LabStreamingLayer Overview

David Medine
dmedine@ucsd.edu
Swartz Center for Computational Neuroscience

EEGLab Workshop
November 20, 2016
0. Introduction

2. Why is LSL

3. A Brief Description of LSL
1. Introduction

What is LabStreamingLayer?

Lab Streaming Layer is an open-source software library that is highly extensible and supports wrappers for numerous languages. It is designed to support an overlay network for realtime data streaming, which facilitates multimodal data stream synchronization without hardware triggers.

![Lab Streaming Layer Overview Diagram]

- EEG Hardware (e.g., BioSemi, MINDO)
- Stimulus Presentation
- Misc Devices (e.g., PhaseSpace)
- Real-Time Viewers
- Recording Program
- Online Processing
- Markers
- Mocap
- EEG
- EEG+Other Control Signals

The diagram illustrates how different devices and processes interact within the Lab Streaming Layer system.
1. Introduction

What is LabStreamingLayer?

- open source software library
1. Introduction

What is LabStreamingLayer?

- open source software library
- highly extensible and supports wrappers for numerous languages
1. Introduction

What is LabStreamingLayer?

- open source software library
- highly extensible and supports wrappers for numerous languages
- overlay network
1. Introduction

**What is LabStreamingLayer?**

- open source software library
- highly extensible and supports wrappers for numerous languages
- overlay network
- realtime data streaming
1. Introduction

What is LabStreamingLayer?

- open source software library
- highly extensible and supports wrappers for numerous languages
- overlay network
- realtime data streaming
- facilitates multimodal data stream synchronization without hardware triggers
2. Why is LSL

Mobile Brain/Body Imaging (MoBI)
2. Why is LSL

Mobile Brain/Body Imaging (MoBI)

(a) Look

3-D Object orienting task

Walk & point

(b) Cortical EEG sources

point left

point right

(c) Independent sources

(d) look left

look right

(e) Neck EMG sources

point left

point right

(f) look left

look right
2. Why is LSL

**Low-Cost Mobile Brain/Body Imaging (MoBI) Platform**

- **ANY EEG system**
- **EyeTribe Eye Tracking ($99)**
- **Kinect Motion Capture**
- **Keyboard & Mouse**
- **LeapMotion Hand Gesture Recognition ($70)**

MoBI in a Box
2. Why is LSL

- Large Scale Experiments

5. Pattern Classification

6. Notifying In-App Feedback

4. Differential Activation

3. EEG Response

1. Pop-up event

2. Perceived / or not
2. Why is LSL

- EEG and ExG
- Full-Body Motion Capture
- Eye-Tracking
- Human Interface Devices, System State, Etc.
The Challenge: To Synchronize Multiple (Asynchronous) Streams
The Challenge: To Synchronize Multiple (Asynchronous) Streams

- EEG/EMG/EOG – i.e. with BioSemi – 512 Hz
2. Why is LSL

Synchronizing

The Challenge: To Synchronize Multiple (Asynchronous) Streams

- EEG/EMG/EOG – i.e. with BioSemi – 512 Hz
- Stimulus Presentation Markers – i.e. with Presentation – random times
The Challenge: To Synchronize Multiple (Asynchronous) Streams

- EEG/EMG/EOG – i.e. with BioSemi – 512 Hz
- stimulus presentation markers – i.e. with Presentation – random times
- Motion Capture – i.e. with PhaseSpace – 400 Hz

2. Why is LSL
The Challenge: To Synchronize Multiple (Asynchronous) Streams

- EEG/EMG/EOG – i.e. with BioSemi – 512 Hz
- Stimulus Presentation Markers – i.e. with Presentation – random times
- Motion Capture – i.e. with PhaseSpace – 400 Hz
- Eyetracking – i.e. with EyeLink – 60 Hz
2. Why is LSL

The Challenge: To Synchronize Multiple (Asynchronous) Streams

- EEG/EMG/EOG – i.e. with BioSemi – 512 Hz
- Stimulus Presentation Markers – i.e. with Presentation – random times
- Motion Capture – i.e. with PhaseSpace – 400 Hz
- Eyetracking – i.e. with EyeLink – 60 Hz
- HCI, GSR, Heart rate, Forceplate, etc. ?? Hz
2. Why is LSL

The Challenge: To Synchronize Multiple (Asynchronous) Streams
2. Why is LSL

**The Challenge: To Synchronize Multiple (Asynchronous) Streams**

- Attach timestamps to data
2. Why is LSL

**Synchronizing**

The Challenge: To Synchronize Multiple (Asynchronous) Streams

- Attach timestamps to data
- Attach timestamps to record times

Measure and test EVERYTHING OVER AND OVER
2. Why is LSL

The Challenge: To Synchronize Multiple (Asynchronous) Streams

- Attach timestamps to data
- Attach timestamps to record times
- Measure and test EVERYTHING
2. Why is LSL

