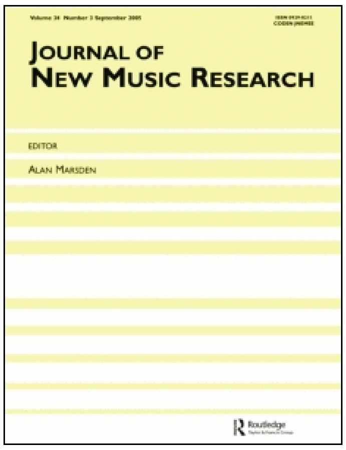

Journal of New Music Research
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713817838

JackTrip: Under the Hood of an Engine for Network Audio

Juan-Pablo Cáceres[a]; Chris Chafe[a]
[a] Stanford University, USA

Online publication date: 09 November 2010

To cite this Article Cáceres, Juan-Pablo and Chafe, Chris(2010) 'JackTrip: Under the Hood of an Engine for Network Audio', Journal of New Music Research, 39: 3, 183 — 187

To link to this Article: DOI: 10.1080/09298215.2010.481361

URL: http://dx.doi.org/10.1080/09298215.2010.481361

Journal of New Music Research
2010, Vol. 39, No. 3, pp. 183–187

JackTrip: Under the Hood of an Engine for Network Audio[†]

Juan-Pablo Cáceres and Chris Chafe

Stanford University, USA

Abstract

The design of a platform for bi-directional musical performance using modern wide area networks (WANs) poses several challenges that are different from related applications, e.g. synchronous local area network (LAN) studio systems or uni-directional WAN streaming. The need to minimize as much as possible audio latency and also maximize audio quality requires specific strategies which are informed, in part, by musical decisions. We present some of the key design elements of the JackTrip application which has evolved through several years of deployment in musical work over wide area networks.

1. Introduction

The SoundWIRE group at the Center for Computer Research in Music and Acoustics (CCRMA) (SoundWIRE Group, 2010) focuses on experiments in bi-directional and n-directional musical performance. Concerts and rehearsals between Stanford and places like New York, Belfast, Banff, or Beijing are now routine (Chafe, 2009).

JackTrip is the application that powers up these on-line collaborations. Presently, it's a Linux and Mac OS X-based system that supports multi-machine network performance over best-effort Internet. The technology being used builds on early work by research groups at McGill University (Xu & Cooperstock, 2000) and Stanford University (Chafe, Wilson, Leistikow, Chisholm, & Scavone, 2000). The basic approach is to send uncompressed audio (avoiding the latency introduced by compression encode/decode algorithms) through high-speed links like *Internet2*. It supports any number of channels (as many as the computers or network paths can handle). Since best-effort network protocols are used, adequate network provisioning is a must.

The subject of this article is JackTrip's design relating to several issues that come up in implementing such a system. It is hoped that these solutions can serve as a point-of-departure for further applications in this same area.

The design achieves (i) the highest audio quality possible, by using uncompressed linear sampling and redundancy to recover from packet loss; (ii) throughput maximization, which gets audio packets onto and off of the network as soon as the sound card can deliver them; (iii) working with any number of channels (depending on available computer processing power and bandwidth); (iv) flexibility in routing and mixing audio channels from and to the different hosts.

Other networking audio packages use different strategies for audio delivery and synchronization. These include the use of compressed audio, artificially increased delays that match one or more musical measures (i.e. musicians play asynchronously with the output that was generated one or more measures before) and one-way recording techniques (with one location/performer at a time) where latency is not an issue (Renaud, Carôt, & Rebelo, 2007). NetJack uses a master/slave approach to synchronize audio clocks. This approach is most suitable for local area networks (LAN) where jitter is smaller (Carôt, Holm, & Werner, 2009). Other systems (nStream, SoundJack and jack-tools) also deliver uncompressed audio (Bouillot & Cooperstock, 2009). JackTrip differs from these applications in that its architecture is optimized to keep decent audio quality at the lowest possible latency. And it is tuneable. Depending on the situation, the user can choose decreased delay at the expense of audio quality for live performance or opt for increased delay to obtain glitch-free audio. Its multi-threaded architecture and buffering mechanisms

[†]Winner of the *Journal of New Music Research* Distinguished Paper Award at the International Computer Music Conference, 2009.

