AVDECC clarifications

Part 1 – ACMP

Revision 2
2/25/2016

1. Introduction

The goal of this document is to clarify some parts of the AVDECC specification (IEEE Std. 1722.1™-2013).

This document tries to describe precisely all the fields of the ACMPDU for all the possible message types. It also describes a few scenarios that raise some questions not precisely answered in the specification.

This document is open for comments and proposals. Please do not hesitate to add/discuss/correct any point if needed. The goal is to finally have everybody agree so that a manufacturer releasing an AVDECC device now can be confident that it will interoperate with any other AVDECC device that will be released in the future.

2. Revision History

Version	Description
1	Initial release of the document February 19th, 2016
2	Updated the “ACMP scenarios” section according to comments from AVnu members. Updated description of the “stream_dest_mac” field of the GET_TX_STATE_RESPONSE and GET_TX_CONNECTION_RESPONSE messages. February 25th, 2016

Contents

3. ACMP PDU format

Explanation of the background colours:

Text on white background: Behaviour described by the specification and consistent

Text on red background: Behaviour described by the specification, but not consistent

Text on yellow background: Proposal of modification of the specification in order to be consistent

3.1. CONNECT_RX_COMMAND and CONNECT_RX_RESPONSE

	CONNECT_RX_COMMAND	**CONNECT_RX_RESPONSE (status=SUCCESS)**	**CONNECT_RX_RESPONSE (status<>SUCCESS)**
controller_entity_id	Entity ID of the controller sending the command	Copy "controller_entity_id" of the CONNECT_RX_COMMAND message	Copy "controller_entity_id" of the CONNECT_RX_COMMAND message
talker_entity_id	Entity ID of the talker which is target of the command	Copy "talker_entity_id" of the CONNECT_RX_COMMAND message	Copy "talker_entity_id" of the CONNECT_RX_COMMAND message
listener_entity_id	Entity ID of the listener which is target of the command	Copy "listener_entity_id" of the CONNECT_RX_COMMAND message	Copy "listener_entity_id" of the CONNECT_RX_COMMAND message
talker_unique_id	Unique ID of the Stream source which is target of the command	Copy "talker_unique_id" of the CONNECT_RX_COMMAND message	Copy "talker_unique_id" of the CONNECT_RX_COMMAND message
listener_unique_id	Unique ID of the Stream sink which is target of the command	Copy "listener_unique_id" of the CONNECT_RX_COMMAND message	Copy "listener_unique_id" of the CONNECT_RX_COMMAND message
stream_id	00:00:00:00:00:00:00:00	Copy "stream_id" of the CONNECT_TX_RESPONSE message	Copy "stream_id" of the CONNECT_TX_RESPONSE message (00:00:00:00:00:00:00:00 if there was no message from the talker)
stream_dest_mac	00:00:00:00:00:00	Copy "stream_dest_mac" of the	Copy "stream_dest_mac" of the

		CONNECT_TX_RESPONSE message	CONNECT_TX_RESPONSE message (00:00:00:00:00:00 if there was no message from the talker)
stream_vlan_id	0	Copy “stream_vlan_id” of the CONNECT_TX_RESPONSE message	Copy “stream_vlan_id” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)
connection_count	0	Copy “connection_count” of the CONNECT_TX_RESPONSE message	Copy “connection_count” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the CONNECT_RX_COMMAND message	Copy “sequence_id” of the CONNECT_RX_COMMAND message
flags.CLASS_B	0	Copy “flags.CLASS_B” of the CONNECT_TX_RESPONSE message	Copy “flags.CLASS_B” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.FAST_CONNECT	0	0	0
flags.SAVED_STATE	0	1 if the listener implements Fast Connect mode, 0 otherwise	1 if the listener implements Fast Connect mode, 0 otherwise
flags.STREAMING_WAIT	0 if the controller wants that the talker starts streaming data and the listener starts playing these data immediately, 1 otherwise	Copy “flags.STREAMING_WAIT” of the CONNECT_TX_RESPONSE message	Copy “flags.STREAMING_WAIT” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.SUPPORTS_ENCRYPTED	0	Copy “flags.SUPPORTS_ENCRYPTED” of the CONNECT_TX_RESPONSE message	Copy “flags.SUPPORTS_ENCRYPTED” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.ENCRYPTED_PDU	0	Copy “flags.ENCRYPTED_PDU” of the CONNECT_TX_RESPONSE message	Copy “flags.ENCRYPTED_PDU” of the CONNECT_TX_RESPONSE message (0 if there was no message from the talker)

flags.TALKER_FAILED	0	1 if the listener is already receiving a Talker Failed attribute for this stream, 0 otherwise	0
Other bits of the flags	0	Copy other bits of the flags of the CONNECT_TX_RESPONSE message	Copy other bits of the flags of the CONNECT_TX_RESPONSE message
reserved	0	0	0

3.2. CONNECT_TX_COMMAND and CONNECT_TX_RESPONSE

	CONNECT_TX_COMMAND	**CONNECT_TX_RESPONSE (status<>TALKER_UNKNOWN_ID)**	**CONNECT_TX_RESPONSE (status=TALKER_UNKNOWN_ID)**
controller_entity_id	Copy "controller_entity_id" of the CONNECT_RX_COMMAND message (saved value from the initial message in case of Fast Connect mode)	Copy "controller_entity_id" of the CONNECT_TX_COMMAND message	Copy "controller_entity_id" of the CONNECT_TX_COMMAND message
talker_entity_id	Copy "talker_entity_id" of the CONNECT_RX_COMMAND message (saved value from the initial message in case of Fast Connect mode)	Copy "talker_entity_id" of the CONNECT_TX_COMMAND message	Copy "talker_entity_id" of the CONNECT_TX_COMMAND message
listener_entity_id	Copy "listener_entity_id" of the CONNECT_RX_COMMAND message (saved value from the initial message in case of Fast Connect mode)	Copy "listener_entity_id" of the CONNECT_TX_COMMAND message	Copy "listener_entity_id" of the CONNECT_TX_COMMAND message
talker_unique_id	Copy "talker_unique_id" of the CONNECT_RX_COMMAND message (saved value from the initial message in case of Fast Connect mode)	Copy "talker_unique_id" of the CONNECT_TX_COMMAND message	Copy "talker_unique_id" of the CONNECT_TX_COMMAND message
listener_unique_id	Copy "listener_unique_id" of the CONNECT_RX_COMMAND message (saved value from the initial message in case of Fast Connect mode)	Copy "listener_unique_id" of the CONNECT_TX_COMMAND message	Copy "listener_unique_id" of the CONNECT_TX_COMMAND message
stream_id	Copy "stream_id" of the CONNECT_RX_COMMAND message (00:00:00:00:00:00:00:00 in case	– If the Talker source is connected to at least one Listener sink: ID of the stream – Otherwise:	Copy "stream_id" of the CONNECT_TX_COMMAND message

