Getting Streams Cloud-Ready and CDN Compatible: Politics and Protocols

Rick Bidlack
Development Engineer, Wheatstone Corporation
New Bern, NC, USA
rick@wheatstone.com

Abstract – The broadcast industry is currently in a transitional phase, heading toward a world in which a large portion, perhaps even a majority, of listeners will receive the station's program via an internet stream rather than terrestrial radio broadcast. Many accomplished engineers and station managers, while experts in traditional radio technology, find themselves feeling like neophytes once again when it comes to internet streaming. This paper aspires to present an instructional overview of what's involved in this generation of an internet audio stream, as well as a better understanding of where standards and practices are still developing.

Broadcast Versus Streamed Content Signal Flow

Streamed content starts out on the same path as broadcast content, typically originating from the automation system if it's music or from the studio microphone if it's voice, or both. One, however, goes to the broadcast chain and the other goes to the streaming encoder, and that one difference introduces new considerations in both processing and metadata for streaming. To understand these considerations, it helps to get a closer look at the signal flow of each.

Broadcast Signal Flow

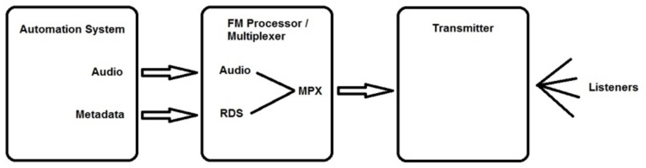

FIGURE 1: TYPICAL BROADCAST SIGNAL FLOW.

For broadcasting purposes, content is fed into an automation system, which sends its audio output to an FM (AM) processor, where it is processed and multiplexed, and then passed on to a transmitter which disseminates the program to the public. As a side channel, the automation system has also been stocked with metadata (artist, title, album, duration, label, ISRC, etc) for every piece of audio in its playlist, and this metadata is also transmitted to the FM processor, where it is turned into an RDS signal which rides as a sideband on the MPX output. There are, of course, variations in this flow path. For example, the metadata from the automation machine may instead be routed to an "aggregator" which cleans and ensures the accuracy and consistency of the metadata before it is forwarded to the FM processor. Or it might not even go to the processor, but to a separate RDS encoder, the output of which is then mixed with the MPX from the processor. The exact details may change, but the steps along the path remain essentially the same.

Stream Signal Flow

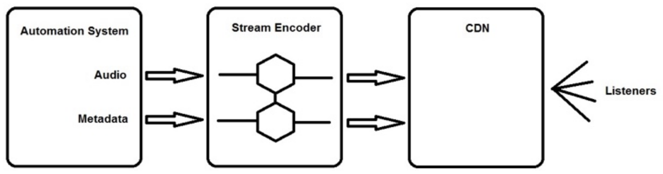

FIGURE 2: TYPICAL STREAM SIGNAL FLOW.

For streaming purposes, content enters the automation system as before, but in this case the output is routed to a stream encoder (also called an “origin server,” for reasons that will be clear later), and from there to a CDN, or Content Distribution Network.

The stream encoder has three jobs:

1. Processes and conditions the audio signals, optimizing it for the compression algorithms.
2. Encodes, packetizes, and transmits the program over the public internet to the destination server (the CDN)
3. Handles the reformatting and forwarding of metadata from the automation system on to the CDN.

CDN Streaming and Add-Ons

Before looking at the inner workings of the stream encoder, let’s examine the Content Distribution Network. The CDN plays the role of the transmitter in the streaming paradigm, but its unique position in the program flow path actually gives it a bigger and more important role. The main function of the CDN is to serve your stream to thousands or tens of thousands of listeners. But the twin facts that A) your program and all associated metadata passes through the CDN’s servers; and B) they know who is listening, from what location, and for how long – gives them an opportunity to provide a whole suite of add-on services.

A big one is ad replacement, which is usually geographically based, but could also be tailored to whatever can be deduced about the individual listener’s tastes and habits. Geo-blocking, logging, skimming, catch-up recording and playback, access to additional metadata (e.g. album art, fan club URLs), listener statistics and click-throughs, customized players, royalty tracking, redundant stream failover, transcoding from one format to another – these are some of the services that CDNs typically provide.

Thus, the CDN basically controls the distribution of the stream to the listening public. It is the responsibility of the stream encoder – the origin server to the CDN’s ingest and distribution servers – to make sure that the CDN gets the right data at the right time and in the right format.

Especially with regard to metadata, the CDN determines what the format will be. The stream encoder is therefore also a mediator/translator between the automation system and the CDN, as it must be able to transform the format of whatever it ingests into the format that the CDN requires. With that in mind, let’s take a closer look at the stream encoder.

