BRAIN PRODUCTS
Solutions for neurophysiological research

LSL: Pitfalls and Tweeks (Plus a Quick RT Programming Guide if We Have Time)

Dr. David Medine
David.Medine@brainproducts.com
Brain Products

October 7, 2019

0. Outline

- 2009-2014—PhD Computer Music, University of California

- 2009-2014—PhD Computer Music, University of California
- 2014-2017—Lab Engineer, Swartz Center for Computational Neuroscience

- 2009-2014—PhD Computer Music, University of California
- 2014-2017—Lab Engineer, Swartz Center for Computational Neuroscience
- 2017-2018—Software Developer, Brain Products

- 2009-2014—PhD Computer Music, University of California
- 2014-2017—Lab Engineer, Swartz Center for Computational Neuroscience
- 2017-2018—Software Developer, Brain Products
- 2018-????—Product Manager BCI, Brain Products

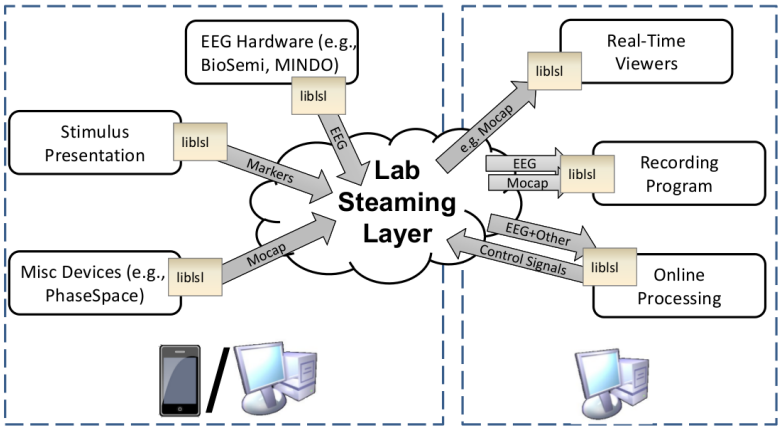
BRAIN PRODUCTS
Solutions for neurophysiological research

LSL Pitfalls and Tweeks

LSL Sync: Recap

What to correct:

- slowly drifting time offsets between CPUs
- jitter (micro fluctations in write-times)



Calculate a Clock Offset with `time_correction()`

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)
  - receive from inlet at outlet (t1)

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)
  - receive from inlet at outlet (t1)
  - immediately send from outlet to inlet (t2)

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)
  - receive from inlet at outlet (t1)
  - immediately send from outlet to inlet (t2)
  - receive from outlet at inlet (t3)

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)
  - receive from inlet at outlet (t1)
  - immediately send from outlet to inlet (t2)
  - receive from outlet at inlet (t3)
- round trip time (RTT) = (t3-t0) - (t2-t1)

Calculate a Clock Offset with `time_correction()`

- Retrieves clock offset during data acquisition using clock filter algorithm (NTP) using `gettimeofday` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet (t0)
  - receive from inlet at outlet (t1)
  - immediately send from outlet to inlet (t2)
  - receive from outlet at inlet (t3)
- round trip time (RTT) = (t3-t0) - (t2-t1)
- clock offset (OFS) = ((t1-t0) + (t2-t3))/2 (for lowest RTT)

