The feedback dynamics of brain-computer interfaces in a distributed processing environment

Ayatelrahman Elsayed
New Mexico State University
aynasser@nmsu.edu

Chuck Creusere
New Mexico State University
ccreuser@nmsu.edu

Abstract

This paper describes a distributed paradigm for human brain-computer interfaces that can incorporate machine learning-directly stimulus feedback to the subject. Specifically, we use OpenBCI hardware and software to capture real-time EEG (Electroencephalography) waveforms from a subject on a host "client" computer and stream them to another "server" computer which could perform complex analyses on the waveforms prior to sending commands back to the OpenBCI interface directing alterations to the stimulus. In addition to describing the conceptual system framework, we present here the test results quantifying the closed-loop system latencies under various conditions. Quantifying latency in any feedback control loop (in this case, one that actually contains the human subject's brain) is vital since excess latency can destabilize a system.

1. Introduction

Creating hybrid systems that combine the strengths of the human mind with those of digital computers has long been a goal of the research community, with the fundamental limitation being the construction of efficient brain-computer interfaces (BCIs). To date, however, much of the focus on BCIs has been uni-directional: from the human to the machine [1], [2]. Recently, a few researchers have started investigating scenarios in which the perceptual feedback loop is "closed"—i.e., systems in which a perceptual stimulus is altered based on the real-time brain response [3]. Thus far, the scope of such "Brain-in-the-Loop" systems has been very narrow, with such systems as exist now being designed for specific, one-off experiments. Our goal here is to introduce the idea of a generic client-server-based, distributed Brain-in-the-Loop (BitL) paradigm and to study the vital issue of round-trip latency in such a system under various conditions.

In this paper, after first providing background information about the EEG-based brain analysis in Section 2, we introduce our prototype distributed BitL architecture in Section 3. In Section 4, we discuss our approach for evaluating the round-trip latency of the prototype system while the results are presented and interpreted in Section 5. Finally, we discuss our conclusions in Section 6.

2. Background

2.1. Human Brain

The brain is the most complex organ in the human body. The brain is what we are; all emotions, movements, thoughts, memories, and speech are created by electrical signals passing through trillions of differing shaped, sized, and functioned neurons. Over the centuries, numerous scientists have dedicated their time and effort to answer the fundamental question of how the brain works. The earliest brain documented reference dates to the 17th century BC with the Egyptian medical text called "The Edwin Smith Surgical Papyrus" [4]. Later in the 19th century, the first brain-imaging technique was discovered by Angelo Mosso called "human circulation balance" [5]. The basic idea was similar to that of functional magnetic resonance imaging (fMRI), widely in use today, in that it measured the redistribution of blood during emotional and intellectual activity. Thanks to modern brain imaging techniques, we now have a vast knowledge about the brain, including how the neurons connect throughout synapses.

2.2. Brain Data Acquisition

When large populations of neurons inside the brain fire synchronously, they generate signals having both an electrical and a magnetic nature. This phenomenon inspired scientists to create EEG, which measures the voltage differences across the scalp via electrodes. These flowing electrical currents induce magnetic fields, which can be captured by

Magnetoencephalography (MEG). Another existing brain imaging tool is functional magnetic resonance imaging (fMRI), which measures the amount of blood flowing to different portions of the brain and uses that as a proxy for the amount of neural activity present. A significant challenge in using EEG as a brain imaging tool, however, is the large variation in skull and scalp electrical conductivity from person to person which results in inconsistent results [6]. This inconsistency is a significant motivator of our desire to incorporate neurofeedback into the system since such feedback allows the system to adapt the stimulus for each subject over a wide range until a hypothesized response is elicited.

2.3. Neurofeedback

Neurofeedback (NF) is a type of biofeedback that uses real-time monitoring of brain activity to alter a stimulus being applied to the subject. The most common brain-sensing tool for NF applications is EEG due to its low cost and easy application. When used in a neurofeedback application, electrodes placed on the scalp measure the localized electrical brain activity and feed this information into an algorithm that alters the stimulus—a perceived audio or video signal or possibly even an electrical signal applied directly to the scalp. NF has been used, for example, as a peak-performance training tool to enhance cognition for healthy subjects [7]. In a 2014 study, theta-upregulation neurofeedback at EEG electrode Pz was shown to improve memory consolidation [8]. In another study from 2019, human performance in a demanding sensory-motor task was improved via online neurofeedback that focused on the regulation of arousal [9]. NF has also been widely used as a therapeutic tool to control the symptoms of patients by observing their deviating brain activity [10] or by boosting motor imagery practice for stroke recovery patients [11]. Particularly relevant in the current context is a method of neurofeedback known as Brain State-Dependent Brain Stimulation (BSDBS) which uses EEG along with real-time signal analysis to adjust brain stimulation in a specified manner [12].