The Challenge: To Synchronize Multiple (Asynchronous) Streams

- Attach timestamps to data
- Attach timestamps to record times
- Measure and test EVERYTHING OVER AND OVER
3. A Brief Description of LSL

LSL Network View

- **EEG Hardware (e.g., BioSemi, MINDO)**
- **Stimulus Presentation**
- **Misc Devices (e.g., PhaseSpace)**
- **Real-Time Viewers**
- **Recording Program**
- **Online Processing**

The diagram illustrates the flow of data from various sources to the Lab Streaming Layer (LSL), which then passes through different components and processes before reaching the final stages of visualization and data analysis.
liblsl

liblsl is an opensource C/C++ library and API for timestamping and streaming multimodal data across a network.

- Basically provides methods for creating 3 objects:
3. A Brief Description of LSL

liblsl

liblsl is an opensource C/C++ library and API for timestamping and streaming multimodal data across a network.

- Basically provides methods for creating 3 objects:
  - stream_info (metadata)
liblsl

liblsl is an opensource C/C++ library and API for timestamping and streaming multimodal data across a network.

- Basically provides methods for creating 3 objects:
  - `stream_info` (metadata)
  - `stream_outlet` (send data)
3. A Brief Description of LSL

liblsl is an opensource C/C++ library and API for timestamping and streaming multimodal data accross a network.

- Basically provides methods for creating 3 objects:
  - `stream_info` (metadata)
  - `stream_outlet` (send data)
  - `stream_inlet` (receive data)
3. A Brief Description of LSL

**stream_info constructor params**

Specifies the nature of an LSL stream (essentials)
3. A Brief Description of LSL

**stream_info constructor params**

Specifies the nature of an LSL stream (essentials)

- name (`'ActiChamp'`)
- type (`'EEG'`)
- channel count (`'64'`)
- sample rate (`'512'`)
- format (`'float32'`)
- source Id (`'8JIAes263D' some such serial number`)
3. A Brief Description of LSL

**stream_info constructor params**

Specifies the nature of an LSL stream (essentials)

- name (‘ActiChamp’)
- type (‘EEG’)
- channel count (‘64’)
- sample rate (‘512’)
- format (‘float32’)
- source Id (‘8JIAs263D’ some such serial number)

**append meta-data**

XML based information that adheres to a basic template, but can be extended to contain anything, anyhow
3. A Brief Description of LSL

**stream_outlet constructor params**

- (optionally) chunk size
- (optionally) data buffer size

Once created, a `stream_outlet` object sits on an open socket (a data stream publisher) and can be pinged by clients to initiate connection (subscription). A loop on a separate thread can stream data or poll for data (push sample) to output sporadically. Destroy after using...(show some code)
3. A Brief Description of LSL

Details – stream_outlet

stream_outlet constructor params

- stream_info object
3. A Brief Description of LSL

Details – `stream_outlet`

`stream_outlet` constructor params

- `stream_info` object
- (optionally) chunk size
3. A Brief Description of LSL

Details – stream_outlet

stream_outlet constructor params

- stream_info object
- (optionally) chunk size
- (optionally) data buffer size
3. A Brief Description of LSL

Details – `stream_outlet`

**stream_outlet constructor params**
- `stream_info` object
- (optionally) chunk size
- (optionally) data buffer size

**stream_outlet life-cycle**

Once created, a `stream_outlet` object sits on an open socket (a data stream publisher) and can be pinged by clients to initiate connection (subscription). A loop on a separate thread can stream data or poll for data (push sample) to output sporadically. Destroy after using... (show some code)
3. A Brief Description of LSL

**stream_outlet constructor params**

- stream_info object
- (optionally) chunk size
- (optionally) data buffer size

**stream_outlet life-cycle**

- once created, a stream_outlet object sits on an open socket (a data stream publisher) and can be pinged by clients to initiate connection (subscription)
**stream\_outlet\_constructor\_params**

- `stream\_info` object
- (optionally) chunk size
- (optionally) data buffer size

**stream\_outlet\_life\_cycle**

- once created, a `stream\_outlet` object sits on an open socket (a data stream publisher) and can be pinged by clients to initiate connection (subscription)
- a loop on a separate thread can stream data or poll for data (`push\_sample`) to output sporadically
3. A Brief Description of LSL

