the EQAM with no program number remapping. When the session SETUP completes, the EQAM will not know what program numbers the MPTS contains. It is only when the PAT arrives for the Input TS that the EQAM will be able to detect that there is a program number conflict with an existing session.

7.7.3.3 Extension: clab-Reason

The Reason extension header defines the reason for a particular TEARDOWN message. The Teardown-Reason-Code is an integer representation of the reason for the TEARDOWN message. The Reason-Phrase is intended to give a short textual description of the Reason-Code. The Teardown-Reason-Code is intended for use by automata and the Reason-Phrase is intended for the human user. There is no requirement to examine or display the Reason-Phrase.

Reason = "clab-Reason" ":" Teardown-Reason-Code SP Reason-Phrase

Teardown-Reason-Code = 3DIGIT

Reason-Phrase = *255<TEXT, excluding CR, LF, HT>

Example:

clab-Reason:550 Session timeout

Teardown-Reason-code is defined in Table 7–10.

Table 7–10 - Supported Teardown Reason Codes

Teardown Reason Code	Reason Phrase
200	User stopped
204	No user activity
205	Set-top capability mismatch
206	Insufficient priority
207	Network delivery failure
400	Fail to tune
401	Loss of tune
402	Loss of tune
403	RTSP failure
404	Channel failure
405	No RTSP server
408	Unknown
409	Network Resource Failure
420	Settop Heartbeat Timeout
421	Settop Inactivity Timeout
422	Content Unavailable
423	Streaming Failure
424	QAM Failure
425	Volume Failure
426	Stream Control Error
427	Stream Control Timeout
428	Session List Mismatch
502	QAM parameter mismatch
550	Session timeout

If the EQAM receives a teardown with an unrecognized Teardown-Reason-Code (one that is not in the above table), the EQAM MUST attempt to tear down the session.

If an ERM gets a SETUP response from the EQAM with a QAM parameter mismatch, (for example, if the modulation parameters returned in SETUP response do not match what was previously signalled via ERRP), it MUST use Reason-Code 502, ‘QAM parameter mismatch’ if a TEARDOWN request is issued.

7.7.3.4 Extension: clab-SessionGroup

The clab-SessionGroup extension header defines a token passed on a SETUP request that is used to identify a group of sessions. This header then may be used in GET_PARAMETER requests to clarify the request.

The syntax of the clab-SessionGroup header is as follows:

clab-SessionGroup = "clab-SessionGroup" ":" 1*255<ALPHA | DIGIT | safe>

Example:

clab-SessionGroup:clab-SessionGroupTokenExample

The clab-SessionGroup header is used in the process of re-synchronizing session state between two components. The clab-SessionGroup token is generated by an RTSP Client to group its established sessions together. A clab-SessionGroup token is passed to an RTSP Server within SETUP request messages. RTSP Servers remember the clab-SessionGroup token for each established session. RTSP Clients may send a GET_PARAMETER request for the "session_list" parameter, passing the clab-SessionGroup token. The RTSP Server returns the list of RTSP session IDs for currently active sessions that are associated with the session group. See the "clab-session-list" extension, Section 7.8.2.1, for more details. An ERM SHOULD use the clab-SessionGroup header to limit the size of clab-session-list to fit into the 64KB TCP message size. Note that a session may only belong to a single session group.

7.7.3.5 Extension: clab-Priority

The clab-Priority extension header defines a mechanism by which the RTSP Client indicates in a request that certain priority be assigned to a session. This provides a priority for the importance of one session versus another session. For example, all SDV sessions might be assigned to a priority of 1, while all VOD sessions might be of priority 2. The ERM can use the priority to decide if one session should be allowed to use bandwidth over another.

The clab-Priority header provides an integer value from 1 to 9, 1 being the highest priority, and 9 being the lowest (the value 0 is reserved). Its syntax is as follows:

clab-Priority = "clab-Priority" ":" 1DIGIT

Example:

clab-Priority:1

7.7.3.6 Extension: clab-SetupType

The clab-SetupType extension header is used in the RTSP SETUP request message to indicate the type of SETUP request.

The syntax of the clab-SetupType header is as follows:

clab-SetupType = "clab-SetupType" ":" 1DIGIT

Example:

clab-SetupType: 1

Where the clab-SetupType is the type of the SETUP as described in the following table:

Table 7–11 - Description of the clab-SetupType Header

clab-SetupType	Meaning	Description
0	Dynamic	Dynamic session setup

Note the clab-SetupType is an optional header. With its absence, the default clab-SetupType is set to 0, indicating the dynamic session setup.

7.7.3.7 Extension: clab-PidRemap

The clab-PidRemap extension header allows the ERM to indicate to the EQAM as to whether it is required to remap the PIDs in the transport stream or pass them through to the output. In the absence of a clab-PidRemap header, the default PidRemap must be equal to 1, indicating that the EQAM MUST remap PIDs. If the EQAM receives a clab-PidRemap header with a value of zero indicating to pass through the PIDs without remapping them, the ERM MUST ensure that there are no PID conflicts. The Video EQAM MAY attempt to ensure that there are no PID conflicts.

clab-PidRemap = "clab-PidRemap" ":" 1DIGIT

Example:

clab-PidRemap:1

There are several ranges of PIDs that must be avoided by the remapping function of the EQAM. The specific requirements related to PID remapping are provided in [VSI].

Table 7–12 - clab-PidRemap Extension

clab-PidRemap	Meaning	Description
0	Pass-Through	Indication to pass TS through with same PIDs as input stream
1	Re-Map PIDs	Indication to EQAM to remap PIDs

7.7.3.8 Extension: clab-MPTSMode

The clab-MPTSMode extension header defines the QAM channel processing mode for the session as defined in [VSI]. The mode can either be passthrough or multiplexing. In passthrough mode, an MPTS input will be sent to a QAM output without any program or PID remapping. All the PIDs in the MPTS input will be sent to the QAM output. In multiplexing mode, an input program or transport stream will be muxed with other streams and sent to a QAM output. The individual program in the MPTS input can be mapped to a different output program. Passthrough mode and multiplexing mode are mutually exclusive on a QAM channel output and so cannot both be used in the same output simultaneously. Multiplexing mode is the default mode, and is inferred if this header is not included in a SETUP message. The clab-MPTSMode header has the following syntax:

clab-MPTSMode = "clab-MPTSMode" ":" ("passthrough" | "multiplex")

Example:

clab-MPTSMode:passthrough

7.7.3.9 Extension: clab-StatmuxGroup

The clab-StatmuxGroup extension header uniquely identifies the grouping within an ERM and the group bitrate. A statmux group is a subset of programs within an MPTS input stream with a confined aggregated bitrate. When the clab-StatmuxGroup header is used, the bitrate is the aggregated bitrate of the individual programs in the statmux group. The clab-StatmuxGroup has the following syntax:

clab-StatmuxGroup = "clab-StatmuxGroup" ":" "group_id" "=" <group-id> " group_bitrate" "=" <group-bitrate>

group-id = 1*16 CHAR

group-rate=1*8 DIGIT

The group-id identifies the statmux group within an ERM. The group-rate is the aggregated bitrate of all the programs in the statmux group in bps. When clab-StatmuxGroup is included in the session SETUP request, the bit_rate in the transport header MUST be set to zero by the RTSP Client. When clab-StatmuxGroup is included in the session SETUP request, the bit_rate in the transport header SHOULD be ignored by the RTSP Server.

Example:

clab-StatmuxGroup:group_id=GroupA group_bitrate=5000000

7.7.4 SETUP Transport Headers

The Transport header is used to identify parameters associated with the transport of the media stream. The Transport header is used in RTSP SETUP request and SETUP response messages. It indicates the type of transport being requested or granted for the session. An RTSP Server MUST use the Transport headers as defined in [RFC 2326] with the format as defined in Section 7.7.4, and in the manner specified in Section 7.7.4.5. An RTSP Client MUST use the Transport headers as defined in [RFC 2326] with the format as defined in Section 7.7.4, and in the manner specified in Section 7.7.4.5.

7.7.4.1 Transport Header Syntax

This section contains several sections to document the structure of ERMI RTSP transport headers.

- Section 7.7.4.1.1 contains the Transport Header Generic Template.
- Sections 7.7.4.1.2 and 7.7.4.1.3 describe and document the syntax for the EQAM input and output parameters cited in Section 7.7.4.1.1.
- Sections 7.7.4.2, 7.7.4.3, and 7.7.4.4 do not use the precise syntax from section 12.39 of [RFC 2326]. However, they define the structure and options which, taken together with Section 7.7.4.1, specify the normative structure for the ERMI RTSP transport headers. Note that the syntax notation used in Sections 7.7.4.2, 7.7.4.3, and 7.7.4.4 does not follow the same augmented BNF as is used in other sections of this document.