Periodically Call `time_correction()`, Map Drifting Clock Values and Fit

Periodically Call `time_correction()`, Map Drifting Clock Values and Fit

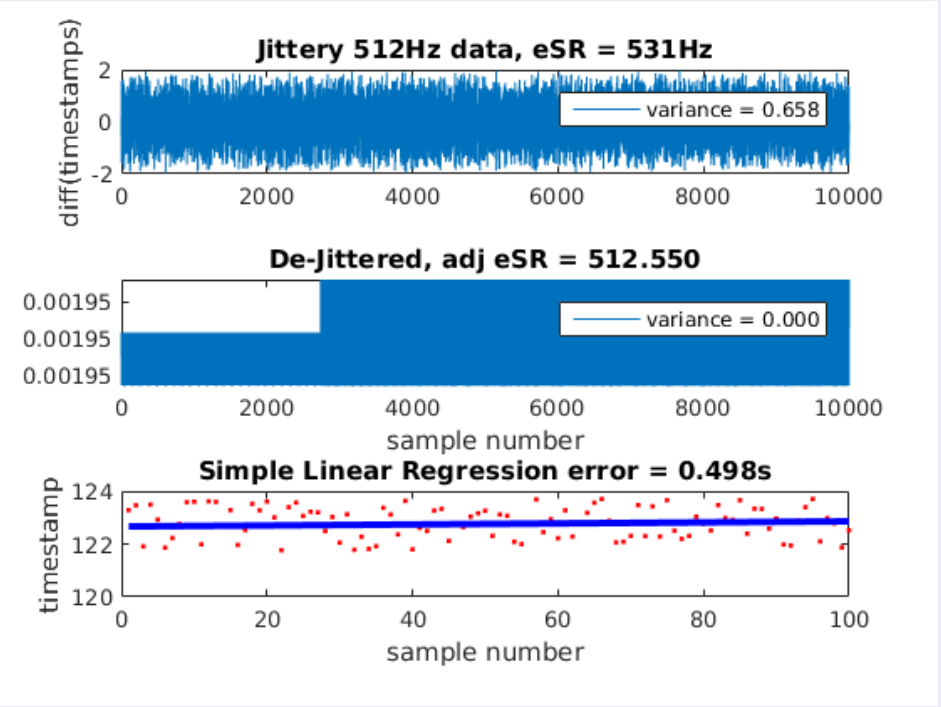
- This is normally done post-hoc in `load_xdf.m` using a fitting procedure. Each map is calculated using an ADMM method incorporating the Huber loss function (`http://web.stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf`)
- Each map is a DC offset and a slope adjustment ($y_n = ax_n + b$) for each intermittent OFS record point (default is 5s between queries in LabRecorder).

Periodically Call `time_correction()`, Map Drifting Clock Values and Fit

- This is normally done post-hoc in `load_xdf.m` using a fitting procedure. Each map is calculated using an ADMM method incorporating the Huber loss function (`http://web.stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf`)
- Each map is a DC offset and a slope adjustment ($y_n = ax_n + b$) for each intermittent OFS record point (default is 5s between queries in LabRecorder).
- This can also be done online, but it will destroy orginal, ground-truth timestamps.

Linearize/De-Jitter the timestamps (if appropriate)

- Simple linear regression (least squares) is very robust:



Python:

```
streams = resolve_stream('type', 'EEG')
inlet = StreamInlet(streams[0], processing_flags=proc_ALL)
```

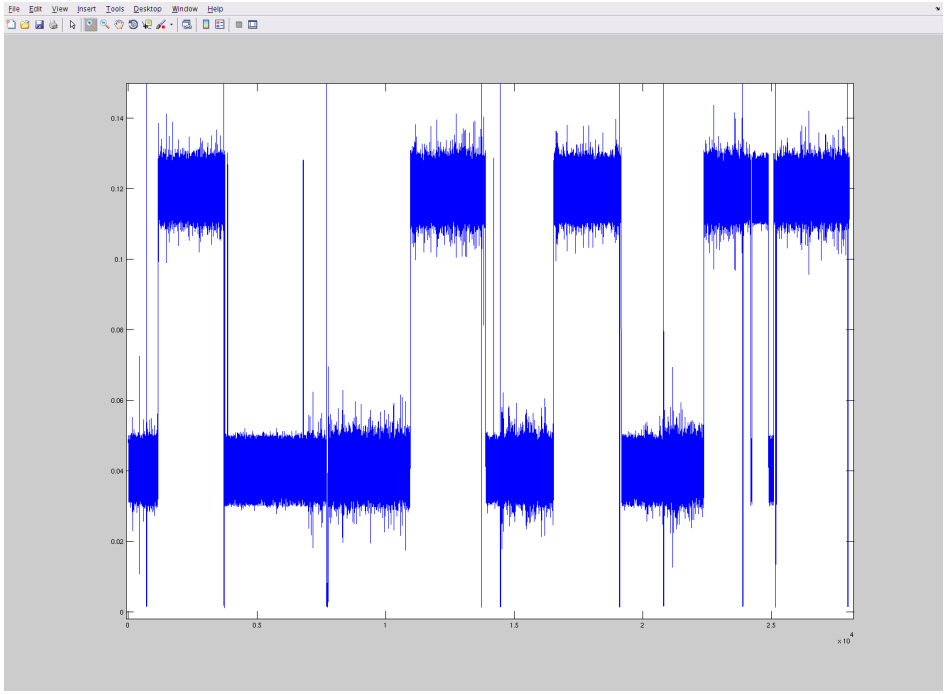
What Can Possibly Go Wrong?

