The Impact of Clock Jitter on MIDI syncable DAWs

Technical Report

Maximilian Rest

E-RM - Erfindungsbüro Rest & Maier

info@midiclock.de
www.midiclock.de

Berlin, Februrary 2014
(updated 4.3.2014)

Contents

1 Introduction

> *"How sour sweet music is, when time is broke and no proportion kept!"*
>
> – William Shakespeare, *King Richard II*, 1595

Already more than 400 years ago, Shakespeare knew about the effects of jitter and phasing in musical performances, and it tells in a nutshell what contemporary musicians still have to take care of today - namely the tight synchronisation of different tracks when using multiple computers as digital audio workstations.

The report at hand explaines a setup of synchronising multiple DAWs, analyses problems most musicians face with MIDI Clock and shows how the E-RM *midiclock* can greatly improve overall timing.

Although *Live* from Ableton AG has been used for the experiments in this paper, all findings hold true in general for all other Music Production Softwares.

If you are interested in the experimental results only, you may jump to Section 5 on Page 11.

2 Problem Setting

The sequencer under examination will be *Live* from Ableton AG, one of the most popular music production softwares currently on the market [4],[8],[12].

Like musicians that play classical instruments, users of DAWs also want to play together with other artists. This is why *Live* has capabilities to synchronise software instances on different computers. One of the easiest solutions is to use the Musical Instrument Digital Interface (MIDI). The MIDI specification offers the use of MIDI Clock signals that keep all connected devices in sync like the drummer in a band [13, p. 30].

MIDI is technically an unidirectional serial interface that always needs a master who sends data, and a slave, who receives it. *Live* can either be configured as a master or a slave concerning MIDI Clock data. A common way to synchronise multiple DAWs is depicted in Figure 1.

"Specifically, the master sends 24 MIDI Clocks, spaced at equal intervals, during every quarter note interval." [1]. This enables the slave to extract tempo and phase information from the incoming Beat Clock signals and play back all audio tracks synchronised to the master.

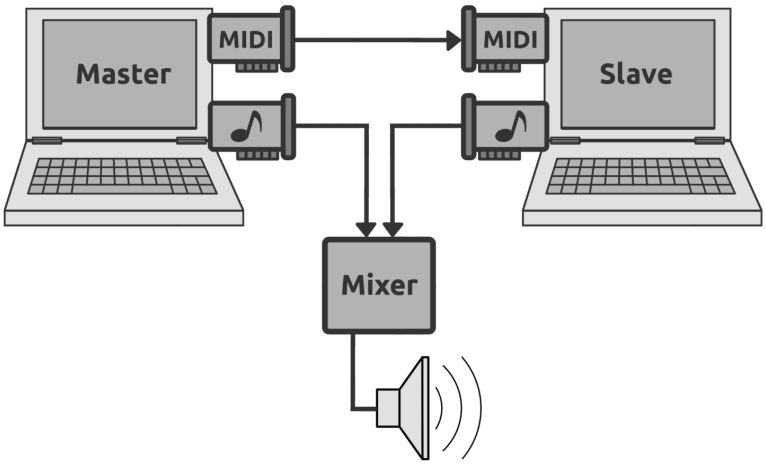

Figure 1: Usual MIDI Beat Clock distribution between multiple DAWs

MIDI Syncronisation and Audio Jitter

A problem which a lot of artists confront when trying to merge audio tracks played by different DAWs (e.g. as in a setup as shown in Fig. 1) is that the resulting synchronisation lacks accuracy.

Although the concept of MIDI Clock appeals through theoretical simplicity, practically the audio tracks from the slave DAW often loose synchronisation and are not played tightly to the master DAW. Ableton introduced clock slave improvements in *Live 8.4.1b1* back in 2011, but states:

> "The stability of the slave algorithm still depends on your system and the quality of your master clock. If you have a real bad master clock or bad MIDI drivers, you probably will not see much of an improvement." [11]

3 Verify Synchronisation Problems between DAWs

To verify this phaenomenon on a scientifical level, a test setup as shown in Figure 2 is evaluated. Two DAWs with *Live 9* are connected through their respective MIDI ports. The MIDI master clock is converted to an audio signal [5] and recorded together with the audio output of the slave DAW with a sampling frequency $f_s = 96kHz$.

