Hierarchical Stream Configuration

G Fedorkow, Adamson Systems, Jul 15, 2010

During the last F2F meeting, we finally got far enough with the coding formats discussion to realize that, given the variety of formats already supported by 1722 and 61883, a single Stream Setup message was simply not going to do the job for 1722.1. Philip Foulkes and Stephen Turner had each noted the hierarchical structure of the formats included in 61883, offering the opportunity to restructure stream setup to reflect the hierarchy of the coding scheme.

Although some of the coding formats used in IEC-61883 are simple unitary streams with a fixed format, e.g., the combined audio/video format used in BluRay DVDs, others are quite complex and flexible, as in the 61883-6 audio formats, including AM824.

This proposal segments the connection setup problem in a way that takes advantage of the modularity and hierarchy of the coding mechanism, allowing new codes to be plugged in as they're developed without having to change the basic stream setup procedure.

The approach uses one common module to set up an AVB stream, independent of media coding format. It then optionally uses one or more format-specific multiplex/demultiplex objects to combine primary media channels into a single AVB stream. The mux/demux blocks may be hierarchically arranged for coding formats that contain multiple primary media channels. Figure 1 shows an example for how AM824 might be configured on the talker side of a AVB stream.

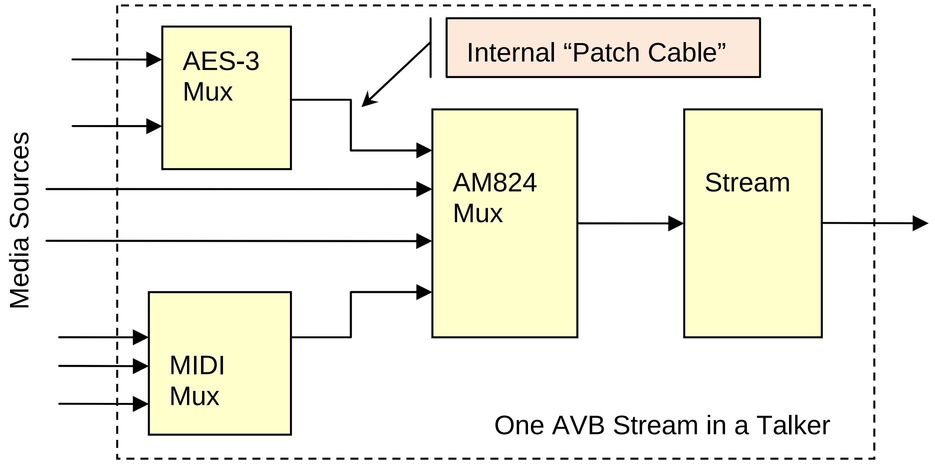


Figure 1: Talker-Side Stream Multiplexing

An important point to note is that each multiplexing block in the hierarchy may have several media inputs on the left side, but one output on the right side that plugs into the next block down the chain. The Listener side is an exact mirror image, with a stream delivering content to the first demux block, whose multiple outputs may go to several different demux blocks until primary channels are derived. This allows the multiplex/demultiplex blocks to be uniquely identified by a “handle” analogous to a Stream Source or Sink, but with no external connection.

The picture shows the blocks in the hierarchy interconnected by “patch cables”, i.e., connections from one block to another internal to the Talker or Listener.

When it comes to setting up streams, these hierarchical blocks can be configured independently, removing the need for one all-inclusive Stream Create procedure that knows every code format. Blocks are created dynamically, starting from the Media Sources, moving down the hierarchy to the AVB Stream. Arguments to each command for dynamic creation of the multiplex blocks give the inputs to the block, and output from the create function would be a handle (similar to a media source, but internal to the AVB block) that identifies the patch cable, plus any bandwidth parameters needed by the next layer down. Outputs from the blocks in Figure 1 are used as inputs to the next block to the right.

Listener configuration is the mirror image, with dynamically created demultiplex blocks starting from the Media Sinks, working back to the received AVB stream.

[add an example for demux-ing MPEG-TS]

[text-fragment: Similarly, if there’s a media stream source that knows what to do with an MPEG transport stream, then connection setup simply patches that media source into the Stream object. If the media source has separate audio and video channels, then something needs to insert an MPEG multiplexer to take the audio streams and video streams and put them together into a transport stream (or more likely take them apart at the Listener).]