2.4. Utility of Closed Loop BSDBS

Open-loop BSDBS monitors and analyzes the current brain state but directs the neural feedback without consideration of the current brain state [13]. For example, specific EEG amplitude or phase oscillations might be used to activate a Transcranial Magnetic Stimulation (TMS) device in a preprogrammed manner, attempting to estimating the state-specific corticospinal excitability [14]. Conversely, in a closed-loop BSDBS system, the current brain state is used to alter the stimuli to drive the brain into a desired state. Thus, a closed-loop BSDBS system aims to self-regulate brain activity using neurofeedback to guide the application of brain stimulation. For example, a biomedical engineering team at Columbia University in 2019 used electroencephalography-based neurofeedback to shift an individual's arousal so that their task performance increases significantly [9]. This work demonstrates a closed-loop brain-computer interface for dynamically tuning arousal to affect online task performance, following the Yerkes and Dodson law. The prototype BitL system described below would, amongst other things, facilitate the implementation of closed-loop BSDBS systems in a distributed manner, allowing more sophisticated machine learning algorithms to be used to more optimally determine how brain stimulation should be adjusted.

2.5. Anatomy of a Closed-Loop Neurofeedback System

All closed-loop neurofeedback systems have five fundamental components:

- **Acquisition of the brain signals**: A brain sensing technique must be used: EEG, MEG, or fMRI. Each technology has pros and cons regarding its use in a neurofeedback system (e.g., temporal versus spatial resolution).
- **Online data pre-processing**: The detection and removal of artifacts (e.g., eye movements); D.C. drift compensation.
- **Feature extraction**: Features are extracted that more compactly represent information useful for analysis and input to machine learning algorithms. Potential features of interest include band power (i.e., delta, theta, alpha, beta, and gamma), Fourier spectrum [15], and wavelet coefficients [16].
- **Generation of the feedback signal**: The extracted feature is mapped to a sensory stimulus by a machine learning algorithm. Some of the more sophisticated techniques use complex deep learning approaches. For example, support vector machines and convolution neural networks have been used for real-time classification in EEG-based emotion recognition systems [17] [18].
- **Online adaptation**: Suppose that the machine learning algorithm that generates the feedback stimulus can alter its behavior based on the

inputs that it is receiving (i.e., the features). In that case, we can say that this learning algorithm is online adaptive. While numerous online adaptive machine learning algorithms have been developed, those based on the concept of 'reinforcement learning' have been applied to EEG waveforms in the past with good results [19] [20].

3. Description of Prototype System

3.1. System Overview

The front end of the proposed system consists of a host computer (i.e., the client) running the OpenBCI GUI that is wirelessly communicating via Bluetooth LE to a Ganglion board which is, itself, physically attached to four electrodes and one ground wire. The GUI accepts the real-time EEG waveforms from the Ganglion board and computes various derivative products, including band powers and short-time Fourier spectra. Software that we have added to the OpenBCI GUI (as a Widget) can stream any of this information (raw EEG, band power, spectrum, etc.) using Lab Streaming Layer (LSL) TCP/IP-based communication protocol to another device. As configured in the current prototype system, we chose to stream the five band power readings from each of the four electrodes. At the prototype system's back end, a server running Matlab receives and processes the transmitted real-time data stream (in our prototype, the band powers for each electrode) from the client. Matlab then sends a corresponding feedback signal back to the front-end client, telling it how to alter the stimulus presented to the subject (see Figure 1).

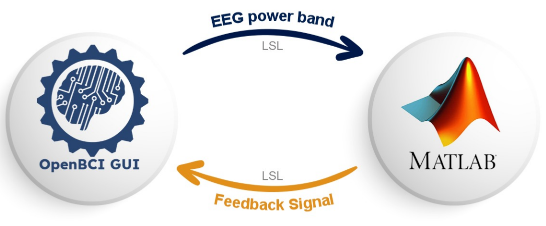


Figure 1: Closed-loop system diagram.