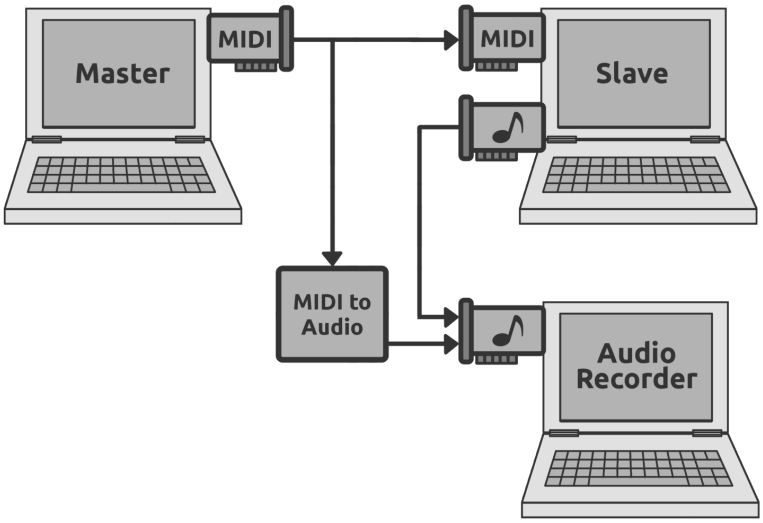

Figure 2: Test setup to verify synchronisation problems between DAWs

Both master and slave DAW are put under 50% CPU load to ensure real-life circumstances. The master entity of *Live* is set to 120 Beats per Minute (BPM) and streams MIDI Clock Ticks to it's MIDI OUT port. On the slave DAW - which locks its playback to the incoming MIDI Clock stream - a short *click*-sound is played eight times per quarter note.

Two tests with MIDI slave receivers in completely different price ranges are performed to compare the gathered data. In Test A, a professional MIDI receiver with a dedicated software driver is used, in Test B a simple MIDI to USB converter with the operating systems class compliant driver is installed. The detailed device configurations are shown in Table 3, page 15. The test duration is 5 minutes and the audio files are recorded in an uncompressed, sample accurate format for further processing.

Test Results

The data has been numerically analyzed with the free software package GNU Octave.

Figure 3 and 4 show BPM(t) behaviour for Test A and Test B. Tempo drops and bursts of approx. ± 0.5BPM centered around the desired playback rate of $\text{BPM}_{set} = 120$ can be observed in both graphs, but more frequent in Test B where a MIDI Interface without a dedicated driver is used.

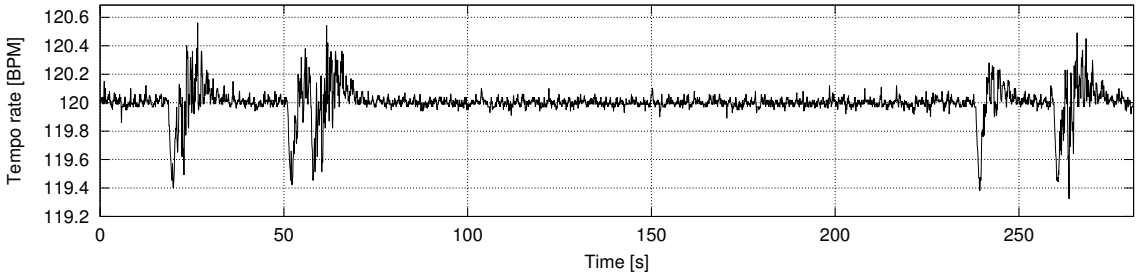

Figure 3: BPM over Time for Test A

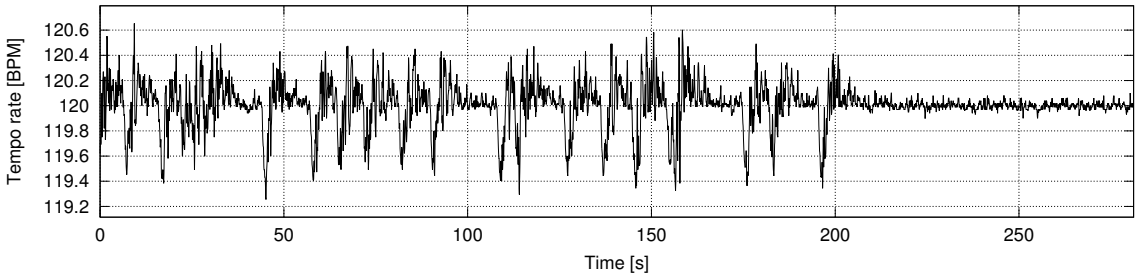

Figure 4: BPM over Time for Test B

Moreover, also outside the range of the bursts the tempo constantly oscillates in both curves with $\pm\frac{3}{100}$BPM. Statistical data can be found in Table 1.

A look at Figures 5 and 6 shows at a glance what artists complain about: The notes of the slave DAW are not played on their desired playback position, which causes the perception of inaccurate synchronisation.