	of Fast Connect mode)	00:00:00:00:00:00:00:00	
stream_dest_mac	Copy “stream_dest_mac” of the CONNECT_RX_COMMAND message (00:00:00:00:00:00 in case of Fast Connect mode)	– If the Talker source is connected to at least one Listener sink: Destination MAC address of the stream – Otherwise: 00:00:00:00:00:00	Copy “stream_dest_mac” of the CONNECT_TX_COMMAND message
stream_vlan_id	Copy “stream_vlan_id” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	– If the Talker source is connected to at least one Listener sink: VLAN ID of the stream (0 indicates default VLAN ID of the SRP domain) – Otherwise: 0	Copy “stream_vlan_id” of the CONNECT_TX_COMMAND message
connection_count	Copy “connection_count” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Total count of Listener sinks connected to this stream source	Copy “connection_count” of the CONNECT_TX_COMMAND message
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the CONNECT_TX_COMMAND message	Copy “sequence_id” of the CONNECT_TX_COMMAND message
flags.CLASS_B	Copy “flags.CLASS_B” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy “flags.CLASS_B” of the CONNECT_TX_COMMAND message – If the Talker source is connected to at least one Listener sink: 0 if the stream is Class A, 1 if the stream is Class B – Otherwise: 0	Copy “flags.CLASS_B” of the CONNECT_TX_COMMAND message
flags.FAST_CONNECT	Copy “flags.FAST_CONNECT” of the CONNECT_RX_COMMAND message (1 in case of Fast Connect mode)	Copy “flags.FAST_CONNECT” of the CONNECT_TX_COMMAND message	Copy “flags.FAST_CONNECT” of the CONNECT_TX_COMMAND message
flags.SAVED_STATE	Copy “flags.SAVED_STATE” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy “flags.SAVED_STATE” of the CONNECT_TX_COMMAND message	Copy “flags.SAVED_STATE” of the CONNECT_TX_COMMAND message

flags.STREAMING_WAIT	Copy “flags.STREAMING_WAIT” of the CONNECT_RX_COMMAND message (saved value from last state in case of Fast Connect mode)	Copy “flags.STREAMING_WAIT” of the CONNECT_TX_COMMAND message	Copy “flags.STREAMING_WAIT” of the CONNECT_TX_COMMAND message
flags.SUPPORTS_ENCRYPTED	Copy “flags.SUPPORTS_ENCRYPTED” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy “flags.SUPPORTS_ENCRYPTED” of the CONNECT_TX_COMMAND message 1 if the talker supports encryption of the PDUs, 0 otherwise	Copy “flags.SUPPORTS_ENCRYPTED” of the CONNECT_TX_COMMAND message
flags.ENCRYPTED_PDU	Copy “flags.ENCRYPTED_PDU” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy “flags.ENCRYPTED_PDU” of the CONNECT_TX_COMMAND message – If the Talker source is connected to at least one Listener sink: 1 if the talker is configured to use encrypted PDUs for this stream, 0 otherwise – Otherwise: 0	Copy “flags.ENCRYPTED_PDU” of the CONNECT_TX_COMMAND message
flags.TALKER_FAILED	Copy “flags.TALKER_FAILED” of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy “flags.TALKER_FAILED” of the CONNECT_TX_COMMAND message	Copy “flags.TALKER_FAILED” of the CONNECT_TX_COMMAND message
Other bits of the flags	Copy other bits of the flags of the CONNECT_RX_COMMAND message (0 in case of Fast Connect mode)	Copy other bits of the flags of the CONNECT_TX_COMMAND message	Copy other bits of the flags of the CONNECT_TX_COMMAND message
reserved	0	0	0

3.3. DISCONNECT_RX_COMMAND and DISCONNECT_RX_RESPONSE

	DISCONNECT_RX_COMMAND	**DISCONNECT_RX_RESPONSE (status=SUCCESS)**	**DISCONNECT_RX_RESPONSE (status<>SUCCESS)**
controller_entity_id	Entity ID of the controller sending the command	Copy "controller_entity_id" of the DISCONNECT_RX_COMMAND message	Copy "controller_entity_id" of the DISCONNECT_RX_COMMAND message
talker_entity_id	Entity ID of the talker which is target of the command	Copy "talker_entity_id" of the DISCONNECT_RX_COMMAND message	Copy "talker_entity_id" of the DISCONNECT_RX_COMMAND message
listener_entity_id	Entity ID of the listener which is target of the command	Copy "listener_entity_id" of the DISCONNECT_RX_COMMAND message	Copy "listener_entity_id" of the DISCONNECT_RX_COMMAND message
talker_unique_id	Unique ID of the Stream source which is target of the command	Copy "talker_unique_id" of the DISCONNECT_RX_COMMAND message	Copy "talker_unique_id" of the DISCONNECT_RX_COMMAND message
listener_unique_id	Unique ID of the Stream sink which is target of the command	Copy "listener_unique_id" of the DISCONNECT_RX_COMMAND message	Copy "listener_unique_id" of the DISCONNECT_RX_COMMAND message
stream_id	00:00:00:00:00:00:00:00	Copy "stream_id" of the DISCONNECT_TX_RESPONSE message	Copy "stream_id" of the DISCONNECT_TX_RESPONSE message (00:00:00:00:00:00:00:00 if there was no message from the talker)
stream_dest_mac	00:00:00:00:00:00	Copy "stream_dest_mac" of the DISCONNECT_TX_RESPONSE message	Copy "stream_dest_mac" of the DISCONNECT_TX_RESPONSE message (00:00:00:00:00:00 if there was no message from the talker)
stream_vlan_id	0	Copy "stream_vlan_id" of the DISCONNECT_TX_RESPONSE message	Copy "stream_vlan_id" of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)

