Contents

1.1 P1722.1 AVB Connection Sequencing Examples

Guy Fedorkow, Adamson Systems, version 3.0

Jun 18, 2010

1.1.1 Use Case

For the pro-audio, live-sound use case, we have a couple of goals:

- The initial desired functionality is a “distributed, peer-to-peer patch panel,” i.e., there’s a list of sources of streaming content, and a list of destinations, and the goal of connection sequencing is to get the right content to the right destinations across named connections.
- It’s important that the system be resilient, and that it recover quickly after failure.
- A system controller (which may be a separate computer, or a functional unit in some other AVB-enabled device) may be required for initial setup, but should not be required for routine operation (and restart) of the system.

A System Controller is desirable for setting up complex systems, but that reliance on a System Controller once configuration is complete is not desirable. To reduce dependency on a System controller, the approach outlined here uses two interlocking phases:

- Part of the procedure discovers and enumerates all the visible AVB components on a network segment, then allows a connection manager to identify and configure talkers and listeners, and to associate them via named connections. A "named connection" might be something like "Right Side Fill", a human-readable name that would be (for example) assigned to an output channel from a mixing desk (the talker) and sent to a group of amplifiers that power side fill speakers on the right side of a stage.

  The result of this first phase would not be actual creation of connections, but rather the configuration of each endpoint with the relevant StreamIDs and parameters of connections it should make.

- In the second part of the procedure Listeners subscribe to the StreamIDs that they've been configured to receive using protocol sequences defined for SRP.

The two phases shouldn't be thought of as distinct and non-overlapping... it's more that the first part identifies which endpoints should participate in what connections, and transmits that information to the endpoints in the form of StreamIDs, while the second part carries out the actual nuts and bolts of making the connection. If the management station changes the connections while a system is operating, the affected endpoints should drop whatever connections they're no longer supposed to have, and start making new ones.

1.1.2 Configuration and Connection Phases

Figure 1 shows that several different configuration techniques could be used, with one uniform method of setting up connections.

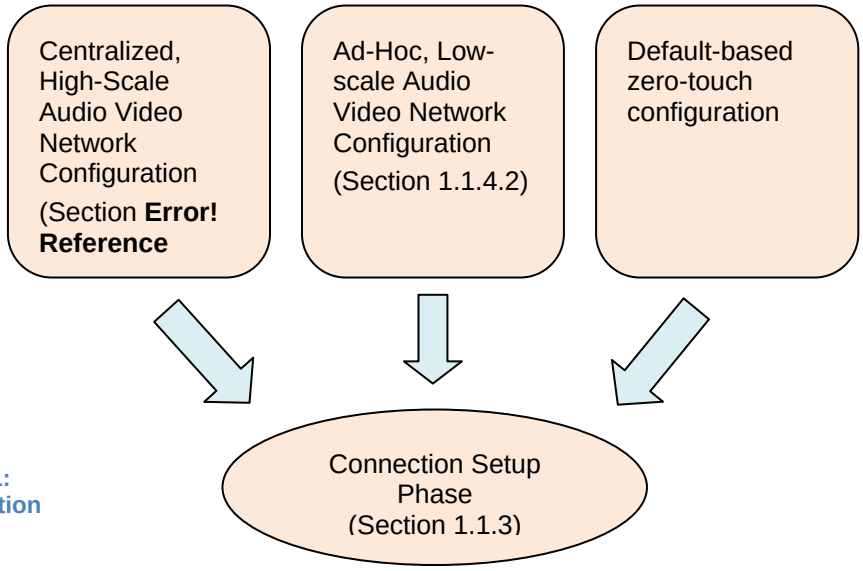


Figure 1: Connection Phases

1.1.3 Connection Setup Phase

At this point, assume that devices all have the StreamIDs configured, and are now starting to set up connections. This stage could be reached either because the connection manager above finished configuring connection(s) or because a previously-operational system is starting up after a power failure.

AVB allows two forms of connection setup, Two-Step and Three Step.