**stream_outlet constructor params**
- stream_info object
- (optionally) chunk size
- (optionally) data buffer size

**stream_outlet life-cycle**
- once created, a stream_outlet object sits on an open socket (a data stream publisher) and can be pinged by clients to initiate connection (subscription)
- a loop on a separate thread can stream data or poll for data (push_sample) to output sporadically
- destroy after using...(show some code)
3. A Brief Description of LSL

resolve_stream
3. A Brief Description of LSL

Details - stream_inlet

**resolve_stream**

- @param prop (‘name’, ‘type’, etc.)
- @param value (‘ActiChamp’, ‘EEG’, etc.)
- @return std::vector<stream_info>
3. A Brief Description of LSL

**resolve_stream**

- @param prop (‘name’, ‘type’, etc.)
- @param value (‘ActiChamp’, ‘EEG’, etc.)
- @return std::vector<stream_info>

**stream_inlet constructor**
3. A Brief Description of LSL

**resolve_stream**
- @param prop (‘name’, ‘type’, etc.)
- @param value (‘ActiChamp’, ‘EEG’, etc.)
- @return std::vector<stream_info>

**stream_inlet constructor**
- @param info (one of the stream_info objects returned by resolve_stream)
- (optional) @params buffer length, chunk length, recover (bool)
**3. A Brief Description of LSL**

**Details – stream_inlet**

**stream_inlet life-cycle**
3. A Brief Description of LSL

**stream_inlet life-cycle**
- automatically connect to corresponding outlet

Details – stream_inlet
3. A Brief Description of LSL

**stream_inlet life-cycle**

- automatically connect to corresponding outlet
- launch a ‘listen’ thread to repeatedly call pull_sample
3. A Brief Description of LSL

**Details - stream_inlet**

**stream_inlet life-cycle**

- automatically connect to corresponding outlet
- launch a ‘listen’ thread to repeatedly call `pull_sample`
- if `recover=true`, automatically respawn if lost
3. A Brief Description of LSL

**stream_inlet life-cycle**

- automatically connect to corresponding outlet
- launch a ‘listen’ thread to repeatedly call `pull_sample`
- if `recover=True`, automatically respawn if lost
- destroy when all done (show some code)
3. A Brief Description of LSL

31 Apps to interface LSL with various devices
31 Apps to interface LSL with various devices

- Link to LSL and vendor libraries (if any)
3. A Brief Description of LSL

31 Apps to interface LSL with various devices

- Link to LSL and vendor libraries (if any)
- Setup ‘streaminfo’ and metadata (xml)
31 Apps to interface LSL with various devices

- Link to LSL and vendor libraries (if any)
- Setup ‘streaminfo’ and metadata (xml)
- Open LSL ‘outlet’
31 Apps to interface LSL with various devices

- Link to LSL and vendor libraries (if any)
- Setup ‘streaminfo’ and metadata (xml)
- Open LSL ‘outlet’
- Launch listener thread to ‘push’ LSL data as it arrives
3. A Brief Description of LSL Overview – Apps
3. A Brief Description of LSL
3. A Brief Description of LSL

Overview – LabRecorder and load_xdf.m

Typical Setup:

```
Typical Setup:

<table>
<thead>
<tr>
<th>Mocap</th>
<th>LAN Con</th>
<th>GSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU 1  
timeofday = 46732.9874

LabRecorder

timeofday = 93746.2874

Mouse

Event Markers

CPU 2  
timeofday = 93746.2874
```
3. A Brief Description of LSL

**Overview – LabRecorder and load_xdf.m**

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**load_xdf.m**
- Organize metadata into a datastructure
- Linearize non-sporadic timestamps
- Use clock offsets and known latencies to synchronize data streams
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header

**load_xdf.m**
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file

**load_xdf.m**
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**Overview – LabRecorder and load_xdf.m**

**load_xdf.m**
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**load_xdf.m**

Overview – LabRecorder and load_xdf.m
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**load_xdf.m**
- Organize meta-data into a datastructure
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**load_xdf.m**
- Organize meta-data into a data-structure
- Linearize non-sporadic timestamps
3. A Brief Description of LSL

**LabRecorder**
- Locate streams, write metadata to output file header
- Read streams, write data to output file
- Periodically check clock offsets (NTP), write offsets to output file footer

**load_xdf.m**
- Organize meta-data into a datastructure
- Linearize non-sporadic timestamps
- Use clock offsets and known latencies to synchronize data streams
3. A Brief Description of LSL

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using the clock filter algorithm (NTP) using `gettimeofday`
- 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 (t3)`
  - `receive from outlet at inlet (t4)`
- Round trip time (RTT) = `(t3-t0) - (t2-t1)`
- Clock offset (OFS) = `((t1-t0) + (t2-t3))/2` (for lowest RTT)
3. A Brief Description of LSL

