8. Connection Management

8.1. Overview

Connection management is the process of setting up, monitoring and tearing down connections between stream sinks and stream sources.

An AVDECC Connection is uniquely identified by the following quadruplet:

— listener_entity_id

— listener_unique_id

— talker_entity_id

— talker_unique_id

A given (listener_entity_id, listener_unique_id) pair can be involved in no more than one connection at the same time. A given (talker_entity_id, talker_unique_id) pair can be involved in any number of connections at the same time.

At a given time, each different AVDECC Entity may have a different view of the set of AVDECC Connections that exist in the network and their attributes, even the two AVDECC Entities that are involved in the same connection. When an AVDECC Controller is present in the network, it can be used to detect such discrepencies and resynchronize (with some input from the user) the AVDECC Entities so that they have the same view of the connections.

The attributes of an AVDECC Connection are as follows:

— stream_id

— stream_dest_mac

— stream_vlan_id

— CLASS_B

— ENCRYPTED_PDU

At a given time, each stream sink of an AVDECC Simple Listener is either "not connected", or "connected". The stream sink goes from "not connected" to "connected" upon successful processing of a CONNECT_TX_RESPONSE message. It goes from "connected" to "not connected" upon successful processing of a DISCONNECT_RX_COMMAND message. Please see the ACMP Simple Listener state machine for details.

At a given time, each stream sink of an AVDECC Smart Listener is either "not connected, or "bound and not connected", or "connected". The "bound and not connected" state is an intermediate state which is specific to AVDECC Smart Listeners. In this state, the AVDECC Smart Listener has received and saved instructions from the AVDECC Controller and is in the process of connecting to the AVDECC Talker. The stream sink goes to "bound and not connected" upon successful processing of a CONNECT_RX_COMMAND message, or after any disruption of the connection detected by the AVDECC Smart Listener. It goes to "connected" upon successful processing of a CONNECT_TX_RESPONSE message. It returns to "not connected" upon successful processing of a DISCONNECT_RX_COMMAND message. Please see the ACMP Smart Listener state machine for details.

At a given time, each stream source of an AVDECC Connection-aware Talker is either "not connected", or "connected". When the stream source is connected, the AVDECC Connection-aware Talker maintains a list of all the connected stream sinks. Please note that in the absence of an AVDECC Controller to periodically resynchronize the views of all the AVDECC Entities in the network, this list may contain AVDECC Listener sinks that are not connected to this stream source, as well as omit AVDECC Listener sinks that are connected to this stream source. The first situation may occur after a reboot of a connected AVDECC Simple Listener. The second situation may occur after a reboot of the AVDECC Connection-aware Talker.

An AVDECC Connection-unaware Talker does not keep any information about connections. All of its output streams are multicast-only and the way they are started and stopped is beyond the scope of this standard.

8.2. AVDECC Connection Management Protocol

8.2.1. AVDECC Connection Management Protocol Data Unit format

The AVDECC Connection Management Protocol Data Unit (ACMPDU) follows the IEEE Std 1722-2016 control AVTPDU header.

The ACMPDU contains the following fields:

— **controller_entity_id**: 64 bits
— **talker_entity_id**: 64 bits
— **listener_entity_id**: 64 bits
— **talker_unique_id**: 16 bits
— **listener_unique_id**: 16 bits
— **stream_dest_mac**: 48 bits
— **connection_count**: 16 bits
— **sequence_id**: 16 bits
— **flags**: 16 bits
— **stream_vlan_id**: 16 bits
— **reserved**: 16 bits

Figure 8-1 shows these fields with offset zero (0) shown as the first octet of the ACMPDU.

The AVDECC Connection Management Protocol redefines the **control_data** as the **message_type** field within the IEEE Std 1722-2016 control AVTPDU header.

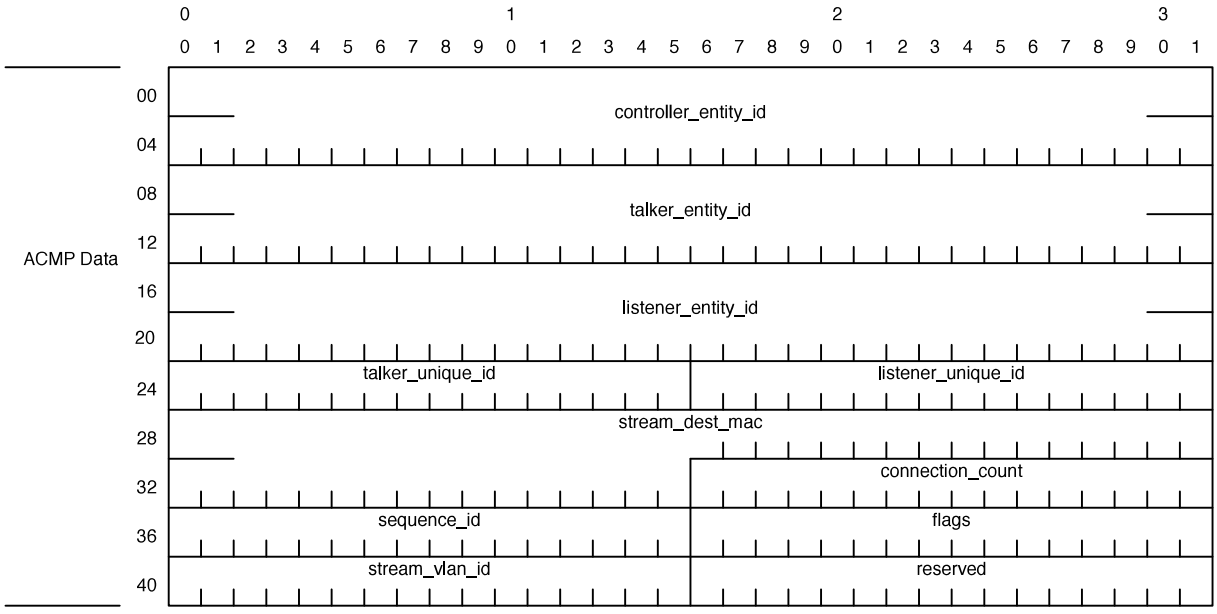

FIGURE 8-1—ACMPDU format

8.2.1.1. cd field

The IEEE Std 1722-2016 control AVTPDU **cd** (control/data) bit is set to one (1).

8.2.1.2. subtype field

The IEEE Std 1722-2016 control AVTPDU **subtype** field is set to the ACMP subtype, $7C_{16}$.

8.2.1.3. sv field

The IEEE Std 1722-2016 control AVTPDU **sv** (stream valid) bit is set to zero (0).

8.2.1.4. version field

The IEEE Std 1722-2016 control AVTPDU **version** field is set to the version of IEEE 1722 being used.

8.2.1.5. message_type (control_data) field

The IEEE Std 1722-2016 control AVTPDU **control_data** field is renamed to **message_type** in ACMP. The message type indicates if it is a command or response message and the type of command. It is set to one of the values as defined in Table 8-1:

TABLE 8-1—message_type field

Value	Function	Meaning
0	CONNECT_TX_COMMAND	Connect/Probe Talker source stream command
1	CONNECT_TX_RESPONSE	Connect/Probe Talker source stream response
2	DISCONNECT_TX_COMMAND	Disconnect Talker source stream command
3	DISCONNECT_TX_RESPONSE	Disconnect Talker source stream response
4	GET_TX_STATE_COMMAND	Get Talker source stream connection state command
5	GET_TX_STATE_RESPONSE	Get Talker source stream connection state response
6	CONNECT_RX_COMMAND	Connect/Bind Listener sink stream command
7	CONNECT_RX_RESPONSE	Connect/Bind Listener sink stream response
8	DISCONNECT_RX_COMMAND	Disconnect/Unbind Listener sink stream command
9	DISCONNECT_RX_RESPONSE	Disconnect/Unbind Listener sink stream response
10	GET_RX_STATE_COMMAND	Get Listener sink stream connection state command
11	GET_RX_STATE_RESPONSE	Get Listener sink stream connection state response
12	GET_TX_CONNECTION_COMMAND	Get a specific Talker connection info command
13	GET_TX_CONNECTION_RESPONSE	Get a specific Talker connection info response
14-15	—	Reserved for future use

8.2.1.6. status field

The IEEE Std 1722-2016 control AVTPDU **status** field is used to carry the result status of the command in the response frame. The status field is set to 0 (SUCCESS) in a command frame and set to an appropriate value from Table 8-2 for a response.