7.7.4.1.1 Transport Header Generic Template

This section provides the formal syntax of the Transport header that compliant ERMI devices must implement. It is compatible with the RTSP Transport header defined in section 12.39 of [RFC 2326]; new transport parameters are specified to be able to establish QAM channels. This header syntax definition must be used in conjunction with Sections 7.7.4.2, 7.7.4.3, and 7.7.4.4.

```
Transport           =   "Transport" ":"
                        1\#transport-spec
transport-spec      =   transport-protocol/profile[/lower-transport]
                        *parameter
transport-protocol  =   "clab-DOCSIS" | "clab-MP2T"
profile             =   "QAM" | "DVBC"
lower-transport     =   "QAM"
parameter           =   ";" ( "unicast" | "multicast" )
                    |   ";" "bit_rate" "=" bit-rate
                    |   ";" "depi_mode" "=" depi-mode
                    |   ";" "source" "=" source-ip
                    |   ";" "source_port" "=" source-port
                    |   ";" "destination" "=" destination-ip
                    |   ";" "destination_port" "=" destination-port
                    |   ";" "multicast_address" "=" multicast-address
                    |   ";" "rank" "=" rank-value
                    |   ";" "mpts_program" "=" mpts-program
                    |   ";" "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
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

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.

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

<source-ip> is the IP address from which the data is streamed. If present in a DOCSIS transport-spec, the value is the IP address of the M-CMTS Core. If present in a Multicast transport-spec, the value is the IP address to use in a Source Specific Multicast join request.

<source-port> is the UDP port from which the media data will be streamed. If present, the value is the UDP port on the M-CMTS Core from which data is sent to the QAM channel.

<destination-ip> is the IP address of the target EQAM input interface.

<destination-port> describes the input UDP port of an EQAM to which the data is streamed. In the unicast case this is the UDP port the stream will be transmitted to. In the multicast case this is the multicast UDP port that the stream will be transmitted to. The ERM MUST NOT use port numbers outside the signaled dynamic port range for dynamic unicast sessions (see Section 6.2.3.13). If no dynamic port range is signaled, then the ERM SHOULD use ports in the range of 49152 - 65535. The EQAM MUST reject a SETUP request that indicates the use of a destination port that is outside the dynamic port range.

<multicast-address> is the IP multicast address of the service.

<rank-value> is a number that identifies the preference order of this transport-spec:

- The rank field is used to order the multicast transport-specs in a session setup for the purpose of redundancy in the case of multiple multicast video sources. When multiple multicast sources are provided, the rank field can be used to identify the primary source versus the secondary or backup sources.
- The order of the transport headers is important when two or more multicast transport-specs have the same rank. The first multicast transport-spec to appear in the message is the first preference, the second is next, etc.

<mpts-program> describes an individual program from an ingress MPTS that should be forwarded to a specific egress MPTS as defined in <qam_destination>. If <mpts-program> is zero, then the ingress TS is a SPTS where no ingress program number is required. If <mpts-program> is non-zero, then <mpts-program> is used to select the

ingress program to de-multiplex and then re-multiplex onto <qam_destination>. The ERM MUST set <mpts-program> to zero when <program-number> of <qam_destination> is zero.

7.7.4.1.3 EQAM Output Parameters

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

<qam-tsid> is the QAM TSID.

<fiber-node> defines one or more fiber nodes names. When fiber-node is used in a SETUP request from the M-CMTS Core to the ERM, it identifies the M-CMTS Core desired fiber nodes for the QAM channel resource. When fiber-node is used in a SETUP response from the ERM to the M-CMTS Core, it identifies the fiber nodes connected to the selected QAM channel. An ERM MUST respond to a SETUP request with all the fiber-nodes to which the QAM channel is connected. For a SETUP response, an ERM SHOULD select a QAM channel that is connected to all the fiber-nodes included in the SETUP request.

Example:

fiber_node="CableLabs.node.1": "CableLabs.node.2": "CableLabs.node.3"

<frequency-range> defines the range of QAM channel frequency. When the frequency-range is used in a SETUP request from the M-CMTS Core to the ERM, it identifies the M-CMTS Core desired frequency range (in Hz) for the QAM channel resource.

Example:

frequency_range=300000000 - 350000000

<qam-name> is the ASCII encoding of a QAM name that consists of <service group>.<TSID>

<qam-destination> describes the "<frequency>.<program-number>" where <frequency> describes the frequency in Hertz with which to tune to the request stream and <program-number> is the program number of the requested egress stream. Note that this is used in a special case whereby the Session Manager requests the UDP and QAM transport information from the ERM. If <program-number> is zero, <qam_destination> is in a pass through mode where all programs and PIDs from a requested ingress transport stream are passed onto the egress transport stream without modification. If <program-number> is non-zero, <qam_destination> is in a multiplexed mode where <program-number> is used to multiplex the requested <mpts-program> onto the egress transport stream of <qam_destination>. For clab-DOCSIS/QAM transport, the <program-number> is set to 0.

<modulation> is the QAM modulation type.

<j83-annex> is the desired QAM operation mode as defined in the Annex section of [J.83].

<taps-value> and <incr-value> are interleaver parameters. The interleaver is defined in [J.83]. If the <taps-value> and <incr-value> fields are used, their values are taken from Table 6–16. The value for taps is taken from the value of I, and the value for increment is taken from the value of J in that table.

<channel-width> is the bandwidth of the QAM channel, in MHz.

<symbol-rate> is the QAM symbol rate in symbols per second. This is only used for video QAMs in J.83 Annex A mode that support variable symbol rate extensions.

7.7.4.2 Transport header format –DOCSIS Data

In the Transport Header format definitions in the following sub-sections, square brackets indicate optional fields, angle brackets delimit variables, and everything else is to be interpreted as literal. If the format shown in this section deviates from the syntax defined in Section 7.7.4.1 and in [RFC 2326], the syntax definition has precedence.

Each Transport header for DOCSIS contains a single transport-spec. Parameter fields within a transport-spec are separated by semicolons, with no space in between.

7.7.4.2.1 ERMI 3 SETUP request from M-CMTS to ERM

Transport:
clab-DOCSIS/QAM
;unicast
;bit_rate=<bit-rate>
;depi_mode=<depi-mode>
[;source=<source-ip>]
[;source_port=<source-port>]
[;qam_tsid=<qam-tsid>]
[;fiber_node=<fiber-node>]
[;frequency_range=<frequency-range>]
[;modulation=<modulation-value>]
[;j83_annex=<j83-annex>]
[;taps=<taps-value>;increment=<incr-value>]
[;channel_width=<channel-width>]

In this request, the M-CMTS Core MUST include either a qam_tsid or a fiber_node parameter to indicate either a particular QAM channel, or a set of fiber nodes that the M-CMTS Core wishes to reach. The physical layer parameters may be provided in order to ensure that the ERM selects a channel that is capable of supporting a certain physical layer configuration.

7.7.4.2.2 ERMI 2 SETUP request from ERM to EQAM

Transport:
clab-DOCSIS/QAM
;unicast
;bit_rate=<bit-rate>
;depi_mode=<depi-mode>
[;source=<source-ip>]
[;source_port=<source-port>]
[;destination=<destination-ip>]
;qam_name=<qam-name>

7.7.4.2.3 ERMI 2 SETUP response from EQAM to ERM

Transport:
clab-DOCSIS/QAM
;unicast
;bit_rate=<bit-rate>
;depi_mode=<depi-mode>
[;source=<source-ip>]
[;source_port=<source-port>]
[;destination=<destination-ip>]
;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>]

7.7.4.2.4 ERMI 3 SETUP response from ERM to M-CMTS

Transport:
clab-DOCSIS/QAM
;unicast
;bit_rate=<bit-rate>
;depi_mode=<depi-mode>
[;source=<source-ip>]
[;source_port=<source-port>]
;destination=<destination-ip>
;qam_tsid=<qam-tsid>
;fiber_node=<fiber-node>
[;frequency_range=<frequency-range>]
;qam_destination=<qam-destination>
;modulation=<modulation-value>
;j83_annex=<j83-annex>
;taps=<taps-value>;increment=<incr-value>
;channel_width=<channel-width>

In this response, the ERM provides the current physical layer parameters for the chosen QAM channel. If these parameters do not match those provided in the corresponding request, the M-CMTS Core may use the mechanism defined in [DEPI] to modify the configuration of the QAM channel.

7.7.4.3 Transport header format – Unicast video

In the Transport Header format definitions in the following sub-sections, square brackets indicate optional fields, angle brackets delimit variables, and everything else is to be interpreted as literal. If the format shown in this section deviates from the syntax defined in Section 7.7.4.1 and in [RFC 2326], the syntax definition has precedence.

Transport headers for unicast video contain one or two comma-separated transport-specs. Parameter fields within a transport-spec are separated by semicolons, with no space in between.

The request and response from a video session manager to the ERM are out of scope, so this section only shows the ERMI 2 request and response.

7.7.4.3.1 ERMI 2 SETUP request from ERM to EQAM

Transport:
clab-MP2T/DVBC/QAM
;qam_name=<qam-name>
;qam_destination=<qam_destination>,
clab-MP2T/DVBC/UDP
;unicast
;bit_rate=< bit-rate>
;destination=<destination-ip>
;destination_port=<destination-port>
[;mpts_program=<mpts-program>]

7.7.4.3.2 ERMI 2 SETUP response from EQAM to ERM

The EQAM MAY include the Transport header in the ERMI 2 SETUP response for unicast sessions.

Transport:
clab-MP2T/DVBC/QAM
;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>]

7.7.4.4 Transport header format – Multicast video

In the Transport Header format definitions in the following sub-sections, square brackets indicate optional fields, angle brackets delimit variables, and everything else is to be interpreted as literal. If the format shown in this section deviates from the syntax defined in Section 7.7.4.1 and in [RFC 2326], the syntax definition has precedence.

Transport headers for multicast video contain one or more comma-separated transport-specs. Parameter fields within a transport-spec are separated by semicolons, with no space in between.

The request and response from a video session managers to the ERM are out of scope, so this section only shows the ERMI 2 request and response.

In order to support redundancy for SDV sessions, the session SETUP request can contain multiple UDP multicast transport-spec options. The priority order in which the ERM desires the multicast transport options to be used is signaled by the <rank-value> field and the order in which the transport-spec entries appear. See Section 7.7.4.1 for more discussion of <rank-value>. This section only shows one multicast request.

7.7.4.4.1 ERMI 2 SETUP request from ERM to EQAM

Transport:

clab-MP2T/DVBC/QAM
;qam_name=<qam-name>
;qam_destination=<qam_destination>,
clab-MP2T/DVBC/UDP
;multicast
;bit_rate=< bit-rate>
[;source=<source-ip>]
;destination=<destination-ip>
;destination_port=<destination-port>
;multicast_address=<multicast-address>
;rank=<rank-value>
[;mpts_program=<mpts-program>]

7.7.4.4.2 ERMI 2 SETUP response from EQAM to ERM

The EQAM MAY include the QAM transport-spec in the ERMI 2 SETUP response.

Transport:

[clab-MP2T/DVBC/QAM
;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>],]
clab-MP2T/DVBC/UDP
;multicast
;bit_rate=< bit-rate>
[;source=<source-ip>]
;destination=<destination-ip>
;destination_port=<destination-port>

;multicast_address=<multicast-address>
;rank=<rank-value>
[;mpts_program=<mpts-program>]

7.7.4.5 Transport Header Use

For DOCSIS applications, the M-CMTS Core, the ERM, and the M-CMTS EQAM MUST support the transport formats defined in Section 7.7.4.2. For video applications, the ERM and Video EQAM MUST support the transport types defined in Sections 7.7.4.3 and 7.7.4.4.

The RTSP Client MUST include the Transport header in the SETUP request to the RTSP Server. The values in the Transport header in a SETUP request indicate the desired resource parameters. The values in the Transport header in a SETUP response indicate the parameters of the selected resource.

In ERMI-2, the RTSP Server (*i.e.,* the EQAM) SHOULD include one or more Transport headers in the SETUP response if the response code is 200. In ERMI-3, the ERM MUST include one Transport header in the SETUP response if the response code is 200. The RTSP Server MUST NOT include the transport types defined in Sections 7.7.4.3 and 7.7.4.4 in the SETUP response if the response code is not 200.

If the M-CMTS Core acting as the RTSP Client is able to utilize a QAM channel resource regardless of the value of a particular optional parameter, then it SHOULD NOT send the corresponding optional field in a SETUP request to the ERM. This increases the RTSP Server 's flexibility in choosing a QAM channel resource. If the M-CMTS Core RTSP Client requires a particular value for an optional field, it MUST include the desired value for that optional field in the SETUP request to the ERM.

In ERMI-2, if an RTSP Server includes a Transport header in a SETUP response to an RTSP Client to indicate success, it MUST include the following fields in the Transport header:

- destination
- qam_name

In ERMI-3, if an RTSP Server includes a Transport header in a SETUP response to an RTSP Client to indicate success, it MUST include the following fields in the Transport header:

- destination
- qam_tsid
- modulation
- j83-annex
- taps
- increment
- channel-width
- fiber_node

For each field, if it was present in the request, the same field and value are both present in the response. If that optional field was not present in the request, the RTSP Server MUST include the field in the response with a value that reflects the current configuration of the QAM channel.

Multiple multicast transport-specs MAY be specified within a MulticastTransport header. In a response to a multicast request, the EQAM MUST include all the multicast transport-specs the EQAM has selected to join, within the considerations of Appendix I.8.1.

The ERM enables the flow and routing of an EQAM that supports the video profile’s UDP-unencapsulated MPEG-2 traffic to an output QAM channel with an RTSP SETUP Request Message for Video on Demand or other unicast services. It also supports the signaling to enable the EQAM to join an IP Multicast Stream and output to the appropriate QAM channel for multicast services. The EQAM configures the flow of traffic and stream processing

based on parameters in the SETUP Request Message, and returns an RTSP SETUP Response Message indicating the success/failure of the request and any ancillary information.

It is the ERM's responsibility to select the EQAM that will carry the session to the appropriate service group. An ERM may consider multiple possible options for stream resource allocation, but when the ERMI interface is used it is an explicit directive to allocate or de-allocate stream resources on a single specific QAM interface.

The ERM MUST NOT attempt to create sessions that would exceed the bandwidth of an input interface or a QAM channel.

The ERM MUST NOT attempt to create sessions that would map more than one input program to the same output program number on any individual QAM channel.

7.8 RTSP Entity Body

Entity body is optional in RTSP messages, and if present it follows the RTSP Headers and is preceded by a blank line.

7.8.1 Entity Body - text/xml

The text/xml Content-Type is used in SETUP methods to pass XML parameters to the device. The outer tag "ermi:clab-ermi" serves as the container for one or more XML elements defined by this specification. The detailed definition of the ermi:clab-ermi element is provided in Annex A.

One or more XML elements may be contained within the ermi:clab-ermi outer tag. The following example shows four XML elements: ermi:EncryptionData (which is defined in Annex A) and three other (as yet undefined) XML elements.

The format is as follows:

