DJ Link Packet Analysis

James Elliott
Deep Symmetry, LLC

April 30, 2016

Abstract

The protocol used by Pioneer professional DJ equipment to communicate and coordinate performances can be monitored to provide useful information for synchronizing other software, such as light shows and sequencers. By creating a "virtual CDJ" that sends appropriate packets to the network, other devices can be induced to send packets containing even more useful information about their state. This article documents what has been learned so far about the protocol, and how to accomplish these tasks.

1 Mixer Startup

When the mixer starts up, after it obtains an IP address (or gives up on doing that and self-assigns an address), it sends out what look like a series of packets[1] simply announcing its existence to UDP port 50000 on the broadcast address of the local network.

These have a data length[2] of 37 bytes, appear roughly every 300 milliseconds, and have the content shown in Figure 1.

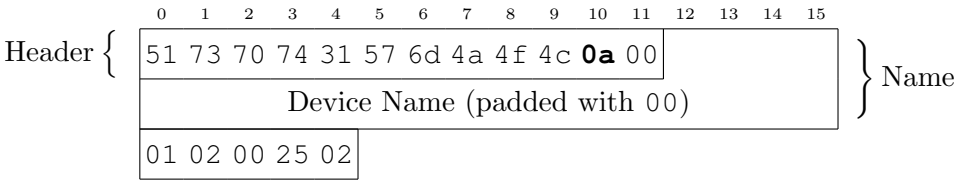

Figure 1: Initial announcement packets from Mixer

[1]The packet capture described in this analysis can be found at `https://github.com/brunchboy/dysentery/raw/master/doc/assets/powerup.pcapng`

[2]Values within packets are shown in hexadecimal, while packet lengths and byte offsets are discussed in decimal.

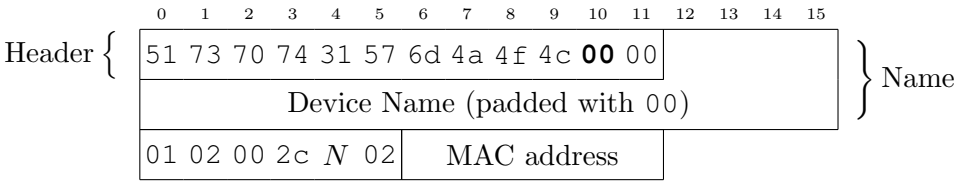

Figure 2: First-stage Mixer device number assignment packets

The tenth byte (inside what is labeled the header) is bolded because its value changes in the different types of packets which follow.

After about three of these packets are sent, another series of three begins. It is not clear what purpose these packets serve, because they are not yet asserting ownership of any device number; perhaps they are used when CDJs are powering up as part of the mechanism the mixer can use to tell them which device number to use based on which network port they are connected to?

In any case, these three packets have a data length of 44 bytes, are again sent to UDP port 50000 on the local network broadcast address, at roughly 300 millisecond intervals, and have the content shown in Figure 2.

The value N at byte 36 is 1, 2, or 3, depending on whether this is the first, second, or third time the packet is sent.

After these comes another series of three numbered packets. These appear to be claiming the device number for a particular device, as well as announcing the IP address at which it can be found. They have a data length of 50 bytes, and are again sent to UDP port 50000 on the local network broadcast address, at roughly 300 millisecond intervals, with the content shown in Figure 3.

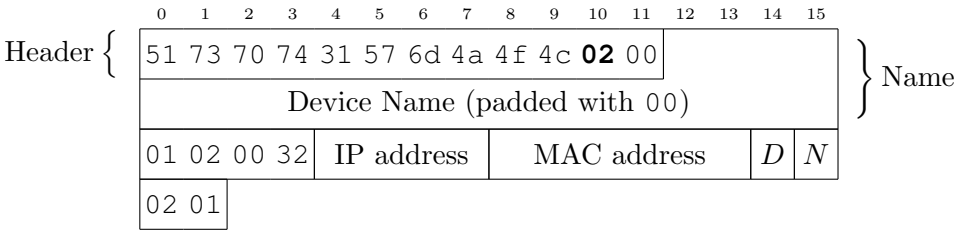

Figure 3: Second-stage Mixer device number assignment packets

I identify these as claiming/identifying the device number because the value D at byte 46 is the same as the device number that the mixer uses to identify itself (`0x21`) and the same is true for the corresponding