connection_count	0	Copy “connection_count” of the DISCONNECT_TX_RESPONSE message	Copy “connection_count” of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the DISCONNECT_RX_COMMAND message	Copy “sequence_id” of the DISCONNECT_RX_COMMAND message
flags.CLASS_B	0	Copy “flags.CLASS_B” of the DISCONNECT_TX_RESPONSE message	Copy “flags.CLASS_B” of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.FAST_CONNECT	0	0	0
flags.SAVED_STATE	0	1 if the listener implements Fast Connect mode, 0 otherwise	1 if the listener implements Fast Connect mode, 0 otherwise
flags.STREAMING_WAIT	0	Copy “flags.STREAMING_WAIT” of the DISCONNECT_TX_RESPONSE message	Copy “flags.STREAMING_WAIT” of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.SUPPORTS_ENCRYPTED	0	Copy “flags.SUPPORTS_ENCRYPTED” of the DISCONNECT_TX_RESPONSE message	Copy “flags.SUPPORTS_ENCRYPTED” of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.ENCRYPTED_PDU	0	Copy “flags.ENCRYPTED_PDU” of the DISCONNECT_TX_RESPONSE message	Copy “flags.ENCRYPTED_PDU” of the DISCONNECT_TX_RESPONSE message (0 if there was no message from the talker)
flags.TALKER_FAILED	0	0	0
Other bits of the flags	0	Copy other bits of the flags of the DISCONNECT_TX_RESPONSE message	Copy other bits of the flags of the DISCONNECT_TX_RESPONSE message
reserved	0	0	0

3.4. DISCONNECT_TX_COMMAND and DISCONNECT_TX_RESPONSE

	DISCONNECT_TX_COMMAND	**DISCONNECT_TX_RESPONSE (status<>TALKER_UNKNOWN_ID)**	**DISCONNECT_TX_RESPONSE (status=TALKER_UNKNOWN_ID)**
controller_entity_id	Copy "controller_entity_id" of the DISCONNECT_RX_COMMAND message (saved value from the current connection in case of Fast Disconnect mode)	Copy "controller_entity_id" of the DISCONNECT_TX_COMMAND message	Copy "controller_entity_id" of the DISCONNECT_TX_COMMAND message
talker_entity_id	Copy "talker_entity_id" of the DISCONNECT_RX_COMMAND message (saved value from the current connection in case of Fast Disconnect mode)	Copy "talker_entity_id" of the DISCONNECT_TX_COMMAND message	Copy "talker_entity_id" of the DISCONNECT_TX_COMMAND message
listener_entity_id	Copy "listener_entity_id" of the DISCONNECT_RX_COMMAND message (saved value from the current connection in case of Fast Disconnect mode)	Copy "listener_entity_id" of the DISCONNECT_TX_COMMAND message	Copy "listener_entity_id" of the DISCONNECT_TX_COMMAND message
talker_unique_id	Copy "talker_unique_id" of the DISCONNECT_RX_COMMAND message (saved value from the current connection in case of Fast Disconnect mode)	Copy "talker_unique_id" of the DISCONNECT_TX_COMMAND message	Copy "talker_unique_id" of the DISCONNECT_TX_COMMAND message
listener_unique_id	Copy "listener_unique_id" of the DISCONNECT_RX_COMMAND message (saved value from the current connection in case of Fast Disconnect mode)	Copy "listener_unique_id" of the DISCONNECT_TX_COMMAND message	Copy "listener_unique_id" of the DISCONNECT_TX_COMMAND message
stream_id	Copy "stream_id" of the DISCONNECT_RX_COMMAND message (00:00:00:00:00:00:00:00 in case	– If the Talker source is still connected to at least one Listener sink: ID of the stream – Otherwise:	Copy "stream_id" of the DISCONNECT_TX_COMMAND message

	of Fast Disconnect mode)	00:00:00:00:00:00:00:00	
stream_dest_mac	Copy “stream_dest_mac” of the DISCONNECT_RX_COMMAND message (00:00:00:00:00:00 in case of Fast Disconnect mode)	– If the Talker source is still connected to at least one Listener sink: Destination MAC address of the stream – Otherwise: 00:00:00:00:00:00	Copy “stream_dest_mac” of the DISCONNECT_TX_COMMAND message
stream_vlan_id	Copy “stream_vlan_id” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	– If the Talker source is still connected to at least one Listener sink: VLAN ID of the stream (0 indicates default VLAN ID of the SRP domain) – Otherwise: 0	Copy “stream_vlan_id” of the DISCONNECT_TX_COMMAND message
connection_count	Copy “connection_count” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Total count of Listener sinks connected to this stream source	Copy “connection_count” of the DISCONNECT_TX_COMMAND message
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the DISCONNECT_TX_COMMAND message	Copy “sequence_id” of the DISCONNECT_TX_COMMAND message
flags.CLASS_B	Copy “flags.CLASS_B” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Copy “flags.CLASS_B” of the CONNECT_TX_COMMAND message – If the Talker source is still connected to at least one Listener sink: 0 if the stream is Class A, 1 if the stream is Class B Otherwise: 0	Copy “flags.CLASS_B” of the DISCONNECT_TX_COMMAND message
flags.FAST_CONNECT	Copy “flags.FAST_CONNECT” of the DISCONNECT_RX_COMMAND message (1 in case of Fast Disconnect mode)	Copy “flags.FAST_CONNECT” of the DISCONNECT_TX_COMMAND message	Copy “flags.FAST_CONNECT” of the DISCONNECT_TX_COMMAND message
flags.SAVED_STATE	Copy “flags.SAVED_STATE” of the DISCONNECT_RX_COMMAND message (0 in case of Fast	Copy “flags.SAVED_STATE” of the DISCONNECT_TX_COMMAND message	Copy “flags.SAVED_STATE” of the DISCONNECT_TX_COMMAND message