```
Content-Type: text/xml
Content-Length: 1234

<?xml version="1.0"?>
<ermi:clab-ermi xmlns:ermi="urn:cablelabs:namespaces:DOCSIS:xsd:EQAM:ermi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ermi:EncryptionData>
    <ermi:encryptSess>true</ermi:encryptSess>
    <ermi:casId>SCTE-52</ermi:casId>
    <ermi:clientMac> client MAC/UA </ermi:clientMac>
    <ermi:cciLevel>Copy No More</ermi:cciLevel>
    <ermi:apsLevel>Disabled</ermi:apsLevel>
    <ermi:CIT>Set</ermi:CIT>
    <ermi:encryptESK>encrypter session key </ermi:encryptESK>
    <ermi:vendorOpaque> unique vendor opaque </ermi:vendorOpaque>
  </ermi:EncryptionData>
  <any_element1/>
  <any_element2/>
  <any_element3/>
</ermi:clab-ermi>
```

7.8.2 Entity Body - text/parameters

The text/parameters Content-Type is used in the GET_PARAMETER and SET_PARAMETER methods to retrieve parameters or set parameters on a device. This specification defines three parameters.

Parameter Type	RTSP Method	Description
clab-session-list	GET_PARAMETER	List of current active sessions
clab-connection-timeout	GET_PARAMETER	Timeout setting for activity on a connection
clab-sessiongroup-list	SET_PARAMETER	A list of session groups to be associated with the connection

7.8.2.1 Parameter: clab-session-list

After re-establishing a broken TCP connection to an RTSP Server, an RTSP Client SHOULD send GET_PARAMETER request to obtain a list of active RTSP sessions on the RTSP Server that were initiated by the RTSP Client prior to the break. The RTSP Client SHOULD use this method to synchronize its session state with the RTSP Server following an RTSP Client reboot.

The RTSP Server MUST support the clab-session-list parameter. As an optional message header to the GET_PARAMETER clab-session-list request, the RTSP Server MUST support the clab-SessionGroup header. The RTSP Server MUST send a GET_PARAMETER response with a clab-session-list value that conveys the list of session IDs for sessions that are active and were setup with the clab-SessionGroup value. If the clab-SessionGroup header is omitted, all sessions will be returned. See Section 7.11.4.

GET_PARAMETER Request:

To request the list of sessions, the content of the message body is: "clab-session-list"

GET_PARAMETER Reply:

The message body syntax for the return value for a GET_PARAMETER clab-session-list is as follows.

clab-session-list = "clab-session-list" ":" clab-session *[";" clab-session]
clab-session = rtsp-session-id [":" clab-client-session-id]
rtsp-session-id = 1*(ALPHA | DIGIT | safe)
clab-client-session-id = 10HEX

where rtsp-session-id is the RTSP Server's session ID and clab-client-session-id is the clab-ClientSessionId originally provided by the RTSP Client. For video sessions, the clab-client-session-id is included. For DOCSIS sessions, the clab-client-session-id should not be included. An example message body containing a return value for a GET_PARAMETER session_list follows:

clab-session-list:12345:00AF123456DE00000001;12346:00BD123456C200000021;
12347:00CE123456AA00000A01

For more information on how the clab-session-list information is used, please see the extension clab-SessionGroup in Section 7.7.3.4.

7.8.2.2 Parameter: clab-connection-timeout

The clab-connection-timeout parameter is used in GET_PARAMETER methods to ask the server for its connection timeout. The RTSP Server MUST return a parameter value for "clab-connection-timeout" in response to a GET_PARAMETER request. See Section 7.1.2 for further elaboration.

The return value from a GET_PARAMETER clab-connection-timeout is as follows.

clab-connection-timeout = "clab-connection-timeout" ":" timeout
timeout=*DIGIT

where timeout is an integer representing seconds.

An example of the message body of a GET_PARAMETER clab-connection_timeout response follows.

clab-connection-timeout:300

7.8.2.3 Parameter: clab-sessiongroup-list

Normally the connection between the RTSP Server and Client is persistent and ANNOUNCE messages will be sent across it. However, if the connection is broken due to an unexpected failure, there needs to be a way for the RTSP Server to re-associate sessions with a particular RTSP Client connection. When either the original RTSP Client or a designated replacement (vendor innovation) connects, it sends the list of session groups for which it is responsible. When the RTSP Server needs to send an ANNOUNCE for a session that was created on a now broken connection, it matches the sessions' group to the group lists for all current connections. If a match is found, the ANNOUNCE message will be sent over that connection. The RTSP Server sends an ANNOUNCE (per the notes in Section 7.7.3.2) to inform the RTSP Client about active video inputs.

The format is as follows:

clab-sessiongroup-list = "clab-sessiongroup-list" ":" session-group *[SP session-group]
session-group =1*255<ALPHA | DIGIT | safe>

An example SET_PARAMETER request is as follows:

```
SET_PARAMETER rtsp://192.0.2.2 RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Content-Type: text/parameters
Content-Length: 48

clab-sessiongroup-list:ERM.SG1 ERM.SG2 ERM.SG3
```

7.9 Session Keepalives and Message Timeout

7.9.1 Session Keepalives and Timeout

A SET_PARAMETER request (with no body) is used as a session keepalive mechanism between the client and server. The issuer of the request includes a Session header with the session ID of the session that is being kept alive. In addition, a CSeq header and a Require header are included in the keepalive request. The response carries only two 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 include a Reason-phrase that provides a detailed description of the error condition. The client is not required to examine the Reason-phrase text that follows the response code.

Table 7–13 - Supported RTSP Response Codes

Response Code	Reason	Comment
200	OK	
400	Bad Request	
403	Forbidden	
404	Not Found	QAM not found TSID not found; TSID does not exist on specified EQAM.
405	Method Not Allowed	Can be used if a dynamic session setup is requested on a device that cannot support it.
408	Request Timeout	

Response Code	Reason	Comment
412	Precondition Failed	Errors in the EncryptionData: Invalid Encryption Type; Encryption Resource Unavailable on the encryptor to encrypt session; Incompatible Encryption Credentials; CCI Mode Unsupported - not available on the encryption device; The unit or MAC address is incorrect;
451	Parameter Not Understood	[RFC 2326] (beginning of section 12) specifies that non-standard header fields should be ignored by the recipient. Output program number conflict.
453	Not Enough Bandwidth	Insufficient QAM channel bandwidth
454	Session Not Found	
456	Header Field Not Valid for Resource	Unicast destination address and UDP port number already in use for an existing session. Clab-MPTSMode not consistent with QAM channel mode.
457	Invalid Range	
461	Unsupported Transport	
462	Destination Unreachable	Invalid Destination IP Address in Transport header.
501	Not Implemented	
503	Service Unavailable	Dynamic session setup not supported for EQAM or QAM channel in static mode.
505	RTSP Version Not Supported	
551	Option Not Supported	

7.11 RTSP Message Examples

7.11.1 SETUP Message Examples

7.11.1.1 Session Setup for Unicast

For dynamic session setup, the ERM must send a SETUP command to the EQAM for each stream associating an input interface (IP address and UDP port) on the EQAM with a specific QAM channel output.

An example of a session SETUP message from the ERM to the EQAM is shown below. The message requests that program 15 of a QAM channel with a frequency of 550MHz and a qam_name of MSO.Division.Hub.1234 be allocated 2.7Mbps of unicast streaming capacity. The message also specifies the destination 192.0.2.1 port 4000. Note that the Require header in the SETUP message must indicate ERMI is the extension required.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name= MSO.Division.Hub.1234
                ;qam_destination=550000000.15,
        clab-MP2T/DVBC/UDP
```

```
                ;unicast
                ;bit_rate=2700000
                ;destination=192.0.2.1
                ;destination_port=4000
clab-ClientSessionId:8cd50800200c9a66abcd
Content-Type: text/xml
Content-Length: 574

<?xml version="1.0"?>
<ermi:clab-ermi xmlns:ermi="urn:cablelabs:namespaces:DOCSIS:xsd:EQAM:ermi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ermi:EncryptionData>
    <ermi:encryptSess>true</ermi:encryptSess>
    <ermi:casId>SCTE-52</ermi:casId>
    <ermi:clientMac> 8cd50800200c </ermi:clientMac>
    <ermi:cciLevel>Copy No More</ermi:cciLevel>
    <ermi:apsLevel>Disabled</ermi:apsLevel>
    <ermi:CIT>Set</ermi:CIT>
    <ermi:encryptESK>284F9EAB396D21 </ermi:encryptESK>
  </ermi:EncryptionData>
</ermi:clab-ermi>
```

The body of the session setup request may optionally contain XML data with the root level element <ermi:clab-ermi> for the EQAM that supports embedded encryption and registers with the ERM through the ERRP interface.

Several parameters in the above example are optional as specified in the RTSP header table. If the SETUP request succeeds, the session setup response from EQAM to ERM should look like:

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

If the SETUP request fails, the EQAM's session SETUP response gives a response code to show the reason for the failure. For example:

```
RTSP/1.0 453 Not Enough Bandwidth
CSeq: 314
```

7.11.1.2 Session Setup for Multicast

For dynamic session setup, the ERM must send a SETUP command to the EQAM for joining and processing the input IP Multicast streams. The following is an example shows a SETUP command specifying more than one possible multicast source.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 313
Require: com.cablelabs.ermi
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.15,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=2700000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
```

```
                ;multicast_address=232.1.1.1
                ;rank=1,
        clab-MP2T/DVBC/UDP
                ;multicast;
                ;bit_rate=2700000
                ;source_address=4.4.4.4
                ;destination=192.0.2.11
                ;destination_port=102
                ;multicast_address=232.3.3.3
                ;rank=2
clab-ClientSessionId:8cd50800200c9a66abcd
```

In this example the EQAM should send join message for one or both multicast groups and pass only one to the output. In case of failure it switches immediately to the redundant source.

Following is an example SETUP response from ERMI with the additional Transport parameters added.

```
RTSP/1.0 200 OK
CSeq: 313
Session: 47112344
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.15,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=2700000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=2
ClientSessionId:8cd50800200c9a66abcd
```

The SETUP response MUST contain all the multicast transports which were joined by the EQAM as the sources. However, if the SETUP response indicates a failure, then the MulticastTransport header MAY be omitted.

7.11.1.3 Setup for MPTS Sessions

MPTS processing assumptions:

- The ERM knows the program numbers used in the MPTS stream and will guarantee that the program number it gives to EQAM for SPTS stream will not conflict with the MPTS stream.
- There is a pre-configured pool of PIDs per QAM for no-PID remap application. This pool of PIDs is known to the MPTS source, which guarantees that the PIDs used in the MPTS will be from this pool and will not conflict if clab-PidRemap is turned off.

When an MPTS stream is delivered to an output without grooming, only a single session SETUP message is sent from the ERM to an EQAM. This applies to both Passthrough mode and Multiplexing mode. The bitrate is the aggregated bandwidth of the MPTS. In addition, the clab-PidRemap must be turned off. An example of MPTS session SETUP is:

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 313
Require: com.cablelabs.ermi
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.0,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=10000000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=0,
        clab-MP2T/DVBC/UDP
                ;multicast;
                ;bit_rate=10000000
                ;source_address=4.4.4.4
                ;destination=192.0.2.11
                ;destination_port=102
                ;multicast_address=232.1.1.1
                ;rank=2
                ;mpts_program=0
clab-ClientSessionId:8cd50800200c9a66abcd
clab-PidRemap:0
clab-MPTSMode:passthrough
```

Here is the response from EQAM:

```
RTSP/1.0 200 OK
CSeq: 313
Session: 12345678
Transport:
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=10000000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=0
clab-ClientSessionId:8cd50800200c9a66abcd
```

When grooming is used for an MPTS stream, each individual program is signaled with a separate session SETUP; the bitrate is the peak bandwidth of the program signaled. This is the default mode of operation for MPTS mux. The following is an example SETUP messages for an MPTS mux with two programs in the MPTS. The first SETUP message maps the program 1 of input transport to the program 2 of QAM output.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 313
Require: com.cablelabs.ermi
Transport:
```

```
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.2,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=5000000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=1
clab-ClientSessionId:8cd50800200c9a66abcd
```

Here is the response from EQAM:

```
RTSP/1.0 200 OK
CSeq: 313
Session: 12345678
Transport:
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=5000000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=1
clab-ClientSessionId:8cd50800200c9a66abcd
```

The second SETUP message maps the program 2 of the input transport to the program 3 of QAM output.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.3,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=5000000
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=2
clab-ClientSessionId:8cd50800200c9a66abce
```

Here is the response from EQAM:

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

Session: 12345679
Transport:
clab-MP2T/DVBC/UDP
;multicast
;bit_rate=5000000
;source_address=2.2.2.2
;destination=192.0.2.10
;destination_port=100
;multicast_address=232.1.1.1
;rank=1
;mpts_program=2
clab-ClientSessionId:8cd50800200c9a66abce

An optional header clab-StatmuxGroup can be used to support the grouping of subset of streams within an MPTS input stream. When the clab-statmux-group header is used, the bitrate is the aggregated bitrate of the individual programs in the statmux-group. The statmux group header uniquely identifies the grouping within an MPTS input stream. The following example is a modified version of the above MPTS mux with program 1 and program 2 within the same statmux-group and a combined bitrate of 8Mbps.