Correspondence: Juan-Pablo Cáceres and Chris Chafe, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, The Knoll, 660 Lomita Dr., Stanford, CA 94305-8180, USA. E-mail: jcaceres@ccrma.stanford.edu, cc@ccrma.stanford.edu

DOI: 10.1080/09298215.2010.481361 © 2010 Taylor & Francis

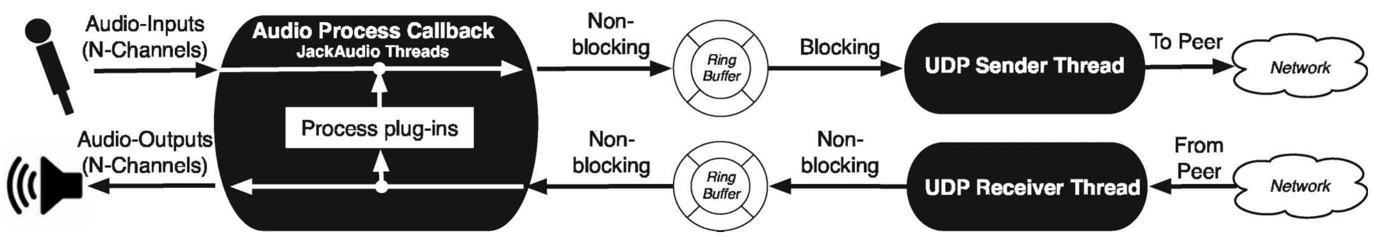

Fig. 1. JackTrip architecture overview.

(explained in detail in Section 2) have been designed, adapted and improved towards this goal.

1.1 Peer-to-peer network audio latency

Wide area network (WAN) connections inevitably introduce transmission delays between two or more hosts. For non-interactive and 'soft' real-time applications, this delay is less of a problem than for high-quality collaborative music performance. The latter places extremely stringent bounds on latency and jitter. The longer the audio latency between musicians, the harder it is for them to play synchronously (Chafe, Cáceres, & Gurevich, 2010). Time delays as short as 25 ms are already problematic for professional ensembles like string quartets.[1]

It is the total delay between sound capture and sound projection which counts. This splits out into (i) acoustic (air path) delays, e.g. the distance between an instrument and the capture microphone and between the speakers and ears; (ii) analogue-to-digital and digital-to-analogue conversion (ADC/DAC) delay, i.e. the time it takes for an analogue source to be transformed into digital and back; (iii) settings chosen for audio quality and packetization, including audio sampling rate and bit depth resolution, buffer and packet sizes, and others; (iv) network transmission delays, including physical (geographical) distance, transmission delays induced by switches, routers, firewall and network congestion among others.

The default transport protocol in JackTrip is User Datagram Protocol (UDP), a low-overhead, fast but unreliable mechanism for transmitting network packets. There is no guarantee that a UDP packet will ever reach its destination. Transmission Control Protocol (TCP) provides an alternative with an acknowledgment scheme which guarantees correct delivery, re-transmitting lost packets and fixing out-of-order ones (see Comer (2005) for a good description of the different protocols). The retransmission feature in TCP comes at a cost in terms of audio delay. Its 'elastic' behaviour produces an ever-changing delay that is extremely disturbing for musicians in real-time performance. Using UDP, latency can be constrained within predefined bounds, but it requires the implementation of its own strategies to deal with late and lost packets. The implementation of these techniques (see Sections 2.3 and 2.4) gives the developer better control of the trade-offs to be made specifically for live musical performance.

The application's own header data accompanies each audio packet to describe local properties like audio buffer size, sampling rate, bit depth, number of channels, a sequence number and a time stamp.

Currently, JackTrip uses Jack (Davis, 2009) as its default host audio server. Jack has several advantages: it runs on Linux and Mac OS X, it has the ability to make audio connections between many different audio clients on the same host, and its current implementation takes advantage of multi-processor machines (Letz , Orlarey, & Fober, 2005). An option to use native sound servers through RtAudio (Scavone, 2002) is also available.