3.2. Hardware Architecture

We evaluated our distributed BitL prototype concept using various devices and operating systems to verify its cross-platform capabilities. Specifically, we considered two Linux-based laptops as client and server, two Windows laptops, and a Windows client laptop with a CentOS 8-based Linux server. It worked successfully on them all. In the interest of brevity, however, we will only present here the results for two of the configurations: two Windows-based laptops as illustrated in Figure 2 and a Windows-based laptop communicating with a Linux server as shown in Figure 3. Figure 2 shows the laptop to laptop hardware configuration where the two laptops are connected with LAN cables to a private network provided by a Belkin AC 1200 DB Gigabit router. OpenBCI four-channel headband is connected to the ganglion board, communicating with the front-end client laptop using Bluetooth. A Windows 10 HP laptop with an Intel core i5-1035G1 CPU running at 1.00GHz and having 8 GB RAM serves as the front-end device. The second connected laptop (the back-end device) is a Windows 10 Dell laptop with an Intel Core i7-7500U CPU running at 2.70GHz and having 8 GB RAM.

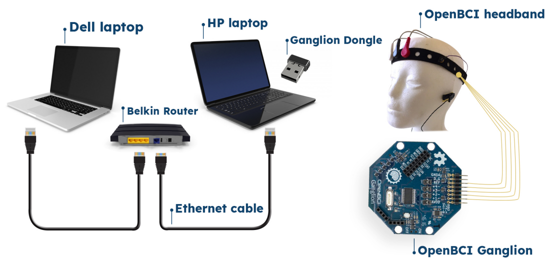


Figure 2: Hardware setup between the two laptops.

Figure 3 shows the client laptop and Linux server connecting over a public Ethernet service. Here, the back end computer used is a Centos 8-based Polywell server with an Intel Core i7-5820K CPU running at 3.30GHz and having 64 GB RAM. The Ethernet router is part of the public NMSU network.

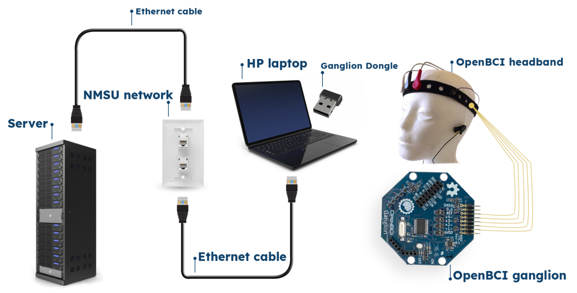


Figure 3: Hardware setup between the laptop and the server.

OpenBCI Board: The OpenBCI Ganglion is a low-cost but high-quality bio-sensing device. It has four channels which it samples at a rate of 200 Hz, and is compatible with EMG, EEG, and Electrocardiography (ECG) electrodes. The Ganglion relies on a standard Bluetooth 4.n (BLE) connection; therefore, it is compatible with any BLE device. For a computer to communicate with the Ganglion board, however, a ganglion dongle must be used which would normally limit the sampling rate to 100 Hz. Fortunately, the use of a delta compression protocol by the OpenBCI software interface brings the rate back up to 200 Hz. In addition to the four scalp electrodes, the Ganglion board also connects to two ear clips which provide the ground reference. To facilitate communication between the Ganglion board and the OpenBCI GUI, the OpenBCI 'Electron Hub' is used. This hub is a TCP/IP server listening on port 109996 at the clients' loopback address (127.0.01).

3.3. Software Architecture

3.3.1. Overview

The two computers, one at the back-end and one at the front-end, communicate together using Lab streaming layer (LSL). The software system relies on five stack layers as illustrated in Figure 4 [21]. OpenBCI GUI acts as the front-end application at the application layer, while Matlab represents the back-end. Both applications request and send data to one another in addition to performing other processing tasks. The next lower layer is the transport layer which creates the TCP or UDP header, and below it is the Network layer where the source and destination IP addresses are added to generate a packet. This packet is then sent to the Data Link layer, where the MAC address information is added to create a frame which is finally sent to the Physical layer which transmits the actual message bits.