Stream Encoder Job-One: Data Compression

The stream encoder performs various functions. The central function, which all other functions are designed in relation to, is to apply data compression to the audio stream, thereby reducing the bandwidth required both to transmit the stream up to the CDN, and for the listener to receive it.

By far the most widely used compression codec for high quality audio storage and transmission is AAC, but its predecessor MP3 is still used for legacy streams (to support legacy players). All compression algorithms operate by reducing the information contained in the audio signal to as few bits as possible; you therefore want to maximize the value of those bits by removing extraneous artifacts from the audio signal before it hits the compression codec. That cleansing and conditioning of the raw audio signal is the job of the DSP section at the input.

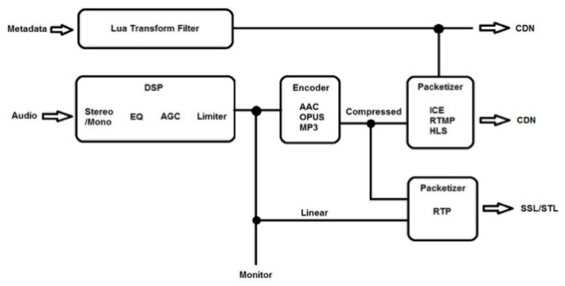

FIGURE 3: THE ENCODER'S CENTRAL FUNCTION IS DATA COMPRESSION.

Encoder Handoff to CDN

The compression codecs produce periodic output in the form of chunks of data, each of which represents a small segment (typically 5 to 20 milliseconds) of the continuous audio signal. These chunks are wrapped in a transport format and transmitted to the CDN's ingest server as discrete, time-stamped packets. Common transport protocols include Icecast, RTMP, HLS, MPEG-DASH and RTP.

Metadata is typically received by the stream encoder on a TCP or UDP socket, and most commonly arrives formatted as XML. What happens after that depends on the transport protocol being used. For Icecast streams, metadata updates (including ad triggers) are sent to the server out-of-band, as separate HTTP messages. For RTMP, the metadata update is embedded in the transport stream itself, as a special INFO type packet. For HLS, metadata may be reformatted as ID3 data and embedded into a separate channel of the underlying MPEG2 Transport stream, and ad triggers can be woven into the manifest file as SCTE-35 "program replacement opportunities."

For all of these methods, the exact details can differ from one CDN to the next since there are no universally accepted standards for handling metadata, so setting up a stream and getting the metadata to update correctly and in sync with the audio stream often requires a short period of trial and error while negotiating protocols with the CDN.

RTP is a special case. RTP is the only stream format that is commonly used to carry uncompressed, full bandwidth audio (in the form of AES67, for example), although it can be used to carry virtually any kind of data, including AAC, Opus or MP3 audio. RTP is typically used in a studio-to-studio or studio-to-transmitter link, rather than as an ingest feed to a CDN.

Metadata Formatting

The final area we want to examine is the metadata itself. We've been talking about transforming it from one format to another, but what does that mean, and why do we need to do it? To answer that question, let's look at some examples of raw metadata received from various automation systems. (Names and numbers have been changed to protect the innocent.)

Some systems export simple tagged text:

```
artist=Sonny Rollins
title=God Bless The Child
length=00:07:29
```

More commonly, metadata arrives in XML or XML-like form. Note that many fields may be empty:

```
<nowplaying><sched_time>220200</sched_time><air_time>402000</air_time><stack_pos></
stack_pos><title>TheSkyIsANeighborhood</title><artist>FooFighters</
artist><trivia>*</trivia><category>MNJ</category><cart>R585</cart><intro>14000</
intro><end></end><station>93.5HD1</station><duration>243300</
duration><media_type>SONG</media_type><milliseconds_left></
milliseconds_left><Album>ConcreteandGold</Album><Field2></
Field2><ISRC>USRW9170028</ISRC><Label>RCA</Label><Tempo></Tempo><Year>2017</Year></
nowplaying>
```

The HTML ampersand entity in this one will require special handling:

```
<audio ID="id_3155460704_30756200">
<type>Song</type>
<status>Playing</status>
<title>JINGLE BELL ROCK</title>
<artist>Hall &amp; Oates</artist>
<length>00:02:03</length>
<category>Classic</category>
</audio>
```

Here's a station ID and a liner from the same source. Some CDN's may want this data, some may not:

```
<audio>
<type>Link</type>
<status>Playing</status>
<artist></artist>
<title>ROCK91.7 | LEGAL ID2</title>
<number>120002</number>
<length>00:00:11</length>
</audio>

<audio>
<type>VoiceTrack</type>
<status>Playing</status>
<artist></artist>
<title>VT TLA3 - Rock: 2021-03-25 08:37</title>
<number></number>
<length>00:00:06</length>
</audio>
```