2. JackTrip's multi-threaded architecture

JackTrip's multi-threaded design is implemented in C++ using the Qt library (Nokia Corporation, 2008–2009) and also can take advantage of multi-core machines. Figure 1 shows the multi-threaded architecture of the application. There is one thread that processes Jack's audio via a callback function. The other processing threads in JackTrip are the *Sender*, which wraps audio packets from the audio thread with the header information into UDP packets, and the *Receiver* that unwraps the packet and has it ready when the audio process callback needs it.

Inter-thread communication is implemented using ring (or circular) buffers as shown in Figure 1. This is one of the critical latency-reducing parts of the design. The only thread that blocks against its input from the ring buffer is the UDP Sender thread; there's no need to send audio that hasn't been generated. Every time a buffer is available on the ring buffer, the sender thread immediately sends it as a UDP packet. Conversely, the receiving ring buffer cannot block, since local audio must obtain a packet from the ring buffer when it is scheduled to—otherwise audio glitches will be heard. JackTrip maximizes reliability, audio flexibility and minimizes as much as possible peer-to-peer audio delay. Two parameters which affect local audio latency are sampling rate and buffer size. For example, using a sampling

[1] Recordings of experiments with the St Lawrence String Quartet are available at http://ccrma.stanford.edu/groups/soundwire/research/slsq/

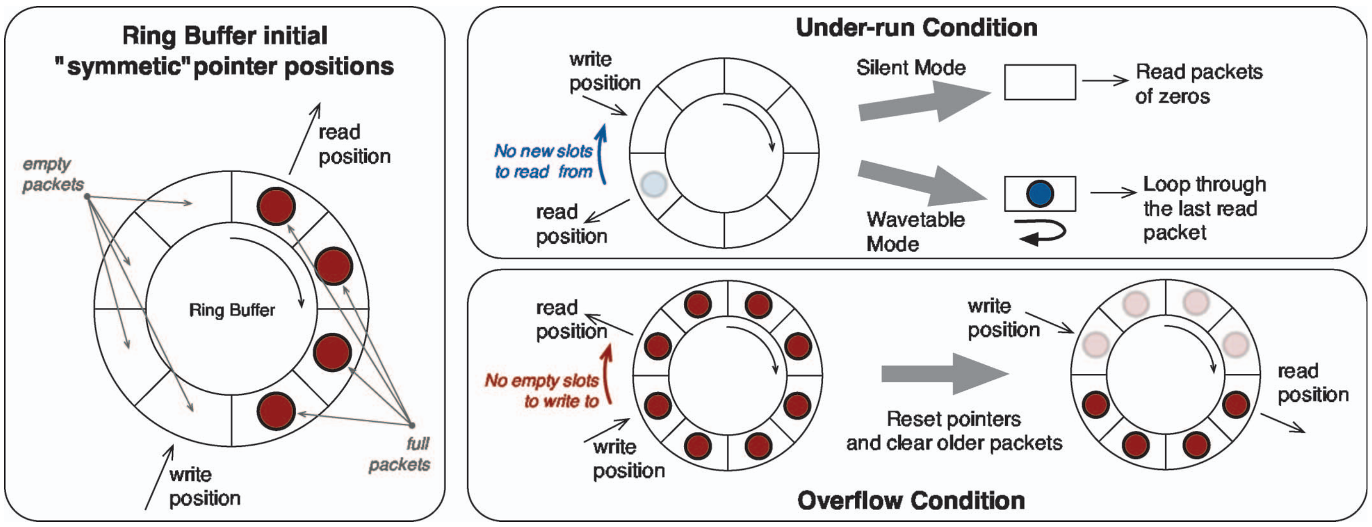

Fig. 2. Ring buffers.

rate of 96 kHz and an audio buffer of 64 frames (or samples), the rate of audio packet delivery is every $64/96000 \times 1000 = 0.67$ ms.[2]

2.1 Thread scheduling

Threads in JackTrip are scheduled as *real-time* priority, i.e. jack audio and socket threads will take priority over any other non-critical process. This avoids interruptions during time-critical tasks.