Here is the first SETUP message:

SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 313
Require: com.cablelabs.ermi
Transport:
clab-MP2T/DVBC/QAM
;qam_name=Division.Hub.20
;qam_destination=550000000.2,
clab-MP2T/DVBC/UDP
;multicast
;bit_rate=0
;source_address=2.2.2.2
;destination=192.0.2.10
;destination_port=100
;multicast_address=232.1.1.1
;rank=1
;mpts_program=1
clab-StatmuxGroup: group_id = 1234; group_rate=8000000
clab-ClientSessionId:8cd50800200c9a66abcd

Here is the response from EQAM:
RTSP/1.0 200 OK
CSeq: 313
Session: 12345678
Transport:
clab-MP2T/DVBC/UDP
;multicast
;bit_rate=0
;source_address=2.2.2.2
;destination=192.0.2.10
;destination_port=100
;multicast_address=232.1.1.1
;rank=1
;mpts_program=1
clab-ClientSessionId:8cd50800200c9a66abcd

The second SETUP message maps program 2 of the input transport to the program 3 of QAM output.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Transport:
        clab-MP2T/DVBC/QAM
                ;qam_name=Division.Hub.20
                ;qam_destination=550000000.3,
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=0
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=2
clab-StatmuxGroup: group_id = 1234; group_rate=8000000
clab-ClientSessionId:8cd50800200c9a66abce
```

Here is the response from EQAM:

```
RTSP/1.0 200 OK
CSeq: 314
Session: 12345679
Transport:
        clab-MP2T/DVBC/UDP
                ;multicast
                ;bit_rate=0
                ;source_address=2.2.2.2
                ;destination=192.0.2.10
                ;destination_port=100
                ;multicast_address=232.1.1.1
                ;rank=1
                ;mpts_program=2
clab-ClientSessionId:8cd50800200c9a66abce
```

7.11.1.4 Session Setup for M-CMTS

Suppose that an M-CMTS Core wishes to create a MAC domain. It sends a SETUP request to request a QAM channel with a bit rate of 38 Mbps. The QAM channel is to be chosen from a service group, which is identified by a list of QAM channels. Suppose that the service group contains two QAM channels, with QAM names of Division.Hub.123 and Division.Hub.456. The DEPI mode for the session (see [DEPI]) is to be DOCSIS-MPT. The ERM's signaling IP address is 192.0.2.2.

The SETUP request from the M-CMTS Core to the ERM looks like this:

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Transport:
        clab-DOCSIS/QAM
                ;unicast
                ;bit_rate=38000000
                ;depi_mode=docsis_mpt
```

```
            ;qam_tsid=123,
      clab-DOCSIS/QAM
            ;unicast
            ;bit_rate=38000000
            ;depi_mode=docsis_mpt
            ;qam_tsid=456
```

The ERM searches its resource database, in the order in which the channels are presented in the received Transport header, and discovers that the QAM channel with QAM TSID 456 is already in use but the channel with QAM TSID 123 is available. The ERM therefore tentatively selects the latter QAM channel to satisfy the request, and sends a request to the EQAM for this resource.

In this request, the ERM specifies a bit rate of 38 Mbps. The ERM knows the EQAM's signaling IP address (192.0.2.2) through the service registration interface (ERMI-1).

In the SETUP request from the ERM to the EQAM, the ERM uses QAM name to identify the QAM channel.

```
SETUP rtsp://192.0.2.2/ RTSP/1.0
CSeq: 101
Require: com.cablelabs.ermi
Transport:
      clab-DOCSIS/QAM
            ;unicast
            ;bit_rate=38000000
            ;depi_mode=docsis_mpt
            ;qam_name=Division.Hub.123
```

The SETUP response from the EQAM to the ERM indicates the QAM channel is indeed available.

```
RTSP/1.0 200 OK
CSeq: 101
Session: 12345679
Transport:
      clab-DOCSIS/QAM
            ;unicast
            ;bit_rate=38000000
            ;depi_mode=docsis_mpt
            ;qam_name=Division.Hub.123
```

The SETUP response from the ERM to the M-CMTS Core, indicates that one of the requested resources is available for use. It fills in values for the Transport header from the information in its database that was passed to it from the EQAM at the time that the resource was registered.

```
RTSP/1.0 200 OK
CSeq: 314
Session: 47223344
Transport:
      clab-DOCSIS/QAM
            ;unicast
            ;bit_rate=38000000
            ;depi_mode=docsis_mpt
            ;destination=192.0.2.2
            ;qam_tsid=123
            ;qam_destination=690000000.0
            ;modulation=256
            ;j83_annex= Annex_B
            ;taps=16
            ;increment=8
```

```
;channel_width=6
;fiber_node = Division.Node1
```

7.11.2 TEARDOWN Message Examples

7.11.2.1 Introduction

The ERM interacts with the EQAM with RTSP TEARDOWN Request and Response messages to teardown existing sessions.

7.11.2.2 Message Headers

The example below shows a message request to tear down an ERMI RTSP session with session ID of 47112345. The reason for this teardown is user stop (reason code 200). The request also indicates ERMI extension is required in the Require header. The presence of the clab-Reason header and clab-clientSessionId indicates this is a Video message.

```
TEARDOWN rtsp://192.0.2.2/ RTSP/1.0
CSeq: 315
Require: com.cablelabs.ermi
clab-Reason: 200 "User stop"
Session: 47112345
clab-clientSessionId:11d98cd50800200c9a66
```

The example response is:

```
RTSP/1.0 200 OK
CSeq: 315
Session: 47112345
clab-clientSessionId:11d98cd50800200c9a66
```

7.11.2.3 Session Teardown

When the M-CMTS deletes the DOCSIS MAC domain, it sends a TEARDOWN request to the ERM to release the QAM channel. The TEARDOWN message contains the RTSP session ID from the SETUP response:

```
TEARDOWN rtsp://192.0.2.2/ RTSP/1.0
CSeq: 316
Require: com.cablelabs.ermi
Session: 47223344
```

The ERM now sends a corresponding TEARDOWN request to the EQAM:

```
TEARDOWN rtsp://192.0.52.12/ RTSP/1.0
CSeq: 102
Require: com.cablelabs.ermi
Session: 47112344
```

The EQAM release the QAM channel and sends a response to confirm its release:

```
RTSP/1.0 200 OK
CSeq: 102
Session: 47112344
```

The ERM sends a corresponding response to the M-CMTS Core:

```
RTSP/1.0 200 OK
```

CSeq: 316
Session: 47223344

7.11.3 SET_PARAMETER Keepalive Message Examples

7.11.3.1 Introduction

Session keepalive will use the RTSP SET_PARAMETER mechanism defined in Section 7.9.

7.11.3.2 Interaction Diagram

The below diagram depicts the interaction between the ERM and the EQAM for a keepalive interaction.

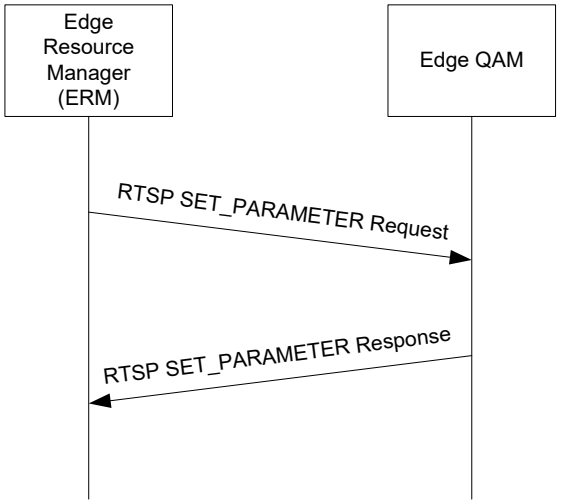


Figure 7–2 - Keepalive Interaction Diagram

7.11.3.3 Message Headers

The following example shows the session keepalive messages using the RTSP SET_PARAMETER message. This message notifies the message receiver that the session with session ID of 47112344 is alive. Both the request message and the response message are given.

SET_PARAMETER rtsp://192.0.2.2/ RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Session: 47112344

RTSP/1.0 200 OK
CSeq: 314
Session: 47112344

7.11.3.4 Keepalive Interaction Scenario

Session keepalive will use the RTSP SET_PARAMETER mechanism defined in Section 7.9.

Step 1 – ERM sends RTSP SET_PARAMETER Request

To request stream information the ERM sends a RTSP SET_PARAMETER Request to the EQAM. After sending the message the ERM starts a timer.

If the timer expires prior to reception of a SET_PARAMETER Response the ERM considers the keepalive request failed. If the ERM receives a RTSP SET_PARAMETER Response for the request after the timer has expired, it may ignore the response.

Step 2 – EQAM sends RTSP SET_PARAMETER Response

The EQAM receives RTSP SET_PARAMETER Request from an ERM.

If the EQAM has issues with the format or content of the RTSP SET_PARAMETER Request or its parameters, it responds to the ERM with a RTSP SET_PARAMETER Response with a response code 451.

Upon reception of a valid RTSP SET_PARAMETER Request the EQAM will send a RTSP SET_PARAMETER Response to the ERM with response code 200 (OK).

7.11.3.5 Session Keepalive

Session Keepalive requests from the RTSP Client to the RTSP Server look like this:

```
SET_PARAMETER rtsp://192.0.2.3/ RTSP/1.0
CSeq: 123
Require: com.cablelabs.ermi
Session: 1234567
```

Keepalive responses from the RTSP Server to the RTSP Client look like this:

```
RTSP/1.0 200 OK
CSeq: 123
Session: 1234567
```

7.11.4 GET_PARAMETER Message Examples

7.11.4.1 Introduction

The ERM interacts with the EQAM via the RTSP GET_PARAMETER Request and Response messages to retrieve information about the sessions. Once connection is lost and re-established, the ERM will sync all sessions from the EQAM and decide whether to leave them as they are or tear them down.

7.11.4.2 Interaction Diagram

The below diagram depicts the interaction between the ERM and the EQAM for a GET_PARAMETER interaction.

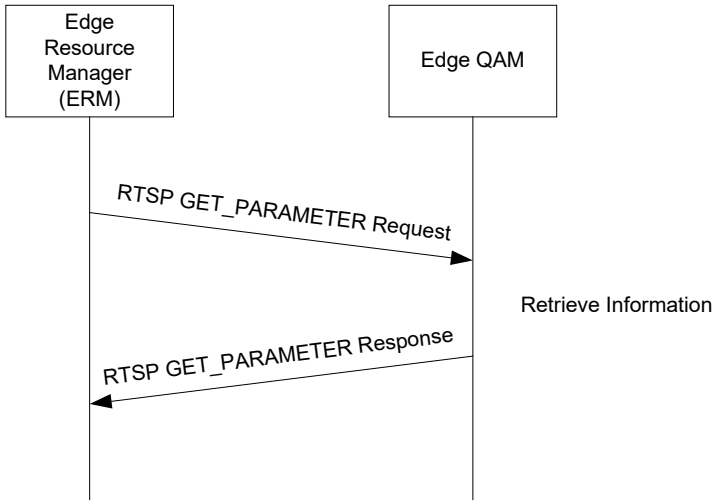


Figure 7–3 - GET_PARAMETER Interaction Diagram

7.11.4.3 Message Headers

The RTSP Entity Body section specifies the supported parameters for the GET_PARAMETER method, per Section 7.8.2.