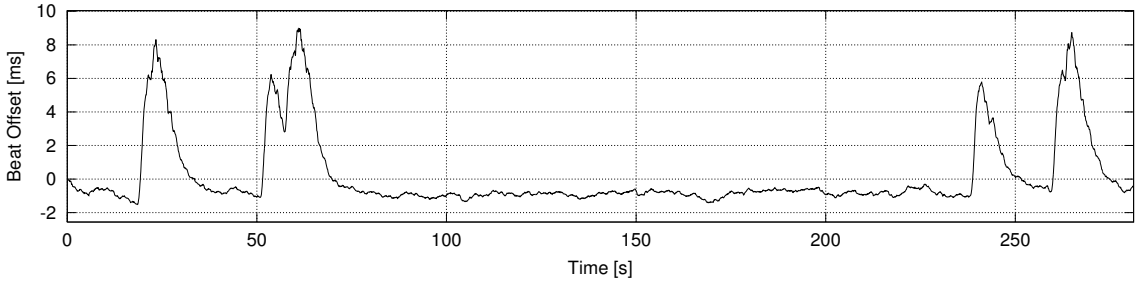

Figure 5: Beat Offset over Time for Test A

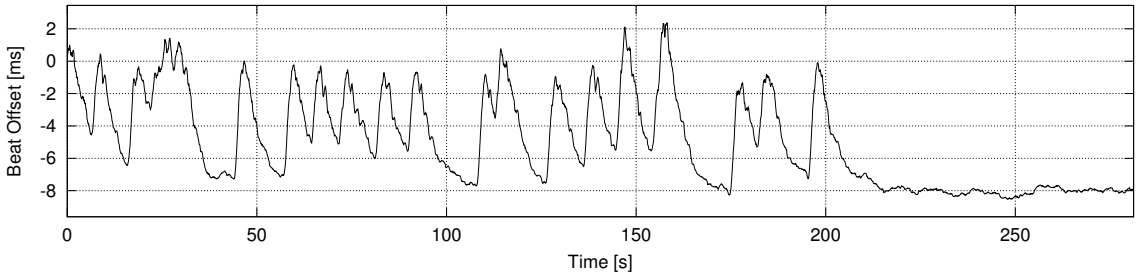

Figure 6: Beat Offset over Time for Test B

The problem is worse with the cheap MIDI interface than with the professional grade device, but both setups have severe timing issues. Whereas the graph from Test A (Fig. 5) at least settles back to a constant offset of roughly $-1ms$ after drops and bursts in the tempo rate, the graph from Test B (Fig. 6) settles at $-7ms$, $-7.5ms$ and $-8ms$.

Setup B most probably even looses MIDI Ticks, the equation to calculate Beat Offset acts like an integrator and the constant negative offest increases over time (see appendix, p. 13). The PLL unlocks on a lost MIDI Tick and locks again afterwards, in an unknown phase state, which results in a different offset [2, p. 103].

The histograms in Figure 7 show the distribution of Beat Offset for both tests. The interpretation of the mean value deserves our particular interest, as the integration constant depends on the start position of the measured data and is not set at the beginning of the data processing.

Test A may look okay at the first glance due to its high peak at a constant offset of $-1ms$, but nevertheless the standard deviation is $2.28ms$ and peaks of up to $9ms$ occur (Tab. 1).

For Test B, the spread of the *min* and *max* values is almost the same as in Test A, and so is the standard deviation. Differences can be seen in the histrogram, which shows a more widespread distribution with a much lower peak at $-8ms$ relative to the other Beat Offsets which occupy a larger area.

The *Ableton Reference Manual* states that "(Beat) Jitter, much more so than latency, creates the feeling that (..) timing is "sloppy" or "loose." " [3, p. 592]. The

Setups in Test A and B would most probably fail a listening test, as a Beat Offset of $-8ms$ equals 13% of the ideal period of $T_{ideal}/f_s = 62.5ms$ between adjacent notes.

The cycle-to-cycle jitter of the MIDI Clock signal from the master DAW can be seen in Figure 8. It is a direct measure of instantaneous jumps of frequency in the clock stream [7, p. 3]. MIDI data from Test A and Test B looked almost the same regarding cycle-to-cycle jitter so that only data from Test A is shown.

The ideal period between adjacent MIDI Clock Ticks at BPM_{set} is $60s/(120 \cdot 24) = 20.1ms$, cycle-to-cycle jitter has peaks of $-38ms$ (Tab. 1).

In this magnitude, C2C jitter obviously exceeds the lock-range of the slave PLL, which gets unlocked from the master tempo and cycles start to slip through [10, p. PLL/6] [2, p. 89]. Unfortunately, one can't precisely distinguish between lock-outs due to ticks that got lost in the slave DAW and those due to excessive cycle-to-cycle jitter.