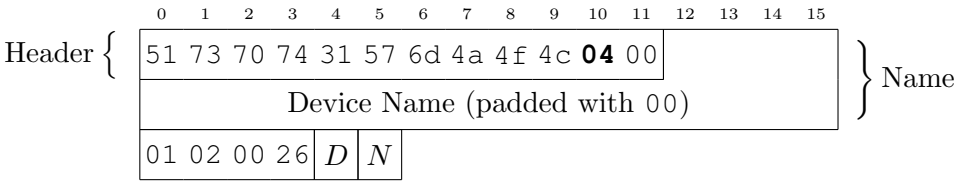

Figure 4: Final-stage Mixer device number assignment packets

packets seen from the CDJs (they use device numbers 2 and 3, as they are connected to those ports/channels on the mixer).

As with the previous series of three packets, the value N at byte 47 takes on the values 1, 2, and 3 in the three packets.

These are followed by another three packets, perhaps the last stage of claiming the device number, again at 300 millisecond intervals, to the same port 50000. These shorter packets have 38 bytes of data and the content shown in Figure 4.

As before the value D at byte 36 is the same as the device number that the mixer uses to identify itself (`0x21`) and N at byte 37 takes on the values 1, 2, and 3 in the three packets.

Once those are sent, the mixer seems to settle down and send what looks like a keep-alive packet to retain presence on the network and ownership of its device number, at a less frequent interval. These packets are 54 bytes long, again sent to port 50000 on the local network broadcast address, roughly every second and a half. They have the content shown in Figure 5.

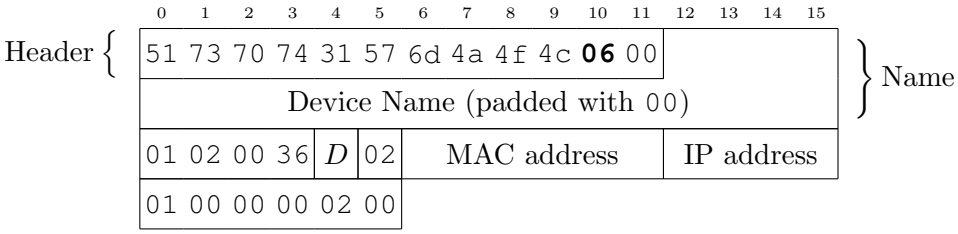

Figure 5: Mixer keep-alive packets

2 CDJ Startup

When a CDJ starts up the procedure and packets are nearly identical, with groups of three packets sent at 300 millisecond intervals to port 50000 of the local network broadcast address. The only difference

between Figure 6 and Figure 1 is the final byte, which is `0x01` for the CDJ, and was `0x02` for the mixer.

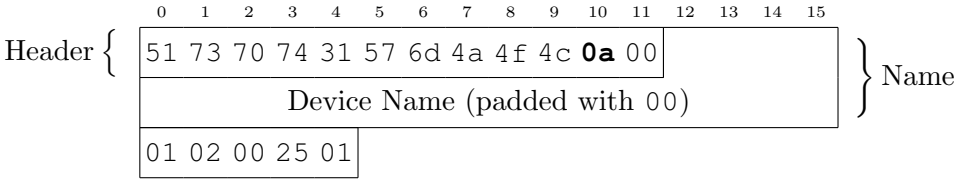

Figure 6: Initial announcement packets from CDJ

Similarly, the next series of three packets from the CDJ are nearly identical to those from the mixer. The only difference between Figure 7 and Figure 2 is byte 37 (immediately after the packet counter N), which again is `0x01` for the CDJ, and was `0x02` for the mixer.

However it appears that in this capture the CDJ skips the second stage of claiming a device number, probably because it is configured to be automatically assigned a device number based on the port of the mixer to which it is connected, and we cannot see a packet that the mixer sent it assigning it that device number. Instead, it jumps right to the end of the third and final stage, sending a single 38-byte packet with header byte 10 set to tt 04 (which identified the three packets of the third stage when the mixer was starting up), with content identical to Figure 4.