Changing Sample Rate

This is bad:

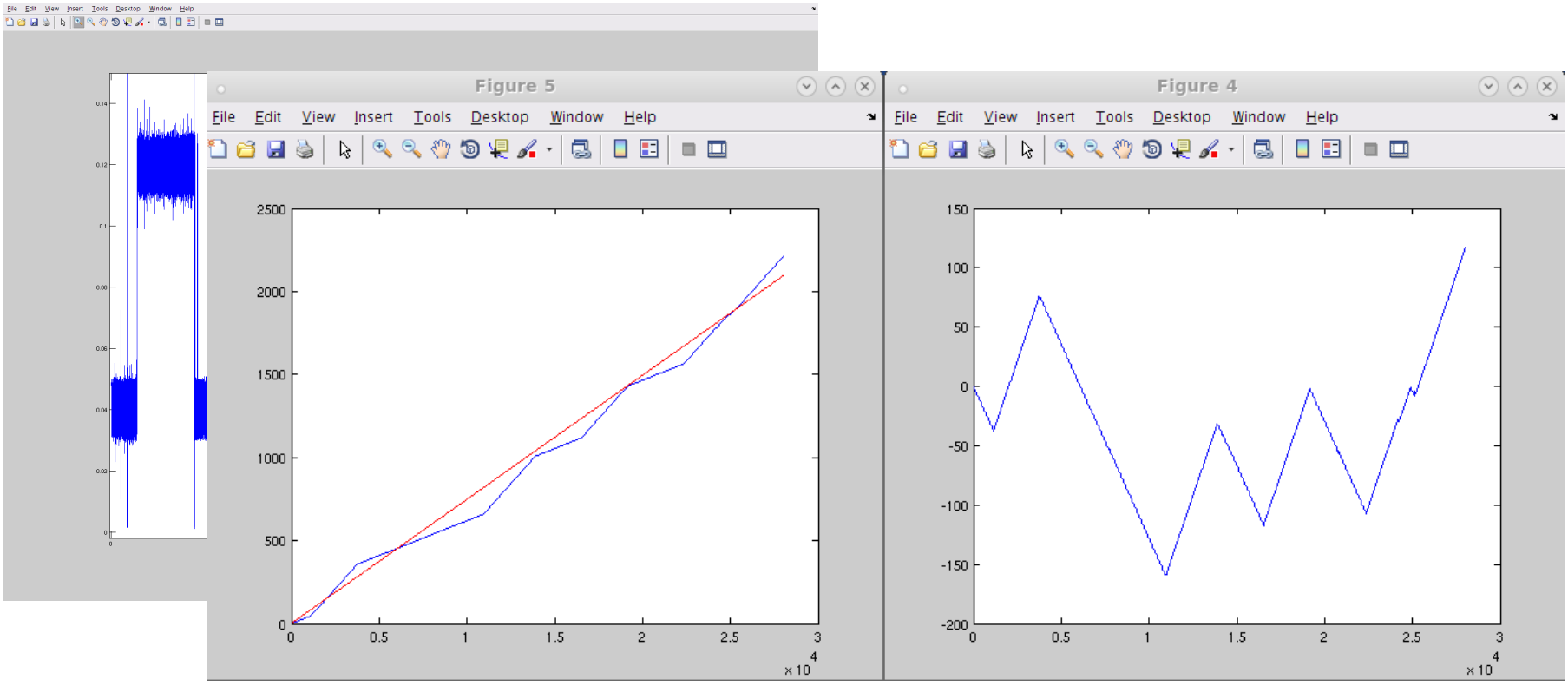
Changing Sample Rate

This is bad:



Changing Sample Rate

This is bad:



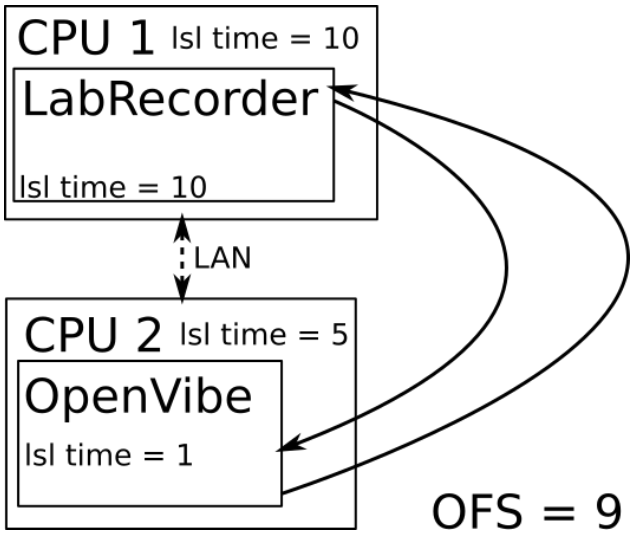
Red line is linearized timestamps, blue is raw. On the right is the difference: between +/-150s !!! .