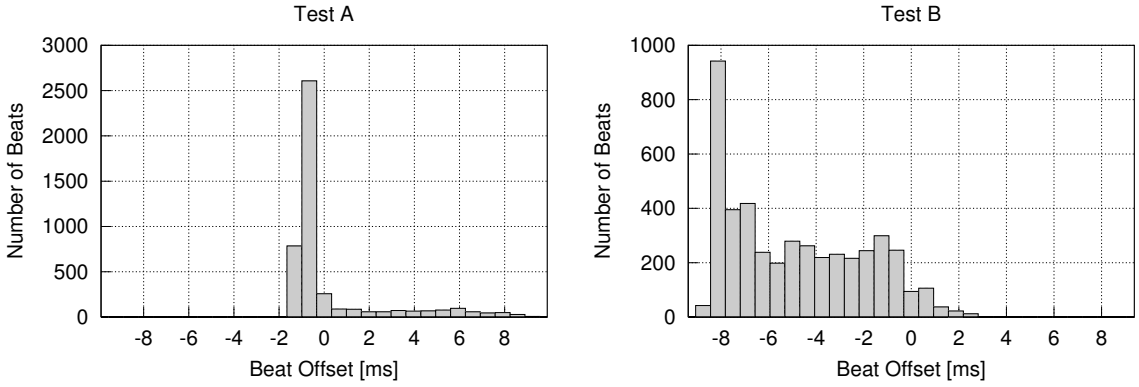

Figure 7: Histograms of Beat Offset distribution for Test A and Test B

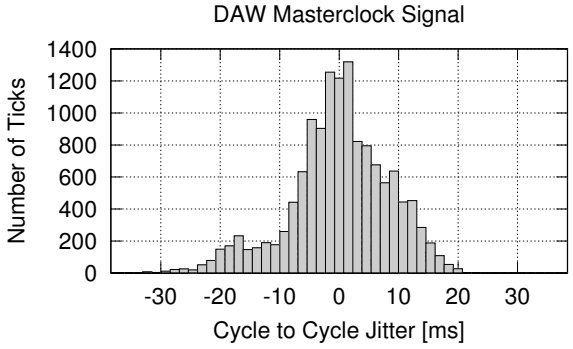

Figure 8: Histogram of DAW generated MIDI Master Clock cycle-to-cycle jitter

Test: Data	**N**	**unit →**	**mean**	**$\sqrt{\text{var}}$**	**min**	**max**
A: Tempo Rate	4500	BPM	120.000	0.114	119.26	120.62
A: Beat Offset	4500	ms	0.231	2.278	-1.51	8.99
B: Tempo Rate	4500	BPM	120.000	0.179	119.23	120.68
B: Beat Offset	4500	ms	-5.066	2.869	-8.51	2.36
A+B: MIDI C2C	13500	ms	0	8.433	-38.36	22.24

Table 1: Statistical data for Test A and Test B

4 Enhance Synchronisation with the E-RM *midiclock*

To show the improvements of DAW-Sync by the E-RM *midiclock*, a test setup as shown in Figure 9 is used. The E-RM *midiclock* acts as the Master Clock, just like the Master DAW in Section 3, Fig. 2.

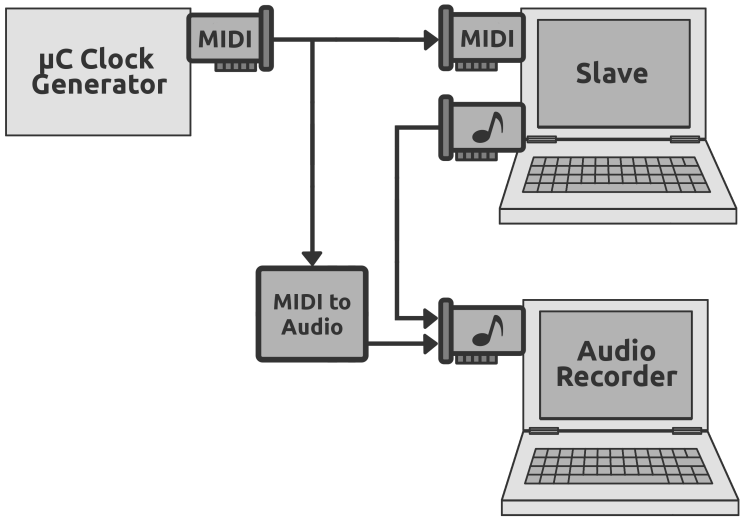

Figure 9: Test setup to verify synchronisation enhancement between DAWs by the E-RM *midiclock*

Again, two different MIDI slave receivers are used, the professional grade receiver for Test C, the cheap & simple one for Test D. The rest of the devices is also the same as in in Section 3 (see Table 3 on page 15 for details).

The recording frequency is $f_s = 96kHz$, the test duration 5 minutes to make the results comparable to Test A and Test B.

As the sampling frequency of the audio recorder in the setup of Fig. 9 is only $f_s = 96kHz$, the MIDI Clock stream from the E-RM *midiclock* is analyzed furthermore

with a logic analyzer at $f_s = 24MHz$, as the maximum time resolution at $f_s = 96kHz$ is only $T_{res} = 1/f_s = 10.42us$.

Test Results

BPM(t) is calculated as in Equation 6 (Appendix, p. 13) and compared against the former results from Test A and Test B without a stable MIDI Clock signal.

Most obviously, no drops and bursts in the tempo rate can be found in the data for both Test C/D (Fig. 10 and 11). The tempo rate is stable at 120BPM $\pm$ 0.05% (Table 2).