Basic two-step SRP setup is already completely defined by Stream Reservation Protocol (802.1Qat D6.1); in this case, the Listener allocates a VLAN and makes a reservation using the StreamID of the desired stream, as shown in Figure 2 below.

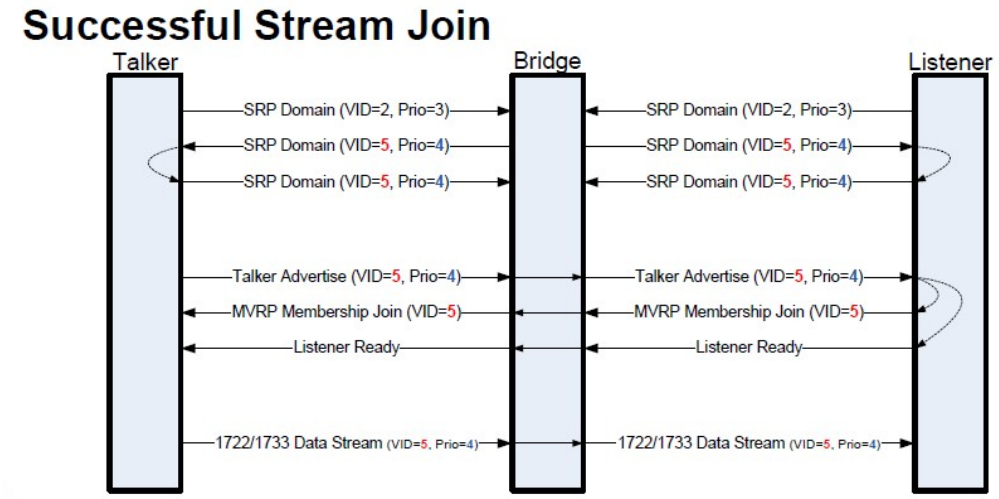


Figure 2: Two-Step Stream Setup

(picture from http://www.ieee802.org/1/files/public/docs2010/at-cgunther-srp-d50-to-d61-0610.pdf)

For Three-step setup, switches use Multicast Pruning, so the Listener won't see SRP advertisements from the Talker without registering for the Talker's mcast MAC address. To obtain that MAC address, the Listener must ask the Talker for the appropriate MAC address.

In this case, a unicast Protocol X sequence is used on startup for the Listener to ask the Talker for the required MAC address. Once the Talker MAC has been obtained, the Listener can register with the local switch to receive multicasts on that address, and it can then use the StreamID to register with the talker.