Here's a sweeper and an ad from another source, below. Notice that the <media_type> tag distinguishes one from the other. The arrival of a SPOT is often used to trigger an ad replacement by the CDN. Durations shown here are in milliseconds.

```
<nowplaying><sched_time>67199000</sched_time><air_time>67267000</
air_time><stack_pos></stack_pos><title>CYQTONEQUICKSWEEP#12DRY</
title><artist>201971108:06:28</artist><trivia></trivia><category>IM5</
category><cart>BIBE</cart><intro>0</intro><end></end><station>95.7KQED</
station><duration>3900</duration><media_type>UNSPECIFIED</
media_type><milliseconds_left></milliseconds_left><Album></Album><Field2></
Field2><ISRC></ISRC><Label></Label><Tempo></Tempo><Year></Year></
nowplaying><nowplaying><sched_time>33716000</sched_time><air_time>33721000</
air_time><stack_pos></stack_pos><title>ShopLocalAWestfieldMerchantsFriday</
title><artist>WestfieldMerchantsJune</artist><trivia>shoplocalgreencountry</
trivia><category>COM</category><cart>4652</cart><intro>0</intro><end></
end><station>95.7KQED</station><duration>30000</duration><media_type>SPOT</
media_type><milliseconds_left></milliseconds_left><Album></Album><Field2></
Field2><ISRC></ISRC><Label></Label><Tempo></Tempo><Year></Year></nowplaying>
```

None of the metadata messages in these examples can be transmitted to the CDN ingest server in the form they are in. Instead, the relevant data must be stripped out and reformatted into a protocol that the ingest server can understand. This protocol often requires that additional information, which is not available in the original metadata received from the automation system (such as login credentials, the server URL, and query parameters) be woven into the message. For an Icecast stream, that might take a form like this:

```
http://source:abc123@ice5.streamnet.com/admin/metadata?mount=/
BRAVOFM1im&mode=updinfo&song=Foo%20Fighters|The%20Sky%20Is%20A%20Neighborhood|
00:04:03|SONG
```

Likewise, an ad trigger might look like this:

```
http://source:abc123@ice5.streamnet.com/admin/metadata?mount=/
BRAVOFM1im&mode=updinfo&song=|Shop%20LocalA%20%2D%20WestfieldMerchantsFriday|
00:00:30|COM
```

In the stream transmitted by the CDN, the local ad from the origin server might be replaced with another ad targeted to the geographic location of the listener. The CDN may use the duration of the ad (30 seconds in this example) as the queue for switching back to the original program, or it may wait for the next SONG update to switch back.

For RTMP and HLS streams, the metadata is reformatted in a particular manner and “injected” into the stream as in-band data. For example, for RTMP, a “setDataFrame” message is assembled, typically containing an array of three strings: title, artist and “url” (a container for everything else). With the Foo Fighters example from above as our input, the schematic form of the output, ignoring the header bits demarcating the various sections, would look more or less like this:

```
@setDataFrame
onMetaData
title:The Sky Is A Neighborhood
artist:Foo Fighters
url:http://www.blaze105.com?autoID=R585&autoCat=SONG&cat=MNJ&album=Concreteand
Gold&label=RCA&ISRC=USRW9170028
```

Note that the mapping of the tags in the original message to some of the query parameters in the output “url” string is neither obvious nor predictable, and in this case was in fact determined by the CDN. The underlying implication is that the manufacturer of the stream encoder cannot know ahead of time what format either the incoming or the outgoing metadata will look like.

A common solution for effecting the transformation of arbitrary input to arbitrary output is a programmable, embedded scripting language such as Lua. Current stream encoders often provide a menu of Lua filters which are able to transform common patterns of metadata input into similarly common output patterns. Usually all that is required to customize the transform filter for the station's own particular needs are a few tweaks to the script.

In closing, we'll show an example of one of the pitfalls of relying on metadata for accuracy, and why many stations are now employing metadata cleaners and aggregators in their flow path. This is a real message received from an automation system (which shall remain anonymous):

```
<audio>
<type>Song</type>
<status>Playing</status>
<artist>Beyoncï¿½</artist>
<title>Crazy in Love</title>
<number></number>
<length>00:03:56</length>
</audio>
```

Obviously, Beyoncé is the intended artist, but how would a computer know that? The odd character sequence at the end of her name is actually a single character: the Unicode Replacement Character / uFFFD, or one representation of it. (The lozenge-question-mark � is another representation of the same character.) How did it get there? Some automated transcription program back in the day encountered a character it wasn't designed to handle (in this case the é), and replaced it with a wild-card. The problem for any metadata transform filter is that the wild-card can refer to *anything*—there's actually no way to know except from the context what the original character was supposed to be.

And so there we are, at the interface of data and chaos. Even in the age of digital streaming, keeping entropy at bay is still the constant battle.