TABLE 8-2—status field

Value	Function	Meaning
0	SUCCESS	Command executed successfully
1	LISTENER_UNKNOWN_ID	Listener does not have the specified unique identifier
2	TALKER_UNKNOWN_ID	Talker does not have the specified unique identifier
3	TALKER_DEST_MAC_FAIL	Talker could not allocate a destination MAC for the stream
4	TALKER_NO_STREAM_ID	Talker does not have an available stream ID for the stream
5	TALKER_NO_BANDWIDTH	Talker could not allocate bandwidth for the stream
6	TALKER_EXCLUSIVE	Talker already has an established stream and only supports one Listener
7	LISTENER_TALKER_TIMEOUT	Listener had timeout for all retries when trying to send command to Talker
8	LISTENER_EXCLUSIVE	The AVDECC Listener already has an established connection to a stream.
9	STATE_UNAVAILABLE	Could not get the state from the AVDECC Entity
10	NOT_CONNECTED	Trying to disconnect when not connected or not connected to the AVDECC Talker specified.
11	NO_SUCH_CONNECTION	Trying to obtain connection info for an AVDECC Talker connection which does not exist.
12	COULD_NOT_SEND_MESSAGE	The AVDECC Listener failed to send the message to the AVDECC Talker.
13	TALKER_MISBEHAVING	Talker was unable to complete the command because an internal error occurred.
14	LISTENER_MISBEHAVING	Listener was unable to complete the command because an internal error occurred.
15	—	Reserved for future use.

continued on next page

TABLE 8-2—status field (*continued*)

Value	Function	Meaning
16	CONTROLLER_NOT_AUTHORIZED	The AVDECC Controller with the specified Entity ID is not authorized to change stream connections.
17	INCOMPATIBLE_REQUEST	The AVDECC Listener is trying to connect to an AVDECC Talker that is already streaming with a different traffic class, etc. or does not support the requested traffic class.
18-30	—	Reserved for future use
31	NOT_SUPPORTED	The command is not supported.

8.2.1.7. control_data_length field

The IEEE Std 1722-2016 control AVTPDU **control_data_length** field for ACMP is set to 44 for this version.

8.2.1.8. stream_id field

The IEEE Std 1722-2016 control AVTPDU **stream_id** field is used to identify and transfer the associated stream ID where suitable. **stream_id** is used mostly in responses rather than commands.

8.2.1.9. controller_entity_id field

The **controller_entity_id** field is used to identify the AVDECC Controller responsible for sending the command so that it can match a response. This field is set to the entity_id of the AVDECC Controller initiating the command.

Fast Connect and Fast Disconnect commands shall set this to the Entity ID of the initiator of the connection.

8.2.1.10. talker_entity_id field

The **talker_entity_id** field is used to identify the AVDECC Talker being targeted by the command. In the case of Talker commands this is the AVDECC Entity receiving the command, in the case of Listener commands this is the AVDECC Entity that any Talker commands is to be sent to. This field is either the Entity ID of the AVDECC Entity being targeted or zero (0).

8.2.1.11. listener_entity_id field

The **listener_entity_id** is used to identify the AVDECC Listener being targeted by the command. In the case of Talker commands this field is ignored, in the case of Listener commands this is the AVDECC Entity receiving the command. This field is either the Entity ID of the AVDECC Entity being targeted or zero (0).

8.2.1.12. talker_unique_id field

The **talker_unique_id** field is a 16 bit value used to uniquely identify the stream source of the AVDECC Talker.

For entities using the AVDECC Entity Model, this corresponds to the descriptor index of the STREAM_OUTPUT descriptor in the currently active CONFIGURATION.

For entities using IEEE 1394 AV/C, this corresponds to the output isoch plug number.

For other Entity models the meaning of the value of this field is left to the model to determine.

8.2.1.13. listener_unique_id field

The **listener_unique_id** field is a 16 bit value used to uniquely identify the stream sink of the AVDECC Listener.

For entities using the AVDECC Entity Model, this corresponds to the descriptor index of the STREAM_INPUT descriptor in the currently active CONFIGURATION.

For entities using IEEE 1394 AV/C, this corresponds to the input isoch plug number.

For other Entity models the meaning of the value of this field is left to the model to determine.

8.2.1.14. stream_dest_mac field

The **stream_dest_mac** field is used to convey the destination MAC address for a stream from the AVDECC Talker to the AVDECC Listener, or from either to the AVDECC Controller.

8.2.1.15. connection_count field

In a GET_RX_STATE_RESPONSE message sent by an AVDECC Simple Listener, the **connection_count** field is used to indicate whether the stream sink of the AVDECC Listener is connected or not.

In a GET_RX_STATE_RESPONSE message sent by an AVDECC Smart Listener, the **connection_count** field is used to indicate whether the stream sink of the AVDECC Listener is bound or not.

In a GET_TX_STATE_RESPONSE message sent by an AVDECC Connection-aware Talker, the **connection_count** field is used to return the number of Listeners the AVDECC Talker thinks are connected to its stream source, i.e. the number of connect TX stream commands it has received less the number of disconnect TX stream commands it has received. This number may not be accurate since an AVDECC Entity may not have sent a disconnect command if its network connection was disconnected or it was abruptly powered down.

In a GET_TX_STATE_RESPONSE message sent by an AVDECC Connection-unaware Talker, the **connection_count** field is always set to 0.

In a GET_TX_CONNECTION_COMMAND message, the **connection_count** field is used to indicate the index of the connection the AVDECC Controller wishes to query. The AVDECC Connection-aware Talker copies the contents of this field into the associated GET_TX_CONNECTION_RESPONSE message.

8.2.1.16. sequence_id field

The **sequence_id** field is incremented on each command sent, and is used to identify the command that a response is for. The **sequence_id** may be initialized to any value after power up.

8.2.1.17. flags field

The **flags** field is used to indicate attributes of the connection or saved state, it is a bit field and is set to an appropriate combination of flags as defined in Table 8-3:

TABLE 8-3—flags field

Bit	Field Value	Function	Meaning
15	0001_{16}	CLASS_B	Indicates that the stream is Class B instead of Class A (default 0 is class A)
14	0002_{16}	FAST_CONNECT	Indicates that the AVDECC Listener is executing a Fast Connect sequence instead of a Controller Connect sequence.
13	0004_{16}	SAVED_STATE	Connection has saved state (used only in GET_RX_STATE_RESPONSE)

continued on next page

TABLE 8-3—flags field (*continued*)

Bit	Field Value	Function	Meaning
12	0008_{16}	STREAMING_WAIT	The AVDECC Listener, if it supports this option, does not process the stream packets until explicitly being told to by the control protocol, and the AVDECC Talker, if it supports this option, does not start streaming until explicitly being told to by the control protocol.
11	0010_{16}	SUPPORTS_ENCRYPTED	Indicates that the stream supports streaming with encrypted PDUs.
10	0020_{16}	ENCRYPTED_PDU	Indicates that the stream is using encrypted PDUs.
9	0040_{16}	REGISTRATION_FAILED	Indicates that the listener has registered an SRP Talker Failed attribute for the stream (used in GET_RX_STATE_RESPONSE) or that the talker has registered an SRP Listener Asking Failed attribute for the stream (used in GET_TX_STATE_RESPONSE and GET_TX_CONNECTION_RESPONSE).
0-8	—	—	Reserved for future use

8.2.1.18. stream_vlan_id

The **stream_vlan_id** field is used to convey the VLAN ID for a stream from the AVDECC Talker to the AVDECC Listener, or from either to the AVDECC Controller.

The **stream_vlan_id** field shall be set to zero (0) to indicate that the VLAN ID specified in the SRP Domain attribute is being used, otherwise it is set to the statically assigned VLAN ID used for the stream.

