From Jack to UDP packets to sound, and back

Fernando Lopez-Lezcano
CCRMA, Stanford University
Stanford, CA, USA
nando@ccrma.stanford.edu

Abstract

The Mamba Digital Snakes are commercial products created by Network Sound, that are used in pairs to replace costly analog cable snakes by a single Ethernet cable. A pair of boxes can send and receive up to 64 channels at 48KHz sampling rate packed with 24 bit samples. This paper describes the evolution of jack-mamba, a small jack client that can send and receive UDP packets to/from the box through a network interface and transforms it into a high channel count soundcard.

Keywords

Jack, UDP, Networking, Soundcard, USB

1 Introduction

The jack-mamba program is the offshoot of a project that is still in progress, with the goal of creating a small trial Wave Field Synthesis system at CCRMA (a 32 channel system for experimentation and possible future expansion).

The conventional solution in most current WFS systems is to use high channel count PCI or PCIe sound cards, usually from the RME MADI family. The MADI output of the soundcard (64 channels) is then split into 8 ADAT 8 channel ports with a MADI to ADAT bridge, and each ADAT lightpipe is fed into 8 channel ADAT D/A converters that drive the speakers. This solution is proven and reliable, easy to synchronize across multiple hosts using a word clock signal, but expensive. For a base 32 channel system the price hovered around $180 per channel (64 channel systems would have a lower cost per channel, of course).

We though it was worthwhile to explore other non-conventional solutions that might lower the cost of the system.

2 A network sound card

Another possibility would be to eliminate the sound card entirely and explore delivery of audio through network packets to custom built boxes that would include a network port and D/A converters. There are already many systems that deliver high quality audio over Ethernet, but most of them use proprietary protocols (a Wikipedia article includes pointers to most[1]). Of course there is also the very complete and complex IEEE 1722 protocol[2]. This topic (an Ethernet “soundcard”) has also surfaced on the LAD, LAU and Jack mailing lists several times on recent years (see, for example, a long thread in 2009[3] started by Will Godfrey - Folderol, another one in mid-2011[4] by Dan Swain). As far as I'm aware this has not yet crystallized into working hardware.

3 The Mamba digital snake

A local Silicon Valley company (Network Sound) has created a family of products that replace high cost analog cable snakes with A/D and D/A boxes connected through a single CAT5 Ethernet cable (the Mamba Digital Snake [5][7]). A pair of boxes can interconnect two locations with up to 64 channels of 48KHz low latency 24 bit audio through a dedicated Ethernet link. Their solution is dedicated and point to point, and thus does not implement resource discovery, overall sampling rate synchronization and hence the communication protocol is very simple.

We thought it would be interesting to explore the

possibility of using one half of a 32 channel digital snake as a "soundcard". A rough cost estimate for that solution seemed to be about 1/3 of the traditional soundcard solution (this ignores for now the problem of synchronizing multiple units driven by different computers so that all the analog outputs are sample synchronous, a requirement of a WFS system).

The box is controlled by an FPGA, has 32 A/D and D/A converters and analog I/O ports, and an Ethernet connector. It is meant to be driven by an identical system, and when running as a slave it recovers its audio clock from the timing of the received UDP packets. The formatting of the UDP packets is very simple and there are no protocols to implement. We just need to send UDP packets with the proper timing and contents to get 32 analog outputs. Network Sound was kind enough to provide us with the packet format specification they receive and transmit.

3.1 The software

We started looking around for open source free software we could use as a starting point. The first candidate was Jacktrip, a system designed at CCRMA to do multichannel high quality long distance audio collaboration[6]. A we started reading the source we found that it was a complex system, difficult to tweak for our purposes, and mostly written using Qt classes, with which the author was not familiar (the original goal was to write a simple command line jack client using plain C).

In the meanwhile LAC 2011 took place and Jörn Netingsmeier pointed out the existence of *jack.tools* by Rohan Drape[7], a set of GPL jack command line tools. One of them is jack.udp and it seemed to be an almost perfect fit for the task at hand:

"jack.udp is a UDP audio transport mechanism for JACK. The send mode reads signals from a set of JACK input ports and sends UDP packets to the indicated port at the indicated host at a rate determined by the local JACK daemon. The "recv" mode reads incoming packets at the indicated port and writes the incoming data to a set of JACK output ports at a rate that is determined by the local JACK daemon. "

After looking at the source (simple C, very readable) it seemed that it would be relatively simple to modify the send and receive functions and adapt them to the packet and sample formats needed by the Mamba box, replacing the sending side of the digital snake with a general purpose Linux computer.