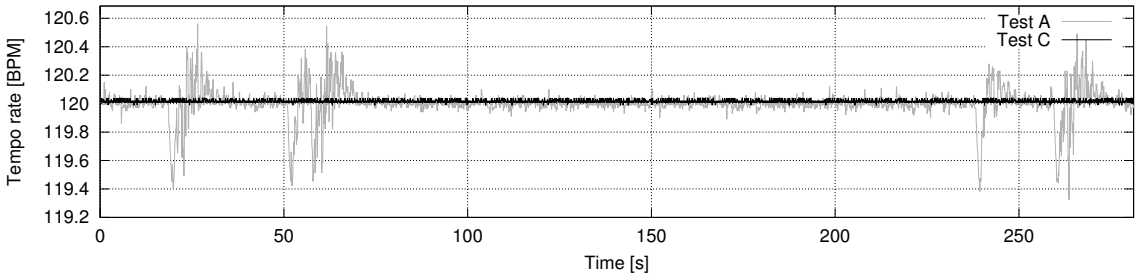

Figure 10: BPM over Time for Test C

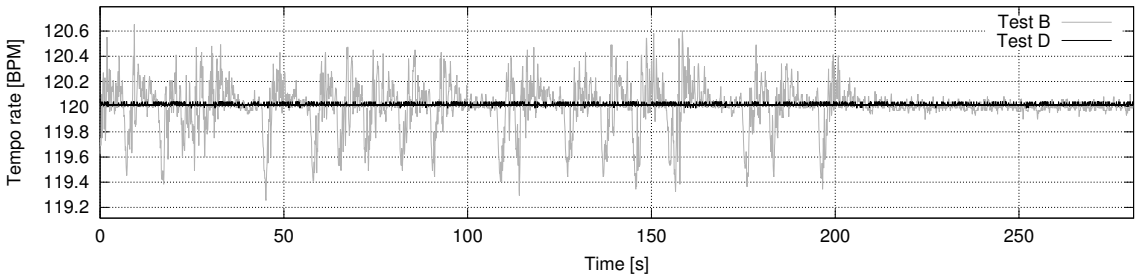

Figure 11: BPM over Time for Test D

There is a serious Beat Offset enhancement as can be seen from Figure 12 and 13. No more dropouts, no loosing-track of the desired beat frequency. The stable MIDI Clock signal drastically enhances synchronisation of the slaved DAW.

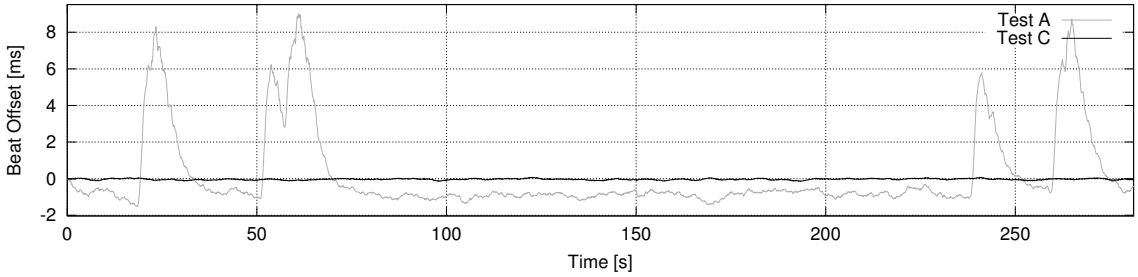

Figure 12: Beat Offset over Time for Test C

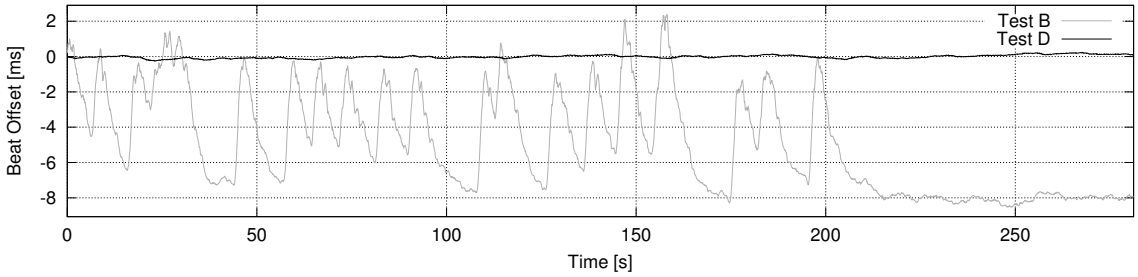

Figure 13: Beat Offset over Time for Test D

The gathered MIDI data reveals MIDI cycle-to-cycle jitter, the histogram in Figure 14 shows the result.