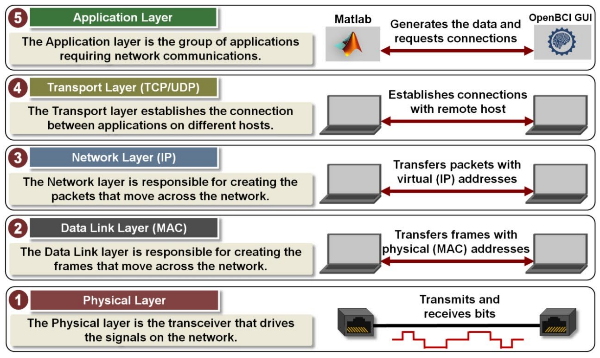

5
Application Layer
The Application layer is the group of applications requiring network communications.
Matlab
Generates the data and requests connections
OpenBCI GUI
4
Transport Layer (TCP/UDP)
The Transport layer establishes the connection between applications on different hosts.
Establishes connections with remote host
3
Network Layer (IP)
The Network layer is responsible for creating the packets that move across the network.
Transfers packets with virtual (IP) addresses
2
Data Link Layer (MAC)
The Data Link layer is responsible for creating the frames that move across the network.
Transfers frames with physical (MAC) addresses
1
Physical Layer
The Physical layer is the transceiver that drives the signals on the network.
Transmits and receives bits

Figure 4: Software stack setup.

3.3.2. OpenBCI GUI

OpenBCI GUI is a powerful open-source tool for visualizing, streaming, or even recording bio-signal data (EEG in our case) from an OpenBCI hardware device. In particular, it can stream real-time data to compatible third-party software (e.g., Matlab, python, Neuromore, OpenViBE, BrainBay, and BioEra). It can be used as a standalone application or as a sketch in the Java "Processing" integrated development environment (IDE). The Processing IDE is a powerful open-source tool for visualization and software prototyping used by research labs of well-known companies like Google and Intel for prototyping new interfaces and services. OpenBCI GUI supports 'widgets,' which allow new capabilities to be easily added to it. OpenBCI GUI already has many widgets that are part of the standard distribution, including the time series, band power, FFT, and networking widgets. This latter widget supports data streaming to other Apps or devices using one of four possible protocols: Serial, UDP, Open Sound Control (OSC), or Lab Streaming Layer (LSL). We chose here to use LSL over the other available network protocols. The serial protocol requires a dedicated hardwired connection. It was thus not a viable choice for future cloud-based implementations while OSC was not a candidate because it does not support real-time communication. As to the tradeoff between LSL and UDP, we chose LSL because it supports both TCP and UDP protocols and is thus more flexible than just using UDP by itself.

3.3.3. Lab Streaming Layer

The Lab Streaming Layer is an open-source network protocol for synchronized data streaming, supporting sampling rates from 44100 HZ for audio to 24 frames/second for video. The streaming Layer API, the **liblsl** library, provides two data representations for the client to stream the data where a "sample" within the OpenBCI architecture contains the measured voltages of all channels at a given time instant. A "chunk" represents multiple samples and is used to improved throughput. Every successful connection starts with the "Outlet", the stream source, advertising itself on the network. The streaming Outlet can use IPv4, IPv6, or both IP stacks in parallel, and it also uses UDP for broadcasting on port 16571, creating multiple sockets. Each socket listens on a different multicast address in the preferred range.

In parallel, the Outlet creates a TCP server socket in the port range of 16572-16604 to handle data transmission requests. The transmission requests can be for data streaming when the inlet pulls samples or for meta-data, where the inlet retrieves the meta-data describing a stream. On the receiver side, a stream resolver is created to resolve the transmitted stream. For synchronization, a built-in clock provides time stamping for the transmitted stream, achieving sub-millisecond accuracy. The built-in clock is based on Network Time Protocol (NTP).

3.4. Simultaneous Communication

3.4.1. Modifying the Network Widget Since the OpenBCI GUI networking widget does not support incoming data streams, we first needed to modify it to create our prototype BitL system. Multiple changes in multiple source files are required to accomplish this task. The modifications that we made are summarized as follows:

In W_Networking.pde file

- Adding LSL stream inlet and stream Info as LSL objects.
- Creating a receiving function called “ReceiveData(),” where an LSL stream inlet is opened, and inlet samples are pulled. This function receives any data in the inlet identified by the stream name.
- A “Receive” option is added to the widget’s dropdown menu.

To add “Receive” to the dropdown menu, the following changes are required.

In SoftwareSettings.pde file

- “Receive” is added in the nwDataTypesArray array of strings.
- ‘Private final String kJSONKeyReceive = “Receive”’ is added to the file.