3.2 Initial tests

The Mamba box uses a simple packet format, in the case of the 32 channel version each UDP packet holds 8 32 channel 24 bit audio frames. Our jack client program would have to send packets at the rate of one packet every 166.67 uSecs.

The jack.udp program has (as a sender) a separate thread for sending UDP packets, which is fed samples from the Jack callback function by a lock-free ring buffer. The Jack callback pushes its samples into the lock-free ring buffer and wakes the UDP send thread through a write to a pipe. The UPD thread, which was waiting on the pipe, reads the samples in packet-sized chunks from the ring buffer and sends them out to the proper IP address and port, where normally a second instance of jack.udp in receive mode does the opposite.

It was easy to replace the packet assembly and disassembly functions with versions that packed 24 bit samples into 8 channel frames with the proper header information. We were able to test sending and receiving audio between two computers as with the original program, except that in this case the formatting of the packets corresponded to the Mamba specifications.

In May 2011 we visited Network Solutions for a first test. We used the Ethernet port of a laptop configured as a statically addressed interface to send the packets.

The first try were less than successful as the UDP packets seemed to go nowhere. Tcpdump made it was easy to see why, as nobody was answering ARP requests. The Network Sound engineers burned a different version of the FPGA code that included ARP generation, and we were able to send packets to the box. For these initial tests sending on the box was

disabled (it only received UDP packets), and internal buffering was maximized to make it as robust as possible to latency problems in the jack client program.

After that, we managed to send a sine wave to the Mamba box and got a clean output, albeit with quite frequent dropouts. But we knew that the packets were getting to the box, and that the formatting of the packets and the samples was correct.

3.3 Clock jitter issues

There was a mismatch between the modified jack.udp and the internals of the Mamba box. All UDP packets packed with samples from a Jack callback were sent together at the beginning of the callback as in the original jack.udp code. As the Mamba box recovers its audio clock from the timing of the received UDP packets the uneven timing creates jitter in the recovered audio clock.

To see if the problem could be minimized we added delays between packets to space them more evenly within a jack cycle. The timing on the receiving end was definitely better after that change.

The fact that the modified jack.udp is a normal Jack client was also going to have extra second order jitter effects. The absolute starting point of the Jack callback in time could move from period to period as Jack clients enter or leave the graph, or when connections are changed so that the order of execution of existing clients change. Those changes, while minimal, would also have an effect in the transmit timing of the UDP packets. In addition to that, the wakeup latency of the UDP send thread itself would add an additional (small) delay to the first packet being sent out in a given Jack period.

As a first approximation the code was good enough to verify that the network stack and high resolution timers in Linux could deliver packets reliably to the box every 166.6 uSecs.

3.4 More testing

Another test on June 1st was much more successful. The glitches that had plagued the previous test were the result of a coding error. The UDP send thread was actually not running in the SCHED_FIFO scheduling ring (booo!). With that fix in place the audio output was much more stable.

We received a loaner box from Network Sound with the custom FPGA code and we were able to keep testing at CCRMA, this time using a desktop machine with a second network interface dedicated to the audio link. This combination did much better than the laptop and there were almost no dropouts even when loading the computer and transferring big files through the regular eth0 interface.

3.5 Receiving audio

Receiving packets from the Mamba box did not work and it took a long time to debug the problem. We could see packets being received by the Linux machine with tcpdump, but nothing was received by the program. Eventually we found out by using netstat that the packets being sent by the Mamba box were not legal (we could see the network stack error counters being incremented). A hex disassembly of a dumped packet confirmed that the checksum was wrong. A bug in the FPGA code, which was promptly fixed by the Network Sound engineers.

Now we were at the end of June. Another flashed FPGA and a test at Network Sound and we managed to receive audio from the Mamba box with one instance of the modified jack.udp program. A jaaa process showed a perfect sine wave being received at our end (one of the channels of the Mamba box was being fed by a high quality audio analyser test signal).

3.6 Looking at the recovered audio clock jitter

We fired another instance of the modified jack.udp to *send* packets to the mamba box. And almost immediately jaaa showed visible side bands around the single spike that represented the sine wave. Puzzling, until one of the engineers (I forget her name) pointed out the obvious cause, this was the confirmation that jitter in the received packets in the Mamba box caused jitter in the recovered audio clock. See below for measurements.