2.2 Buffering mechanism

Two types of scheduling problems can occur on the receive side, illustrated in Figure 2.

- **Overrun condition.** The receiving ring buffer is full, i.e. there is no space to write new buffers coming from the UDP Receiver thread. This normally happens when asynchronous clocks drift, e.g. the peer's clock runs faster than the local clock.
- **Under-run condition.** The receiving ring buffer is empty, i.e. there are no new packets to read. This is caused either because there are packets that are delayed or lost in the network or because the clocks of the two machines have drifted the other way.

This is different from common streaming applications that can stop playback (e.g. audio-video playback on browsers) when they reach an under-run condition, and won't have the overflow problem because real-time is not a concern. Typically, these applications adaptively increase or reduce their buffer size. In JackTrip, latency needs to be constant and another method is needed to deal with both under and over-run conditions.

Ring buffers have a *read* and a *write* pointer (Figure 2). On initialization, both the read and write pointers are in a 'symmetric' position. The longer the buffer size, the higher the latency.[3] The length of the buffer is a provision for network jitter; slight variations around this symmetry are produced by packets not arriving at exactly the same frequency. In an ideal situation, where both machines have clocks that match exactly and no packets are lost, the symmetric position will be maintained on average throughout the connection. The higher the jitter the longer the ring buffer needs to be to avoid glitches.

2.3 Buffer glitches

Primarily as a result of receive buffers not being sized to accommodate network jitter, glitches occur and have to be dealt with. The application has two different modes that respond to under-runs (Figure 2).

- **Silent mode.** Send a packet of zeros (silence) to the process callback.
- **Wavetable mode.** Re-send the last available packet to the process callback. This will produce a wavetable synthesizer type of sound when there are no new packets available for some time, since it is going to loop on the last received one.

For under-runs, the pointers are not reset because we always want to be able to *read* the most recent packet.

To deal with buffer overflows, the ring buffer *read pointer* is reset to the symmetric position with respect to

[2] Internal redundancy and other factors can make the actual local latency approximately the double of this number, but the delivery rate, i.e. the rate at which packets are sent and received, corresponds to that number.

[3] This latency can be visualized as the signed 'distance' between the *read* and *write* pointer.

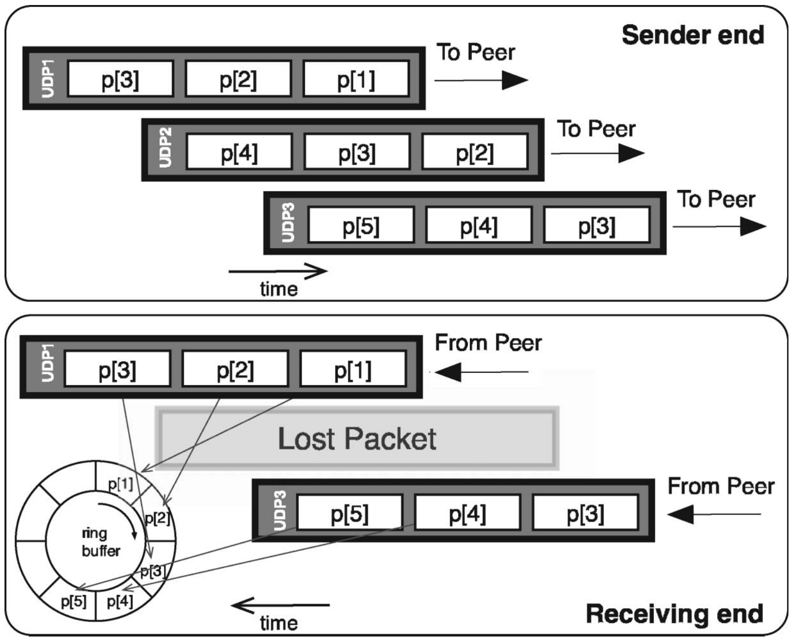
Sender end
UDP1 p[3] p[2] p[1] To Peer
UDP2 p[4] p[3] p[2] To Peer
UDP3 p[5] p[4] p[3] To Peer
time
UDP1 p[3] p[2] p[1] From Peer
Lost Packet
UDP3 p[5] p[4] p[3] From Peer
ring buffer
p[1] p[2] p[3] p[4] p[5]
time
Receiving end

Fig. 3. UDP redundancy, *RedunFactor* = 3.

the *write pointer*. Some packets[4] will be lost in the process but the clock drift will be reset to its original position, thus avoiding another glitch for an extended amount of time.