To alter the stimulus in another widget, a global variable must be created *In DataProcessings.pde file*, and that variable name must be used in both the network widget and the widget altering the stimuli: e.g.,

- A global Integer variable is created with a default value of zero (Delta).
- Received_Data variable is assigned in ReceiveData() function with the value of feedback signal from Matlab corresponding to maximum power band.
- Casting received “float” number from Matlab to “integer” variable which is then assigned to Received_Data Variable.

3.4.2. Back-End Compute Server We wrote a Matlab script for simultaneously sending and receiving real-time data that uses the lab streaming layer library functions on the server end. A sending stream is created with two channels, a sampling rate of 100Hz, and a source id of ‘sdfwerr32432’. For receiving, we designed a resolver to receive the OpenBCI GUI sending stream, identified by the name of ‘obci_eeg2’. The received data stream is pulled in by Matlab in chunks, increasing the throughput.

A problem arises, however, that can cause the sending and receiving processes in the OpenBCI GUI not to operate simultaneously. The size of the data matrix received by the Matlab from the OpenBCI GUI varies from 4x0 up to 4x32 (4 elements with either 0 or 32 bits each). If no data is received (i.e., matrix of size 4x0), Matlab will not transmit any data back to the OpenBCI GUI, and this will cause the OpenBCI GUI to freeze as it is stuck trying to pull in the expected Matlab frame. When this occurs, no data will be transmitted by the GUI. Since Matlab is also waiting for data (from the OpenBCI GUI), it also gets locked up, and thus the entire system becomes paralyzed. To overcome this problem, we have forced Matlab to push a dummy data “111,111” to its Outlet when it receives a matrix of size 4x0 from the GUI.

4. Experimental Setup: Closed-Loop Delay Calculations

The time difference between the OpenBCI GUI client sending a message to the back-end Matlab server and it receiving back a reply is a critical parameter for any feedback ‘control’ system as it affects the stability of such systems. The timing diagram illustrated in Figure 5 demonstrates the process of calculating the round-trip loop delay (L_D), which is the difference in time between the GUI’s receipt of the reply (t_{RG}) and its prior transmission of the original message (t_{TG}). To calculate this, several steps are required. First, the OpenBCI GUI client sends a flag of “-1” for each of the five sending frames. At the back-end, the Matlab server sends back one of three different frames, depending on what it received from the client: 1) if it received nothing, it sends a frame of [111,111]; 2) if it received the -1 flag frames, it sends a flagged frame of [-1,-1]; otherwise, it sends a [888,999] frame. OpenBCI GUI records the time it transmitted its flag frame, t_{TG}, and the time it received the reply from the Matlab server, t_{RG}. The GUI then stores these times in a text file along with the EEG band power and frame rate information. A mandatory pause in the Matlab program running on

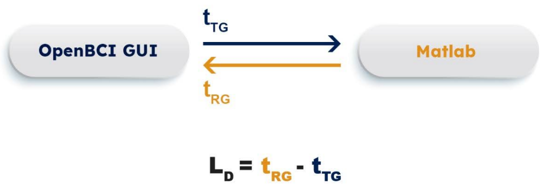


Figure 5: Closed-loop system timing diagram, where L_D is the round-trip loop delay.

the server must be inserted after each transmission to prevent system failure. How the length of this pause affects the system's performance is quantified in the results section below. A python script extracts the flag transmission and reception times from the stored file and calculates the statistics presented below.

5. Results

5.1. Impact of Server Processing Delays on the Round-Trip Delay

We stated in the previous section that we needed to insert a 'pause' into our Matlab server program for the closed-loop system to function correctly. We will discuss the reasons for this requirement in the next section after reviewing the results here. It is important to note that the inserted 'pause' is equivalent to the processing delay that a complex machine learning program might incur as it analyzes the incoming EEG stream and formulates the optimal reply to the OpenBCI GUI client. Thus, the potential utility of these results goes well beyond the simplistic framework used here to calculate them.

For visualizing the loop delay in relation to the induced Matlab delay, we use the boxplot tool. A boxplot is one of the best statistical tools for visualization since it summarizes five vital statistics (i.e., maximum, minimum, median, Q1, Q2) in one plot, as shown in Figure 6 [22].