Even though the value of N is `01`, this is the only packet in this series that the CDJ sends. It would probably behave differently if configured to assign its own device number (behaving like we saw the mixer behave in claiming its device number).

The CDJ then moves to the keep-alive stage, sending out 54-byte packets with the content shown in Figure 8.

As seems to always be the case when comparing mixer and CDJ packets, the difference between this and Figure 5 is that byte 37 (following the device number D) has the value `01` rather than `02`, and the same is true of the second-to-last byte in each of the packets. (Byte

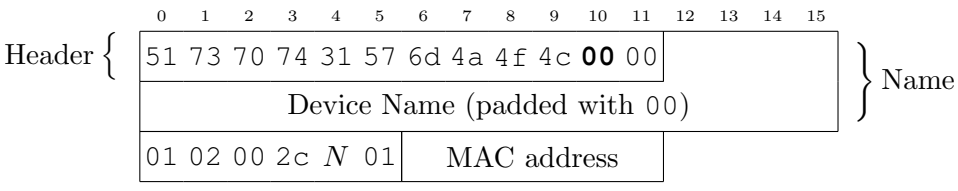

Figure 7: First-stage CDJ device number assignment packets

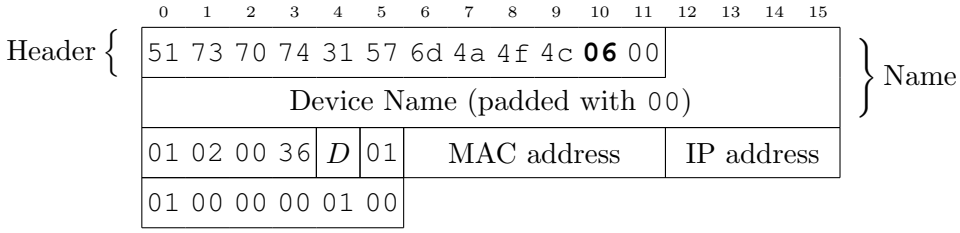

Figure 8: CDJ keep-alive packets

52 is `01` in Figure 8 and `02` in Figure 5.

3 Tracking BPM and Beats

For some time now, Afterglow[3] has been able to synchronize its light shows with music being played on Pioneer equipment by observing packets broadcast by the mixer to port 50001. Until recently, however, it was not possible to tell which player was the Master, so there was no way to determine the down beat (the start of each measure). This section will be expanded and more details provided as Afterglow is updated to take advantage of the discoveries described in the next section.

Until then, here is a summary of what is currently done. A socket is opened and bound to port 50001. Whenever a packet from the mixer is received on this socket, if the length is 96 bytes, it is known to contain beat and BPM information. The current BPM can be obtained as:

$$\frac{byte[90] \times 256 + byte[91]}{100}$$

These packets are sent on each beat, and the current beat number (1, 2, 3 or 4) is sent in *byte*[92]. However, the beat number is *not* synchronized with the master player, and so it is not useful for much. We expect to make use of the Virtual CDJ technique to determine the actual beat number soon.

4 Creating a Virtual CDJ

Although some useful information can be obtained simply by watching broadcast traffic on a network containing Pioneer gear, in order to

[3] `https://github.com/brunchboy/afterglow#afterglow`

get important details it is necessary to cause the gear to send you information directly. This can be done by simulating a "Virtual CDJ".[4]

To do this, bind a UDP server socket to port 50002 on the network interface on which you are receiving DJ-Link traffic, and start sending keep-alive packets to port 50000 on the broadcast address as if you were a CDJ. Follow the structure shown in Figure 8, but use the actual MAC and IP addresses of the network interface on which you are receiving DJ-Link traffic, so the devices can see how to reach you.

You can use a value like 5 for D (the device/player number) so as not to conflict with any actual players you have on the network, and any name you would like. As long as you are sending these packets roughly every 1.5 seconds, the other players and mixers will begin sending packets directly to the socket you have opened on port 50002.

We are just beginning to analyze all the information which can be gleaned from these packets, but here is what we know so far.[5]

4.1 Mixer Status Packets

Packets from the mixer will have a length of 56 bytes and the content shown in Figure 9.