	Disconnect mode)		
flags.STREAMING_WAIT	Copy “flags.STREAMING_WAIT” of the DISCONNECT_RX_COMMAND message (saved value from last state in case of Fast Disconnect mode)	Copy “flags.STREAMING_WAIT” of the DISCONNECT_TX_COMMAND message	Copy “flags.STREAMING_WAIT” of the DISCONNECT_TX_COMMAND message
flags.SUPPORTS_ENCRYPTED	Copy “flags.SUPPORTS_ENCRYPTED” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Copy “flags.SUPPORTS_ENCRYPTED” of the DISCONNECT_TX_COMMAND message 1 if the talker supports encryption of the PDUs, 0 otherwise	Copy “flags.SUPPORTS_ENCRYPTED” of the DISCONNECT_TX_COMMAND message
flags.ENCRYPTED_PDU	Copy “flags.ENCRYPTED_PDU” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Copy “flags.ENCRYPTED_PDU” of the DISCONNECT_TX_COMMAND message – If the Talker source is still connected to at least one Listener sink: 1 if the talker is configured to use encrypted PDUs for this stream, 0 otherwise – Otherwise: 0	Copy “flags.ENCRYPTED_PDU” of the DISCONNECT_TX_COMMAND message
flags.TALKER_FAILED	Copy “flags.TALKER_FAILED” of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Copy “flags.TALKER_FAILED” of the DISCONNECT_TX_COMMAND message	Copy “flags.TALKER_FAILED” of the DISCONNECT_TX_COMMAND message
Other bits of the flags	Copy other bits of the flags of the DISCONNECT_RX_COMMAND message (0 in case of Fast Disconnect mode)	Copy other bits of the flags of the DISCONNECT_TX_COMMAND message	Copy other bits of the flags of the DISCONNECT_TX_COMMAND message
reserved	0	0	0

3.5. GET_RX_STATE_COMMAND and GET_RX_STATE_RESPONSE

	GET_RX_STATE_COMMAND	**GET_RX_STATE_RESPONSE (status=SUCCESS)**	**GET_RX_STATE_RESPONSE (status<>SUCCESS)**
controller_entity_id	Entity ID of the controller sending the command	Copy "controller_entity_id" of the GET_RX_STATE_COMMAND message	Copy "controller_entity_id" of the GET_RX_STATE_COMMAND message
talker_entity_id	00:00:00:00:00:00:00:00	– If the Stream sink is connected to a Talker source: Entity ID of the talker – If not connected: 00:00:00:00:00:00:00:00	00:00:00:00:00:00:00:00
listener_entity_id	Entity ID of the listener which is target of the command	Copy "listener_entity_id" of the GET_RX_STATE_COMMAND message	Copy "listener_entity_id" of the GET_RX_STATE_COMMAND message
talker_unique_id	0	– If the Stream sink is connected to a Talker source: Unique ID of the talker source – If not connected: 0	0
listener_unique_id	Unique ID of the Stream sink which is target of the command	Copy "listener_unique_id" of the GET_RX_STATE _COMMAND message	Copy "listener_unique_id" of the GET_RX_STATE _COMMAND message
stream_id	00:00:00:00:00:00:00:00	– If the Stream sink is connected to a Talker source: ID of the stream – If not connected: 00:00:00:00:00:00:00:00	00:00:00:00:00:00:00:00
stream_dest_mac	00:00:00:00:00:00	– If the Stream sink is connected to a Talker source: Destination MAC	00:00:00:00:00:00

		address of the stream – If not connected: 00:00:00:00:00:00	
stream_vlan_id	0	– If the Stream sink is connected to a Talker source: VLAN ID of the stream (0 indicates default VLAN ID of the SRP domain) – If not connected: 0	0
connection_count	0	Number of Listener sinks connected to this Talker source 1 if the Stream sink is connected to a Talker source, 0 otherwise	0
sequence_id	changed by the sender at each new command sent	Copy "sequence_id" of the GET_RX_STATE_COMMAND message	Copy "sequence_id" of the GET_RX_STATE_COMMAND message
flags.CLASS_B	0	– If the Stream sink is connected to a Talker source : 0 if the stream is Class A, 1 if the stream is Class B – If not connected: 0	0
flags.FAST_CONNECT	0	0	0
flags.SAVED_STATE	0	1 if the listener implements Fast Connect mode, 0 otherwise	0
flags.STREAMING_WAIT	0	0	0
flags.SUPPORTS_ENCRYPTED	0	0	0
flags.ENCRYPTED_PDU	0	– If the Stream sink is connected to a Talker source : 1 if the stream is encrypted, 0 otherwise – If not connected: 0	0

flags.TALKER_FAILED	0	– If the Stream sink is connected to a Talker source : 1 if the listener is receiving a Talker Failed attribute for this stream, 0 otherwise – If not connected: 0	0
Other bits of the flags	0	0	0
reserved	0	0	0

3.6. GET_TX_STATE_COMMAND and GET_TX_STATE_RESPONSE

	GET_TX_STATE_COMMAND	**GET_TX_STATE_RESPONSE (status<>TALKER_UNKNOWN_ID)**	**GET_TX_STATE_RESPONSE (status=TALKER_UNKNOWN_ID)**
controller_entity_id	Entity ID of the controller sending the command	Copy "controller_entity_id" of the GET_TX_STATE_COMMAND message	Copy "controller_entity_id" of the GET_TX_STATE_COMMAND message
talker_entity_id	Entity ID of the talker which is target of the command	Copy "talker_entity_id" of the GET_TX_STATE_COMMAND message	Copy "talker_entity_id" of the GET_TX_STATE_COMMAND message
listener_entity_id	00:00:00:00:00:00:00:00	Copy "listener_entity_id" of the GET_TX_STATE _COMMAND message	Copy "listener_entity_id" of the GET_TX_STATE _COMMAND message
talker_unique_id	Unique ID of the Stream source which is target of the command	Copy "talker_unique_id" of the GET_TX_STATE _COMMAND message	Copy "talker_unique_id" of the GET_TX_STATE_COMMAND message
listener_unique_id	0	Copy "listener_unique_id" of the GET_TX_STATE _COMMAND message	Copy "listener_unique_id" of the GET_TX_STATE _COMMAND message
stream_id	00:00:00:00:00:00:00:00	– If the Stream source is connected to at least one Listener sink: ID of the stream – If not connected: 00:00:00:00:00:00:00:00	Copy "stream_id" of the GET_TX_STATE _COMMAND message
stream_dest_mac	00:00:00:00:00:00	– If the Stream source is connected to at least one Listener sink: Destination MAC address of the stream (00:00:00:00:00:00 if the MAAP range previously allocated by the Talker has been lost and a new range has	Copy "stream_dest_mac" of the GET_TX_STATE _COMMAND message