1.1.1 A Note on Type-Checking Media Channels

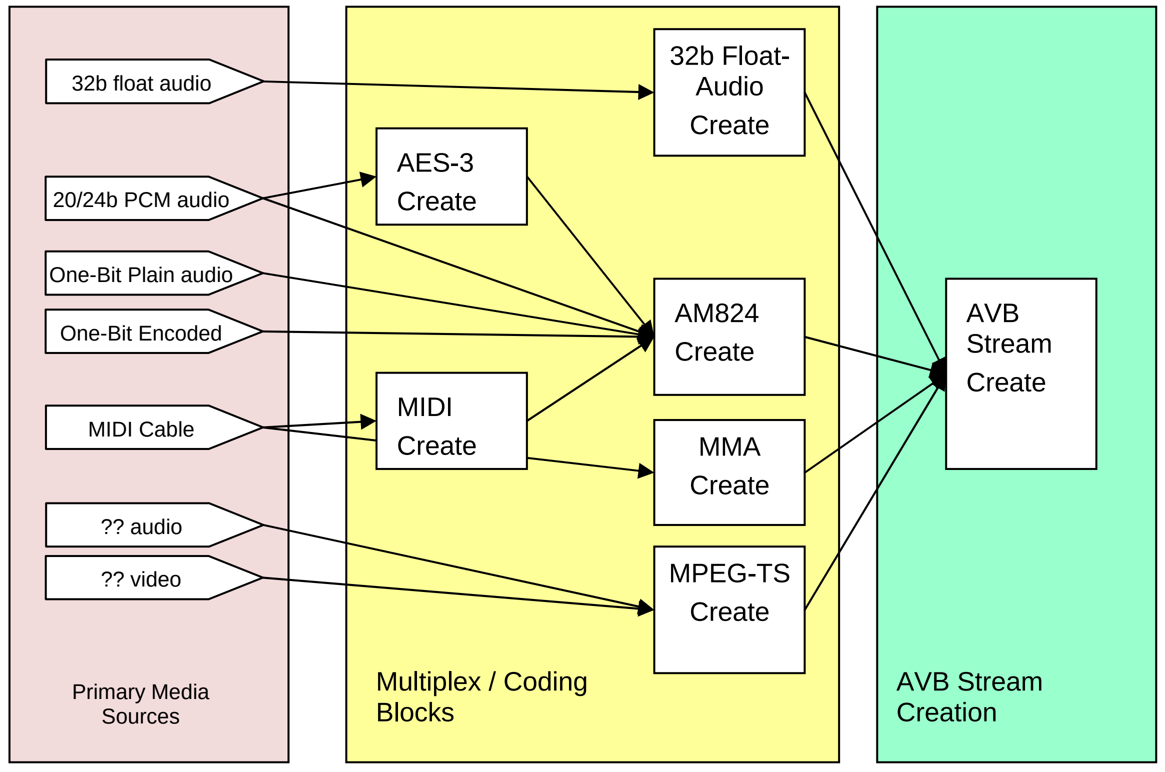
While the blocks in each level of the hierarchy should be flexible, it should be noted that this is not an “anything plugs into anything” hierarchy… the output of an AM824 block can only go directly to one stream, and a single stream cannot accept input from multiple AM824 blocks. Similarly, the block that combines two unformatted 24-bit PCM audio streams and converts it to an AES-3 format in 61883-6 can only plug into two slots of an AM824 stream.

We should further note that, although 1722.1 defines Media Sources and Media Sinks as if they’re all interchangeable, of course that’s not true either. A compressed video stream source can’t be plugged into AM824, and 24-bit PCM audio source can’t plug into a 61883-5 SDL-DVCR stream.

But by plugging different coding blocks into the hierarchy, the AVB subsystem can multiplex different numbers and types of media stream inputs and outputs, with flexible formats.

To ensure that each level of the coding hierarchy knows what media format to expect, and to check for incompatible requests, we can combine a Coding Type tag with a Media Source ID number, in a way that’s analogous to the tag format used in AM824. Implementations can add a degree of consistency checking by making sure that Coding Tag part of the Media Source and Media Sink identifiers line up with the stream setup blocks to which they’re being routed. So for example, an XLR connector on the back of a mic preamp might be represented internally as having an ID number corresponding to the plug, and Coding Tag of 24-bit PCM. When a stream setup is done routing the mic-preamp to an AM824 multiplexer, then the AM824 mux knows what it’s putting in that slot. But if the stream setup routes the mic preamp to a MIDI multiplexer, an error should be generated.

Figure 2 shows how the coding hierarchy works. In this picture, it should be noted that this is not a diagram of data flow, or of a specific stream configuration; although this picture shows which coding types *could* be multiplexed into which other types, it does not show which combinations can be combined in a single connection (for example, AM824 can multiplex AES3, MIDI and PCM audio all at once, but further to the right, an AVB stream can only carry one of AM824, 32-bit Float audio, MPEG or MMA).



[This picture doesn't contain all the coding options and combinations yet!]