** What is the intent of using value 0 in stream_vlan_id ??? It it to allow the controller to decide whether the talker should use a statically imposed value or take the vlan id from the Domain atttribute? Or is for the talker to be able to to follow the variations of the Domain attribute ? In the latter case, is it expected that the talker withdraws and declares a new Talker attribute when the Domain attribute changes? Is it expected that the listener automatically joins the new VLAN when its Domain attribute changes also? What if the Default VLAN ID in the Domain attributes are different at the talker's and the listener's ??? **

8.2.2. Protocol Specification

All ACMPDUs are transmitted to the ACMP multicast destination MAC address defined in Table 2-1.

ACMP uses timeouts, sequence IDs and a retry to provide a reliability mechanism. Commands and responses can be matched by looking primarily at the AVDECC controller_entity_id, sequence_id and message_type fields but the talker_entity_id, talker_unique_id, listener_entity_id and listener_unique_id fields can also be used for additional matching.

The timeout periods used by ACMP vary based on the command type. Table 8-4 details the timeout periods for each command.

TABLE 8-4—ACMP command timeouts

Function	Timeout Period	Notes
CONNECT_TX_COMMAND	2000 milliseconds	The AVDECC Talker may need to perform IEEE Std 1722-2011 MAAP allocation which takes at minimum 1.5 seconds.
DISCONNECT_TX_COMMAND	200 milliseconds	—
GET_TX_STATE_COMMAND	200 milliseconds	—
CONNECT_RX_COMMAND	4500 milliseconds	The AVDECC Listener may need to perform a retry of the CONNECT_TX_COMMAND.
DISCONNECT_RX_COMMAND	500 milliseconds	The AVDECC Listener may need to perform a retry of the DISCONNECT_TX_COMMAND.
GET_RX_STATE_COMMAND	200 milliseconds	—
GET_TX_CONNECTION_COMMAND	200 milliseconds	—

8.2.2.1. ACMP Sequences

There are nine different ACMP sequences: Controller Connect, Controller Disconnect, Controller Bind, Controller Unbind, Fast Connect, Fast Disconnect, Get Sink State, Get Source State and Get Connection Info.

8.2.2.1.1. Controller Connect

The Controller Connect sequence is used to connect a stream between an AVDECC Simple Listener and an AVDECC Talker.

An AVDECC Simple Listener relies on the AVDECC Controller to manage errors during or after the connection process with the AVDECC Talker, and does not save the connection parameters in a non volatile memory.

The Controller Connect sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Simple Listener a CONNECT_RX_COMMAND containing the connection parameters. Then it waits for the response from the AVDECC Simple Listener.

b) The AVDECC Simple Listener does basic verifications and propagates the command to the AVDECC Talker, after changing the type from CONNECT_RX_COMMAND to CONNECT_TX_COMMAND. Then it waits for the response from the AVDECC Talker. In case of a timeout of the first CONNECT_TX_COMMAND, the AVDECC Simple Listener sends a single retry. If this command fails then the AVDECC Simple Listener returns failure to the AVDECC Controller.

c) The AVDECC Talker does basic verifications, potentially initiates stream reservation/transmission and sends to the AVDECC Simple Listener a CONNECT_TX_RESPONSE containing the stream parameters.

d) The AVDECC Simple Listener does basic verifications, extracts the stream parameters, potentially initiates stream reservation, prepares to receive the stream and propagates the response to the AVDECC Controller, after changing the type from CONNECT_TX_RESPONSE to CONNECT_RX_RESPONSE.

After the sequence above is finished, the AVDECC Controller is responsible to handle any connection error or connection disruption that might happen for any reason.

Please note that the CONNECT_TX_COMMAND message transmitted by the AVDECC Simple Listener has the FAST_CONNECT bit set to 0.

8.2.2.1.2. Controller Disconnect

The Controller Disconnect sequence is used to disconnect a stream from an AVDECC Simple Listener.

The Controller Disconnect sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Simple Listener a DISCONNECT_RX_COMMAND mentioning which connection has to be closed. Then it waits for the response from the AVDECC Simple Listener.

b) The AVDECC Simple Listener does basic verifications, potentially stops stream reservation and stream reception, and propagates the command to the AVDECC Talker, after changing the type from DISCONNECT_RX_COMMAND to DISCONNECT_TX_COMMAND. Then it waits for the response from the AVDECC Talker. In case of a timeout of the first DISCONNECT_TX_COMMAND, the AVDECC Simple Listener sends a single retry. If this command fails then the AVDECC Simple Listener returns failure to the AVDECC Controller.

c) The AVDECC Talker does basic verifications, potentially stops stream reservation/transmission and sends to the AVDECC Simple Listener a DISCONNECT_TX_RESPONSE.

d) The AVDECC Simple Listener does basic verifications and propagates the response to the AVDECC Controller, after changing the type from DISCONNECT_TX_RESPONSE to DISCONNECT_RX_RESPONSE.

After the sequence above is finished, the AVDECC Controller is responsible to handle any potential disconnection error.

8.2.2.1.3. Controller Bind

The Controller Bind sequence is used to provide an AVDECC Smart Listener with the connection parameters and let it manage the connection by itself.

An AVDECC Smart Listener is capable of autonomously recovering from errors during the connection/disconnection process and any disruption of the connection that might happen for any reason at any time. An AVDECC Smart Listener also saves the connection parameters in a non volatile memory and automatically restores its active connections after a power cycle. Due to this controller-like set of capabilities, an AVDECC Smart Listener is not subject to the same requirements as an AVDECC Simple Listener. Please see the state machines for details.

The Controller Bind sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Smart Listener a CONNECT_RX_COMMAND containing the connection parameters. Then it waits for the response from the AVDECC Smart Listener.

b) The AVDECC Smart Listener does basic verifications, notes that the target sink is bound, saves the connection parameters in a non volatile memory, and sends a CONNECT_RX_RESPONSE to the AVDECC Controller. Please note that this message does not contain the Stream parameters (stream_id, stream_dest_mac, stream_vlan_id) as the AVDECC Smart Listener has not inquired the AVDECC Talker yet.

After the sequence above is finished, the AVDECC Smart Listener is responsible to establish the connection with the AVDECC Talker (see the Fast Connect sequence) and handle any connection error or connection disruption that might happen for any reason. After receiving the CONNECT_RX_RESPONSE, the AVDECC Controller is not necessary for monitoring the connection and can safely be unplugged from the network.

Note: the way the AVDECC Smart Listener detects and handles errors or disruptions is beyond the scope of this standard.

** How does the AVDECC Controller know that the AVDECC Listener is a smart listener? Should it detect that by inspecting the CONNECT_RX_RESPONSE (the stream_id, stream_dest_mac and stream_vlan_id are all zeros? Or the FAST_CONNECT bit is set to 1?) or should we add a bit in the capabilities of the listener? **

8.2.2.1.4. Controller Unbind

The Controller Unbind sequence is used to clear the saved connection parameters of an AVDECC Smart Listener.

The Controller Unbind sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Smart Listener a DISCONNECT_RX_COMMAND mentioning which sink has to be unbound. Then it waits for the response from the AVDECC Smart Listener.

b) The AVDECC Smart Listener does basic verifications, clears the saved connection parameters for this sink, notes that the sink is unbound and sends a DISCONNECT_RX_RESPONSE to the AVDECC Controller.

After the sequence above is finished, the AVDECC Smart Listener is responsible to withdraw the connection with the AVDECC Talker. How it does that is beyond the scope of this standard.

8.2.2.1.5. Fast Connect

The Fast Connect sequence is used by AVDECC Smart Listeners in the "bound and not connected" state in order to set up a connection with the AVDECC Talker.

Note: this standard does not specify when exactly an AVDECC Smart Listener triggers a Fast Connect sequence. A reasonable implementation would trigger this after a Controller Bind sequence, after a reboot or after somehow detecting a disruption of the connection. The implementation may use ADP to check that the AVDECC Talker is present on the network and using the same gPTP domain before triggering the Fast Connect sequence.