**Synchronization**

1: **Calculate Clock Offsets**

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` 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 (t₃)
  - Receive from outlet at inlet (t₄)
  
  Round trip time (RTT) = (t₃ - t₀) - (t₂ - t₁)
  
  Clock offset (OFS) = ((t₁ - t₀) + (t₂ - t₃))/2 (for lowest RTT)
3. A Brief Description of LSL

Synchronization

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` 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 (t3)
  - receive from outlet at inlet (t4)

- round trip time (RTT) = (t3-t0) - (t2-t1)
- clock offset (OFS) = ((t1-t0) + (t2-t3))/2 (for lowest RTT)
3. A Brief Description of LSL

Synchronization

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` 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 (t3)
  - receive from outlet at inlet (t4)

\[
\text{round trip time (RTT)} = (t_3 - t_0) - (t_2 - t_1)
\]

\[
\text{clock offset (OFS)} = \frac{(t_1 - t_0) + (t_2 - t_3)}{2} \quad \text{(for lowest RTT)}
\]
3. A Brief Description of LSL

**1: Calculate Clock Offsets**

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet \((t_0)\)
  - receive from inlet at outlet \((t_1)\)
  - immediately send from outlet to inlet \((t_3)\)
  - round trip time \((RTT) = (t_3 - t_0) - (t_2 - t_1)\)
  - clock offset \((OFS) = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}\) (for lowest RTT)
3. A Brief Description of LSL

Synchronization

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` 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 (t3)
  - receive from outlet at inlet (t4)

Round trip time (RTT) = (t3-t0) - (t2-t1)

Clock offset (OFS) = ((t1-t0) + (t2-t3))/2 (for lowest RTT)
3. A Brief Description of LSL

Synchronization

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet \( (t_0) \)
  - receive from inlet at outlet \( (t_1) \)
  - immediately send from outlet to inlet \( (t_3) \)
  - receive from outlet at inlet \( (t_4) \)

- round trip time (RTT) \( = (t_3-t_0) - (t_2-t_1) \)
3. A Brief Description of LSL

### Synchronization

1: Calculate Clock Offsets

- Record clock offsets periodically during data acquisition using clock filter algorithm (NTP) using `get_time_of_day` to record sets of 4 timestamps in rapid succession:
  - send from inlet to outlet \((t_0)\)
  - receive from inlet at outlet \((t_1)\)
  - immediately send from outlet to inlet \((t_3)\)
  - receive from outlet at inlet \((t_4)\)
- round trip time \((RTT) = (t_3-t_0) - (t_2-t_1)\)
- clock offset \((OFS) = ((t_1-t_0) + (t_2-t_3))/2\) (for lowest RTT)
2: Map drifting clock values and fit

This is normally done post-hoc in `xdf.m` using a fitting procedure. Each map is calculated using an ADMM method incorporating the Huber loss function (http://www.stanford.edu/~boyd/papers/distr_opt_stat_learning_admm.html). 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).

The latest version of LSL has methods for doing this online, but it is not yet validated.
3. A Brief Description of LSL

2: 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://www.stanford.edu/~boyd/papers/distr_opt_stat_learning_admm.html)

- 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).
2: 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://www.stanford.edu/~boyd/papers/distr_opt_stat_learning_admm.html)

- 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).

- The latest version of LSL has methods for doing this online, but it is not yet validated.
3. A Brief Description of LSL

3: Linearize/De-Jitter the timestamps (if appropriate)

- Simple linear regression is very robust:

![Graph showing linearization of timestamps](image)

- 200 percent jittery Data, SR = 512Hz
- De-Jittered, adj SR = 511.497
3. A Brief Description of LSL

Synchronization

3: Linearize/De-Jitter the timestamps (if appropriate)

- Simple linear regression is very robust:

This will fail if the sampling rate changes!
3. A Brief Description of LSL

This is bad:
3. A Brief Description of LSL

This is bad:
Red line is linearized timestamps, blue is raw. On the right is the difference: between +/-150s !!!.
3. A Brief Description of LSL

Determine device Lag:
3. A Brief Description of LSL

Determine device Lag:

![Figure 1: Synchronization Graph](image1)

- 5th centile = 1.916
- Mean = 2.545
- Median = 2.487
- 95th centile = 3.402
- Standard Deviation = 0.432 ms

![Figure 2: Histogram](image2)
3. A Brief Description of LSL

Determine device Lag:

To complete the synchronization, simply subtract the mean.
3. A Brief Description of LSL

Synchronization

(a) Look

3-D Object orienting task

Walk & point

(b) Cortical EEG sources

point left

point right

(c) Independent sources

(d) Look left

Look right

(e) Neck EMG sources

point left

point right

(f) Look left

Look right
Thank You!