2.4 Packet redundancy algorithm

As an unreliable transport mechanism, UDP has no provisions to notify the sender when or if a packet was successfully delivered, or if the receiving order matches the sender's. With today's good network Quality of Service (QoS), we generally experience a very low number of lost or out-of-order packets. But, since even one misplaced packet will be perceived as a glitch, JackTrip includes a mechanism to recover (within certain bounds) lost or unordered packets.

Jacktrip's redundancy algorithm is used when sufficient bandwidth is available. The technique is illustrated on Figure 3. The sender bundles *RedunFactor* copies of every audio+header packet into a bigger UDP packet (with *RedunFactor* $\in \mathbb{Z}^+$). This is done for every new audio+header buffer, so each UDP packet has an overlap of *RedunFactor* − 1 buffers, as illustrated in the figure. On the receiving end, Algorithm 1 reads a UDP packet and determines if it has not already received the extra copies. New copies are sent to the ring buffer, and extra copies are discarded. Lost packets are recovered as illustrated in Figure 3.

Algorithm 1 Packet redundancy receiving end

```
For every new UDP packet
$CurSeqNum \leftarrow$ Packet highest sequence num
if $(CurSeqNum - LastSeqNum) \leq RedunFactor$ then
    $NumNewPackets = CurSeqNum - LastSeqNum$
else
    $NumNewPackets = RedunFactor$
end if
for $i = (NumNewPackets - 1)$ to 0 do
    Send $P[CurSeqNum - i]$ to Ring Buffer
end for
$LastSeqNum \leftarrow CurSeqNum$
```

2.5 Processing plugins

JackTrip also has the ability to dynamically add plugins into the audio process callback (Figure 1). One plugin implements loopback mode, i.e. audio received from a peer is immediately sent back. This allows a location to listen to its echo from a remote peer. The aural evaluation of network quality (Chafe & Leistikow, 2001) or the synchronization of music through 'feedback locking' (Cáceres, Hamilton, Iyer, Chafe, & Wang, 2008) are two practical applications that use this approach.

Plugins can also be used for 'Internet acoustics' or sonification through physical models (Chafe, Wilson, & Walling, 2002), e.g. a network implementation of the Karplus-Strong algorithm for strings and drums synthesis.

3. Conclusions and future work

'Broader' broad-band networks have the capacity to support high-quality audio. JackTrip serves to illustrate

[4] Half of the ring buffer for even buffer sizes, and half minus 1 for odd buffer sizes, to be precise.

some of the software design decisions for achieving low-latency, bi-directional audio using these networks. Its particular uses have different requirements: collaborative music making is different from, for example, one-way remote studio recording where latency is not the issue but packet loss is. Depending on the application, JackTrip allows the user to tune its configuration, for example trading off some reliability (allowing for minor glitches) in favour of tighter latency.

At the design stage, the engineer must provide the methods that support the highest-quality musical performance. In particular, JackTrip deals with packet loss by providing a redundancy algorithm, and deals with clock drifts and late or unrecoverable packets by using lower-level strategies in ring buffers that can, e.g. sound like a wavetable synthesizer, thus extending the musical sonority of the moment. Clock drift between remote WAN machines is still an unsolved issue and there are presumably new techniques to be tried in the future, like adaptive re-sampling, packet cross-fading, and others.

The current work of JackTrip is focused on the application layer, but new network projects like OpenFlow (OpenFlow Consortium, 2008) and Dynamic Circuit Network (Internet2, 2009) provide the opportunity to start experimenting with lower layers; it would be possible to dynamically specify network paths to minimize latency, or to obtain dedicated bandwidth for a more reliable QoS.