An AVDECC Smart Listener performing a Fast Connect sequence sends a CONNECT_TX_COMMAND message that has the same format as that used in the Controller Connect sequence, except that the FAST_CONNECT bit is set to 1. If the first command times out in the normal timeout period, then the command is retried. When the AVDECC Smart Listener receives a successful response from the AVDECC Talker, it extracts the stream parameters, initiates stream reservation and prepares to receive the stream. The AVDECC Smart Listener does not send any asynchronous message to notify the AVDECC Controller.

Note: the behavior of the AVDECC Smart Listener in case of a second timeout or a failing response in the Fast Connect sequence is not specified by this standard.

8.2.2.1.6. Fast Disconnect

The Fast Disconnect sequence is used by AVDECC Listeners that support the Clean Shutdown feature to automatically disconnect a connected stream sink at poweroff time.

An AVDECC Listener performing a Fast Disconnect sequence sends a DISCONNECT_TX_COMMAND message to the AVDECC Talker before powering off. If the first command times out in the normal timeout period, then the command is retried. The AVDECC Listener does not send any asynchronous message to notify the AVDECC Controller.

Note: contrary to the Fast Connect sequence, the usage of the Fast Disconnect sequence is not reserved to AVDECC Smart Listeners. Any AVDECC Listener supporting the Clean Shutdown feature can use it.

** Should we add a bit in the capabilities of the listener to indicate that the listener supports Clean Shutdown? **

8.2.2.1.7. Get Sink State

The Get Sink State sequence is used to get the state of a sink of an AVDECC Listener.

The Get Sink State sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Listener a GET_RX_STATE_COMMAND mentioning the target sink. Then it waits for the response from the AVDECC Listener.

b) The AVDECC Listener does basic verifications, retrieves the current state of the sink (connected/bound state, connection parameters and stream parmeters when applicable) and sends a GET_RX_STATE_RESPONSE to the AVDECC Controller.

8.2.2.1.8. Get Source State

The Get Source State sequence is used to get the state of a source of an AVDECC Talker.

The Get Source State sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Talker a GET_TX_STATE_COMMAND mentioning the target source. Then it waits for the response from the AVDECC Talker.

b) The AVDECC Talker does basic verifications, retrieves the current state of the source (number of connections, if supported, and stream parameters when applicable) and sends a GET_TX_STATE_RESPONSE to the AVDECC Controller.

8.2.2.1.9. Get Connection Info

The Get Connection Info sequence is used to get information about a specific connection between a source of an AVDECC Connection-aware Talker and a particular sink of an AVDECC Listener.

The Get Connection Info sequence is as follows:

a) The AVDECC Controller sends to the AVDECC Connection-aware Talker a GET_TX_CONNECTION_COMMAND mentioning the target source and the target connection. The target connection is identified by a number between 0 and the number of connections the AVDECC Connection-aware Talker thinks it has, minus 1. Then the AVDECC Controller waits for the response from the AVDECC Talker.

b) The AVDECC Connection-aware Talker does basic verifications, retrieves the current information about the target source and the target connection (Entity ID and Unique ID of the connected AVDECC Listener) and sends a GET_TX_CONNECTION_RESPONSE to the AVDECC Controller.

Note: AVDECC Connection-unaware Talkers do not support the GET_TX_CONNECTION_COMMAND.

8.2.2.2. State machine types

8.2.2.2.1. ACMPCommandResponse

The ACMPCommandResponse type is a structure containing the fields from the ACMPDU required to be able to construct or use an ACMPDU.

The ACMPCommandResponse structure contains the following fields:

— **message_type**: 4 bits
— **status**: 5 bits
— **stream_id**: 64 bits
— **controller_entity_id**: 64 bits
— **talker_entity_id**: 64 bits
— **listener_entity_id**: 64 bits
— **talker_unique_id**: 16 bits
— **listener_unique_id**: 16 bits
— **stream_dest_mac**: 48 bits
— **stream_vlan_id**: 16 bits
— **connection_count**: 16 bits
— **sequence_id**: 16 bits
— **flags**: 16 bits

8.2.2.2.2. InflightCommand

The InflightCommand type is a structure which is used to save the state of commands which are in flight (the command has been sent but the response has not yet been received and retries have not yet finally timed out). It includes:

— **timeout**: A timer (timeout value) for when the command will timeout.

— **retried**: 1 bit (boolean) indicating if a retry has been sent.

— **command**: The ACMPCommandResponse that was sent.

— **original_sequence_id**: 16 bits, the sequence_id from the command which generated this command (for Listener commands which generate Talker commands).

8.2.2.2.3. StreamParams

The AVDECC StreamParams type is a structure containing the description of a stream. It includes:

— **stream_id**: 64 bits

— **stream_dest_mac**: 48 bits

— **stream_vlan_id**: 16 bits

— **CLASS_B**: 1 bit

— **ENCRYPTED_PDU**: 1 bit

8.2.2.2.4. ListenerBindingParams

The AVDECC ListenerBindingParams type is a structure containing the description of the connection as requested by an AVDECC Controller to an AVDECC Listener. It includes:

— **talker_entity_id**: 64 bits

— **talker_unique_id**: 16 bits

— **controller_entity_id**: 64 bits

8.2.2.2.5. SimpleListenerSinkContext

The AVDECC SimpleListenerSinkContext type is a structure containing the information required for the AVDECC Simple Listener to maintain state on a stream sink. It includes:

— **inflight_used**: 1 bit

— **inflight**: InflightCommand variable.

— **connected**: 1 bit

— **binding_params**: ListenerBindingParams variable.

— **stream_params**: StreamParams variable.

— **STREAMING_WAIT**: 1 bit

— **REGISTERING_FAILED**: 1 bit

8.2.2.2.6. SmartListenerSinkContext

The AVDECC SmartListenerSinkContext type is a structure containing the information required for the AVDECC Smart Listener to maintain state on a stream sink. It includes:

— **bound**: 1 bit

— **connected**: 1 bit

— **binding_params**: ListenerBindingParams variable.

— **stream_params**: StreamParams variable.

— **STREAMING_WAIT**: 1 bit

— **REGISTERING_FAILED**: 1 bit

8.2.2.2.7. ListenerPair

The AVDECC ListenerPair type is a structure containing two fields to track a connected Listener. It includes:

— **listener_entity_id**: 64 bits

— **listener_unique_id**: 16 bits

8.2.2.2.8. ConTalkerSourceContext

The AVDECC ConTalkerSourceContext type is a structure containing the information required for the AVDECC Connection-aware Talker to maintain state on a stream source. It includes:

— **connected_listeners**: dynamic array of ListenerPair structs.

— **stream_params**: StreamParams variable.

— **STREAMING_WAIT**: 1 bit

— **REGISTERING_FAILED**: 1 bit

8.2.2.2.9. NconTalkerSourceContext

The AVDECC NconTalkerSourceContext type is a structure containing the information required for the AVDECC Connection-unaware Talker to maintain state on a stream source. It includes:

— **stream_params**: StreamParams variable.

— **REGISTERING_FAILED**: 1 bit

8.2.2.2.10. ACMPCommandParams

The ACMPCommandParams type is a structure containing the information required for the AVDECC Controller to make a command to be sent. It includes:

— **message_type**: 4 bits

— **talker_entity_id**: 64 bits

— **listener_entity_id**: 64 bits

— **talker_unique_id**: 16 bits

— **listener_unique_id**: 16 bits

— **connection_count**: 16 bits

— **flags**: 16 bits

— **stream_vlan_id**: 16 bits ** Why do we need this field ??? Can we remove it from this structure ? **

8.2.2.3. State machine global variables

8.2.2.3.1. my_id

The my_id variable is a 64 bit value containing the AVDECC Entities Entity ID.

8.2.2.3.2. rcvdCmdResp

The rcvdCmdResp variable is a structure of type ACMPCommandResponse containing the next received ACMPDU to be processed.

8.2.2.4. ACMP Controller state machine

The AVDECC Controller state machine describes the active participation of the AVDECC Controller in the ACMP conversation. An AVDECC Controller may, independently of these state machines, monitor all received ACMP messages for tracking the state of connections on the network.