The jitter in the recovered audio clock is not a problem in the real digital snake as the timing of the packets sent by the master box is hardware driven.

3.7 Thread priorities

It is necessary to optimize the priority of all SCHED_FIFO threads involved in this network based "audio stack".

As the timing of the outgoing packets is critical to minimize audio clock recovery jitter, the UDP send thread runs with a priority that is higher (by one) than the priority of the main SCHED_FIFO thread of jackd. Receiving packets is very important but the timing is not so critical, so that the UDP receiving thread runs with higher priority than the jackd client thread of jack-mamba (it has less priority than sending).

There are two additional sets of threads that can use priority tweaking, although the result is not as drastic as the main UDP send and receive threads inside the modified jack-udp program.

As in the case of Jack, the IRQ thread for the hardware Ethernet interface dedicated to audio can be raised in priority. It should be higher than the priority of jackd but probably lower than the priority of the IRQ thread that takes care of the soundcard hardware (more experimentation is needed to fine-tune this – this will probably depend on the reliability of the hardware drivers for sound and network packets).

The kernel also has network transmit and receive software interrupts that run with SCHED_FIFO priority. They can also be optimized to run at higher priority than other software interrupts.

4 Getting rid of the pipe

With the previous tests we had confirmed that it was possible to send and receive packets from a stock Linux machine running an RT patched kernel with very occasional packet delays. Almost all of the delays were short enough that the buffering inside the Mamba box could cover for them.

To get better performance and to minimize the jitter in the recovered audio clock we had to change the internal architecture of the original jack.udp program (now renamed jack-mamba). We wanted to do away with the pipe based wakeup calls that the jack callback function used to drive the UDP send thread. After all, the UDP send thread needed to send packets at a regular rate that should be locked to the sampling rate of the soundcard being used by Jack.

If the UDP send thread were to send all packets at exactly the right *time,* there should be no need for wakeup calls, the Jack callback could feed the ring buffer and the UDP send thread could empty it at exactly the same rate, and both halves would not need any additional synchronization. Enough extra buffering in the ring buffer in the form of a fixed time offset between arrival of samples to the Jack callback and delivery of those samples in UDP packets would provide for a cushion against latency variations in the Jack callbacks and uneven scheduling latencies in all processes involved, including the UDP send thread.

Based on Jack time (*jack_last_frame_time* tells us when the current Jack cycle started) we can know exactly when the next packet should be sent. The UDP thread can then use absolute time *clock_nanosleep* calls instead of relative *nanosleep* waits for delivering the packets at the right time. And we can pass the absolute time stamp for each packet in the same ring buffer that sends samples from the jack callback to the UDP send thread.

The UDP thread is now an infinite loop that checks for availability of data in the ring buffer in each iteration. If there is enough data available to fill one UDP packet, it reads it together with the absolute time when it should be sent, and waits with a call to a *clock_nanosleep* delay (with a TIMER_ABSTIME argument). The packet is then sent at the right time and the loop is iterated again.

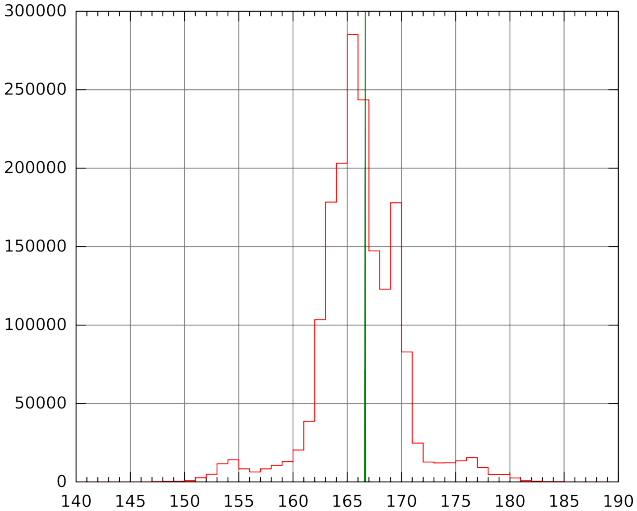
If there is not enough data available in the ring buffer the thread waits using a relative *clock_nanosleep* call that delays the thread for the duration of the pre-calculated inter-packet delay.