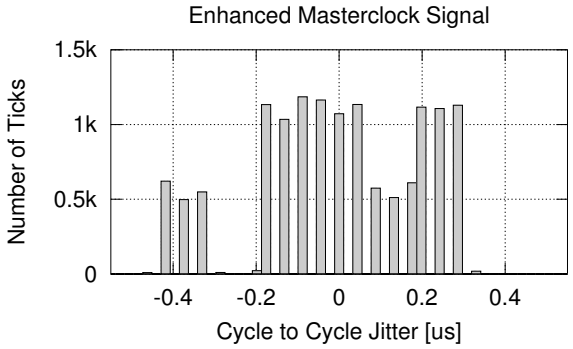

Figure 14: Histogram of *midiclock* cycle-to-cycle jitter in [us]

The former maximum MIDI Clock jitter of 38ms is reduced by three magnitudes down to 400ns by the E-RM *midiclock*.

A look at the histogram plots of Beat Offset in Figure 15 reveals even more evidence that a stable MIDI Clock signal improves audio playback on a slaved DAW. The distributions are almost gaussian and centered very narrow around their mean value.

A relative enhancement ratio is defined by comparing the highest magnitudes of Beat Offset of related tests:

$$e = \frac{\max(\text{mean}_{old} - \min_{old}, \max_{old} - \text{mean}_{old})}{\max(\text{mean}_{new} - \min_{new}, \min_{new} - \text{mean}_{max})} \tag{1}$$

For Test A/C, Beat Offset enhances by a factor of $e = 79.37$, for Test B/D by $e = 29.94$.

As the Clock data from the E-RM *midiclock* which gets fed into the slave DAW can be considered almost 'jitter-free', the biggest portion of remaining beat jitter is inducted by the slave DAW and it's MIDI interface and driver.

It is important to note that the relative enhancement ratio e is higher for Test C. A professional MIDI Interface with dedicated drivers apparently adds less jitter to the incoming messages.

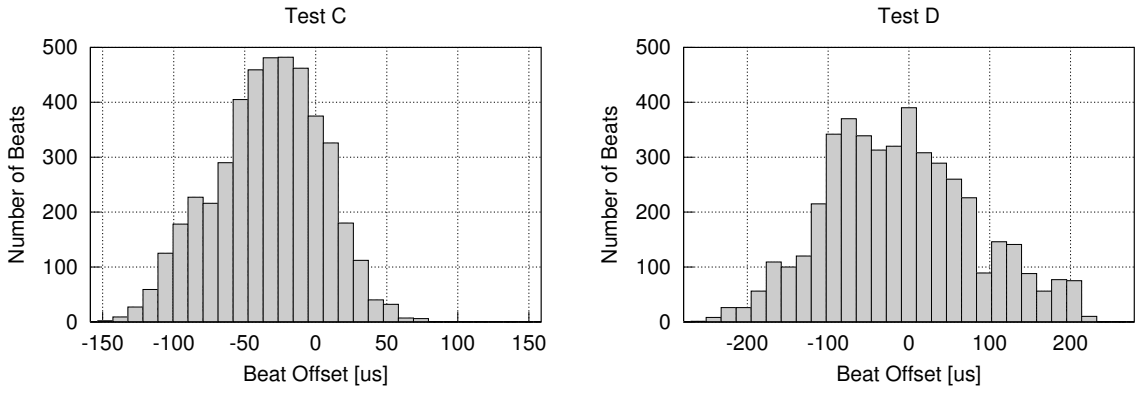

Figure 15: Normalized Histograms of Audio TIE distribution for Test C and Test D

Test: Data	**N**	**unit →**	**mean**	**$\sqrt{\text{var}}$**	**min**	**max**
C: Tempo Rate	4500	BPM	120.020	0.018	119.96	120.06
C: Beat Offset	4500	us	-33.895	37.841	-144.25	71.22
D: Tempo Rate	4500	BPM	120.020	0.018	119.96	120.06
D: Beat Offset	4500	us	-5.374	93.755	-253.44	228.49

Table 2: Statistical data for Test C and Test D

5 Conclusion

To ensure tight synchronisation of MIDI syncable DAWs, a stable master clock signal must be provided. With the E-RM *midiclock*, a simple solution is available, which eliminates frequently changing beat phasing and forces all slaved DAWs to follow the given tempo.

Reducing the former maximum MIDI Clock jitter of $38ms$ by three magnitudes down to $400ns$ caused a cut down of Beat Offset spread from $10ms$ by a factor of 48 to $210us$ with a professional MIDI receiver.

With the E-RM *midiclock*, an endless number of slaves can be connected using MIDI distribution boxes, which will all run in sync with the master clock and thus in sync with each other.