8.2.2.4.1. State machine variables

8.2.2.4.1.1. inflight

The inflight variable is a dynamic list of InflightCommands which are in the process of being performed.

8.2.2.4.1.2. rcvdResponse

The rcvdResponse variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with an AVDECC Controller response ACMPDU (either of GET_TX_STATE_RESPONSE, CONNECT_RX_RESPONSE, DISCONNECT_RX_RESPONSE, GET_RX_STATE_RESPONSE or GET_TX_CONNECTION_RESPONSE).

8.2.2.4.2. State machine functions

8.2.2.4.2.1. txCommand(messageType, command, retry)

The txCommand function transmits a command of type messageType. It sets the ACMPDU fields to the values from the command ACMPCommandResponse parameter and the **message_type** field to the value of messageType.

If this function successfully sends the message and it is not a retry then it adds an InflightCommand entry to the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime + the appropriate timeout for the messageType (see Table 8-4), the retried field set to FALSE and the **sequence_id** field set to the **sequence_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime + the appropriate timeout for the messageType (see Table 8-4) and the retried field set to TRUE. This starts the timeout timer for this command.

If this function fails to send the message it calls the txResponse function with the appropriate response code for the messageType (messageType + 1), the passed in command and the status code of COULD_NOT_SEND_MESSAGE. If this was a retry then the InFlightCommand entry corresponding to the command is removed from the inflight variable.

8.2.2.4.2.2. cancelTimeout(commandResponse)

The cancelTimeout function stops the timeout timer of the inflight entry associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

8.2.2.4.2.3. removeInflight(commandResponse)

The removeInflight function removes an entry from the inflight variable associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

8.2.2.4.2.4. processResponse(commandResponse)

The processResponse function is the AVDECC Controllers hook into the state machine for handling responses to commands. This function may update saved state or perform other actions as necessary for the AVDECC Controller implementation to function.

8.2.2.4.2.5. makeCommand(params)

The makeCommand function is the AVDECC Controllers hook into the state machine for sending a command. The AVDECC Controller provides the params argument which is a structure of type ACMPCommandParams containing all of the parameters required to construct a ACMPCommandResponse to be passed to the txCommand function.

8.2.2.4.3. State machine diagram

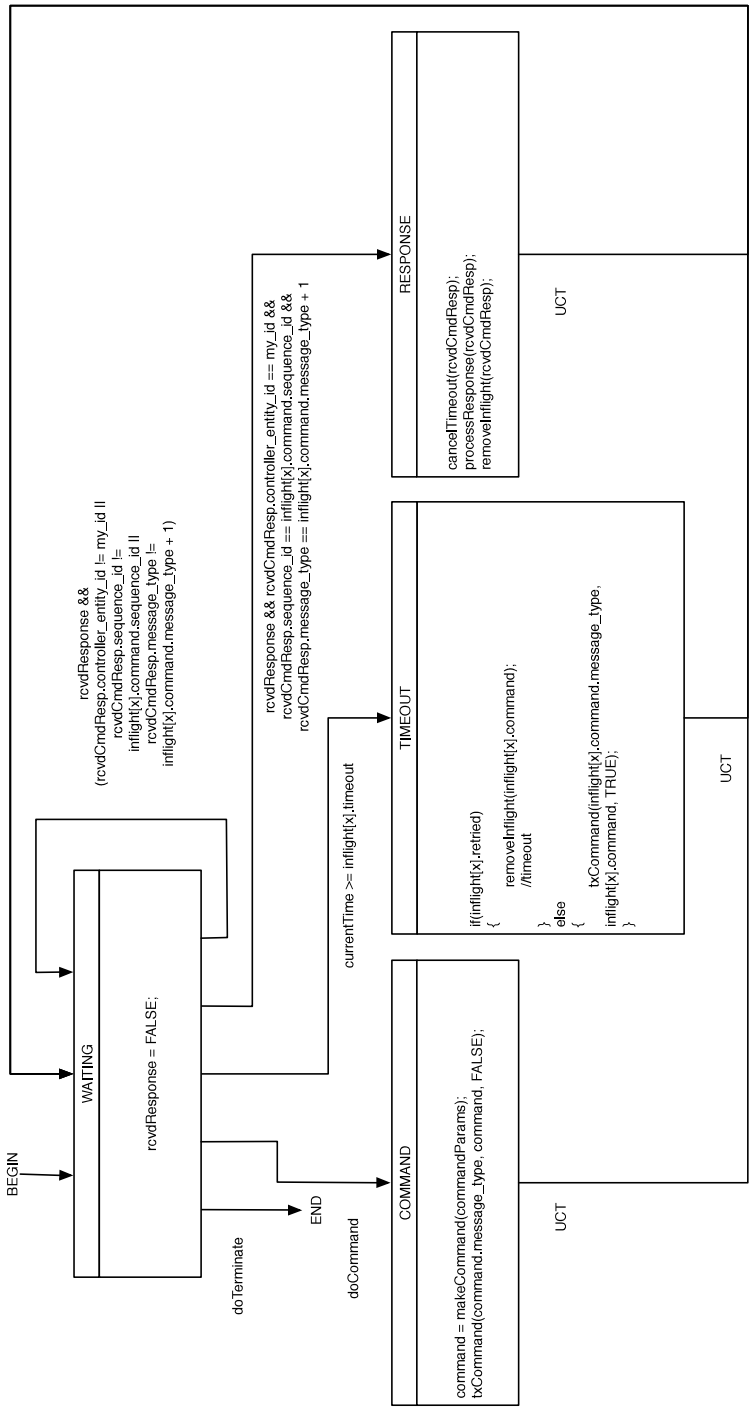

FIGURE 8-2—ACMP Controller state machine

8.2.2.5. ACMP Simple Listener State Machine

8.2.2.5.1. State machine variables

8.2.2.5.1.1. listenerSinkContexts

The listenerSinkContexts variable is an array of SimpleListenerSinkContext structures, one per Listener unique ID.

For a given Listener Unique ID, the inflight_used field is set to TRUE, and the inflight field is initialized, by the txCommand function, when the AVDECC Simple Listener sends a first command to an AVDECC Talker from the CONNECT_RX_COMMAND or DISCONNECT_RX_COMMAND state. The retried field is updated when the AVDECC Simple Listener sends a second command to the AVDECC Talker from the CONNECT_TX_TIMEOUT or DISCONNECT_TX_TIMEOUT state. The inflight_used field is set to FALSE in the CONNECT_TX_RESPONSE or DISCONNECT_TX_RESPONSE state, when the AVDECC Simple Listener receives a matching response from the AVDECC Talker, and in the CONNECT_TX_TIMEOUT or DISCONNECT_TX_TIMEOUT state, when the timeout expires for the second time.

The connected field is set to TRUE, and the binding_params, stream_params and STREAMING_WAIT fields are initialized by the connectListener function, when the AVDECC Simple Listener receives an expected CONNECT_TX_RESPONSE message. The same fields of the SimpleListenerSinkContext structure ar all cleared by the disconnectListener function, when the AVDECC Simple Listener receives a DISCONNECT_RX_COMMAND message.

The connectListener function may be called several times without any call to disconnectListener in between. In this case, the fields of the SimpleListenerSinkContext structure are reinitialized each time with potentially new values.

During the lifetime of a connection in the AVDECC Simple Listener (that is between a call to connectListener and a call to disconnectListener, or between two calls to connectListener), the inflight_used and inflight fields are not used and the connected, binding_params and stream_params fields remain stable. The STREAMING_WAIT bit may be updated by the processing of the AECP AEM START_STREAMING and STOP_STREAMING commands. The REGISTERING_FAILED field may be updated by asynchronous events received from the SRP protocol.

8.2.2.5.1.2. rcvdConnectRXCmd

The rcvdConnectRXCmd variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a CONNECT_RX_COMMAND ACMPDU.

8.2.2.5.1.3. rcvdDisconnectRXCmd

The rcvdDisconnectRXCmd variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT_RX_COMMAND ACMPDU.