		not been allocated yet) – If not connected: 00:00:00:00:00:00	
stream_vlan_id	0	– If the Stream source is connected to at least one Listener sink: VLAN ID of the stream (0 indicates default VLAN ID of the SRP domain) – If not connected: 0	Copy “stream_vlan_id” of the GET_TX_STATE _COMMAND message
connection_count	0	Number of Listener sinks connected to this Talker source	Copy “connection_count” of the GET_TX_STATE _COMMAND message
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the GET_TX_STATE_COMMAND message	Copy “sequence_id” of the GET_TX_STATE_COMMAND message
flags.CLASS_B	0	Copy “flags.CLASS_B” of the GET_TX_STATE_COMMAND message – If the Talker source is connected to at least one Listener sink: 0 if the stream is Class A, 1 if the stream is Class B – Otherwise: 0	Copy “flags.CLASS_B” of the GET_TX_STATE_COMMAND message
flags.FAST_CONNECT	0	Copy “flags.FAST_CONNECT” of the GET_TX_STATE_COMMAND message	Copy “flags.FAST_CONNECT” of the GET_TX_STATE_COMMAND message
flags.SAVED_STATE	0	Copy “flags.SAVED_STATE” of the GET_TX_STATE_COMMAND message	Copy “flags.SAVED_STATE” of the GET_TX_STATE_COMMAND message
flags.STREAMING_WAIT	0	Copy “flags.STREAMING_WAIT” of the GET_TX_STATE_COMMAND message	Copy “flags.STREAMING_WAIT” of the GET_TX_STATE_COMMAND message
flags.SUPPORTS_ENCRYPTED	0	Copy “flags.SUPPORTS_ENCRYPTED” of the GET_TX_STATE_COMMAND	Copy “flags.SUPPORTS_ENCRYPTED” of

		message 1 if the talker supports encryption of the PDUs, 0 otherwise	the GET_TX_STATE_COMMAND message
flags.ENCRYPTED_PDU	0	Copy “flags.ENCRYPTED_PDU” of the GET_TX_STATE_COMMAND message – If the Talker source is connected to at least one Listener sink: 1 if the talker is configured to use encrypted PDUs for this stream, 0 otherwise – Otherwise: 0	Copy “flags.ENCRYPTED_PDU” of the GET_TX_STATE_COMMAND message
flags.TALKER_FAILED	0	Copy “flags.TALKER_FAILED” of the GET_TX_STATE_COMMAND message	Copy “flags.TALKER_FAILED” of the GET_TX_STATE_COMMAND message
Other bits of the flags	0	Copy other bits of the flags of the GET_TX_STATE_COMMAND message	Copy other bits of the flags of the GET_TX_STATE_COMMAND message
reserved	0	0	0

3.7. GET_TX_CONNECTION_COMMAND and GET_TX_CONNECTION_RESPONSE

	GET_TX_CONNECTION_COMMAND	**GET_TX_CONNECTION_RESPONSE (status<>TALKER_UNKNOWN_ID)**	**GET_TX_ CONNECTION _RESPONSE (status=TALKER_UNKNOWN_ID)**
controller_entity_id	Entity ID of the controller sending the command	Copy "controller_entity_id" of the GET_TX_CONNECTION_COMMAND message	Copy "controller_entity_id" of the GET_TX_ CONNECTION _COMMAND message
talker_entity_id	Entity ID of the talker which is target of the command	Copy "talker_entity_id" of the GET_TX_ CONNECTION _COMMAND message	Copy "talker_entity_id" of the GET_TX_ CONNECTION _COMMAND message
listener_entity_id	00:00:00:00:00:00:00:00	Entity ID of the connected Listener (00:00:00:00:00:00:00:00 if status=NO_SUCH_CONNECTION)	Copy "listener_entity_id" of the GET_TX_ CONNECTION _COMMAND message
talker_unique_id	Unique ID of the Stream source which is target of the command	Copy "talker_unique_id" of the GET_TX_ CONNECTION _COMMAND message	Copy "talker_unique_id" of the GET_TX_ CONNECTION _COMMAND message
listener_unique_id	0	Unique ID of the connected Listener sink (0 if status=NO_SUCH_CONNECTION)	Copy "listener_unique_id" of the GET_TX_ CONNECTION _COMMAND message
stream_id	00:00:00:00:00:00:00:00	– If the Stream source is connected to at least one Listener sink: ID of the stream – If not connected: 00:00:00:00:00:00:00:00	Copy "stream_id" of the GET_TX_ CONNECTION _COMMAND message
stream_dest_mac	00:00:00:00:00:00	– If the Stream source is connected to at least one Listener sink: Destination MAC address of the stream (00:00:00:00:00:00 if the MAAP range previously allocated by the Talker has been lost and a new range has	Copy "stream_dest_mac" of the GET_TX_ CONNECTION _COMMAND message

		not been allocated yet) – If not connected: 00:00:00:00:00:00	
stream_vlan_id	0	– If the Stream source is connected to at least one Listener sink: VLAN ID of the stream (0 indicates default VLAN ID of the SRP domain) – If not connected: 0	Copy “stream_vlan_id” of the GET_TX_ CONNECTION _COMMAND message
connection_count	Index of the connection which is target of the command (the first connection of the list has index 0)	Number of Listener sinks connected to this Talker source	Copy “connection_count” of the GET_TX_ CONNECTION _COMMAND message
sequence_id	changed by the sender at each new command sent	Copy “sequence_id” of the GET_TX_ CONNECTION _COMMAND message	Copy “sequence_id” of the GET_TX_ CONNECTION _COMMAND message
flags.CLASS_B	0	Copy “flags.CLASS_B” of the GET_TX_CONNECTION_COMMAND message – If the Talker source is connected to at least one Listener sink: 0 if the stream is Class A, 1 if the stream is Class B – Otherwise: 0	Copy “flags.CLASS_B” of the GET_TX_ CONNECTION _COMMAND message
flags.FAST_CONNECT	0	Copy “flags.FAST_CONNECT” of the GET_TX_ CONNECTION _COMMAND message	Copy “flags.FAST_CONNECT” of the GET_TX_ CONNECTION _COMMAND message
flags.SAVED_STATE	0	Copy “flags.SAVED_STATE” of the GET_TX_ CONNECTION _COMMAND message	Copy “flags.SAVED_STATE” of the GET_TX_ CONNECTION _COMMAND message
flags.STREAMING_WAIT	0	Copy “flags.STREAMING_WAIT” of the GET_TX_ CONNECTION _COMMAND message	Copy “flags.STREAMING_WAIT” of the GET_TX_ CONNECTION _COMMAND message
flags.SUPPORTS_ENCRYPTED	0	Copy “flags.SUPPORTS_ENCRYPTED” of the	Copy “flags.SUPPORTS_ENCRYPTED” of the GET_TX_ CONNECTION