Acknowledgements

Fundamental contributors to design and coding have included Scott Wilson, Randal Leistikow and Daniel Walling. At CCRMA, Fernando Lopez-Lezcano and Carr Wilkerson have provided continuous technical support and advice. Various individuals have also contributed in testing the software during several concerts, in particular Alain Renaud and Jonas Braasch. Starting in the year 2000, The Banff Centre has provided a second home for development and concert testing through residencies and artistic projects. We are particularly indebted to Theresa Leonard, director of audio programs, for her continuous support.

References

Bouillot, N., & Cooperstock, J.R. (2009). Challenges and performance of high-fidelity audio streaming for interactive performances. In *NIME '09: Proceedings of the 9th International Conference on New Interfaces for Musical Expression*, Pittsburgh, PA, USA, pp. 135–140.

Cáceres, J.-P., Hamilton, R., Iyer, D., Chafe, C., & Wang. G. (2008). To the edge with China: Explorations in network performance. In *ARTECH 2008: Proceedings of the 4th International Conference on Digital Arts*, Porto, Portugal, pp. 61–66.

Carôt, A., Hohn, T., & Werner, C. (2009). Netjack—remote music collaboration with electronic sequencers on the Internet. In *Proceedings of the Linux Audio Conference*, Parma, Italy.

Chafe, C. (2009). Tapping into the Internet as an acoustical/musical medium. *Contemporary Music Review*, *28*(4), 413–420. Retrieved from http://www.informaworld.com/10.1080/07494460903422362

Chafe, C., Cáceres, J.-P., & Gurevich, M. (2010). Effect of temporal separation on synchronization in rhythmic performance. *Perception*, *39*(7), 982–992.

Chafe, C., & Leistikow, R. (2001). Levels of temporal resolution in sonification of network performance. In *Proceedings of the 7th International Conference on Auditory Display* (pp. 50–55). Helsinki: Laboratory of Acoustics and Audio Signal Processing and the Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology.

Chafe, C., Wilson, S., Leistikow, R., Chisholm, D., & Scavone, G. (2000, December). A simplified approach to high quality music and sound over IP. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Verona, Italy.

Chafe, C., Wilson, S., & Walling, D. (2002). Physical model synthesis with application to internet acoustics. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (Vol. 4, pp. IV-4056–IV-4059). Los Alamitos, CA: Institute of Electrical and Electronics Engineers.

Comer, D.E. (2005). *Internetworking with TCP/IP, Volume 1* (5th ed.). New York: Prentice Hall.

Davis, P. (2009). JACK: Connecting a world of audio. Retrieved from http://jackaudio.org/

Internet2. (2009). Internet2 Dynamic Circuit Network. Retrieved from http://www.internet2.edu/network/dc/

Letz, S., Orlarey, Y., & Fober, D. (2005). Jack audio server for multi-processor machines. In *Proceedings of International Computer Music Conference*, Barcelona, Spain, pp. 1–4.

Nokia Corporation. (2008–2009). Qt Software. Retrieved from http://www.qtsoftware.com/

OpenFlow Consortium. (2008). The OpenFlow Switch Consortium. Retrieved from http://www.openflowswitch.org/

Renaud, A.B., Carôt, A., & Rebelo, P. (2007). Networked music performance: State of the art. In *Proceedings of the AES 30th International Conference*, Saariselkä, Finland.

Scavone, G.P. (2002). RtAudio: A cross-platform C^{++} class for realtime audio input/output. In *Proceedings of International Computer Music Conference*, Gothenburg, Sweden, pp. 196–199.

SoundWIRE Group. (2010). SoundWIRE research group at CCRMA, Stanford University. Retrieved from http://ccrma.stanford.edu/groups/soundwire/

Xu, A., & Cooperstock, J.R. (2000). Real-time streaming of multichannel audio data over Internet. In *Proceedings of the 108th Convention of the Audio Engineering Society*, Paris, France, pp. 627–641.