8.2.2.5.1.4. rcvdConnectTXResp

The rcvdConnectTXResp variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a CONNECT_TX_RESPONSE ACMPDU.

8.2.2.5.1.5. rcvdDisconnectTXResp

The rcvdDisconnectTXResp variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT_TX_RESPONSE ACMPDU.

8.2.2.5.1.6. rcvdGetRXState

The rcvdGetRXState variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_RX_STATE_COMMAND ACMPDU.

8.2.2.5.2. State machine functions

8.2.2.5.2.1. validListenerUnique(ListenerUniqueId)

The validListenerUnique function returns a Boolean indicating if the AVDECC ListenerUniqueId passed in is valid for the AVDECC Entity.

8.2.2.5.2.2. listenerIsAcquiredLocked(command)

The listenerIsAcquiredLocked function returns a Boolean indicating if the AVDECC Simple Listener is acquired or locked by a controller other than the one specified by the **controller_entity_id** in the command.

This function returns TRUE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is currently acquired or locked by a controller other than the one specified by the **controller_entity_id** in the command, otherwise it returns FALSE.

This function returns FALSE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is currently not acquired nor locked by anyone.

8.2.2.5.2.3. listenerIsConnected(command)

The listenerIsConnected function returns a Boolean indicating if the AVDECC Simple Listener is already connected to a stream source other than the one specified by the **talker_entity_id** and **talker_unique_id** in the command.

This function returns TRUE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is connected to a stream source other than the one specified by the **talker_entity_id** and **talker_unique_id** in the command, otherwise it returns FALSE.

This function returns FALSE when being asked if it is connected to the same stream so that after an unclean disconnection (the AVDECC Talker disappearing and then reappearing without an intermediate DISCONNECT_RX_COMMAND being sent) the next connection attempt by the AVDECC Controller to restore the connection will succeed.

8.2.2.5.2.4. listenerIsConnectedTo(command)

The listenerIsConnectedTo function returns a Boolean indicating if the AVDECC Simple Listener is already connected to the stream source specified by the **talker_entity_id** and **talker_unique_id** in the command.

This function returns FALSE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is unconnected or already connected to a different **talker_entity_id** and **talker_unique_id**, otherwise it returns TRUE.

8.2.2.5.2.5. txCommand(messageType, command, retry)

The txCommand function transmits a command of type messageType. It sets the ACMPDU fields to the values from the command ACMPCommandResponse parameter and the **message_type** field to the value of messageType.

If this function successfully sends the message and it is not a retry (inflight.retry=FALSE) then it sets inflight_used to TRUE and initializes the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime + the appropriate timeout for the messageType (see Table 8-4), the retried field set to FALSE and the sequence_id field set to the **sequence_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry (inflight.retry=TRUE) then it updates the inflight variable by setting the timeout field to the value of currentTime + the appropriate timeout for the messageType (see Table 8-4) and the retried field set to TRUE. This starts the timeout timer for this command.

If this function fails to send the message it calls the txResponse function with the appropriate response code for the messageType (messageType + 1), the passed in command and the status code of COULD_NOT_SEND_MESSAGE. If this was a retry (inflight.retry=TRUE) then inflight_used is set to FALSE.

8.2.2.5.2.6. txResponse(messageType, response, error)

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message_type** field to the value of messageType and the status field to the value of the error parameter.

8.2.2.5.2.7. connectListener(response)

The connectListener function uses the passed in response structure to connect a stream sink to the AVDECC Talker.

The process of connecting the stream may succeed or fail depending on various conditions (not all mentioned here), and may result in initiating an SRP Listener registration. When it succeeds, the function sets the fields of the AVDECC SimpleListenerSinkContext entry for the **listener_unique_id** to the following values:

— connected is set to TRUE.

— the fields of binding_params are set to the equivalent fields of the response structure.

— the fields of stream_params are set to the equivalent fields of the response structure.

— STREAMING_WAIT is set to the equivalent field of the response structure.

The connectListener function returns the response structure, with REGISTRATION_FAILED properly updated. The connectListener function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.5.2.8. disconnectListener(command)

The disconnectListener function uses the passed in command structure to disconnect a stream sink from an AVDECC Talker.

The process of disconnecting the stream may succeed or fail depending on various conditions (not all mentioned here), and may result in initiating an SRP Talker de-registration. When it succeeds, the function sets the connected, binding_params, stream_params, STREAMING_WAIT and REGISTERING_FAILED fields of the AVDECC SimpleListenerSinkContext entry for the **listener_unique_id** to zero (0) or FALSE.

The disconnectListener function returns the command structure. The disconnectListener function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.5.2.9. cancelTimeout(ListenerUniqueId)

The cancelTimeout function stops the timeout timer of the inflight variable associated with the AVDECC ListenerUniqueId passed in.

8.2.2.5.2.10. getState(command)

The getState function never fails and returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent field of the stream_params structure.

— **talker_entity_id** and **talker_unique_id** are set to the equivalent field of the binding_params structure.

— **STREAMING_WAIT** and **REGISTERING_FAILED** are set as in the SimpleListenerSinkContext structure.

— **connection_count** is set to 0 if connected=FALSE, 1 if connected=TRUE.

— **FAST_CONNECT** and **SAVED_STATE** are set to 0.

— **SUPPORTS_ENCRYPTED** is set to 1 if the AVDECC Simple Listener supports receiving encrypted PDUs, 0 otherwise. **is this correct???**

8.2.2.5.3. State machine diagram

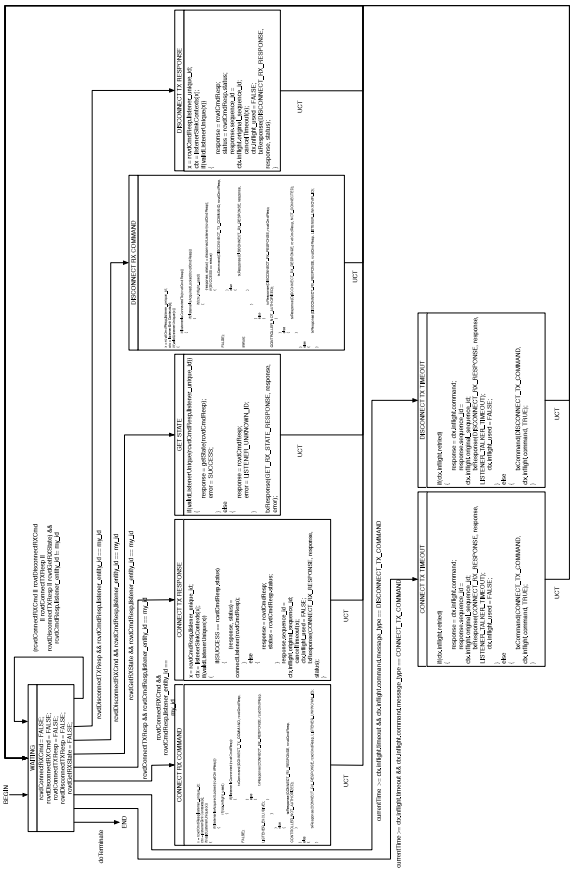

FIGURE 8-3—ACMP Simple Listener state machine

8.2.2.6. ACMP Smart Listener State Machine

8.2.2.6.1. State machine variables

8.2.2.6.1.1. listenerSinkContexts

The listenerSinkContexts variable is an array of SmartListenerSinkContext structures, one per Listener unique ID.

The bound field is set to TRUE, and the binding_params and STREAMING_WAIT feilds are initialized by the bindListener function, when the AVDECC Smart Listener receives a CONNECT_RX_COMMAND. All the fields of

the ListenerSinkContext structure are cleared by the clearListener function, when the AVDECC Simple Listener receives a DISCONNECT_RX_COMMAND message.

The bindListener function may be called several times without any call to clearListener in between. In this case, the fields of the SmartListnerSinkContext structure are reinitialized each time with potentially new values.