		GET_TX_CONNECTION_COMMAND message 1 if the talker supports encryption of the PDUs, 0 otherwise	_COMMAND message
flags.ENCRYPTED_PDU	0	Copy “flags.ENCRYPTED_PDU” of the GET_TX_CONNECTION_COMMAND message – If the Talker source is connected to at least one Listener sink: 1 if the talker is configured to use encrypted PDUs for this stream, 0 otherwise – Otherwise: 0	Copy “flags.ENCRYPTED_PDU” of the GET_TX_ CONNECTION _COMMAND message
flags.TALKER_FAILED	0	Copy “flags.TALKER_FAILED” of the GET_TX_ CONNECTION _COMMAND message	Copy “flags.TALKER_FAILED” of the GET_TX_ CONNECTION _COMMAND message
Other bits of the flags	0	Copy other bits of the flags of the GET_TX_ CONNECTION _COMMAND message	Copy other bits of the flags of the GET_TX_ CONNECTION _COMMAND message
reserved	0	0	0

4. ACMP scenarios

4.1. Acquirement/lock and connections – Listener refusal

In this scenario, there are 4 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- Two controllers: *Controller1* and *Controller2*

The goal of this scenario is to show that a listener shall refuse a connection/disconnection request from a controller if it has been acquired/locked by another controller.

1) *Controller1* acquires or locks successfully one or several STREAM_INPUT descriptor(s) of *Listener* through AECP
2) *Controller2* sends **CONNECT_RX_COMMAND** with **listener_entity_id** equal to the Entity ID of *Listener* and **listener_unique_id** equal to the index of one of the STREAM_INPUT descriptors acquired/locked by *Controller1*
3) *Listener* refuses the request and replies immediately **CONNECT_RX_RESPONSE** with **status=CONTROLLER_NOT_AUTHORIZED** (*Listener* doesn't even send **CONNECT_TX_COMMAND** to *Talker*)

Notes:

- In order to acquire/lock a STREAM_INPUT descriptor, *Controller1* may either acquire/lock the STREAM_INPUT descriptor only, or the AUDIO_UNIT descriptor associated to this STREAM_INPUT descriptor, or the ENTITY descriptor.
- If *Controller1* has acquired/locked a subtree of *Listener*'s AEM model which doesn't contain the STREAM_INPUT descriptor targeted by the **CONNECT_RX_COMMAND**, *Listener* shall not refuse the request.
- This behaviour should be agreed and implemented by every listener. Otherwise, strange behaviours may appear when the talker implements it but not the listener for example.

4.2. Acquirement/lock and connections – Talker refusal

In this scenario, there are 4 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*

- Two controllers: *Controller1* and *Controller2*

The goal of this scenario is to show that a talker shall refuse a connection/disconnection request from a controller if it has been acquired/locked by another controller.

1) *Controller1* acquires/locks successfully one or several STREAM_OUTPUT descriptor(s) of *Talker* through AECP
2) *Controller2* sends **CONNECT_RX_COMMAND** to *Listener* with **talker_entity_id** equal to the Entity ID of *Talker* and **talker_unique_id** equal to the index of one of the STREAM_OUTPUT descriptors acquired/locked by *Controller1*
3) *Listener* sends **CONNECT_TX_COMMAND** to *Talker*
4) *Talker* refuses the request and replies **CONNECT_TX_RESPONSE** with **status=CONTROLLER_NOT_AUTHORIZED**
5) *Listener* receives **CONNECT_TX_RESPONSE** with **status=CONTROLLER_NOT_AUTHORIZED**
6) *Listener* doesn't connect and sends **CONNECT_RX_RESPONSE** with **status=CONTROLLER_NOT_AUTHORIZED**

Notes:
- In order to acquire/lock a STREAM_OUTPUT descriptor, *Controller1* may either acquire/lock the STREAM_OUTPUT descriptor only, or the AUDIO_UNIT descriptor associated to this STREAM_OUTPUT descriptor, or the ENTITY descriptor.
- If *Controller1* has acquired/locked a subtree of *Talker*'s AEM model which doesn't contain the STREAM_OUTPUT descriptor targeted by the **CONNECT_TX_COMMAND**, *Talker* shall not refuse the request.

4.3. State of the Listener during Fast Connect attempt

In this scenario, there are 3 AVDECC entities:
- A listener which implements the Fast Connect mode: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that a Listener performing a Fast Connect shall advertise to the controller that it is not connected.

1) *Controller* successfully establishes a connection between *Talker* and *Listener*
2) Power is switched off (every device shuts down)
3) *Talker* is removed from the network
4) Later, power is switched on again
5) *Listener* reboots and sends some **CONNECT_TX_COMMAND** messages to *Talker* with **flags.FAST_CONNECT=1**

6) During this time, *Controller* sends **GET_RX_STATE_COMMAND** to *Listener*
7) *Listener* replies **GET_RX_STATE_RESPONSE** with **connection_count=0** and **flags.FAST_CONNECT=0**

Note: there are two ways for the controller to know that the listener is currently attempting to connect in Fast Connect mode:

- Either sniff the **CONNECT_TX_COMMAND** messages on the network and see that **flags.FAST_CONNECT** is set. This is the best way to do because the **CONNECT_TX_COMMAND** message contains the Entity IDs of both the listener and the talker, and it also contains the Unique IDs of the sink and the source.
- Either send a GET_STREAM_INFO command to the listener. The listener will set flags.CONNECTED=0 and flags.FAST_CONNECT=1. The drawback of this method is that the controller doesn't have any information about the talker source to which the listener is trying to connect to.