Figure 2: AVB Stream Creation Hierarchy

Media Source/Sink Channel Code (MSCC)

Media Sources and Sinks, as well as Patch Cables used within the coding hierarchy, are all specified as 32-bit numbers, which are structured as follows, with a part that identifies the channel, and a tag that identifies the type of coding carried in that channel:

32 bits, split as follows
- o 16 bits define 'channel within media source'
- o 8 bits define media source within device [we should work on this split with JeffK]
- o 8 bits define Channel Coding Format (CCF)

Every 32-bit slot that will be assigned in an AM824 packet must have an MSCC in the *create* message, corresponding to the content that will be put in the slot.

CCF	Function	
0	Don't use this code point	
1	Unused Slot	Use this code when there's a slot reserved, but nothing in it. The talker should talk zero in this slot, the listener should ignore it.[1]
Primary Media Channel Types[2]		

[1] This function might be helpful on the talker if there's some reason for an empty slot. But it's critical on the listener side to allow the listener to pick off one or two channels from a big bundle. Fill in Media Sink information for the slots you want to hear, and Empty for the rest.

2	24-bit linear MBLA	
3	20-bit linear MBLA	
4	16-bit linear MBLA	
5	One-bit audio	
6	One bit audio encoded	
7	MIDI Data	AM824 tags are inserted at run-time depending on whether there's MIDI data to send or not.
8	First Word, high-precision MBLA	
9	Second Word, high-precision MBLA	
…		
15	Eighth Word, high-precision MBLA	
16	32-bit float	
Multiplexed Channel Types		
17	AM824-muxed channels	
18	MPEG-muxed channels	
	… plus many others for other 61883 formats …	

Table 1: (Incomplete list of) Channel Coding Formats

1.1.2 Stream Creation Commands

To set up a complete stream (on the talker side), the Controller must start from the left, "creating" the functional units. Each unit takes a list of inputs as its arguments, and returns an output handle (I've called it a Patch Cable for now). The next unit to the right takes the patch cable(s) returned by the previous one(s) as part of its input arg list.

Each unit may return some kind of bandwidth specification as well to feed into the next block down the chain.

Note that these commands deal only with *configuration*, not data flow. Endpoint designers are free to optimize their data paths any way they want; all these commands do is to say which multiplexers send media content to what other blocks in the chain *inside a single endpoint*.

[2] I've used "Primary" to describe channels that enter or leave the AVB subsystem, and Multiplexed to describe channels that have undergone one or more stages of multiplexing within the AVB subsystem. Note that these types aren't immutable – for example, and AVB Endpoint with a capability to encode and decode MPEG transport streams might consider MPEG as a "multiplexed" internal channel format, while and AVB endpoint that could not decode MPEG could treat an MPEG TS as a "primary" channel type.

1.1.2.1Stream Create Commands

The first level of hierarchical stream create is just the AVB stream itself. This module takes inputs that identify the single source of media for the stream, as well as the traffic parameters required by AVB.

1.1.2.1.1 Talker_Stream_Create

Args:

- Stream Class (A or B)
- TSpec (Traffic Specification; this number gives the maximum size of the packets to be sent, and the number of packets per "Class Interval", i.e. number of packets per 125 usec interval for Class A)
- Data Frame Priority and Rank
- Presentation time offset (default 2 msec)
- Stream Name
- Patch Cable input
  Acceptable Input coding formats:
  - AM824, MMA (Midi), MPEG-TS, 32-bit Float (and a couple of others)

Returns

- StreamID
- MAC Addr

Syntax

/avb/source/create/stream

[insert a proposal for command syntax here!]

If the requested stream has already been successfully created and the client requests a stream source to be created with the exact same name and parameters, then the server will respond with the existing stream information.

Stream source names must be unique within a single talker device.

Listener Stream Create

[The listener gets the Tspec, Class and Rank from the SRP advertisements, right?]

Args:

- StreamID
- MAC Address (if using Three-Step)
- Patch Cable output

Returns

- Ok/notOK

1.1.2.2 Media Coding and Multiplexing Block Creation Commands

1.1.2.2.1 AM824_Mux_Create

Args

- Sample Rate

- Channel Map with media sources or patch cables
- Acceptable Input coding formats:
    - MBLA, OneBit, OneBit-Encoded, MIDI, high-precision MBLA, etc

Returns

- Patch Cable output
- Bandwidth params / TSpec

Syntax

/avb/source/create/am824 (Talker-side command)

Create an AM824 multiplexer to plug into an AVB stream source:

1 **TX:** ["/avb/source/create/am824" ,ib (number of channels) (media source channel code)...]

3 **RX:** ["/avb/source/create/am824" ,iiib (handle) (bandwidth param) (number of channels) (media source channel code)...]