During the lifetime of a binding (that is between a call to bindListener and a call to clearListener, or between two calls to bindListener), the bound and binding_params fields remain stable. The STREAMING_WAIT bit may be updated by the processing of the AECP AEM START_STREAMING and STOP_STREAMING commands. The REGISTERING_FAILED field may be updated by asynchronous events received from the SRP protocol. The connected and stream_params fields are initialized and updated as a result of the execution of the Fast Connect sequence; the way an AVDECC Smart Listener is updating these fields is beyond the scope of this standard.

8.2.2.6.1.2. rcvdConnectRXCmd

The rcvdConnectRXCmd variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a CONNECT_RX_COMMAND ACMPDU.

8.2.2.6.1.3. rcvdDisconnectRXCmd

The rcvdDisconnectRXCmd variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT_RX_COMMAND ACMPDU.

8.2.2.6.1.4. rcvdGetRXState

The rcvdGetRXState variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_RX_STATE_COMMAND ACMPDU.

8.2.2.6.2. State machine functions

8.2.2.6.2.1. validListenerUnique(ListenerUniqueId)

The validListenerUnique function returns a Boolean indicating if the AVDECC ListenerUniqueId passed in is valid for the AVDECC Entity.

8.2.2.6.2.2. listenerIsAcquiredLocked(command)

The listenerIsAcquiredLocked function returns a Boolean indicating if the AVDECC Smart Listener is acquired or locked by a controller other than the one specified by the **controller_entity_id** in the command.

This function returns TRUE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is currently acquired or locked by a controller other than the one specified by the **controller_entity_id** in the command, otherwise it returns FALSE.

This function returns FALSE if the stream sink identified by the **listener_entity_id** and **listener_unique_id** is currently not acquired nor locked by anyone.

8.2.2.6.2.3. txResponse(messageType, response, error)

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message_type** field to the value of messageType and the status field to the value of the error parameter.

8.2.2.6.2.4. bindListener(command)

The bindListener function uses the passed in command structure to bind a stream to the AVDECC Talker.

The process of binding the stream is immediate and never fails. The function sets the binding_params variable of the AVDECC SmartListenerSinkContext entry for the **listener_unique_id** to the values of the equivalent fields in the command structure, sets STREAMING_WAIT as in the command structure and sets the bound field to TRUE.

The bindListener function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter and with the **connection_count** field set to 1.

8.2.2.6.2.5. clearListener(command)

The clearListener function uses the passed in command structure to clear a stream binding.

The process of clearing the binding of a stream is immediate and never fails. The function sets all fields of the AVDECC SmartListenerSinkContext entry for the **listener_unique_id** to zero (0) or FALSE.

The clearListener function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter and with the **connection_count** field set to 0.

8.2.2.6.2.6. getState(command)

The getState function never fails and returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent field of the stream_params structure.

— **talker_entity_id** and **talker_unique_id** are set to the equivalent field of the binding_params structure.

— **STREAMING_WAIT** and **REGISTERING_FAILED** are set as in the SmartListenerSinkContext structure.

— **connection_count** is set to 0 if bound=FALSE, 1 if bound=TRUE.

— **FAST_CONNECT** is set to 1.

— **SAVED_STATE** is set to 1. **what is the exact meaning of SAVED_STATE???**

— **SUPPORTS_ENCRYPTED** is set to 1 if the AVDECC Smart Listener supports receiving encrypted PDUs, 0 otherwise. **is this correct???**

8.2.2.6.3. State machine diagram

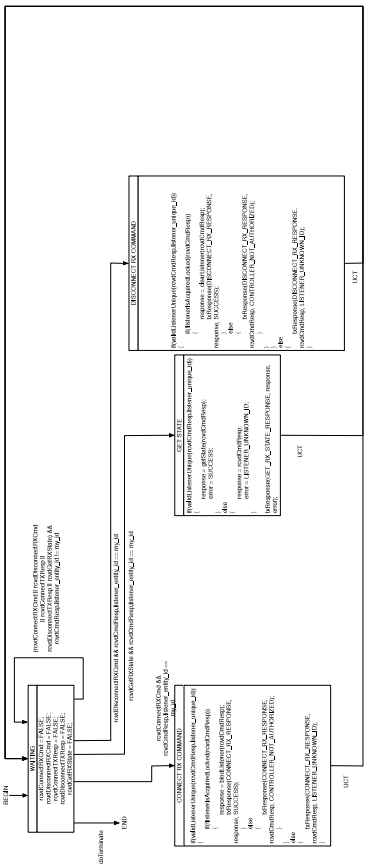

FIGURE 8-4—ACMP Smart Listener state machine

8.2.2.7. ACMP Connection-aware Talker State Machine

8.2.2.7.1. State machine variables

8.2.2.7.1.1. talkerSourceContexts

The talkerSourceContexts variable is an array of ConTalkerSourceContext structures, one per Talker unique ID.

The connected_listeners field of the TalkerSourceContext structure associated with a Talker Unique ID is updated by the connectTalker and disconnectTalker functions, when the AVDECC Connection-aware Talker receives a CONNECT_TX_COMMAND or DISCONNECT_TX_COMMAND message.

The stream_params field may be updated by the AVDECC Connection-aware Talker asynchronously of any received ACMP message. The way this is done is beyond the scope of this standard. At a given time, each of the fields of the stream_params structure may be valid or not. The connectTalker function returns an error if not all of these fields are valid at the end of the execution of the function.

The STREAMING_WAIT field is set by the connectTalker function and may be updated by the processing of the AECP AEM START_STREAMING and STOP_STREAMING commands. The REGISTERING_FAILED field may be updated by asynchronous events received from the SRP protocol.

8.2.2.7.1.2. rcvdConnectTX

The rcvdConnectTX variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a CONNECT_TX_COMMAND ACMPDU.

8.2.2.7.1.3. rcvdDisconnectTX

The rcvdDisconnectTX variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT_TX_COMMAND ACMPDU.

8.2.2.7.1.4. rcvdGetTXState

The rcvdGetTXState variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_TX_STATE_COMMAND ACMPDU.

8.2.2.7.1.5. rcvdGetTXConnection

The rcvdGetTXConnection variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_TX_CONNECTION_COMMAND ACMPDU.

8.2.2.7.2. State machine functions

8.2.2.7.2.1. validTalkerUnique(TalkerUniqueId)

The validTalkerUnique function returns a Boolean indicating if the AVDECC TalkerUniqueId passed in is valid for the AVDECC Entity.

8.2.2.7.2.2. connectTalker(command)

The connectTalker function uses the passed in command structure to connect a stream to the AVDECC Listener.

If this is the first stream sink connecting to the stream source identified by the AVDECC Talker unique ID, then the AVDECC Connection-aware Talker may allocate a stream ID and destination multicast MAC address and may initiate an SRP Talker registration.

The process of connecting the stream may succeed or fail depending on various conditions (not all mentioned here). In particular, it fails if the AVDECC Connection-aware Talker is not ready to transmit the requested stream (i.e. one of the fields of stream_params is not valid). When it succeeds, the connectTalker function may update the contents of the ConTalkerSourceContext entry describing this stream source. In particular, if the **listener_entity_id** and **listener_unique_id** of the command are not in the connected_listeners field of the AVDECC ConTalkerSourceContext entry describing this stream, then they are added to the field.

The connectTalker function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— **connection_count** is set to the current length of the connected_listeners array.

— STREAMING_WAIT and REGISTERING_FAILED are set as in the ConTalkerSourceContext structure.

The connectTalker function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.7.2.3. disconnectTalker(command)

The disconnectTalker function uses the passed in command structure to disconnect the stream from an AVDECC Listener.

The process of disconnecting the stream may succeed or fail depending on various conditions (not all mentioned here). When it succeeds, the disconnectTalker function may update the contents of the ConTalkerSourceContext entry describing this stream. In particular, if the **listener_entity_id** and **listener_unique_id** of the command are in the list of connected_listeners, then they are removed from the list.

If after the disconnection there is no more stream connected to the stream source identified by the AVDECC Talker Unique ID, then the AVDECC Connection-aware Talker may deallocate the stream ID and destination multicast MAC address and may initiate an SRP Talker de-registration.

The disconnectTalker function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— **connection_count** is set to the current length of the connected_listeners array.