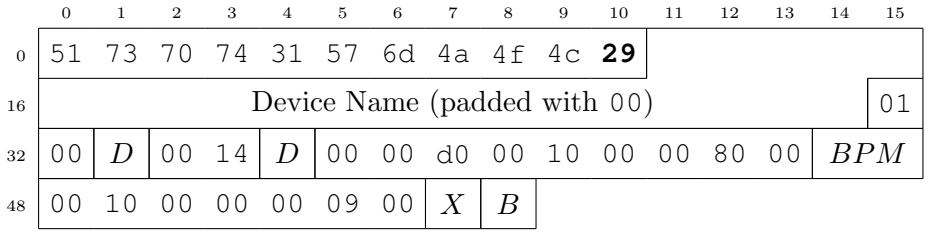

Figure 9: Mixer status packets

Packets coming from a DJM-2000 nexus connected as the only mixer on the network contain a value of 33 (`0x21`) for their Device Number D (bytes 33 and 36).

The current tempo in beats-per-minute identified by the mixer can be obtained as:

$$\frac{byte[46] \times 256 + byte[47]}{100}$$

[4]Thanks are due to Diogo Santos for discovering the trick of creating a virtual CDJ in order to receive detailed status information from other devices.

[5]Examples of packets discussed in this section can be found in the capture at `https://github.com/brunchboy/dysentery/raw/master/doc/assets/to-virtual.pcapng`

This value is labeled BPM in Figure 9. Unfortunately, this BPM seems to only be valid when a synced source is playing; when the mixer is doing its own beat detection from unsynced audio sources, even though it displays the detected BPM on the mixer itself, and uses that to drive its beat effects, it does not send that value in these packets.

Mixer status packets are sent on each beat, but seemingly not with as much precision as those sent to port 50001, and the current beat number (1, 2, 3 or 4) is sent in *byte*[55], labeled B. However, the beat number is *not* synchronized with the master player, and so it is not useful for much. The beat number should be determined, when needed, from packets that are sent by the master player.

The value at *byte*[54], labeled X, has an unknown meaning. It seems to start out with the value `0x00`, and then change when a player starts playing to the value `0xff`, but it may well do other things as well.

4.2 CDJ Status Packets

Packets from a CDJ will have a length of 212 bytes and the content shown in Figure 10.

The Device Number in D (bytes 33 and 36) is the Player Number as displayed on the CDJ itself. In the case of this capture, the CDJs were assigned Player Numbers 2 and 3.

The activity flag A at byte 39 seems to be 0 when the player is idle, and 1 when it is playing, searching, or loading a track.

The track number being played (its position within a playlist or other scrolling list of tracks, as displayed on the CDJ) can be found at bytes 50 and 51, labeled $Track$. (It may be a 4-byte value and also include bytes 48 and 49, but that would seem an unmanageable number of tracks to search through.)

There are a number of bytes, labeled t_2 through t_7, whose purpose is as yet undetermined. They are all zero when there is no track loaded, but take different values when a track is loaded. (Actually, there are many other bytes than these which behave like this, so they will probably be removed from the chart in a future version of this document.)

Byte 106, labeled U_a (for "USB activity"), alternates between the values 4 and 6 when there is USB activity—it may even alternate in time with the flashing USB indicator LED on the player. Byte 111 (U_l for "USB local") has the value 4 when there is no USB media loaded, 0 when USB is loaded, and 2 when the USB Stop button has been pressed and the USB media is being unmounted.

Byte 117, labeled U_g (for "USB global"), appears to have the value 1 whenever USB media is present in any player on the network, whether