The following example shows the how the ERM requests the connection timeout parameter for this socket connection. The EQAM responds to the request and notifies the sender that the connection timeout value for it is 180 seconds.

```
GET_PARAMETER rtsp://192.0.2.2 RTSP/1.0
CSeq: 314
Require: com.cablelabs.ermi
Content-Type: text/parameters
Content-Length: 25

clab-connection-timeout
```

The response message can be described in the following:

```
RTSP/1.0 200 OK
CSeq: 314
Content-Type: text/parameters
Content-Length: 29

clab-connection-timeout:180
```

The following example shows the how the ERM requests the list of current active sessions. The EQAM responds to the request by listing the sessions.

```
GET_PARAMETER rtsp://192.0.2.2 RTSP/1.0
CSeq: 318
Require:com.cablelabs.ermi
Content-Type: text/parameters
Content-Length: 19

clab-session-list
```

The response message can be described in the following:

```
RTSP/1.0 200 OK
CSeq: 318
Content-Type: text/parameters
Content-Length: 47

clab-session-list:12345:00AF123456DE00000001;67890:00AF1234567800000024
```

The following example shows the how the ERM requests the list of current active sessions for a particular Session Group. The EQAM responds to the request by listing the appropriate sessions.

```
GET_PARAMETER rtsp://192.0.2.2 RTSP/1.0
CSeq: 321
Require:com.cablelabs.ermi
clab-SessionGroup:sessionGroup1
Content-Type: text/parameters
Content-Length: 19

clab-session-list
```

The response message can be described in the following:

```
RTSP/1.0 200 OK
CSeq: 321
clab-SessionGroup:sessionGroup1
Content-Type: text/parameters
Content-Length: 47

clab-session-list:12345:00AF123456DE00000001
```

7.11.4.4 GET_PARAMETER Interaction Scenario

The following shows the typical interaction scenario for RTSP GET_PARAMETER:

Step 1 – ERM sends RTSP GET_PARAMETER Request

To request stream information the ERM sends a RTSP GET_PARAMETER Request to the EQAM. After sending the message the ERM starts a timer.

If the timer expires prior to reception of a GET_PARAMETER Response the ERM considers the GET_PARAMETER request failed. If the ERM receives a RTSP GET_PARAMETER Response for the request after the timer has expired, it may ignore the response.

Step 2 – EQAM sends RTSP GET_PARAMETER Response

The EQAM receives RTSP GET_PARAMETER Request from an ERM.

If the EQAM has issues with the format or content of the RTSP GET_PARAMETER Request or its parameters, it responds to the ERM with a RTSP GET_PARAMETER Response with response code of 400 indicating "Bad Request".

Upon reception of a valid RTSP GET_PARAMETER Request the EQAM will retrieve the requested information and will send a RTSP GET_PARAMETER Response to the ERM with the appropriate parameters.

7.11.4.5 Get Parameter

The following is an example of GET_PARAMETER request from an RTSP Client to an RTSP Server to get a list of current active session ids.

```
GET_PARAMETER rtsp://192.0.2.2 RTSP/1.0
CSeq: 130
Require:com.cablelabs.ermi
Content-Type: text/parameters
Content-Length: 19

clab-session-list
```

The RTSP response from the RTSP Server to the RTSP Clients looks like this:

```
RTSP/1.0 200 OK
CSeq: 130
Content-Type: text/parameters
Content-Length: 47

clab-session-list:12345:00AF123456DE00000001
```

7.11.5 ANNOUNCE Message Examples

7.11.5.1 Introduction

On some occasions the EQAM will send unsolicited messages to the ERM regarding active sessions. These messages will largely reflect events that may or will have impact to an active streaming session. These messages are sent via RTSP ANNOUNCE Request and Response messages as detailed below.

Reasons for the EQAM to send an ANNOUNCE message include a QAM channel failure, or join redundant IP Multicast.

7.11.5.2 Interaction Diagram

The below diagram depicts the interaction between the ERM and the EQAM to notify the ERM of information relating to the session.

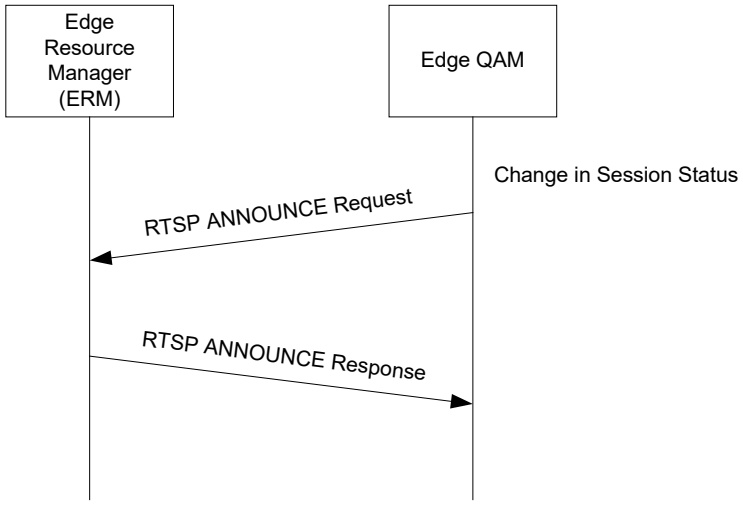


Figure 7–4 - ANNOUNCE Interaction Diagram

7.11.5.3 Downstream Failure Message Header

A new header beyond the original RTSP [RFC 2326]. called clab-Notice (per Section 7.7.3.2) is used for RTSP ANNOUNCE message. The text sequence below shows an example of an ANNOUNCE message. This message announces a downstream failure for a session with session ID 47112344. The time for the announce event is also given. The Require header also indicates that ERMI extension is required.

```
ANNOUNCE rtsp://192.0.2.2 RTSP/1.0
CSeq: 316
Require:com.cablelabs.ermi
Session: 47112344
clab-Notice:5401 "Downstream Failure"
 event-date=19930316T064707.735Z npt
clab-ClientSessionId:11d98cd50800200c9a66
```

The response to ANNOUNCE will be:

```
RTSP/1.0 200 OK
CSeq: 316
Session: 47112344
clab-ClientSessionId:11d98cd50800200c9a66
```

7.11.5.4 ANNOUNCE Interaction Scenario

The following shows the typical interaction scenario for RTSP ANNOUNCE:

Step 1 – EQAM sends RTSP ANNOUNCE Request to ERM
The EQAM sends a RTSP ANNOUNCE Request message to the ERM.

No timers are needed as RTSP ANNOUNCE messages are intended for near real-time information.

Step 2 – ERM sends RTSP ANNOUNCE Response to EQAM
The ERM receives a RTSP ANNOUNCE Request from the EQAM.

If the ERM has issues with the format or content of the RTSP ANNOUNCE Request or its parameters, it responds to the EQAM with a RTSP ANNOUNCE Response with response code of 400 indicating "Bad Request". If the ERM doesn't know of the session, it responds to the EQAM with response code of 452 indicating "Session not found."

The ERM will send a RTSP ANNOUNCE Response back to the EQAM with appropriate parameters.

7.11.5.5 Session Announce

The following is an example ANNOUNCE request sent from the ERM to an M-CMTS Core to ask the M-CMTS Core to release the RTSP session:

```
ANNOUNCE rtsp://192.0.2.2 RTSP/1.0
CSeq: 130
Require:com.cablelabs.ermi
Session: 12345678
clab-Notice:5401 "Downstream Failure"
event-date=19930316T064707.735Z npt
clab-ClientSessionId:11d98cd50800200c9a66
```

If the M-CMTS Core agrees to tear down the session, the response looks like this:

```
RTSP/1.0 200 OK
CSeq: 130
Session: 12345678
clab-ClientSessionId:11d98cd50800200c9a66
```

This is then followed by a TEARDOWN message from the M-CMTS Core to the ERM (see section <Session Teardown>).

If, however, the M-CMTS Core does not agree to tear down the session following receipt of the first message in this section, the response looks like this:

```
RTSP/1.0 503 Service Unavailable
CSeq: 130
Session: 12345678
clab-ClientSessionId:11d98cd50800200c9a66
```

7.12 DOCSIS Resource allocation operation

7.12.1 Resource Allocation

When the M-CMTS Core needs to add a QAM channel resource to a DOCSIS MAC domain, it initiates an RTSP session with a SETUP request in order to obtain downstream QAM resources for the domain. Similarly, when the M-CMTS Core wishes to remove a QAM channel resource from a DOCSIS MAC domain, it concludes the RTSP session by sending a TEARDOWN request to release that downstream QAM channel resource.

When setting up a session, the sequence of messages shown in Figure 7–5 is followed.

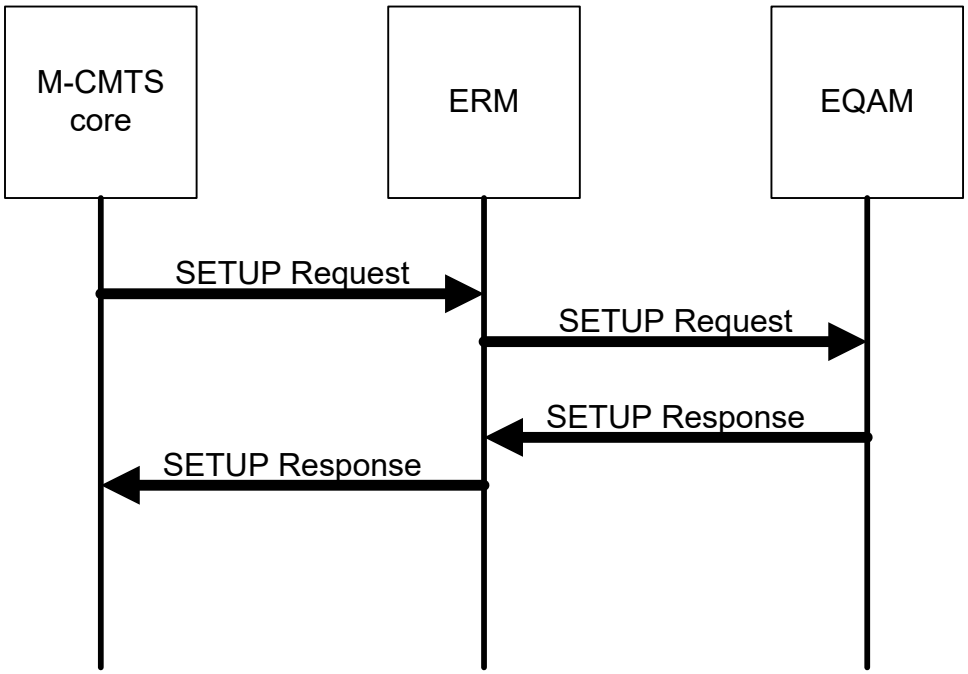


Figure 7–5 - RTSP SETUP Message Flow

The M-CMTS Core MUST include the following information in its SETUP request:

- Fiber node names or a list of QAM channels;
- QAM channel bit rate;
- QAM channel depi mode.

If fiber nodes are used, the M-CMTS Core MUST use the fiber-node Output parameter per Section 7.7.4.1.3. The QAM channel selected by the ERM must be connected to at least one of the fiber nodes indicated in the fiber-node Output parameter. If an explicit QAM channels list is used, the M-CMTS Core MUST represent each QAM channel by a single transport-spec. The ERM MUST select a QAM channel from the list presented to it in the Transport header of the SETUP request.

The parameters of the QAM channel are described in the Transport header. The ERM MUST NOT select a QAM channel that does not support all the listed parameters.

The M-CMTS Core MUST NOT use different DOCSIS parameters in different transport-specs of a single RTSP request.

If the ERM cannot locate a QAM channel that satisfies all the parameters listed in the Transport header, it MUST respond to the request with a SETUP response with an appropriate error code as defined in Section 7.10.

After a downstream QAM channel has been selected by the ERM, the ERM MUST send a SETUP request to the EQAM that contains the selected QAM channel and containing at least the following information:

- QAM name;
- QAM channel bit rate;
- QAM channel parameters.

After the EQAM receives this RTSP SETUP request, it MUST verify that the downstream QAM channel resource is available. If the resource is available, the EQAM MUST send an RTSP SETUP response to the ERM with success status. If the resource is not available the EQAM MUST return a SETUP response indicating failure. The EQAM MUST also respond with a failure notification if the requested QAM channel parameters indicated in Transport header cannot be supported without disrupting the operation of any other QAM channels in the EQAM. The EQAM MAY configure the QAM channel using the values indicated in the Transport header before it returns the SETUP response to the ERM. The EQAM MAY wait for DEPI operation to perform reconfiguration of QAM channel parameters. The M-CMTS Core MUST reconfigure the QAM channel (if necessary) before the QAM channel is used.