This can happen when there is a Jack xrun or the Jack callback thread gets delayed for a long time. After the wait the loop is iterated again.

In this way there is no additional scheduling delay for the first packet at the beginning of a jack cycle due to the pipe wakeup, all packets are scheduled at the right time (in the steady state condition), the scheduling time only depends on the start of each Jack cycle and we can add a time offset for additional buffering (and latency) in the ring buffer.

The jitter in packet delivery will of course be impacted by any scheduling delays in the UDP send thread, but with a properly configured RT patched kernel and correct IRQ priorities it works fine.

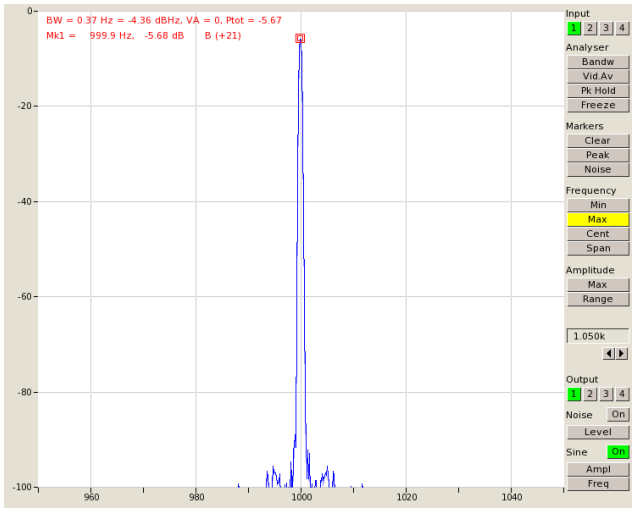
The next graph shows inter-packet scheduling delay measurements using tcpdump (with inter-packet time stamps, the "-ttt" option) on the sending machine over a period of 5 minutes, The X axis is measured packet spacing (the theoretical delay in this case should be 166.667 uSecs running Jack at 48KHz with 128 x 2) and the Y axis is number of packets that fall in a particular microsecond-wide bin:



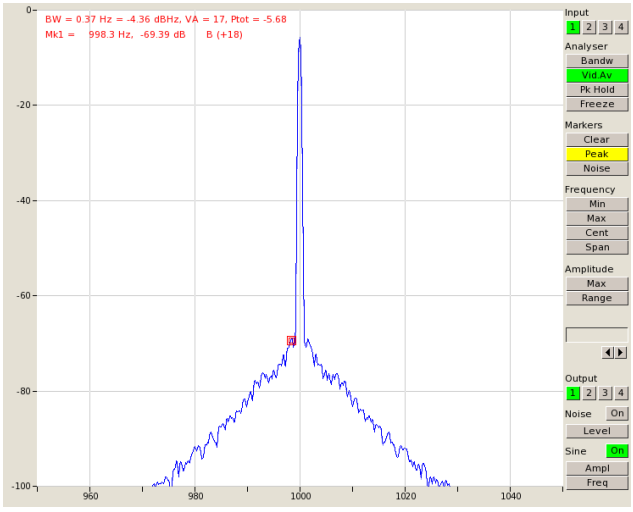
Time delay between sent packets

Even with a properly tuned system there are still occasional cases of long scheduling delays. A more complete investigation needs to be done, adding code to trigger Jack stack traces in those cases to try to see why the scheduling latencies are happening.

The figure below show a 1KHz sine wave from a Minirator MR-PRO generator being fed into channel #1 of the Mamba box and received by a single instance of jack-mamba:



The next figure shows the effect of recovered audio clock jitter due to received packet timing differences when we add another instance of jack-mamba sending silence to the Mamba box (no other changes). This is using jaaa peak hold and averaging functions to see the worst long term distortion products:



The bandwidth of the measurement is very small

(0.366Hz) and the distortion products are at or below 70dB below full scale. These measurements were taken on a workstation with quad core Q9450 Intel processors @ 2.66GHz, running Fedora 14 with a 2.6.33.19-rt31 kernel while logged into an NFS served account through the normal eth0 interface.

4.1 Managing latency

The fact that there are elastic ring buffers between the reception and transmission of the UDP packets, and the Jack callbacks means we have potentially variable input and output audio latencies.