Imposition of External Timestamps

- Some applications (notably OpenViBE) implement LSL's capability of imposing timestamps other than calls to `lsl_local_time()`.
- This makes calls to `time_correction()` incorrect:

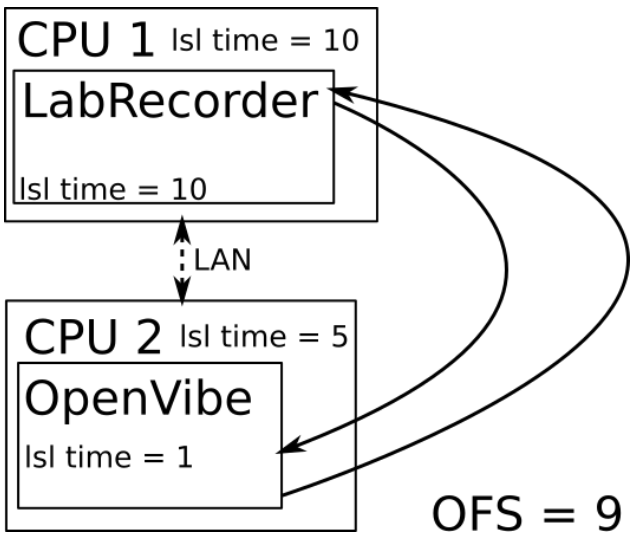
Imposition of External Timestamps

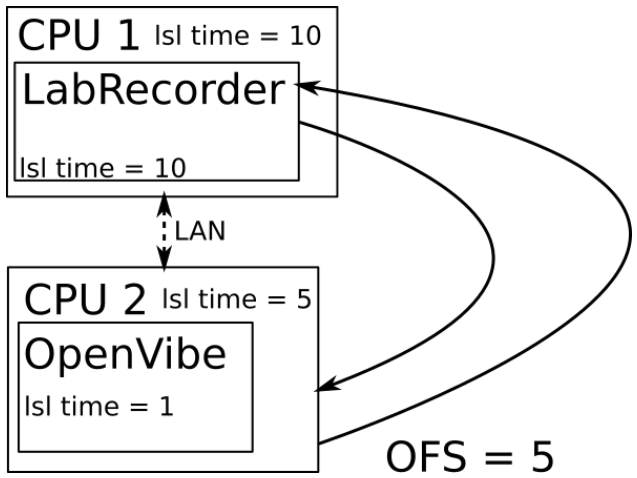
- Some applications (notably OpenViBE) implement LSL's capability of imposing timestamps other than calls to `lsl_local_time()`.
- This makes calls to `time_correction()` incorrect:



Imposition of External Timestamps

- Some applications (notably OpenViBE) implement LSL's capability of imposing timestamps other than calls to `lsl_local_time()`.
- This makes calls to `time_correction()` incorrect:





Problems with WLAN

- calls to `time_correction()` can overload the network and cause gaps in the data
- this can lead to all kinds of problems synchronizing offline; and, obviously this is not acceptable in an online application
- this is a danger in heavily taxed network scenarios (i.e. many streams) when these calls are frequent
- I don't have lot's of details about this issue, but I can report that in the case of a single 128 channel EEG amplifier operating at 256 Hz with a 'good router' this is not an issue
- when possible, go wired
- incidentally, a mobile hotspot can be used to transmit LSL messages in a pinch (which is really cool)—but this is not appropriate for sampled data!!!