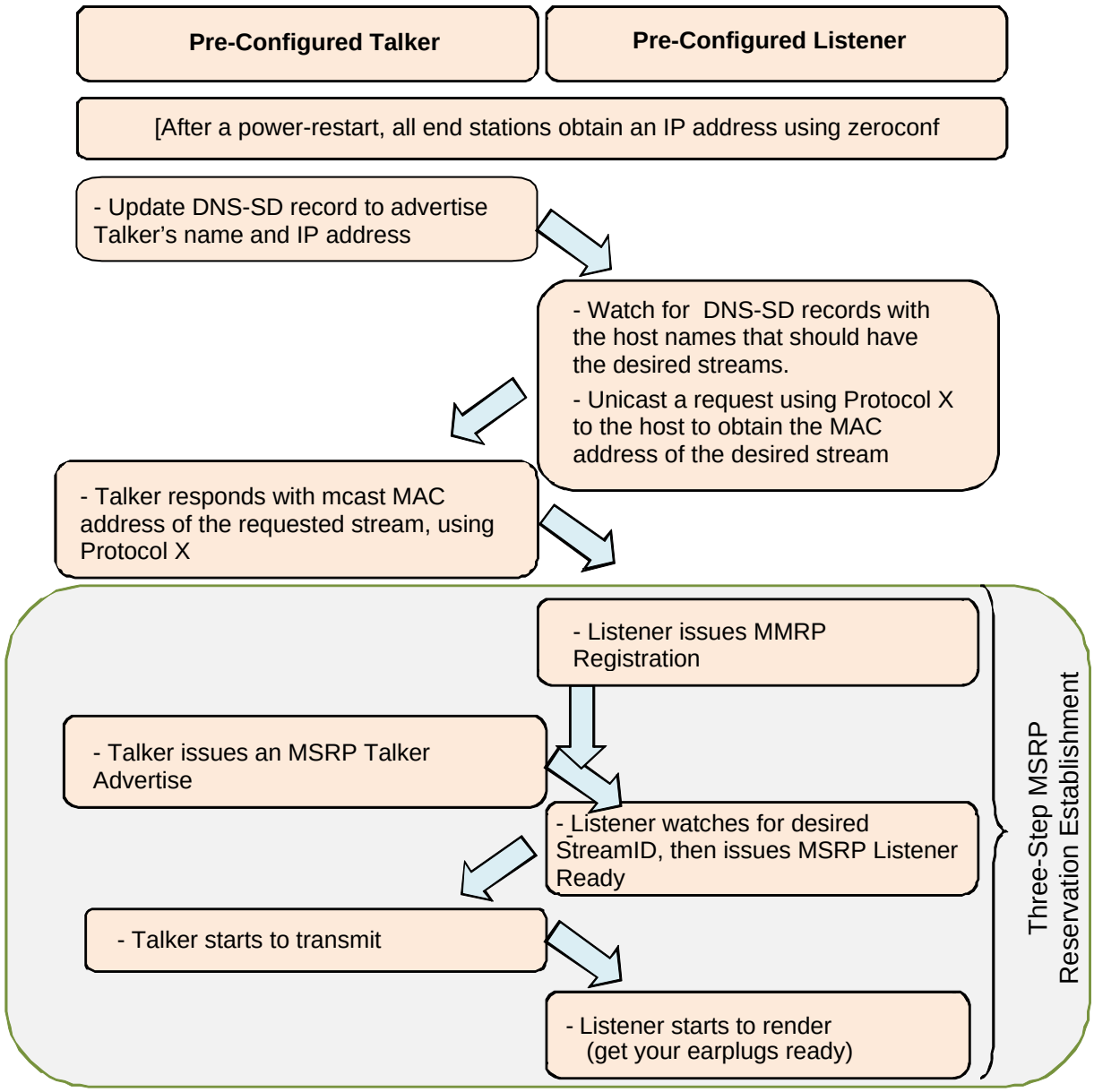
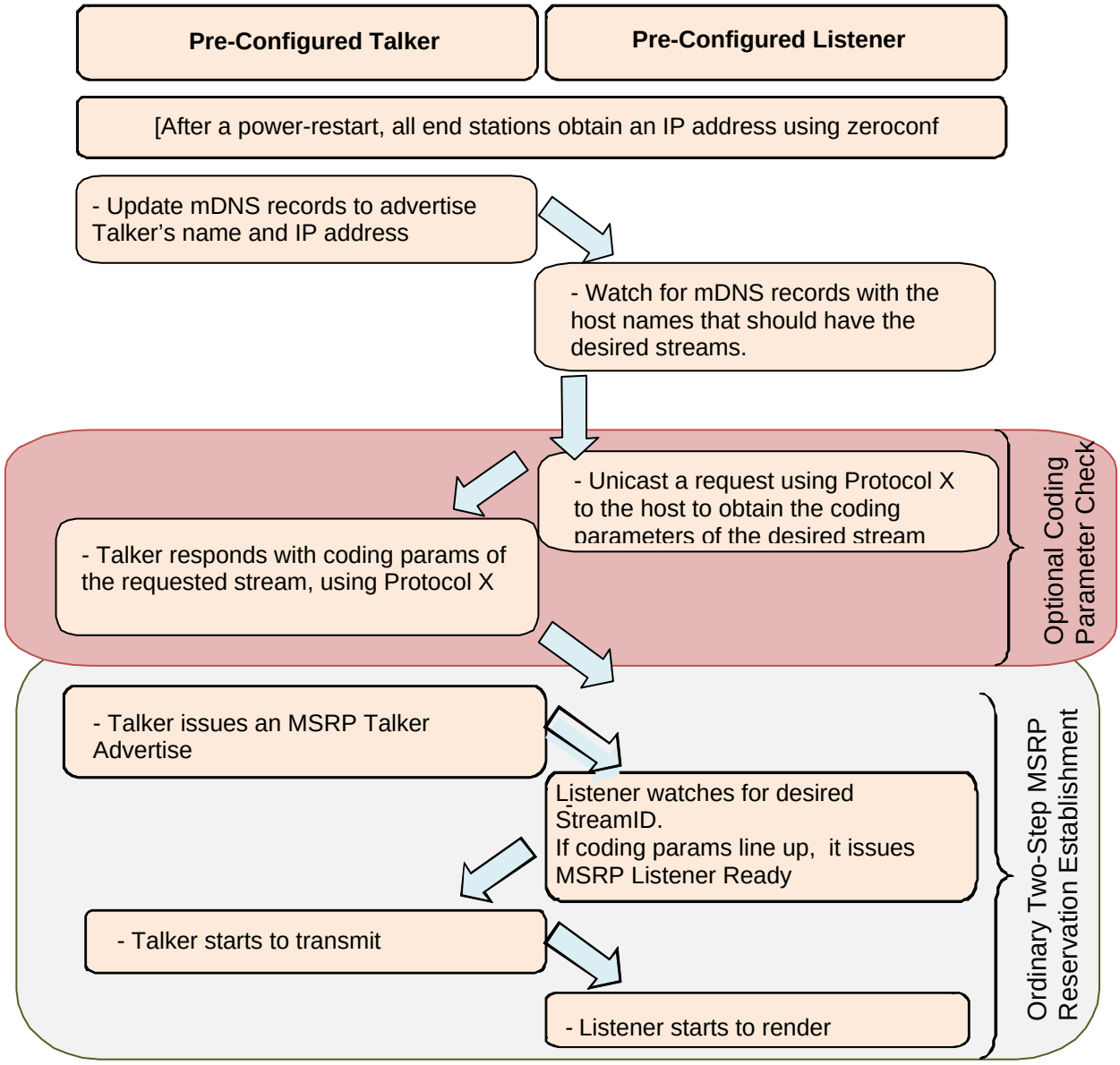