Considering first the hardware setup for Dell-Hp private router configuration of Figure 2, Figure 7a shows box-plots for 20 , 30 and 31 milliseconds of Matlab-induced delay. As shown, the loop delay is very high, ranging from 99 up to 6717 milliseconds. The main common theme for these three cases is that the number of sent flags differs dramatically from the number received. The difference between the number of received flags and sent flags decreases as the pause increases, reaching a difference of 11 flags in the case of a pause of 31 ms. This difference in flags sent versus flags received vanishes when the Matlab pause

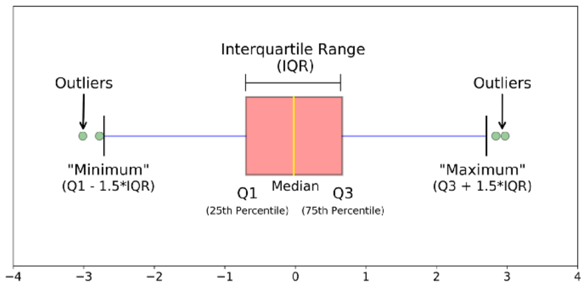


Figure 6: Box-plot illustration.

is between 32 and 200 ms. Figure 7b shows a boxplot of the round-trip loop delay with Matlab pauses of between 32 and 50 ms, and we see that, as expected, the median of the loop delay increases as the pause length increases. Figure 7c shows the boxplot of the loop delays for Matlab pauses of 100 and 200 ms, and we see that the median of loop delay is almost equal to the pause lengths in both cases. Obviously, in the case of longer Matlab pause lengths, the length of this pause dominates the round-trip loop delay. Studying these figures, we note that for Matlab pauses of greater than 32 ms, the increase in the median loop delay as the pause length increases is fairly linear. It is recommended in any statistical study, however, to display results both with and without the outliers. With this in mind, Table 1 shows the mean, the standard deviation (SD), and the 95% confidence intervals (CI) of the loop delay, both with and without outliers. In both cases, the round-trip loop delays are highly linear over most of the range of Matlab pauses, but the results with outliers removed generally have narrower confidence intervals.

These results also show that for the outlier-free case, the difference between the mean loop delays (MLD) and Matlab pause times (MP) is consistently around two milliseconds with a maximum of 2.6 ms and a minimum of 1.1 ms, as shown in Table 1. This difference is explained by the execution time of the Matlab code, which is around 2.56 ms. Thus, the vast majority of the round-trip loop delay is caused by the Matlab code: either intentionally introduced or due to its processing latency. We note, however, that this inherent processing latency is not a fundamental system limitation because it is an order of magnitude smaller than the minimum 32 ms delay that we were required to introduce for the system to operate correctly.

With respect to the Linux-HP/NMSU-network configuration of Figure 3, Figure 8a illustrates the boxplot of round-trip delay corresponding to Matlab pauses of 20, 30, 31, 32, 33 and 35 ms. We note that the average round-trip delays are considerably increased, as

Table 1: Loop delay results' statistics for Dell-HP router setup in Figure 2.

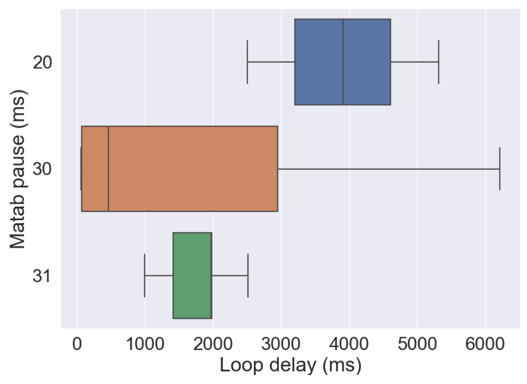
	Loop delay with outliers			Loop delay without outliers			
Matlab pause	**Mean**	**SD**	**CI**	**Mean**	**SD**	**CI**	**Outliers%**
20 ms	3905.0	1981.3	[-13896.4, 21706.4]	3905.0	1981.3	[-13896.4, 21706.4]	0.0 %
30 ms	1710.5	2108.3	[840.2, 2580.7]	1710.5	2108.3	[840.2, 2580.7]	0.0 %
31 ms	1845.6	412.6	[1717.0, 1974.2]	1845.6	412.6	[1717.0, 1974.2]	0.0 %
32 ms	42.5	43.8	[30.1, 54.8]	34.6	0.9	[34.4, 34.9]	9.8 %
33 ms	38.1	13.2	[33.2, 43.1]	34.9	1.3	[34.3, 35.4]	6.7 %
35 ms	40.3	11.7	[36.2, 44.4]	37.6	1.7	[37.0, 38.2]	5.9 %
40 ms	43.2	10.3	[39.9, 46.6]	41.1	1.1	[40.8, 41.5]	7.9 %
43 ms	45.6	4.0	[44.4, 46.8]	45.3	1.5	[44.8, 45.8]	4.4 %
45 ms	48.1	3.5	[47.0, 49.3]	47.5	1.4	[47.0, 47.9]	4.5 %
47 ms	50.9	9.1	[47.7, 54.0]	49.3	1.3	[48.8, 49.8]	2.9 %
50 ms	55.4	12.1	[48.7, 62.1]	52.5	0.8	[52.0, 52.9]	13.3 %
100 ms	101.8	2.7	[100.7, 102.8]	102.2	1.2	[101.7, 102.7]	14.3 %
200 ms	202.1	2.0	[200.3, 203.9]	201.5	1.0	[200.4, 202.6]	3.4 %