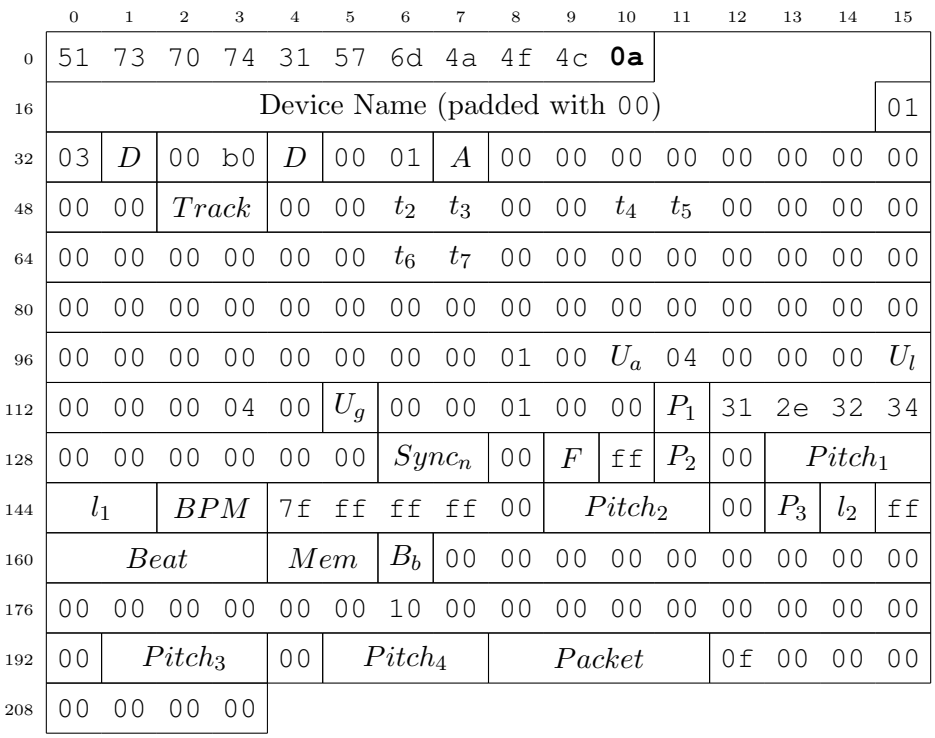

Figure 10: CDJ status packets

7	6	5	4	3	2	1	0
1	Play	Master	Sync	On-Air	1	0	0

Figure 11: CDJ state flag bits

or not the Link option is chosen in the other players, and 0 otherwise. I don't know if that is true of other linkable SD media as well; this needs more investigation.

Byte 123, labeled P_1, appears to describe the current play mode. The values that have been seen so far, and their apparent meanings, are:

0 No track is loaded.

3 The player is playing normally.

4 The player is playing a loop.

5 The player is paused anywhere other than the cue point.

6 The player is paused at the cue point.

9 The player is searching forwards or backwards.

17 The player reached the end of the track and stopped.

The value $Sync_n$ at bytes 134–135 seems to increment whenever the player syncs to a new tempo master (another player or the mixer). I am assuming it is just a 2-byte value, because I tried syncing 256 times and saw the counter expand from byte 135 to include byte 134. It may actually be a 4-byte value and also involve bytes 132 and 133, but I wasn't going to try changing sync 65,536 times or more to find out. Perhaps we could write software to test that someday by forcing tempo master changes.

Byte 137, labeled F, is a bit field containing some very useful state flags, detailed in Figure 11.

> We have not yet seen any other values for bits 0, 1, 2, or 7 in F, so we're unsure if they also carry meaning. If you ever find different values for them, please let us know by filing an Issue! `https://github.com/brunchboy/dysentery/issues`

Byte 139, labeled P_2 seems to be another play state indicator, having the value 126 when stopped and 122 when playing.

There are four different places where pitch information appears in these packets: $Pitch_1$ at bytes 141–143, $Pitch_2$ at bytes 153–155, $Pitch_3$ at bytes 193–195, and $Pitch_4$ at bytes 197–199.

Each of these values represents a three-byte pitch adjustment percentage, where `0x100000` represents no adjustment (0%), `0x000000`

represents slowing all the way to a complete stop (−100%, reachable only in Wide tempo mode), and `0x200000` represents playing at double speed (+100%).

Here is how the pitch adjustment percentage represented by $Pitch_1$ would be calculated:

$$100 \times \frac{(byte[141] \times 65536 + byte[142] \times 256 + byte[143]) - 1048576}{1048576}$$

We don't know why there are so many copies of the pitch information, or all circumstances under which they might differ from each other, but it seems that $Pitch_1$ and $Pitch_3$ report the current pitch adjustment actually in effect (as reflected on the BPM display), whether it is due to the local pitch fader, or a synced tempo master.