Figure 3: Three-Step Setup

1.1.3.1 Optional Stream Checking

In the examples above, Listeners attach to Talkers based on nothing but a cached StreamID, even though configuration changes may have taken place while the parts of the system were disconnected from the network. System designers may want to ensure that the Talker and Listener still agree on coding before enabling a stream.

This can be accomplished in either the two-stage or three-stage cases by inserting another protocol exchange before completing the SRP setup:



1.1.3.2 Device Replacement

We want to configure systems once and have them restart without a management station, and that is accomplished by programming StreamIDs into Talkers and Listeners to cause connections to be set up autonomously at startup.

But devices fail, and must be replaced.

In the cases described above, the management station must be used to program the new device, be it a talker or listener. Listeners need to be told to what StreamID they should listen. And a Talker's streamID may be based on its MAC address, so a replacement is guaranteed to have a different number.

As an *optional* behavior, it would be possible for an endpoint to discover that a speedy connection by StreamID was not working so well, and try to make the connection by name. This would allow new devices to be programmed off-line with connection names, without needing to plug a management station or device into a live network to discover StreamIDs.

The plan would work as follows:

- Talkers and Listeners would have to be programmed by users with predictable and stable host names. Zeroconf takes care of collisions, but the intent would be to have devices named in a way that collisions are unlikely.
- Streams and channels would be named in both the Talker and Listener. Names for streams and channels should be provided anyway to allow someone at a management station to understand how connections are being used.
- The Listener would be programmed to connect to the appropriate StreamID (as above), but would also store the corresponding host name and stream name in non-volatile memory.