The ERM MUST send a SETUP response with an appropriate response code as specified in Table 7–13. The ERM MUST try every QAM channel in the received QAM channel list or in the received fiber node list (in order), until it satisfies the request or it has exhausted the QAM channel list. If no QAM channels are available to satisfy the request, the ERM MUST send response code 453 in the SETUP response.

If the ERM receives a SETUP response with a response code of 200 from the EQAM, it SHOULD send a SETUP response with a response code of 200 to the M-CMTS Core. If the ERM sends a SETUP response with a response code of 200 to the M-CMTS Core, the ERM MUST include a Transport header with at least the following information:

- QAM name;
- QAM TSID;
- QAM channel IP address;
- QAM channel parameters.

If QAM channel allocation is successful and the fiber node configuration of the QAM channel is known to the ERM via ERMI-1 interface, the ERM MUST return the fiber node information by including a fiber-node Output parameter in the response.

If the ERM is unable to find suitable QAM channel (based on requested modulation parameters), it MUST respond to the M-CMTS Core using Response code 503.

The M-CMTS Core initiates a data session to the QAM channel, following the procedure defined in [DEPI].

The ERM SHOULD NOT send the SETUP response to the M-CMTS Core before it receives a SETUP response from the EQAM, unless a timeout occurs or the ERM, after consulting its database of resources, cannot locate a suitable QAM channel.

7.12.2 Resource De-allocation

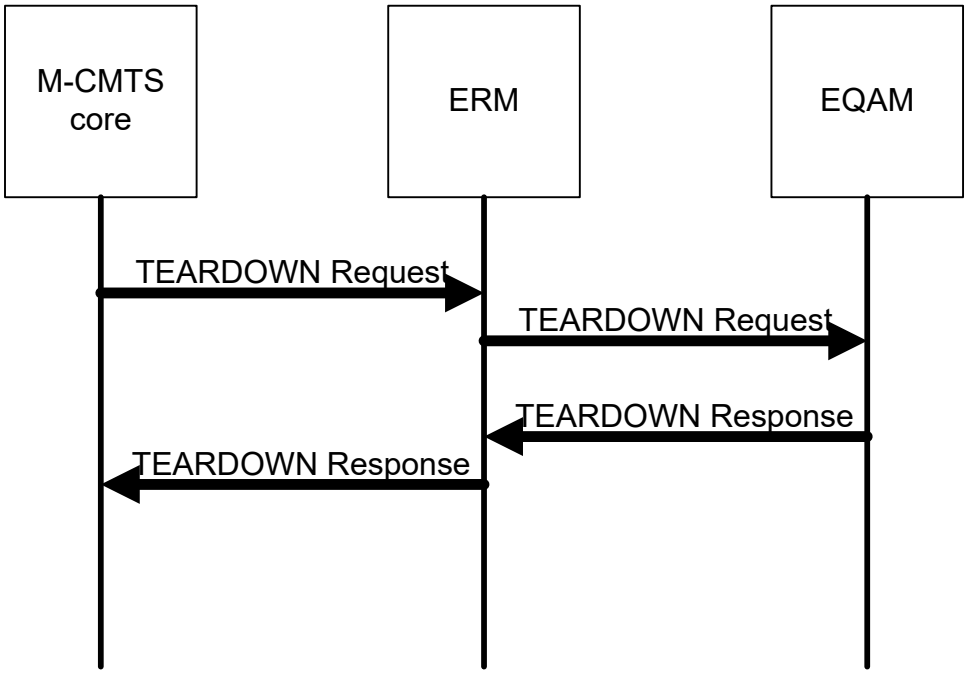


Figure 7–6 - RTSP TEARDOWN Message Flow

If an M-CMTS Core removes a QAM channel resource from a MAC domain, it MUST send a TEARDOWN request to the ERM. This message MUST include the corresponding RTSP session ID obtained from the original SETUP response.

Upon receipt of a TEARDOWN request, the ERM MUST send a TEARDOWN request to the EQAM. The ERM SHOULD NOT send the TEARDOWN response to the M-CMTS Core before it receives a TEARDOWN response from EQAM device, unless a timeout occurs.

If a QAM channel is re-allocated (*e.g.,* between applications that carry video and DOCSIS traffic), some of the channel configuration parameters may change as the channel switches between the two kinds of traffic. The EQAM SHOULD configure QAM channels by default so that their parameters match the requirements of video traffic because some video applications simply assume the default configuration is in effect. When a QAM channel is used for DOCSIS traffic, some of these settings could be changed over the DEPI interface (see [DEPI] for details). Therefore, in order to make this QAM channel available for VOD use after DOCSIS use, the EQAM MUST restore the QAM channel parameters to their prior values before an EQAM sends a TEARDOWN response to ERM to release a QAM channel resource. An EQAM SHOULD NOT restore the QAM channel configuration if the session being torn down is not the only RTSP session in this QAM channel or if any QAM channel dependencies exist that might affect other QAM channels that are in use.

7.12.3 Multiple QAM channels in MAC Domain

In a single DOCSIS MAC domain, there may be multiple downstream QAM channels. To support multiple QAM channels in a single MAC domain, the M-CMTS Core MUST send resource requests for different QAM channels in the MAC domain to the ERM separately, one request for each QAM channel resource.

7.12.4 Synchronization with DEPI control [DEPI]

[DEPI] defines a control interface between the M-CMTS Core and the EQAM. This interface is used to negotiate session parameters for DOCSIS traffic. For example, the UDP port on which the EQAM accepts DOCSIS data for the session is negotiated over this interface. This section is only applicable for DOCSIS sessions and does not apply to video sessions.

If an M-CMTS application has not statically assigned QAM channel resources and must obtain them through the ERM via RTSP, then the M-CMTS Core MUST NOT initiate a DEPI session prior to obtaining QAM channel resources for that session. Similarly, the M-CMTS Core MUST NOT release QAM channel resources before tearing down the corresponding DEPI session. If DEPI session setup fails, the M-CMTS Core SHOULD tear down the RTSP session to release the QAM channel resources.

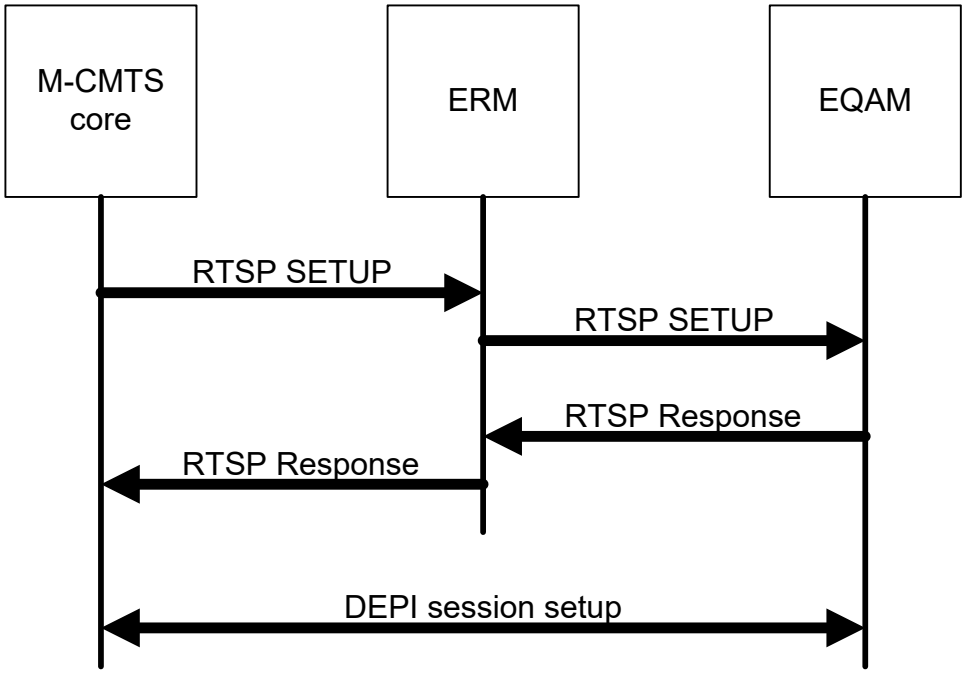


Figure 7–7 - Session Setup Sequence with DEPI

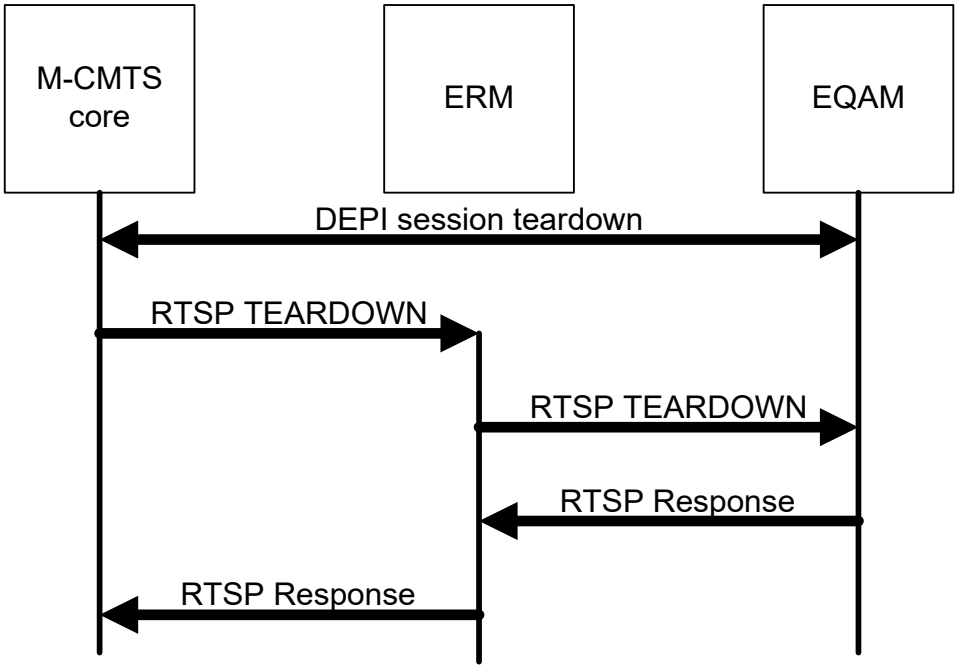


Figure 7–8 - Session Teardown Sequence with DEPI

[DEPI] allows an M-CMTS Core to change QAM channel physical settings directly. The EQAM checks to make sure that changing the setting of one QAM channel does not affect the operation of any other QAM channels in service. Because QAM channels are physically connected to RF ports, there may be limitations on how physical parameters can be changed.

For example, an RF port may require that all the associated QAM channels have the same modulation type. Suppose there are four QAM channels in the RF port and one QAM channel is already in service with a modulation type of QAM64. A new request (for a different QAM channel in this port) with modulation type QAM256 should be rejected. Otherwise, the operation of the existing QAM would be affected.

An M-CMTS Core MUST use separate RTSP requests to obtain QAM channel resource for each DEPI data session.

Annex A XML Extensions

The <ermi:clab-ermi> XML element is defined to carry additional extension information in session SETUP.

In the case when EQAM supports embedded encryption option, the ERM should use the ERMI interface upon session setup to communicate with EQAM on the necessary encryption parameters in the SETUP request message. The EncryptionData xml element informs the EQAM that encryption/conditional access information is being passed to the device.

A.1 EncryptionData Descriptor Definitions

The SETUP message uses the following descriptors for providing the appropriate information to the encryption engine to encrypt/protect the stream. If a particular element is not needed, then it can be omitted.

encrypt Session: indicates whether to encrypt session or pass-through unencrypted:

- true
- false

conditional access ID: indicates conditional access type for session, shown are known settings (others will be added later as needed):

- MediaCipher
- PowerKEY
- SCTE-52

client MAC/UA: represents the session's RTSP Client MAC address or unit address

CCI level, rights data: represents Copy Control Indicator/Digital Rights protection, valid settings are as follows:

- Copy Never
- Copy One Generation
- Copy Freely
- Copy No More

APS level, rights data: represents Analog Protection System/Macrovision protection, valid settings are as follows:

- Enabled
- Disabled

CIT, rights data: represents the Constrained Image Trigger flag, valid settings are as follows:

- Clear
- Set

encrypter session key: provides encrypter Session Key (SK) for the uniquely identified encrypter (network, QAM, etc) in the initial request. The message will be passed to encrypter via the CreateSession message. The SK can be an opaque message blob that has no relevance to any component in the network other than the targeted encrypter.

unique vendor opaque: ***provides*** the encrypter a vendor specific opaque message. The message allows a vendor to provide information that is not described/outlined/provided by one of the other previous messages outlined above.