4.4. Request to stop Fast Connect attempts

In this scenario, there are 3 AVDECC entities:

- A listener which implements the Fast Connect mode: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show how a Controller can ask a Listener to stop attempting to connect in Fast Connect mode.

1) *Controller* successfully establishes a connection between *Talker* and *Listener*
2) Power is switched off (every device shuts down)
3) *Talker* is removed from the network
4) Later, power is switched on again
5) *Listener* reboots and sends some **CONNECT_TX_COMMAND** messages to *Talker* with **flags.FAST_CONNECT=1**
6) During this time, *Controller* sends **DISCONNECT_RX_COMMAND** to *Listener*
7) *Listener* receives **DISCONNECT_RX_COMMAND**
8) *Listener* stops attempting to connect in Fast Connect mode
9) *Listener* replies **DISCONNECT_RX_RESPONSE** with **status=NOT_CONNECTED**

Note: there is no way for the controller to globally disable the Fast Connect feature of a listener. A listener implementing Fast Connect and which is rebooted without being cleanly disconnected will always try to connect in Fast Connect mode (until the controller sends a **DISCONNECT_RX_COMMAND** message).

4.5. Connection succeeded on Talker and failed on Listener

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that the controller shall always request the state of the entities after a connection failure.

1) *Controller* sends **CONNECT_RX_COMMAND** to *Listener*
2) *Listener* receives **CONNECT_RX_COMMAND**
3) *Listener* sends **CONNECT_TX_COMMAND** to *Talker*
4) *Talker* receives **CONNECT_TX_COMMAND**
5) *Talker* executes successfully the "connectTalker" function
6) *Talker* sends **CONNECT_TX_RESPONSE** with **status=SUCCESS**
7) *Listener* receives **CONNECT_TX_RESPONSE** with **status=SUCCESS**
8) *Listener* executes the "connectListener" function and it fails for any reason (it returns status<>SUCCESS)
9) *Listener* sends **CONNECT_RX_RESPONSE** with **status<>SUCCESS**
10) *Controller* receives **CONNECT_RX_RESPONSE** with **status<>SUCCESS**
11) *Controller* gets the state of *Listener* thanks to a **GET_RX_STATE_COMMAND/GET_RX_STATE_RESPONSE** exchange
12) *Controller* gets the state of *Talker* thanks to a **GET_TX_STATE_COMMAND/GET_TX_STATE_RESPONSE** + some **GET_TX_CONNECTION_COMMAND/GET_TX_CONNECTION_RESPONSE** exchanges
13) *Controller* notices that the connection failed on *Listener* and tries again sending a **CONNECT_RX_COMMAND** to *Listener*

Note: if the listener continuously fails to establish the connection, it will be impossible to disconnect the talker. Indeed, the controller cannot send directly a **DISCONNECT_TX_COMMAND** to the talker, and it's no use to send a **DISCONNECT_RX_COMMAND** to the listener because it will always reply with **status=NOT_CONNECTED** without even trying to send a **DISCONNECT_TX_COMMAND** to the talker.

Note2: it would be nice if the listener could handle this case by itself, this means automatically send a **DISCONNECT_TX_COMMAND** to the talker when the "connectListener" function fails (in addition to sending the **CONNECT_RX_RESPONSE** with **status<>SUCCESS** to the controller). Unfortunately a listener behaving this way is not conform to the ACMP Listener state machine specified in IEEE 1722.1-2013.

4.6. Disconnection succeeded on Listener and failed on Talker

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that the controller shall always request the state of the entities after a disconnection failure.

1) *Controller* sends **DISCONNECT_RX_COMMAND** to *Listener*
2) *Listener* receives **DISCONNECT_RX_COMMAND**
3) *Listener* executes successfully the "disconnectListener" function
4) *Listener* sends **DISCONNECT_TX_COMMAND** to *Talker*
5) *Talker* receives **DISCONNECT_TX_COMMAND**
6) *Talker* executes the "disconnectTalker" function and it fails for any reason (it returns status<>SUCCESS)
7) *Talker* sends **DISCONNECT_TX_RESPONSE** with **status<>SUCCESS**
8) *Listener* receives **DISCONNECT_TX_RESPONSE** with **status<>SUCCESS**
9) *Listener* sends **DISCONNECT_RX_RESPONSE** with **status<>SUCCESS**
10) *Controller* receives **DISCONNECT_RX_RESPONSE** with **status<>SUCCESS**
11) *Controller* gets the state of *Listener* thanks to a **GET_RX_STATE_COMMAND/GET_RX_STATE_RESPONSE** exchange
12) *Controller* gets the state of *Talker* thanks to a **GET_TX_STATE_COMMAND/GET_TX_STATE_RESPONSE** + some **GET_TX_CONNECTION_COMMAND/GET_TX_CONNECTION_RESPONSE** exchanges
13) *Controller* notices that the disconnection failed on *Talker* and tries to reconnect *Listener* with a **CONNECT_RX_COMMAND.** Once *Listener* will be connected again, *Controller* will be able to try again a clean disconnection

Note: the controller always has to reconnect the listener before trying again the disconnection. Indeed, the controller cannot send directly a **DISCONNECT_TX_COMMAND** to the talker, and it's no use to send a **DISCONNECT_RX_COMMAND** to the listener because it will always reply with **status=NOT_CONNECTED** without even trying to send a **DISCONNECT_TX_COMMAND** to the talker.

4.7. Talker connected to a ghost Listener

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that there are situations where the controller may be unable to disconnect a talker.

1) *Controller* successfully establishes a connection between *Talker* and *Listener*
2) *Listener* is removed from the network without clean disconnection
3) *Controller* cannot disconnect *Talker* because it is not allowed to send a **DISCONNECT_TX_COMMAND**

Note: the only way to exit from this locked situation is to add back the listener to the network and to send a **DISCONNECT_RX_COMMAND**, or to reboot the talker. While the talker source is connected to at least one listener sink, even if this listener has disappeared, it will continue to advertise its stream and consume some SRP resources in the network.

4.8. ACMP connection succeeds even if no bandwidth for the stream

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that SRP bandwidth allocation error shall not change the status of an ACMP connection.