Appendix

Introduction to Jitter

The problem which one confronts when audio playback on a slaved DAW is out of sync is called jitter:

> "Jitter is defined as the (..) deviation of a signal's transition time from its ideal position in time." [6, p. 122]

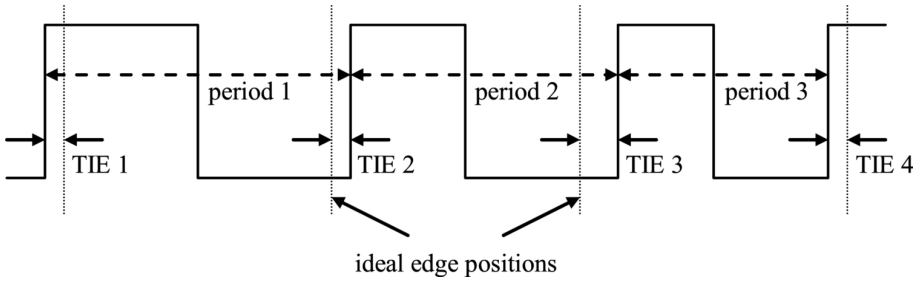

Figure 16: Definition of jitter [6, p. 128]

In the case of audio synchronisation, the N notes recorded from the slave DAW can be treated as the signal under examination. They are compared to an imaginary reference clock with ideal edge positions, predefined by the notes that should regularily be played at the given tempo [6, pp 124-125,127].

As defined by Maichen in *Digital Timing Measurements* [6], this jitter is generally called "Time Interval Error" (TIE, see Fig. 16). In this case, TIE is the deviation of the actual playback time of a note from its ideal position.

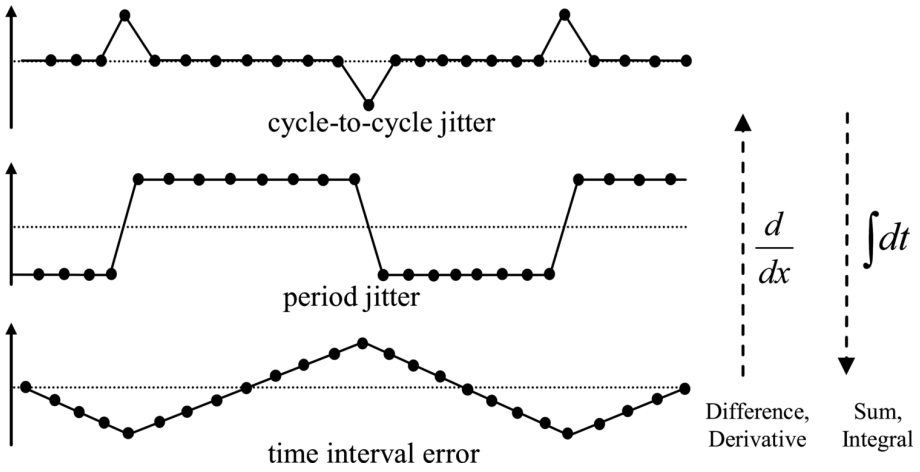

Figure 17: Different jitter trend plots [taken from 6, p. 128]

In the evaluation it became evident that the gathered MIDI data can't be used as a reference clock for audio jitter calculation because of it's own jitter. Since there is no other reference clock recorded in the setup, time interval error must be calculated round the back by period jitter and the desired period between two notes.

For that purpose all positions $pos(n)$ of the notes $n \in [1 : N]$ are extracted with numerical software from the respective recording. As the audio files are uncompressed and sample accurate, the returned position $pos(n)$ is a sample number in the file.

To obtain the period $T(n)$ (in samples) between two notes n and $n + 1$ from the position data, adjacent positions must be subtracted:

$$T(n) = pos(n + 1) - pos(n) \quad , \quad n \in [1 : N - 1] \tag{2}$$

At the playback tempo of $BPM_{set} = 120$ in the tests, the desired period T_{ideal} between two adjacent notes is:

$$T_{ideal} = \frac{f_s}{8 \cdot BPM_{set}} = \frac{96000 \cdot 60s}{8 \cdot 120 \cdot s} = 6000 \tag{3}$$

The factor 8 in the denominator represents the eight notes per quarter with BPM in general defined as the rate of quarter notes per minute.

Period jitter $\Delta_T(n)$ is then calculated as

$$\Delta_T(n) = T(n) - T_{ideal} \quad , \quad n \in [1 : N - 1] \tag{4}$$

Like shown in Figure 17, time intervall error $\Delta_{pos}(n)$ can now easily be calculated from $\Delta_T(n)$ by summation [9, p. 14].

$$\Delta_{pos}(n) = \sum_{i=1}^{n} \Delta_T(i) \quad , \quad n \in [1 : N - 1] \tag{5}$$

T_{ideal} probably needs some adjustment by a few per mill to take the actual playback tempo into account (for example 120.01 instead of 120.00 BPM). Otherwise, the summation in Equ. 5 will add up this error and run high or low.

$T(n)$ and $\Delta_{pos}(n)$ are now a direct measure for the actual tempo rate of the slave and the offset of the desired beat position. To make the results easily comprehensible, $T(n)$ and $\Delta_{pos}(n)$ are translated into BPM and a Beat Offset in seconds respectively.