The number of samples stored by the Jack callback in the outgoing ring buffer determines the additional output latency, and can be controlled. The number of received UDP packets stored in the incoming ring buffer determines the additional input latency. This is set at program start time by discarding packets until the latency corresponds to the desired amount. Currently there is no way to detect dropped incoming packets as there is no packet sequence information being sent by the Mamba box in the header of the packets. More work needs to be done in this regard.

4.2 Jack xruns

If an xrun occurs then the synchronization between the jack callback thread and the UDP send thread is broken. When that happens the reference timing for the two threads is recalculated so that they are synchronous again.

4.3 Error reporting

To report errors to the main program we added another lock-free ring buffer that communicates error messages back to the main program, it samples the error queue and reports statistics to the console.

To be able to properly report errors with readable timestamps we found it necessary to record the offset between the return of the *time* system call and that of *clock_gettime* at the startup of the program to compensate the printed values for an offset between the two clocks.

5 Network packet priorities

Linux provides several ways to prioritize network traffic and we tried to add those options to the program to further optimize performance and timing of the packet delivery.

The first one is setting a higher socket priority for the socket being used to transmit the UDP packets (SO_PRIORITY) using *setsockopt*. We chose the highest priority that we could use without having to run the program with special privileges (6).

The second one is to change the Type of Service (IP_TOS) of the packets. For this option it is necessary for the executable to have special permissions so it is not so easy to use. If that option is used and the function returns an error, the program suggests setting the executable to have the appropriate capability (if file based capabilities are available, of course).

6 Trial by fire

The Transitions Concert, our end of summer concert, was going to have a first night (on September 28th) curated by the author of this paper with a new outdoors sound system that used 16 main speakers in a 3D setup and 4 sub-woofers. Our goal was to have a reliable system that could drive all 20 speakers from the 32 channel Mamba D/A box.

We used a Quad Core Intel machine running Fedora 14 with a 2.6.33.14-rt31 patched kernel. A second PCI express Intel dual port Ethernet card was installed and provided a dedicated point to point connection to the Mamba box. The computer was left in the server room and just three Ethernet cables connected the system to the outdoor concert venue. Two were used to remote the monitor and USB peripherals, and the third one connected the workstation to the Mamba box. Jack-mamba worked perfectly. No xruns or glitches in the 1 1/2 hours the concert lasted.

7 A 32 channel USB soundcard

When using laptops it is normally desired to be able to use the wired Ethernet interface to have a fast network connection to the Internet. While it may be possible to share the interface with the audio packets using proper routing, it would be much better to have

a second Ethernet interface dedicated to audio.

We tried to use a USB 2.0 to Ethernet dongle and the results were very promising. We configured the Ethernet interface using the Network Managwer GUI in a Fedora 14 installation to be tied to the hardware address of the dongle, and use the proper fixed IP address that the Mamba box sends to. We also set up routing so that the default route for Internet access is not changed and the new interface does not interfere with existing network connections.

Once the interface is properly set up, it is possible to plug in the dongle, wait until it is discovered by Network Manager, and start jack-mamba right away. It is almost as easy to use as a real USB soundcard. The dongle can even be disconnected and reconnected and audio starts streaming again with no problems when the interface is again detected.

We could also create udev rules so that the priority of the USB IRQ thread in RT patched kernels is changed from the default when the dongle is connected. No tests have been made on what happens when the USB dongle shares the internal chipset USB hub with other USB ports and peripherals. Maybe this solution will not be workable in all cases.

8 Processor load

On an Intel Quad Core workstation running at 2.66GHz the jack-mamba process uses around 15.9% of a CPU (split into the three main threads that use 7.3%, 5.3% and 3.3%), the IRQ thread for the Ethernet card uses 3.6% and the network software IRQ (sirq-net-rx/1) uses 9.3% (for a total of approximately 28.8%). This is one disadvantage of this “soundcard” as it uses more processing power than a standard PCI[e] interface.

Using the USB Ethernet interface instead, the total load of jack-mamba is 17.9%, the ehci_hcd IRQ thread for the USB interface uses 9%, the uhci_hcd 2%, and assorted sirq-net* and tasklets use 8.8% for a total of around 37%.

While a USB based Ethernet dongle uses 30% more CPU than a PCI express Ethernet interface it is not unreasonable for such an interesting solution. Of course the actual numbers in both cases will change slightly with different hardware drivers.