Tweeks and Tips

Changing Sample Rate

- in an offline scenario `load_xdf()` must be called with `HandleJitterRemoval` set to `false`
- in an online scenario the setup of the inlet receiving the data must have the post processing flags for dejittering and monotinization turned off

C++:

```
inlet.set_postprocessing(0|1|8); // result = 10011
```

Python:

```
streams = resolve_stream('type', 'EEG')
inlet = StreamInlet(streams[0], processing_flags=0|1|8)
```

- LSL can be tweeked with a config file!
- this file must be named `lsl_api.cfg` and live in either `HOME/lsl_api` `/etc/lsl_api` or `ROOT`
- you can also set an environment variable: `LSLAPICFG`
- you can also place the cfg file in the same directory as the copy of liblsl loaded by the app and the changes will be local instead of global
- there are many options which you can read about in the source code (`https://github.com/sccn/liblsl/blob/81b0df6acbc6ca8c0fc1de5b1f91445ab912af7c/src/api_config.cpp`)

You can tell LSL to ignore external timestamps and always use its own:

```
[tuning]
ForceDefaultTimestamps = 1
```

...but beware, unless the cfg file is placed in the same directory as the copy of liblsl.dll loaded by the app, these settings are global!

Recommended tweeks for using WLAN courtesy of Matthew Grivich:

```
[tuning]
TimeProbeMaxRTT = 0.100
TimeProbeInterval = 0.010
TimeProbeCount = 10
TimeUpdateInterval = 0.25
MulticastMinRTT = .100
MulticastMaxRTT = 30
```

Important Caveat!

liblsl-Python/pylsl/examples/ReceiveDataInChunks.py:
https://github.com/labstreaminglayer/liblsl-Python/blob/d003cba234d0685931d42f8d1df0fd776cefc04f/pylsl/examples/ReceiveDataInChunks.py