BPM(t) as the actually measured playback rate at the time t is obtained by rearranging Equ. 3

$$\text{BPM}(t) = \frac{f_s \cdot 60s}{8 \cdot T(n)} \quad \text{with} \quad t = \frac{pos(n)}{f_s} \quad , \quad n \in [1 : N - 1] \tag{6}$$

Beat Offset in seconds is gained by converting the sample count to time:

$$\text{TIE}(t) = \frac{\Delta_{pos}(n)}{f_s} \quad \text{also with} \quad t = \frac{pos(n)}{f_s} \quad , \quad n \in [1 : N - 1] \tag{7}$$

In addition to the representation on the timeline, histograms of both BPM(t) and TIE(t) are plotted, as this gives a clue on the distribution of jitter [6, pp 129,132].

The MIDI Clock Ticks that are fed into the slave DAW are analyzed seperatly. As *Live* is synchronised to incoming clock pulses that travel through a complex computer system, some sort of clock smoothing and phase correction algorithm is used to calculate playback position and rate [11].

Most probably, a Software Phase-Locked-Loop (SPLL) accomplishes the desired filtering. As stated in *Digital Timing Measurements*, Section 8.2.1.4, a very useful parameter to estimate the stability of PLLs is cycle-to-cycle jitter (C2C). It is a direct measure of instantaneous jumps of frequency in the clock stream [7, p. 3]. Large deviations from the ideal clock tick position result in a low frequency jitter which can not be filtered out by the low-pass loop filter anymore, the PLL gets unlocked [10, p. PLL/6].

MIDI Clock Tick cycle-to-cycle jitter C2C(n) can be calculated from the N_M positions $pos(n)$ of the MIDI Ticks

$$\mathrm{C2C}(n) = [pos(n+2) - pos(n+1)] - [pos(n+1) - pos(n)] \tag{8}$$

with $n \in [1 : N_M - 2]$ [9, p. 14].

Devices used during the Test

Function	Machine Configuration	MIDI Interface	Soundcard
Test A			
Master	Samsung Q35 CD 1.66Ghz, 1GB Ram Windows XP, Live 9.0.2	ESI MIDIMATE II USB 2.0	
Slave	Apple MacBook Pro i7 1.66Ghz, 4GB Ram Mac OS X, Live 9.1.1	**RME Fireface UCX, FW400 (dedicated driver)**	RME Fireface UCX
Test B			
Master	*like Test A*		
Slave	*like Test A*	**Elektron TM-1, USB 2.0 (class compliant)**	*like Test A*
Test C			
Master	**E-RM *midiclock***		
Slave	*like Test A*		
Logic Analyzer	Saleae Logic 8ch, max. 24MHz sampling		
Test D			
Master	**E-RM *midiclock***		
Slave	*like Test B*		
L.A.	*like Test B*		
All Tests			
Recorder	IBM X61 C2D 1.6Ghz, 4GB Ram Linux Mint, Audacity 2.0.0		onboard

Table 3: Devices used to verify Synchronisation problems between DAWs

References

[1] Anonymous. *Syncing Sequence Playback*. http://home.roadrunner.com/~jgglatt/tech/midispec/seq.htm; 2014.

[2] Best, Roland E. *Phase-locked Loops - Design, Simulation and Applications*. 6th Edition. McGraw-Hill, 2007. ISBN: 978-0-07-1-49375-8.

[3] Dennis DeSantis et al. *Ableton Reference Manual Version 9*. 2013.

[4] Digital Music Doctor LLC. *Music Software Internet Popularity*. http://www.digitalmusicdoctor.com/popularity; 2013.

[5] Innerclock Systems Pty Ltd. *MIDI Clock to Audio circuitry*. www.innerclocksystems.com/NewICSClockWatch.html; 2014.

[6] Wolfgang Maichen. *Digital Timing Measurements - From Scopes and Probes to Timing and Jitter*. Frontiers in Electronic Testing, Vol. 33. Springer, 2006. ISBN: 978-0-387-31418-1.

[7] Howell Mitchell. *A Primer on Jitter, Jitter Measurement and Phase Locked Loops*. Application Note 687 - Rev. 0.1 5/12. Silicon Labs, 2012.

[8] MusicRadar. *The 15 best DAW software apps in the world today*. http://www.musicradar.com/tuition/tech/the-16-best-daw-software-apps-in-the-world-today-238905/15; 2012.

[9] PCI-SIG. *PCI ExpressTM Jitter Modeling*. White Paper - Revision 1.0RD. 2004.

[10] Prof. Dr.-Ing Klaus Petermann and Dr.-Ing Christian-Alexander Bunge. *Hochfrequenztechnik II - Vorlesungsskript*. 2012.

[11] Nico Starke. *MIDI clock (slave) improvements in Live 8.2.4b1*. https://forum.ableton.com/viewtopic.php?f=1&t=165235; 2011.

[12] TechMedia Network. *2014 Audio Production Software Product Comparisons*. http://audio-production-software-review.toptenreviews.com; 2014.

[13] The MIDI Manufacturers Association. *MIDI 1.0 Detailed Specification*. Document Version 4.2. 1995.