Upon startup:

- A Listener would attempt to connect to a talker using the StreamID (as described above).
- If the connection fails, the Listener can send a Protocol X request containing the connection name to the hostname corresponding to the Talker, to learn the StreamID and channel map for the named connection
- If that works, the Listener can then update its non-volatile copy of the StreamID and complete the connection using the standard procedure.

This behavior does not have to be 'normative' – it does need access to appropriate Protocol X commands to learn the right information, but the algorithmic steps are within the scope of something a Listener could do all on its own.

1.1.4 Configuration Phase Sequencing Steps

Section 1.1.3 above starts with the assumption that endpoints have StreamIDs already configured, and all that's needed is to reserve bandwidth in the network in between. This section outlines how those StreamIDs would be configured.

This section sketches two ways to configure StreamIDs

- One assuming a large centrally-managed system
- The other assuming a small ad-hoc system

The basic flow should be the same for all kinds of devices – high-end pro down to low-cost consumer.

In each case, there will be a network that has various AVB devices attached.

In each case, there will be some device that has a human interface that can be used to initiate the connections. That human interface might be a management workstation in the case of a large pro system, or maybe it would be an integrated component in a simple device like a DVD player. But in either case, a similar set of steps should be executed.

- All devices start by obtaining addresses using Zeroconf procedures.
- All Devices register a host-name using mDNS
- All devices would advertise their AVB capability using DNS-SD
- The management device would go to each device and discover
    - The available media sources and sinks
    - Supported coding options
    - Currently-configured streams and channel maps
- With this information, the management station can display existing allocations of media sources and sinks to streams, and stream connections between devices.
- The management station can also issue commands to create new media streams.

The following sections outline the procedure in more detail.

1.1.4.1 Stream Configuration Use Case

The following case assumes that streams are to be configured using a device like a PC or laptop. The sequence of Protocol X commands does not have to be followed exactly, and different systems might change the order depending on implementation details or scaling requirements.

Upon power-on, all AVB endpoints start up and carry out the routine steps:

- Endpoints make any pre-programmed connections using StreamIDs stored in non-volatile memory
- Endpoints use AutoIP or Zeroconf to obtain an IP address
- Endpoints use mDNS to register an endpoint device name. If a device name has not been configured, use Zeroconf rules to make one up.
- Endpoints use DNS-SD to advertise basic AVB capability, e.g.
    - o "I can do AVB Audio"
    - o "I'm an Adamson Powered Speaker, Model YXXX"

  (See Section 1.2 for details on the Text Record)
- Endpoints then start a Protocol X server, and await commands from the connection manager.

Subsequent action is triggered by the connection manager (CM), which might use the following steps:

1. The CM would start a Zeroconf Browser to display the devices found on the network. See XX for an example of what this could look like.
2. To prepare to set up connections, the CM would collect up a complete list of media channel sources and sinks from all devices, listing each channel by the name assigned in the endpoint.

```
TX: [ "/media/source/*/type" ]
        returns a type1 field for each media source.

TX: [ "/media/source/*/description" ]
        returns a description[2] field for each media source


TX: [ "/media/sink/*/type" ]

TX: [ "/media/sink/*/description" ]
```

3. The CM should also ask each endpoint to identify the coding and bit rate options that it's capable of supporting. Due to difficult-to-quantify internal constraints, the endpoint is not likely to be able to promise to make connections with any conceivable combination of the formats it has available for use, but this step narrows the list to the choices to the ones that are likely to work.[3]

[1] The 'type' field gives the generic purpose of the media source, e.g.., "microphone input", AES/EBU Left Channel Input".
[2] The 'description' field identifies the specific media source, e.g., "back-panel plug #5"
[3] For example, a device might be able to stream all its media channels using 48 kHz sample rates, but only a subset of the media channels if the rate is set to 192 kHz. Or a device might be able to do PCM format or AC3 format, but not both at once.