A.2 XML Schema Definition

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="urn:cablelabs:namespaces:DOCSIS:xsd:EQAM:ermi"
xmlns:ermi="urn:cablelabs:namespaces:DOCSIS:xsd:EQAM:ermi"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:simpleType name="casIdType">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="MediaCipher"/>
			<xsd:enumeration value="PowerKEY"/>
			<xsd:enumeration value="SCTE-52"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="cciLevelType">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Copy Never"/>
			<xsd:enumeration value="Copy One Generation"/>
			<xsd:enumeration value="Copy Freely"/>
			<xsd:enumeration value="Copy No More"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="apsLevelType">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Enabled"/>
			<xsd:enumeration value="Disabled"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="CITType">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Clear"/>
			<xsd:enumeration value="Set"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:element name="encryptSess" type="xsd:boolean"/>
	<xsd:element name="casId" type="ermi:casIdType"/>
	<xsd:element name="clientMac" type="xsd:string" nillable="false" abstract="false"/>
	<xsd:element name="cciLevel" type="ermi:cciLevelType" nillable="false"/>
	<xsd:element name="apsLevel" type="ermi:apsLevelType"/>
	<xsd:element name="CIT" type="ermi:CITType"/>
	<xsd:element name="encryptESK" type="xsd:string"/>
	<xsd:element name="vendorOpaque" type="xsd:string"/>
	<xsd:complexType name="EncryptionDataType">
		<xsd:sequence>
			<xsd:element ref="ermi:encryptSess" minOccurs="0"/>
			<xsd:element ref="ermi:casId" minOccurs="0"/>
			<xsd:element ref="ermi:clientMac" minOccurs="0"/>
			<xsd:element ref="ermi:cciLevel" minOccurs="0"/>
			<xsd:element ref="ermi:apsLevel" minOccurs="0"/>
			<xsd:element ref="ermi:CIT" minOccurs="0"/>
			<xsd:element ref="ermi:encryptESK" minOccurs="0"/>
			<xsd:element ref="ermi:vendorOpaque" minOccurs="0"/>
		</xsd:sequence>
	</xsd:complexType>
	<xsd:element name="EncryptionData" type="ermi:EncryptionDataType"/>
	<xsd:element name="clab-ermi" type="ermi:clab-ermiType"/>
	<xsd:complexType name="clab-ermiType">
```