1) *Controller* sends **CONNECT_RX_COMMAND** to *Listener*
2) *Listener* receives **CONNECT_RX_COMMAND**
3) *Listener* sends **CONNECT_TX_COMMAND** to *Talker*
4) *Talker* receives **CONNECT_TX_COMMAND**

5) *Talker* executes successfully the "connectTalker" function. The "connectTalker" function requests the SRP stack of *Talker* to advertise and register the right MRP attributes as soon as possible in order to advertise its stream and reserve bandwidth for it on the path from *Talker* to *Listener*. Let's suppose that, at this time, the SRP stack of *Talker* already knows that there is no more bandwidth available on the link. Thus it immediately advertises a Talker Failed attribute instead of a Talker Advertise.
6) *Talker* sends **CONNECT_TX_RESPONSE** with **status=SUCCESS** (and not TALKER_NO_BANDWIDTH!!!)
7) *Listener* receives **CONNECT_TX_RESPONSE** with **status=SUCCESS**
8) *Listener* executes successfully the "connectListener" function.
9) *Listener* sends **CONNECT_RX_RESPONSE** with **status=SUCCESS**
10) *Controller* receives **CONNECT_RX_RESPONSE** with **status=SUCCESS**

We see there that the ACMP connection is established with no error between the talker and the listener, but no bandwidth has been reserved for the stream (the listener will be aware of that because it will receive a Talker Failed attribute).

Note: the TALKER_NO_BANDWIDTH status code (value=5) defined by the IEEE 1722.1-2013 standard shall never be used because a bandwidth allocation problem can never make an ACMP connection fail.

4.9. ACMP connection fails because MAAP fails

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that a MAAP error will prevent an ACMP connection to be done (contrary to an SRP error).

1) *Controller* sends **CONNECT_RX_COMMAND** to *Listener*
2) *Listener* receives **CONNECT_RX_COMMAND**
3) *Listener* sends **CONNECT_TX_COMMAND** to *Talker*
4) *Talker* receives **CONNECT_TX_COMMAND**
5) Let's assume here that *Talker* is configured to dynamically allocate a destination MAC address through MAAP (this means that it didn't receive a SET_STREAM_INFO command with stream_dest_mac<>00:00:00:00:00:00 and flags.STREAM_DEST_MAC_VALID=1.). Let's assume also that

Talker doesn't manage to allocate a MAC address through MAAP (even after a 1.5 second timeout). This may happen if there is an aggressive device on the network that always defends the addresses chosen by *Talker*.

6) *Talker* cannot establish the ACMP connection and sends **CONNECT_TX_RESPONSE** with **status=TALKER_DEST_MAC_FAIL**
7) *Listener* receives **CONNECT_TX_RESPONSE** with **status= TALKER_DEST_MAC_FAIL**
8) *Listener* doesn't connect because **status<>SUCCESS**
9) *Listener* sends **CONNECT_RX_RESPONSE** with **status= TALKER_DEST_MAC_FAIL**
10) *Controller* receives **CONNECT_RX_RESPONSE** with **status= TALKER_DEST_MAC_FAIL**

We see there that the ACMP connection is not established due to the fact that the talker has no MAC address for its stream. Please note that if the talker continuously fails in dynamically allocating a MAC address through MAAP, the controller may assign a predefined MAC address to it thanks to the SET_STREAM_INFO command (if the talker implements this command).

4.10. MAAP fails after a successful ACMP connection

In this scenario, there are 3 AVDECC entities:

- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that a MAAP error occurring after a successful ACMP connection will not break the ACMP connection.

1) *Controller* successfully establishes a connection between *Talker* and *Listener*
2) *Talker* loses the MAAP address range it had allocated for this connection
3) *Talker* immediately tries to allocate a new MAAP address range. During this time, *Talker* doesn't send any packet to the old MAC address and asks the SRP stack to stop advertising any Talker attribute.
4) As soon as *Talker* has managed to allocate a new MAAP address range, it asks the SRP stack to advertise a Talker attribute with the new MAC address. Please note that the SRP stack may delay this declaration (up to 30 seconds) due to the inherent constraints of the SRP protocol.

Notes:

- If the talker never manages to allocate a new MAAP address range, it stays connected but doesn't declare any Talker attribute.
- If the talker manages to allocate a new MAAP address range for its stream, all subsequent **CONNECT_TX_RESPONSE**, **GET_TX_STATE_RESPONSE** and **GET_TX_CONNECTION_RESPONSE** messages will carry the new MAC address

- A talker changing its stream destination MAC address will not inform anybody asynchronously through ACMP messages, but it can send an unsolicited GET_STREAM_INFO message to the registered controllers
- The listener must be prepared to take into account SRP stream parameters different from the parameters advertised in the **CONNECT_TX_RESPONSE** message.

4.11. SRP stream parameters different from ACMP stream parameters

In this scenario, there are 3 AVDECC entities:
- A listener: *Listener*
- A talker: *Talker*
- A controller: *Controller*

The goal of this scenario is to show that a Talker shall give precedence to SRP stream parameters over ACMP stream parameters.

1) *Controller* successfully establishes a connection between *Talker* and *Listener*
2) *Listener* initializes the parameters of the stream it is going to receive with the fields of the **CONNECT_TX_RESPONSE** message (**stream_dest_mac**, **stream_vlan_id** and **flags.CLASS_B**). In particular, *Listener* subscribes to the right VLAN ID and starts listening to data packets with the correct destination MAC address and the priority code point associated to the correct SR class.
3) Later, the SRP stack of *Listener* receives a Talker attribute with different stream parameters (destination MAC address and/or VLAN ID and/or SR class). *Listener* must use these SRP parameters rather than the parameters of the **CONNECT_TX_RESPONSE**. In particular, it may have to subscribe to another VLAN ID, listening to another destination MAC address and another priority code point.
4) Later, *Listener* may receive again different stream parameters through SRP. In this case, *Listener* shall use the latest SRP parameters.
5) If at one point, *Listener* doesn't receive any Talker attribute anymore, it keeps the last received SRP parameters and don't fall back to the initial parameters received in the **CONNECT_TX_RESPONSE** (these ones are now completely obsolete).

Note: the listener shall always fill the fields of the **GET_RX_STATE_RESPONSE** message with the stream parameters it is currently using. This means that if these parameters have changed since the ACMP connection has been established, the listener will return the latest received SRP stream parameters.