/avb/sink/create/am824 (Listener-side command)

[insert detailed syntax here]

The key idea for AM824 channel configuration is the creation of the channel map, the list of media sources that will be packed into an AVB packet. Each entry in the channel-map list represents one slot in a 61883-6 packet, in the order that the packet will be assembled.

A Talker can use the Unused_Slot code point as a place holder to indicate that a time slot should be reserved, but it contains only zeros.

More important is the use of the Unused_Slot code at the listener; if a particular Listener wants to pick off the third AM824 slot of six, it would put Unused_Slot as the first two elements in the list, then name the Media Sink to which it wants to route the valid channel, and then Unused_Slot in the remaining three elements of the list.

The Handle that's returned by the AM824 Create command should be passed into a subsequent Stream Create command.

1.1.2.2.2 AES-3_Create

Args

- Media source left
- Media source right
- Acceptable Input coding formats:
    - MBLA

Returns

- Patch Cable Out
- (Bandwidth params are implicit; each AES-3 flow takes two slots in an AM824 Mux)

[And there are lots more commands like this to be figured out]

1.1.2.3 Stream Teardown

Each of the Create commands should be paired with a Delete command.

In addition, a Delete of an element in the hierarchy should imply a Delete of all the elements to the left of the deleted element. In particular, to tear down a complete stream (looking at the Talker side again), use a Delete Stream command; that will remove the stream and all the multiplexing blocks that had been configured to feed media content into the stream.

1.1.2.4 A Note on Parameter Passing

The technique outlined above implies that all of the configuration information passing between blocks is fully visible, so, for example, one Create command may return bandwidth parameters needed by the next Create command in the hierarchy, in addition to returning the handle needed. This offers maximum flexibility and visibility, but requires more information to get passed back and forth.

There are two possible optimizations

a) All the blocks being configured are inside the same endpoint, so the endpoints could associate bandwidth parameters with handles and propagate them automatically as the blocks are assembled. For example, the creation of an AM824 block returns a handle which would be used in the creation of an AVB stream; when the stream create command is issued, it should contain the handle, and the endpoint can look that up to find what bandwidth parameters should be passed to SRP.

b) But a more serious performance issue with the approach outlined above is the number of round trips needed to configure a complex stream. Each block in the hierarchy needs a round trip command and response to deliver the handles needed for the next command.
A different approach would be to specify a command set that explicitly recognizes the hierarchy, allowing a single command with nested components to set up a complex stream.[3]

```
create_stream(create_am824(primary_source1, primary_source2, create_aes2(primary_source3,
primary_source4), create_midi(primary_source5))
```

This approach gets a complex setup done in one step, hides all the bandwidth calculations, and even eliminates the visible “patch cables”. But for AVBC or SNMP messaging, this approach is not so good, as the messages for a complex stream could get quite long, making it much more complicated to stick with UDP-based protocols like SNMP or AVBC.

1.2 Notes

1.2.1 Sections from P802.1Qat (SRP)

[This section quotes material relating to setup parameters from the SRP doc for ready reference]

Virtual Bridged Local Area Networks -Amendment XX: Stream Reservation Protocol (SRP) P802.1Qat/D4.2

35.2.2.8.4 TSpec

The 32-bit TSpec component is the Traffic Specification associated with a Stream. It consists of the following two elements (which are encoded as described in 10.8.1.1):

a) **MaxFrameSize:**

The 16-bit unsigned MaxFrameSize component is used to allocate resources and adjust queue selection parameters in order to supply the quality of service requested by an MSRP Talker Declaration. It represents the maximum frame size that the Talker will produce, excluding any overhead for media specific framing (e.g., preamble, IEEE Std 802.3 header, Priority/VID tag, CRC, interframe gap). As the Talker or Bridge determines the amount of bandwidth

[3] Think of this like a programming language where the argument list for a function call may itself contain function calls.

to reserve on the egress port it will calculate the media specific framing overhead on that port and add it to the number specified in the MaxFrameSize field.
b) **MaxIntervalFrames:**
The 16-bit unsigned MaxIntervalFrames component is used to allocate resources and adjust queue selection parameters in order to supply the quality of service requested by an MSRP Talker Declaration. It represents the maximum number of frames that the Talker may transmit in one “class measurement interval” (34.4) [*i.e., 125 usec for Class A*]

35.2.3.1.1 [Talker] REGISTER_STREAM.request

REGISTER_STREAM.request (
    StreamID,
    Declaration Type,
    DataFrameParameters,
    TSpec,
    Data Frame Priority,
    Rank,
    Accumulated Latency)

35.2.3.1.5 [Listener] REGISTER_ATTACH.request

REGISTER_ATTACH.request(
    StreamID,
    Declaration Type)