— STREAMING_WAIT and REGISTERING_FAILED are set as in the ConTalkerSourceContext structure.

The disconnectTalker function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.7.2.4. getState(command)

The getState function never fails and returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— **connection_count** is set to the current length of the connected_listeners array.

— STREAMING_WAIT and REGISTERING_FAILED are set as in the ConTalkerSourceContext structure.

8.2.2.7.2.5. getConnection(command)

The getConnection function uses the passed in command parameter to return the connection information for an indexed connection.

The **connection_count** field of the command parameter is used to identify a zero (0) based index into the **connected_listeners** array of ListenerPairs in the ConTalkerSourceContext associated with the **talker_unique_id**. If this index is greater than or equal to the length of the array of ListenerPairs, then the getConnection function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, and returns a status code equal to NO_SUCH_CONNECTION.

The getConnection function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— STREAMING_WAIT and REGISTERING_FAILED are set as in the ConTalkerSourceContext structure.

— **listener_entity_id** and **listener_unique_id** are filled with the values from the ListenerPair entry indexed by the **connection_count** of the command in the **connected_listeners** array of the ConTalkerSourceContext entry.

The getConnection function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.7.2.6. txResponse(messageType, response, error)

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message_type** field to the value of messageType and the status field to the value of the error parameter.

8.2.2.7.3. State machine diagram

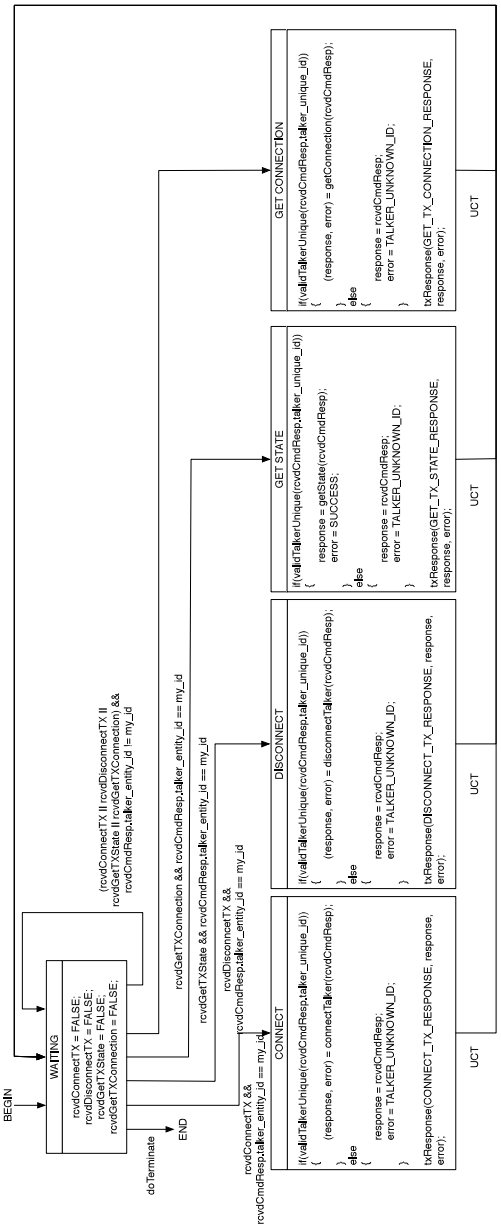

FIGURE 8-5—ACMP Connection-aware Talker state machine

8.2.2.8. ACMP Connection-unaware Talker State Machine

8.2.2.8.1. State machine variables

8.2.2.8.1.1. talkerSourceContexts

The talkerSourceContexts variable is an array of NconTalkerSourceContext structures, one per Talker unique ID.

The stream_params field may be updated by the AVDECC Connection-unaware Talker asynchronously of any received ACMP message. The way this is done is beyond the scope of this standard. At a given time, each of the fields of the stream_params strcuture may be valid or not. The probeTalker function returns an error if not all of these fields are valid at the end of the execution of the function.

The REGISTRATION_FAILED field may be updated by asynchronous events received from the SRP protocol.

8.2.2.8.1.2. rcvdConnectTX

The rcvdConnectTX variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a CONNECT_TX_COMMAND ACMPDU.

8.2.2.8.1.3. rcvdDisconnectTX

The rcvdDisconnectTX variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT_TX_COMMAND ACMPDU.

8.2.2.8.1.4. rcvdGetTXState

The rcvdGetTXState variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_TX_STATE_COMMAND ACMPDU.

8.2.2.8.1.5. rcvdGetTXConnection

The rcvdGetTXConnection variable is a Boolean which is set to TRUE when the rcvdCmdResp variable is set with a GET_TX_CONNECTION_COMMAND ACMPDU.

8.2.2.8.2. State machine functions

8.2.2.8.2.1. validTalkerUnique(TalkerUniqueId)

The validTalkerUnique function returns a Boolean indicating if the AVDECC TalkerUniqueId passed in is valid for the AVDECC Entity.

8.2.2.8.2.2. probeTalker(command)

The probeTalker function returns the stream parameters of the stream source identified in the passed in command structure.

The process of probing the stream may succeed or fail depending on various conditions (not all mentioned here). In particular, it fails if the AVDECC Connection-unaware Talker is currently not ready to transmit the probed stream (i.e. one of the fields of stream_params if not valid).

The probeTalker function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— **connection_count** is set to 0.

— REGISTERING_FAILED are set as in the NconTalkerSourceContext structure.

The probeTalker function also returns a status code as defined in Table 8-2 indicating either SUCCESS or the reason for a failure.

8.2.2.8.2.3. getState(command)

The getState function never fails and returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter except:

— **stream_id**, **stream_dest_mac**, **stream_vlan_id**, **CLASS_B** and **ENCRYPTED_PDU** are set to the equivalent fields of the stream_params structure.

— **connection_count** is set to 0.

— REGISTERING_FAILED are set as in the NconTalkerSourceContext structure.

8.2.2.8.2.4. txResponse(messageType, response, error)

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message_type** field to the value of messageType and the status field to the value of the error parameter.

8.2.2.8.3. State machine diagram

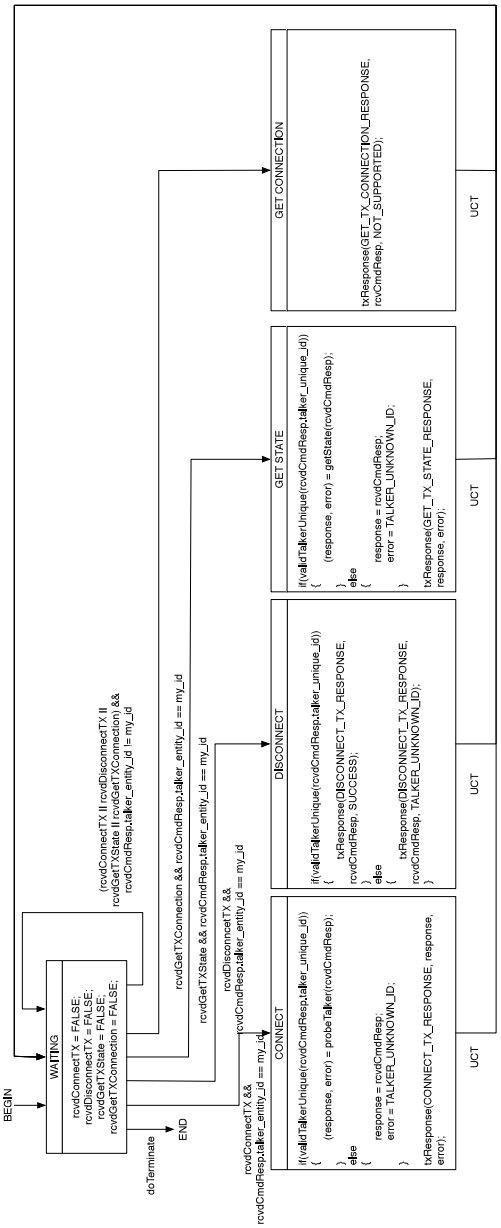

FIGURE 8-6—ACMP Connection-unaware Talker state machine