9 Creating a networked audio hub

Another feasibility test we performed was to run in the same machine jack-mamba and instances of jack_netsource (the netone version of netjack) tied to another dedicated Ethernet port with a DHCP server running on it. With this setup a computer can use the workstation as a networked audio hub and connect to the Mamba box through a network interface. This is the scheme the author uses to implement network audio connectivity in his OpenMixer project (albeit with regular PCI based RME soundcards delivering audio to the speakers).

10 Future development

Currently the determination of the spacing between outgoing UDP packets is based just on the absolute times of the last two jack cycles, a more precise determination should be arrived at, either based on a moving average of jack period lengths, or a proper phase locked loop[8].

A very desirable next step would be to convert the jack client program into a proper jack backend. We would need a stable clock source to drive the audio packet delivery.

A simple option would be to switch the Mamba box into master mode, and use the received UDP packets to trigger the sending of the transmitted UDP packets. In this way the sampling rate would be defined by the Mamba box itself and there would be no jitter noise added due to audio clock recovery.

A Mamba Jack backend would transform the Mamba box into a real "sound card". And a USB to Ethernet dongle would make this "sound card" independent of the availability of a free Ethernet port in the machine (USB ports are plentiful).

The jack-mamba program currently has support for 16, 32, 48 and 64 channel Mamba boxes but we have only tested it with the 32 channel version. It remains

to be seen if a 64 channel box can be made to run reliably and if the CPU load of the network stack processes is low enough to make it practical.

Another task is devising a synchronization scheme so that multiple boxes can have sample synchronous output (maybe impossible). There are protocols that can synchronize the clock of several computers very accurately, perhaps with enough resolution to be able to send UDP packets from different computers to different Mamba boxes at the same absolute time. But the ring buffer induced latencies and the possibility of dropped packets could make this unreliable. Word clock I/O on the Mamba boxes could alleviate these problems.

We also need to see if Network Sound can add sequence numbers to the Mamba UDP packet headers to enable us to detect missing or out of sequence packets. It would also be neat for the box to be able to copy a word from the incoming header to the outgoing header as this would make it possible to try to detect outgoing packet loss.

The newest products from Network Sound can be programmed through special UDP packets (including the send and receive IP addresses and ports, master mode, number of channels, etc). We need to write a small program that can access that functionality.

11 Conclusion

A Jack client has been presented which sends and receives UDP packets from one half of a Mamba digital snake box and slaves its sampling rate to a n existing soundcard. It has been tested with a 32 channel box with no problems in real life situations (concert diffusion), and works even through a cheap USB to Ethernet adapter. It runs in a regular Linux workstation with no special tweaks other than an RT patched kernel and proper IRQ thread priority optimization.

While solving the original problem (a low cost network based delivery of audio for WFS systems) is not yet done, this simple program and its planned migration to a full Jack backend could be useful nevertheless for situations where a high channel count soundcard is desired with low cost.

Jack-mamba will be made available under the GPL in time for LAC2012.

12 Acknowledgements

It would have taken a much longer time to write this program from scratch if it were not for Rohan Drape, his jack.tools package and jack.udp. Of course without Jack itself and all the wonderful audio programs that can use it this would not have been possible either. Many thanks to all the members of the Linux Audio community!

It would also have been impossible to do this without the enthusiastic support of CCRMA and Chris Chafe (CCRMA's Director), and without the help of all the people at Network Sound.

References

[1] Wikipedia: *Audio over Ethernet*
http://en.wikipedia.org/wiki/Audio_over_Ethernet

[2] Wikipedia: *Audio Video Bridging*
http://en.wikipedia.org/wiki/Audio_Video_Bridging

[3] LAU/LAD: “FOSS Ethernet soundcard” thread (Will Godfrey - Folderol):
http://linuxaudio.org/mailarchive/lau/2009/11/23/162370

[4] Jack-Audio-Devel: “Ethernet-based audio interface using (net)jack idea” thread (Dan Swain):
http://permalink.gmane.org/gmane.comp.audio.jackit/24568

[5] Network Sound: Mamba Digital Snake:
http://www.networksound.com/Digsnake.html

[6] Jacktrip and SoundWire:
https://ccrma.stanford.edu/groups/soundwire/software/jacktrip/

[6] Rohan Drape, jack.tools:
http://slavepianos.org/rd/

[7] Network Sound: Mamba Audio Streamer:
http://www.networksound.com/Mamba_AS.html

[7] Using a DLL to filter time, Fons Adriansen,
http://kokkinizita.linuxaudio.org/papers/usingdll.pdf