```
                <xsd:sequence>
                        <xsd:element ref="ermi:EncryptionData"/>
                         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
                </xsd:sequence>
          </xsd:complexType>
</xsd:schema>
```

Appendix I Use Cases

This appendix includes several possible examples of the use of ERMI and the EQAM, ERM and M-CMTS Core devices in operator networks. These examples are not intended to be definitive, but are offered as guidance for implementers and operators.

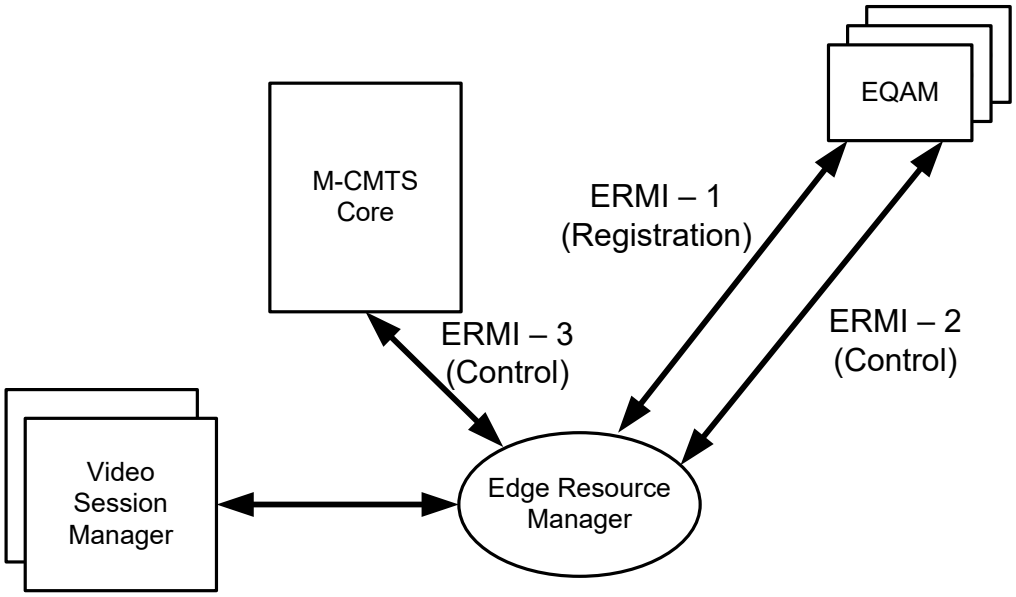


Figure I–1 - Use Case, base architecture

I.1 The M-CMTS obtains a Downstream Resource

At some point, the M-CMTS Core requires access to a new downstream QAM channel to provide data to a service group of modems.

The M-CMTS Core has two ways to obtain QAM channel resources. In the first method, the M-CMTS Core is configured with service group and fiber node information according to DOCSIS 3.0. The method by which it knows the topology of the access network in sufficient detail to create a valid mapping between service groups and fiber nodes is not specified in this document. Most likely, the mapping is pre-configured on to the M-CMTS Core, either manually through a command-line interface or through a management protocol such as SNMPv3.

As an alternative to the previous step, the M-CMTS Core creates a list of the QAM channels that are part of this service group. The M-CMTS Core can use fiber node information to request resources from the ERM because the ERM gains node information from the EQAMs during ERRP registration. The M-CMTS Core would normally remove from this list any QAM channels that it knows will be unavailable (such as ones that it is already using).

The M-CMTS Core must know a valid IP address that it can use to contact the ERM. This address could be manually configured on the M-CMTS Core, or it could be obtained by some automatic means. Reasonable methods by which this could be achieved include the use of options in a DHCP lookup or using SRV records in a DNS lookup. This specification does not define any particular procedure that must be followed by the M-CMTS Core when it obtains the IP address of an ERM.

The M-CMTS Core sends an RTSP SETUP message to the ERM, giving the fiber node list or the details of all the candidate QAM channels in the service group.

The ERM obtains the list of candidate QAM channels either through internal fiber node to QAM channel mapping information or through the explicit QAM channel list in the SETUP request. The ERM examines the list of candidate QAM channels in the SETUP request (which are listed in the order in which the M-CMTS Core prefers

them), and applies any local policy to the listed QAM channel, to select a "best" QAM channel that it believes to be available.

The ERM sends an RTSP SETUP request to the EQAM, identifying the selected QAM channel (only).

The EQAM confirms that the chosen QAM channel is available, and returns a 200 OK response to the SETUP.

The ERM returns a 200 OK response to the M-CMTS Core, and marks this QAM channel as being in use (*i.e.,* unavailable for use if another SETUP arrives and includes this QAM channel in its list). If the EQAM had returned a failure code other than 453 (indicating Insufficient QAM channel bandwidth), the ERM would have selected the "next-best" QAM channel from the candidate list and gone back two steps from here (to step "The ERM sends an RTSP SETUP ") to send an RTSP SETUP request.

Once the M-CMTS Core has received the 200 OK from the ERM, it is free to use the QAM channel.

I.2 The M-CMTS Core releases a Downstream resource

When an M-CMTS Core has finished using a resource, it informs the ERM that it has done so, and the ERM returns it to the pool of resources that may be acquired by M-CMTS Cores.

The M-CMTS Core sends an RTSP TEARDOWN message to the ERM; this message contains the session ID that the M-CMTS Core received from the ERM in the response to its initial SETUP request.

The ERM sends an RTSP teardown message to the EQAM; this message contains the session ID that the ERM received from the EQAM in the response to its initial SETUP request. The EQAM checks whether traffic is still passing on the QAM channel. Assuming that there is no such traffic, it resets the QAM channel configuration parameters to their original values and internally marks the QAM channel as available for use, so that a subsequent request to use it can succeed.

The EQAM sends a successful RTSP TEARDOWN response for the session to the ERM. The ERM marks the QAM channel as being available for subsequent SETUP requests.

The ERM sends a successful RTSP TEARDOWN response to the M-CMTS Core.

I.3 EQAM forces shutdown of a QAM channel

An EQAM may need to shutdown a QAM channel cleanly. Normally, a QAM channel should not be simply removed from service if it is passing traffic; however, this procedure can be used as part of a clean shutdown that will remove a QAM channel from service as soon as it becomes free.

The EQAM sends a ERRP UPDATE message to the ERM. This UPDATE message identifies the QAM channel that is being removed from service. The ERM will update its database to reflect the fact that this QAM channel is no longer available for use.

The EQAM tears down the DEPI control session to the M-CMTS Core (see [DEPI] for details). The M-CMTS Core now initiates the usual teardown sequence:

The M-CMTS Core sends an RTSP TEARDOWN message to the ERM; this message contains the session ID that the M-CMTS Core received from the ERM in the response to its initial SETUP request.

The ERM sends an RTSP teardown message to the EQAM; this message contains the session ID that the M-CMTS Core received from the EQAM in the response to its initial SETUP request. The EQAM checks whether traffic is still passing on the QAM channel. Assuming that there is no such traffic, it resets the QAM channel configuration parameters to their original values and internally marks the QAM channel as available for use, so that a subsequent request to use it can succeed.

The EQAM sends a successful RTSP TEARDOWN response for the session to the ERM. The ERM marks the QAM channel as being available for subsequent SETUP requests.

The ERM sends a successful RTSP TEARDOWN response to the M-CMTS Core.

I.4 Broken connections

I.4.1 ERMI-1 transport connection broken

If the ERRP connection between the EQAM and the ERM unexpectedly goes down, the following series of events and messages would be reasonable.

The ERM detects the ERRP connection broken and does the following for each resource advertised through this ERRP connection:

- If there is no RTSP session using this resource, remove the resource from its database;
- If there is active RTSP sessions using this resource, mark this resource non-operational. The resource is removed from ERM database after all RTSP session timed out or torn down.

The EQAM establishes a new ERRP connection to the ERM.

The EQAM advertises its resources using ERRP UPDATE messages.

For each advertised resource, the ERM performs the following operations: Search in ERM database for this resource. If the resource does not exist, treat this resource as a new resource advertisement and mark it available for use. If the resource does exist in database, change the resource state to operational.

I.4.2 ERMI-2 transport connection broken

If the RTSP transport connection between the EQAM and the ERM unexpectedly goes down. The ERM attempts to establish a new transport connection to EQAM. Once a new transport connection has been created, both devices proceed as if the transport connection was never lost.

The RTSP Client, upon reestablishing the TCP connection, should send a GET_PARAMETER, connection_timeout request and also a GET_PARAMETER session_list request to resynchronize with the RTSP Server per Section 7.11.4.

I.4.3 ERMI-3 transport connection broken

If the RTSP transport connection between the M-CMTS Core and the ERM unexpectedly goes down, the M-CMTS Core establishes a new transport connection to the ERM and both devices proceed as if the transport connection was never lost.

The RTSP Client, upon reestablishing the TCP connection, should send a GET_PARAMETER, connection_timeout request and also a GET_PARAMETER session_list request to resynchronize with the RTSP Server per Section 7.11.4.

I.5 Device failures

I.5.1 Complete EQAM failure

In the event that an EQAM completely fails and there is no automatic failover to an alternative EQAM, the M-CMTS Core will detect that the DEPI session has been broken (see[DEPI]).

The M-CMTS Core sends one or more RTSP TEARDOWN messages to the ERM, tearing down all RTSP sessions that were using QAM channels on the EQAM that has failed. These messages contain the session IDs that the M-CMTS Core received from the ERM in the response to its initial SETUP requests.

ERRP connection will also fail in this situation. The procedure for handling ERRP connection failure should be followed.

I.5.2 Complete M-CMTS Core failure

In the event that an M-CMTS Core completely fails, the EQAM will detect that the DEPI session has been broken (see [DEPI]). The RTSP session between the ERM and the M-CMTS Core will timeout (since Keepalive messages are no longer being sent), which causes the ERM to teardown the RTSP sessions to this M-CMTS Core. ERM also tears down RTSP sessions to EQAM to release all the resources used by this M-CMTS Core.

I.5.3 Complete ERM failure

In the event that an ERM fails completely, the DEPI sessions between the M-CMTS Core and the EQAM are unaffected.

The M-CMTS Core should maintain the RTSP session to the ERM until the DEPI session ends.

The EQAM should maintain the RTSP session to the ERM until the DEPI session ends.

I.6 Device reboots

I.6.1 EQAM reboot

In the event that an EQAM reboots, it will behave exactly as it did on initial boot. The EQAM is not required or expected to maintain any non-configured state information across boots.

I.6.2 M-CMTS Core reboot

In the event that an M-CMTS Core reboots, the EQAM and the ERM will detect loss of the M-CMTS Core and will proceed as in Section I.5.2. The EQAM does not release the QAM channel resources until the ERM instructs it to do so, following the timeout of the RTSP session with the M-CMTS Core. The M-CMTS Core is not required to maintain any non-configured state information across boots.

When the M-CMTS Core reboots, it sends a GET_PARAMETER RTSP request to the ERM to request the session list for each of its associated SessionGroup identifiers.

The ERM responds by identifying the current RTSP sessions between these two devices. If the CMTS persists the state of its allocated sessions across reboots, then it should subsequently send a TEARDOWN for each session ID that it does not recognize. Depending on how quickly the M-CMTS Core reboots and the values of the session time-outs, some of the sessions may have been torn down while it rebooted.

If the CMTS does not persist the state of its allocated sessions across reboots, then it should subsequently send a TEARDOWN for all of the session IDs in order to clean up its previously allocated resources.

I.6.3 ERM reboot

In the event that the ERM reboots, the EQAM and the M-CMTS Core will detect loss of the ERM and will proceed as in sections <ERRP loss, ERMI-2 loss, ERMI-3 loss, ERM failure>. When the ERM reboots, it sends a GET_PARAMETER RTSP request to the EQAM. The EQAM responds by identifying the current RTSP sessions between these two devices. Depending on how quickly the ERM reboots and the values of the session time-outs, some of the sessions may have been torn down while it rebooted. Normally, it will explicitly tear down any sessions identified by the response to the GET_PARAMETER request.

I.7 Video on Demand

In the classic VOD use case, the ERM sends an RTSP SETUP Request Message to allocate bandwidth for a unicast stream. The RTSP SETUP request to the EQAM will include both an MPEG-TS/IP transport header including the IP address + UDP port number of the EQAM selected for the session as well as an MPEG-TS/QAM transport header which specifies the TSID and MPEG program number to be used at the output of the QAM multiplex. In addition, the ERM will specify the bandwidth allocated to the stream with the Bandwidth header. The EQAM may use this bandwidth value to police the MPEG-TS/UDP stream at the input to the multiplex.

The EQAM will set up the datapath from the input IP interface through the multiplex to the outgoing RF interface, and return an RTSP SETUP Response via ERMI. The message must indicate success or failure – if the command execution was successful, the parameters specified in the SETUP may be echoed back in the message.

The ERM must avoid bandwidth and program number conflicts on a specific multiplex during the session setup process.

When the ERM receives a SETUP request from the VOD Session Manager, the ERM will select the best QAM channel to use for the session based on the TSID/QAM Name list passed to it in the resource allocation request and based on the pre-allocated streams on the EQAM. The EQAM sends an RTSP SETUP response message back via

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):

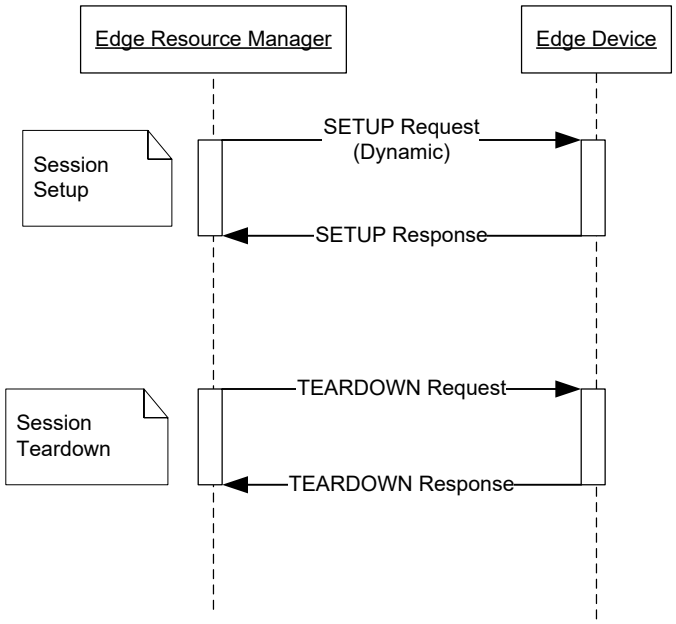


Figure I–2 - Dynamic Session Lifecycle

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 an video channel, the EQAM allocates resources for the stream and issues an 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.

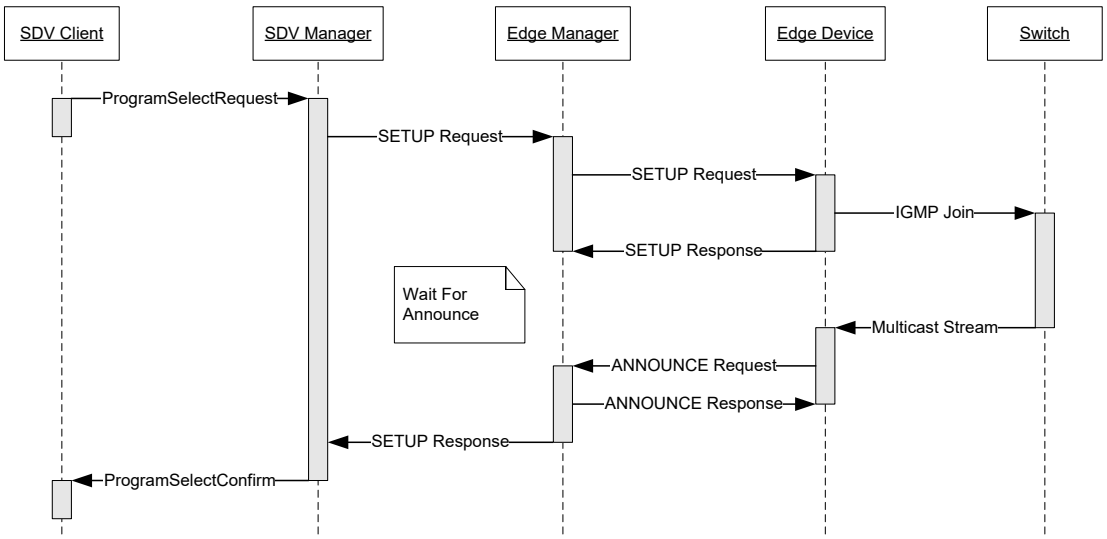


Figure I–3 - Message Flow for Synchronous Mode

The figure below displays the message flow for asynchronous mode.

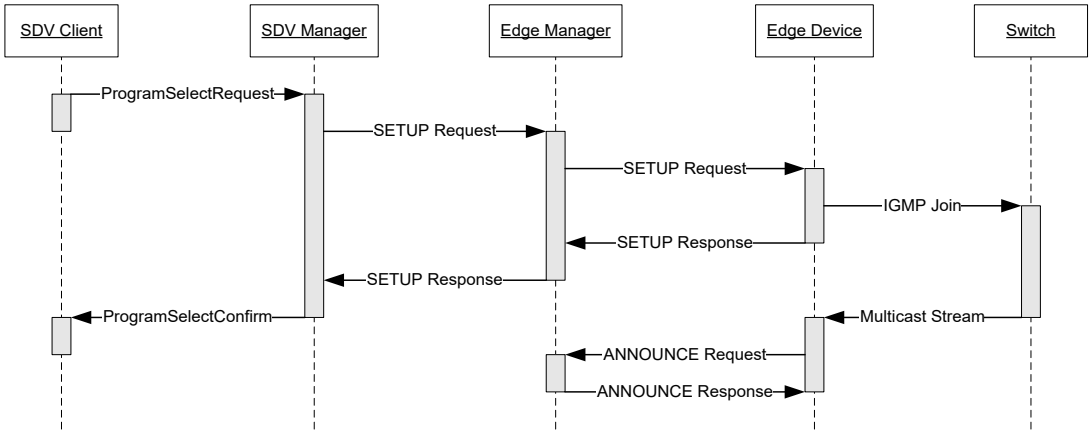


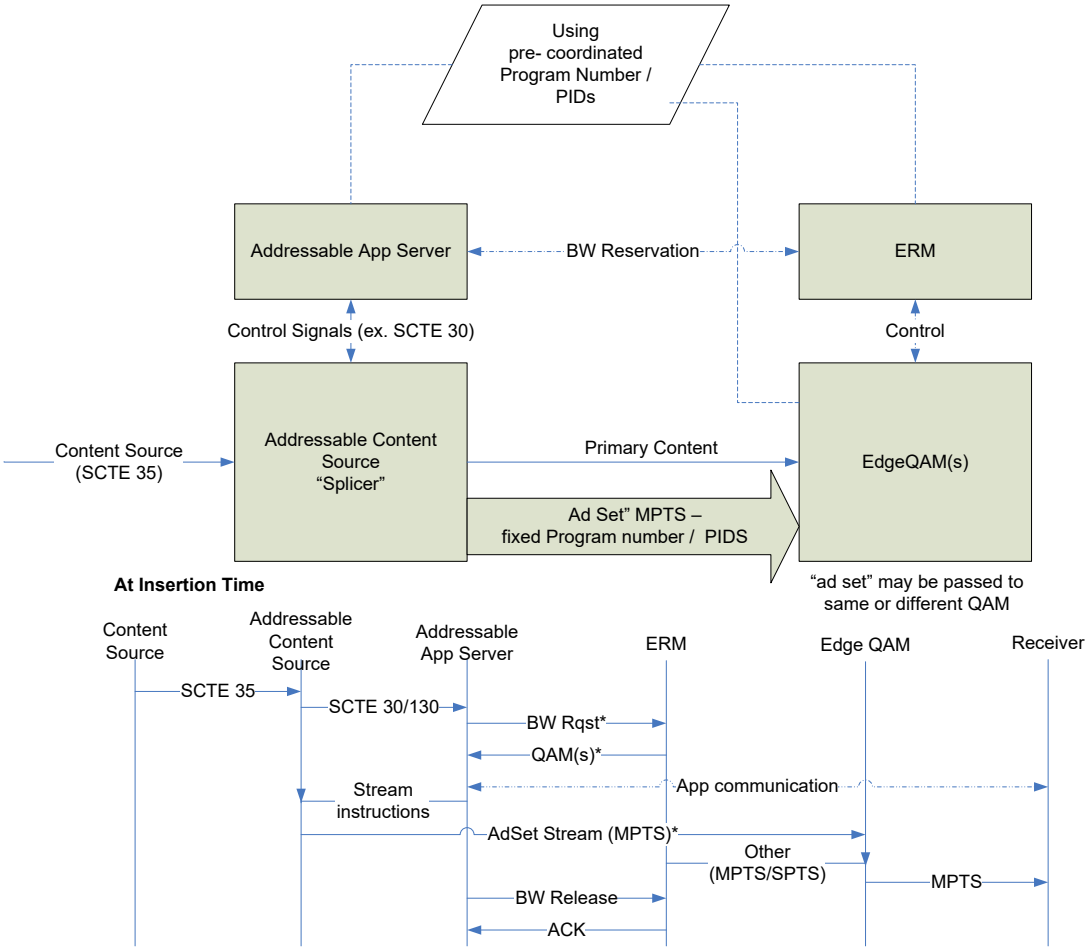
Figure I–4 - Message Flow for Asynchronous Mode

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.