[we still need to figure this out]

4. The CM should ask each endpoint which streams it already has configured, and learn the channel maps for those streams

```
TX: [ "/avb/source/byname/*/" ]
        return all the configured stream names with streamIDs and MAC
        addresses

TX: [ "/avb/source/*/format" ]
        find out the format for each of these streams

TX: [ "/avb/source/*/map" ]
        find the list of channel numbers in each stream.  Index the
        channel numbers into the source/MEDIA_SOURCE_ID list to find
        the names [This listing will have to incorporate a coding
        format element too given AM-824 per-channel coding options]


TX: [ "/avb/sink/byname/*/" ]    - repeat it all for stream sinks

TX: [ "/avb/sink/*/format" ]

TX: [ "/avb/sink/*/map" ]
```

5. At this point, the CM has a list of every media channel and every stream that's already configured. From here, it can use whatever user interface it wants[4] to display the configuration, and to decide what new streams should be configured.
6. The CM can optionally issue proprietary control commands to configure EQ, gain settings, etc.
7. The CM can issue Protocol X commands to configure new streams. The result of these commands to talkers and listeners would be to program each device with the names, StreamIDs and channel maps of the desired connections.
   Talker streams should be configured first, so that Listeners can connect as soon as they're told what streamID to use.

   [Example needed]
8. Once Talker and Listener StreamIDs have been configured, connections can be set up using the procedure in Section 1.1.3 above

1.1.4.2 Small-Scale Stream Configuration

[This section is 'speculative']

Section 1.1.4.1 focuses on large-scale applications with lots of endpoints and a substantial configuration manager, but it would be good to imagine a small-scale use-case as well to make sure the protocols fit.

One possible consumer use-case would be a miniature version of the use-case described above in Section 1.1.4.1, in which there would be a bunch of devices like speakers, media players, set-top boxes, etc, all of which run Zeroconf plus Protocol X and all of which can be controlled by an iPod/iPad style

[4] E.g., read a list of desired connections from a file, or use a drag-n-drop GUI, or use a text-based command line, etc.

device. This is essentially the same as the pro case we've considered, except with ten times fewer channels, streams and devices, and probably more automation with fewer visible choices in selecting values for coding, etc. But from the general flow of user interaction, it's the same:

- The controller device discovers all the AVB devices
- It uses "Protocol X" to collect up the media sources and sinks, and reads back existing stream configurations
- It presents this to the user on an interactive screen, and invites the user to make changes to the connectivity. Changes then get saved back to the devices in terms of connection setup commands and put into effect.

But to push all this a bit further, a tougher use-case might be a couple of speakers with essentially no user interface, and a source of audio or video such as a DVD or CD player. The player might have nothing but an LCD display with a couple of buttons for navigating a menu. In this case, I'd equip the speakers with something like a mechanical rotary switch with half a dozen positions marked "A" through "F". When the speaker starts up, it should register an mDNS name that reads something like "Speaker-B". If two speakers are set to B, then I think Zeroconf would deal with that by arbitrarily making one Speaker-B1 and the other Speaker-B2. So be it.

The player has a tougher job, but it has to do essentially the same job as the controller in the previous example -- prowl the LAN to find the AVB devices, and ask them via Protocol X what they are and what they can do. Using a teeny menu, it would have to display the devices and connections it finds and then offer to configure streams. Although this sounds like a very tough user interface design problem, it seems like the protocol needs are essentially the same. The only difference I can think of (relevant to the Device Category list) is that the DVD player might cut the clutter by refusing to display any devices that it couldn't obviously connect to... e.g., do show anything that claims to be a loudspeaker or video monitor, don't show other media players, synthesizers, MIDI devices, etc.[5],[6]

This use case would benefit from well-defined default values for stream parameters (e.g. coding formats)[7]

[5] If we think this mode of operation might happen, we should scrub the device list with this use in mind.
[6] This doesn't mean that the DVD player can't connect to a player/recorder/ripper from SwissArmyAV Corp, just that you probably need to get out the laptop or iPad to make the connections
[7] Would the two choices be Two-channel, 48 kHz PCM, and Dolby 5.1 [what coding is this?]?