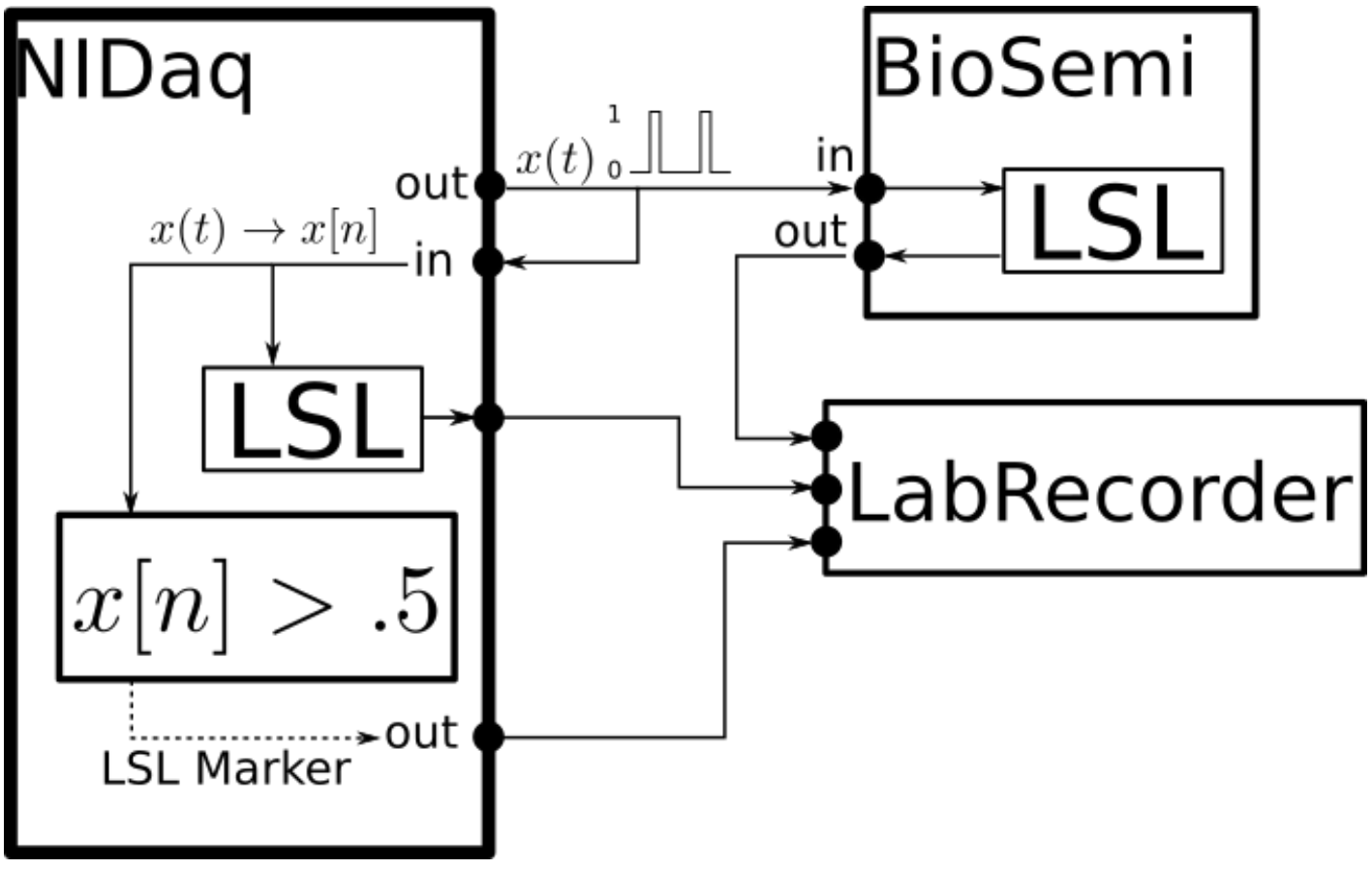
Windows Fun

- Windows does not trust an LSL stream—always disable all firewalls
- Windows 10 does this awesome thing where it spawns random virtual network adaptors
  - until LSL is multiple adaptor ready (?) LSL may want to stream through one of these virtual adaptors
  - you can delete them until are left with only physical adaptors

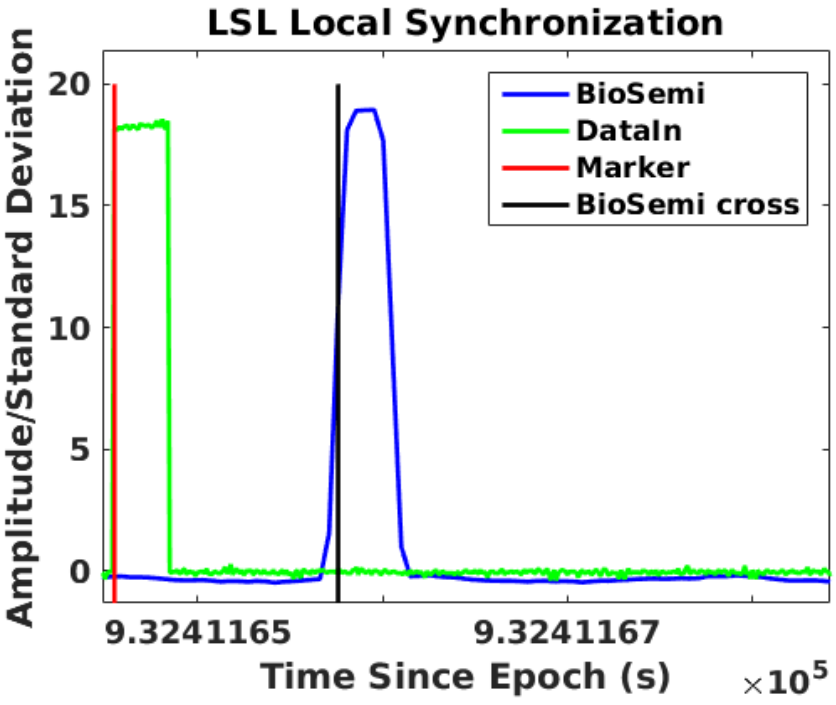
Measuring Things

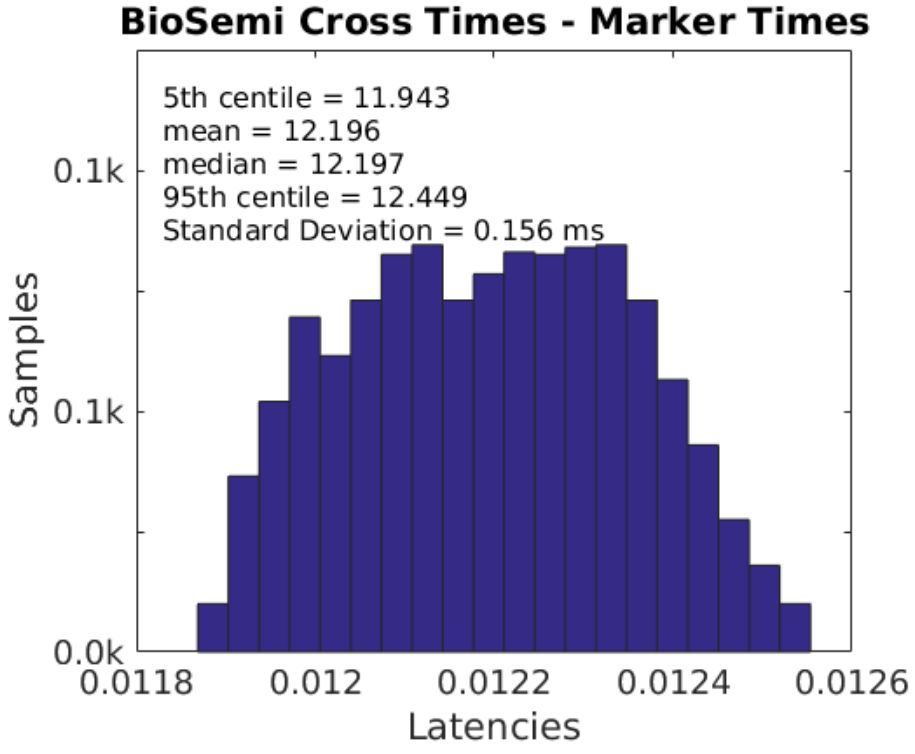
The Grivich Experiment:

The Grivich Experiment:

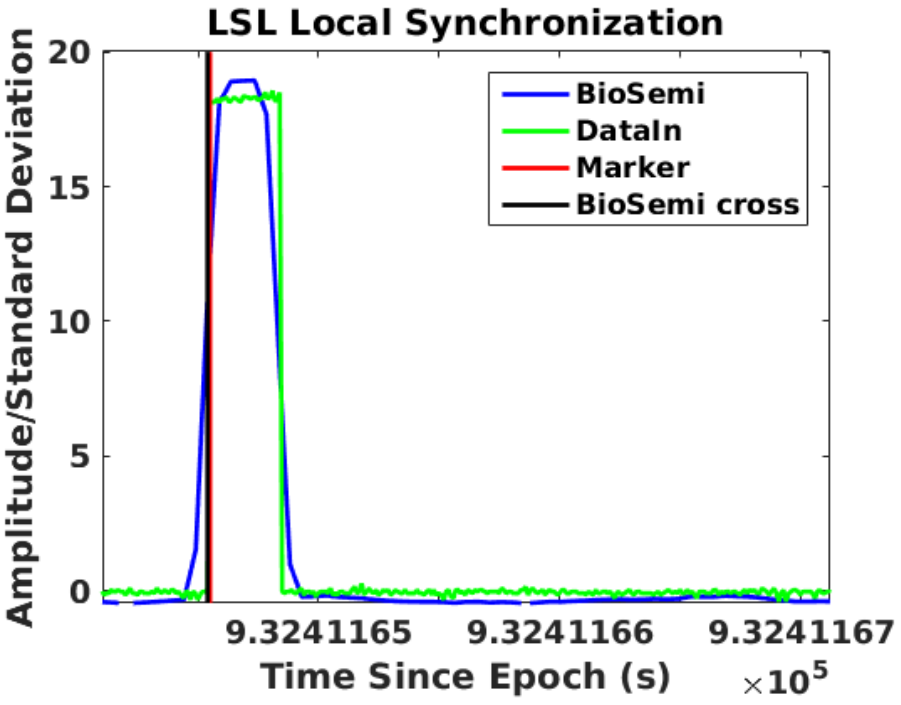


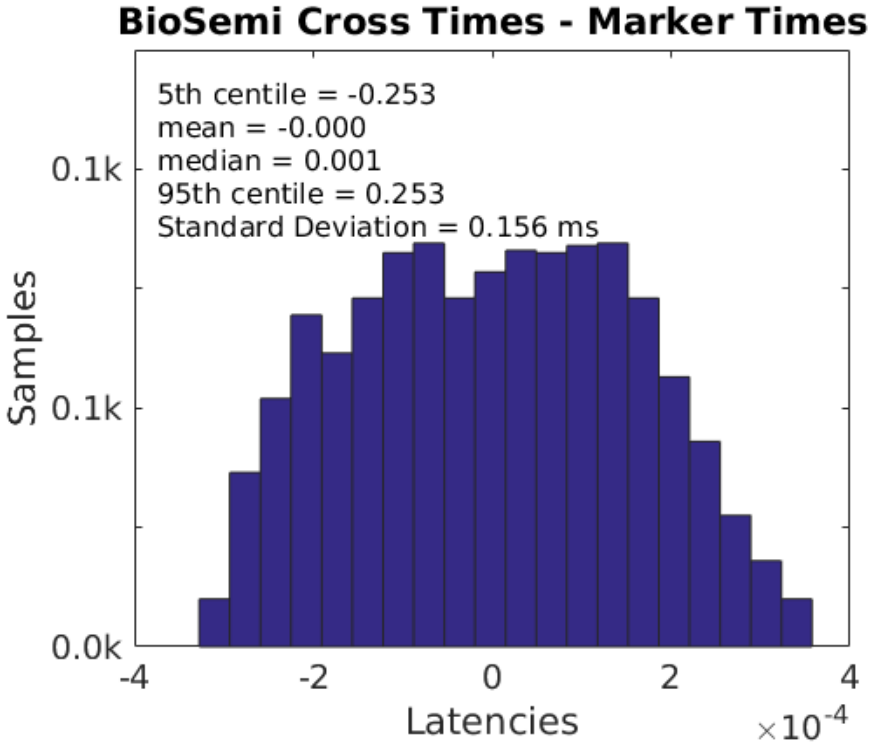
The Grivich Experiment:





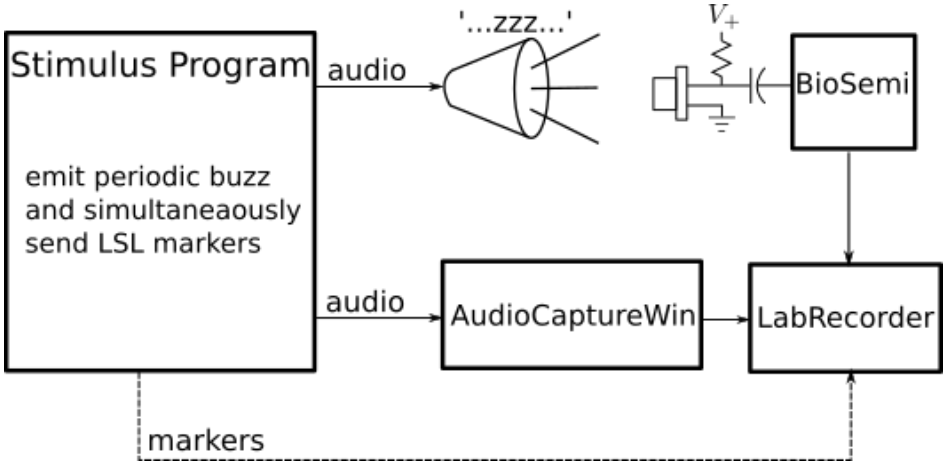
The Grivich Experiment:

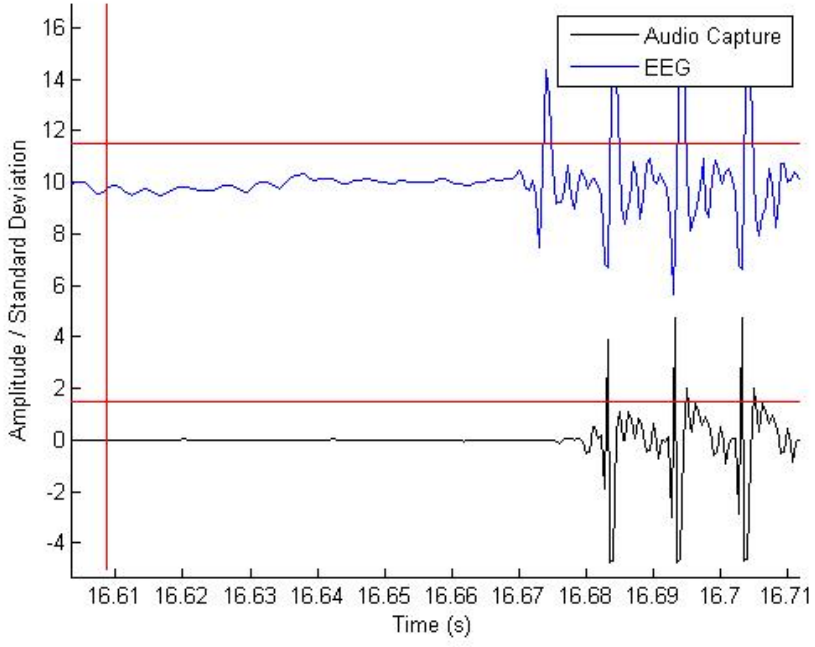




Testing Lag Times

An experiment to measure latency of audio stimuli in LSL:





https://www.neurobs.com/menu_presentation/menu_hardware/labstreamer

Thank You