Table 2: Loop delay results' statistics of Server-HP NMSU-network setup in Figure 3.

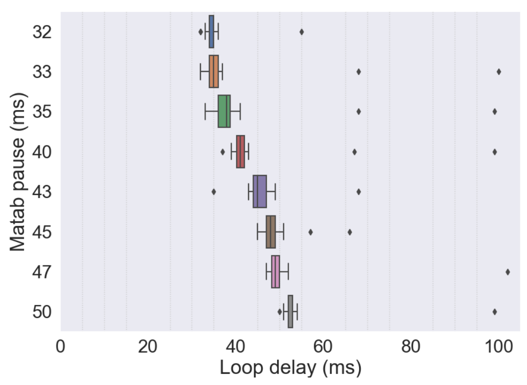
	Loop delay with outliers			Loop delay without outliers			
Matlab pause	**Mean**	**SD**	**CI**	**Mean**	**SD**	**CI**	**Outliers%**
20 ms	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30 ms	10874.5	6952.9	[-189.1, 21938.1]	10874.5	6952.9	[-189.1, 21938.1]	0.0 %
31 ms	4117.0	2511.6	[-18449.2, 26683.2]	4117.0	2511.6	[-18449.2, 26683.2]	0.0 %
32 ms	7617.0	4181.9	[4625.4, 10608.6]	7617.0	4181.9	[4625.4, 10608.6]	0.0 %
33 ms	2899.6	631.4	[2659.5, 3139.8]	2786.6	171.2	[2720.2, 2853.0]	3.4 %
35 ms	9193.7	12074.5	[5762.2, 12625.2]	4848.3	188.4	[4789.6, 4907.0]	16.0 %
40 ms	356.5	286.4	[266.1, 446.9]	356.5	286.4	[266.1, 446.9]	0.0 %
43 ms	359.6	370.3	[244.2, 475.0]	359.6	370.3	[244.2, 475.0]	0.0 %
45 ms	263.3	384.3	[175.5, 351.1]	135.3	201.1	[85.8, 184.7]	13.2 %
47 ms	143.3	248.2	[95.9, 190.6]	47.4	0.9	[47.2, 47.5]	22.2 %
50 ms	642.1	617.3	[449.7, 834.4]	642.1	617.3	[449.7, 834.4]	0.0 %
100 ms	259.9	337.8	[148.9, 370.9]	100.4	1.1	[100.0, 100.8]	23.7 %
200 ms	309.5	279.2	[217.8, 401.3]	200.1	1.1	[199.7, 200.5]	21.1 %

is also the statistical spread for these delays. That is to be expected given that this is a high-traffic public network. We also note that the reliability of the communications is adversely affected as the number of return flags received by OpenBCI GUI is far fewer than the number it sent. In this scenario, we find that a Matlab pause of 20 ms does not provide sufficient time for even one of the 128 flags sent from Matlab to be received by the front-end GUI program. Increasing the Matlab pause to 40, 43, 45, 47, and 50 ms still does not provide sufficient time for the network to transmit all of the sent flags to the GUI, causing the boxplots of the loop delay to be very spread out and have large averages as shown in Figure 8a. Unsurprisingly, the results are not consistent: they change quite a bit each time we run the system which is expected as the network congestion varies continuously. Boxplots for the cases with larger delays of 100 and 200 ms, as shown in Figure 8c, *do* illustrate the consistent linear relationship between the loop delay and the Matlab pause that we previously noted for the private network configuration. This can be explained by the fact that a Matlab pause of 100 ms or longer is long enough for the network to deliver the Matlab flag acknowledging that it received the 'GUI sent' flag before it then sends yet another flag. Table 2 provides the mean, the standard deviation (SD), and the 95% confidence interval (CI) of the loop delay along with the percentage of outliers for each test case.