$Pitch_2$ and $Pitch_4$ are always tied to the position of the local pitch fader, unless Tempo Reset is active, effectively locking the pitch fader to 0% and $Pitch_2$ and $Pitch_4$ to `0x100000`, or the player is paused or the jog wheel is being held down, freezing playback and locking the local pitch to −100%, in which case they both have the value `0x000000`.

When playback stops, either due to the play button being pressed or the jog wheel held down, the value of $Pitch_4$ drops to `0x000000` instantly, while the value of $Pitch_2$ drops over time, reflecting the gradual slowdown of playback which is controlled by the player's brake speed setting. When playback starts, again either due to the play button being pressed or the jog wheel being released, both $Pitch_2$ and $Pitch_4$ gradually rise to the target pitch, at a speed controlled by the player's release speed setting.

If the player is *not* synced, but the current pitch is different than what the pitch fader would indicate (in other words, the player is in the mode where it tells you to move the pitch fader to the current BPM in order to change the pitch), moving the pitch fader changes the values of $Pitch_2$ and $Pitch_4$ until they match $Pitch_1$ and $Pitch_3$ and begin to affect the actual effective pitch. From that point on, moving the pitch fader sets the value of all of $Pitch_1$, $Pitch_2$, $Pitch_3$, and $Pitch_4$. This all seems more complicated than it really needs to be...

The current BPM of the track (the BPM at the point that is currently being played, or at the location where the player is currently paused) can be found at bytes 146–147 (labeled BPM). It is a two-byte integer representing one hundred times the current track BPM. So, the current track BPM value to two decimal places can be calculated as:

$$\frac{byte[146] \times 256 + byte[147]}{100}$$

In order to obtain the actual playing BPM (the value shown in the

BPM display), this value must be multiplied by the current effective pitch, calculated from $Pitch_1$ as described above.

Because Rekordbox and the CDJs support tracks with variable BPM, this value can and does change over the course of playing such tracks. When no track is loaded, BPM has the value `0xffff`.

The meaning of values l_1 (bytes 144–145) and l_2 (byte 158) are not currently known. They may simply reflect whether a track is loaded or not: l_1 seems to have the value `0x7fff` when no track is loaded, and the value `0x8000` when a track is loaded, while l_2 has the value 0 when no track is loaded, 1 when a track is loaded, and 2 when a track is loaded an the player is the sync master... but there is likely more going on here than we have yet figured out.

Byte 157 (labeled P_3) seems to communicate additional information about the current play mode, with the following meanings that we have found so far:

0 No track is loaded.

1 The player is paused or playing in Reverse mode.

9 The player is playing in Forward mode with jog mode set to Vinyl.

13 The player is playing in Forward mode with jog mode set to CDJ.

The 4-byte beat counter (which counts each beat from 1 through the end of the track) is found in bytes 160–163, labeled $Beat$. When the player is paused at the start of the track, this seems to hold the value 0, even though it is beat 1, and when no track it loaded, this holds the value `0xffffffff`.

The counter B_b at byte 166 counts out the beat within each bar, cycling $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ repeatedly, and can be used to identify the down beat (as is used in the Master Player display on the CDJs as a mixing aid). Again, when no track is loaded, this holds the value 0.

A countdown timer to the next saved memory point is available in bytes 164–165 (labeled Mem). If there is no saved memory point after the current play location in the track, or if it is further than 64 bars ahead, these bytes contain the value `0x01ff` and the CDJ displays "-.- Bars". As soon as there are just 64 bars (256 beats) to go before the next memory point, this value becomes `0x0100`. This is the point at which the CDJ starts to display a countdown, which it displays as "63.4 Bars". As each beat goes by, this value decreases by 1, until the memory point is about to be reached, at which point the value is `0x0001` and the CDJ displays "0.1 Bars". On the beat on which the memory point was saved the value is `0x0000` and the CDJ displays "0.0 Bars". On the next beat, the value becomes determined by the next memory point (if any) in the track.

Bytes 200-203 seem to contain a 4-byte packet counter $Packet$, which is incremented for each packet sent by the player. (I am just

guessing it is four bytes long, I have not yet watched long enough for the count to need more than the last three bytes).

> This analysis is incomplete; there are many bytes whose purpose is not yet known, not all of which are even mentioned here yet.