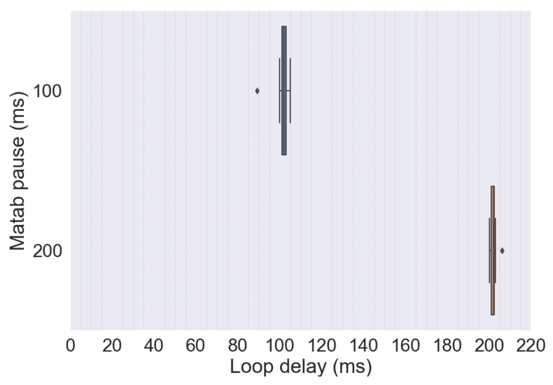
The table shows that the network lag affects all of these results, leading to dramatic increases in the round-trip loop delays. The results are highly inconsistent, however, although outlier removal does



(a) Box-plot for very small delay.

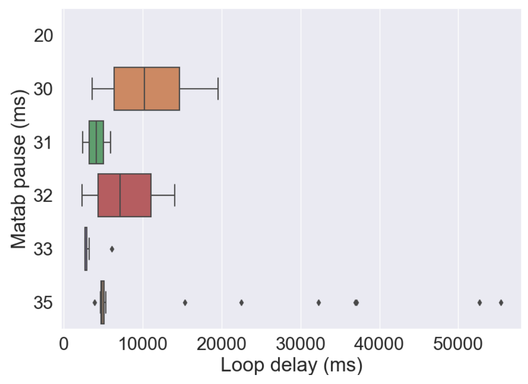


(b) Box-plot for medium delay.

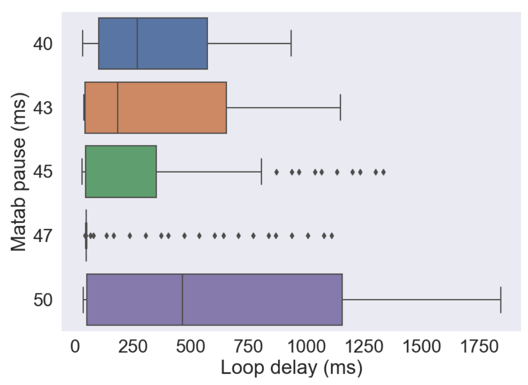


(c) Box-plot for large delay

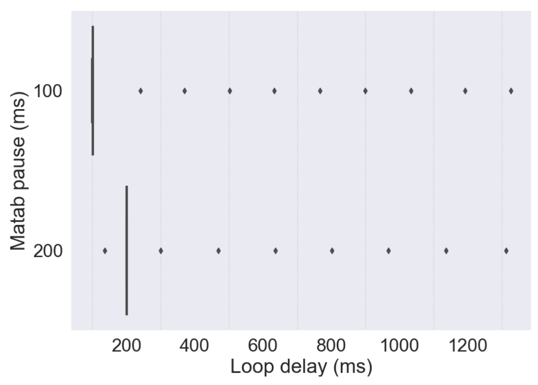
Figure 7: Box-plot for loop delay of Dell-Hp router setup in figure 2.



(a) Box-plot for very small pause.



(b) Box-plot for medium pause.



(c) Box-plot for large pause.

Figure 8: Box-plot for loop delay of server-Hp NMSU-network setup in figure 3.

help in certain cases. Consistent with the box plots, it is not until the Matlab-induced delay exceeds 100 ms that the standard deviation consistently achieves small values. Even that requires outlier removal due to the large network delay fluctuations. The case where the Matlab delay is 46 ms appears to be the one exception to the rule. Still, it is unclear that we can reliably achieve good performance with this short delay under a broader range of possible network conditions.

Interestingly, the differences between the mean loop delay and the Matlab pause for the 100 ms and 200 ms cases are 0.4 ms and 0.1 ms, respectively. These are both significantly lower than the 1.1 ms minimum difference of Dell-HP private router setup. We believe this difference can be explained by the superior response time of the Polywell server relative to that of the Dell laptop. This belief is born out by observing that the Matlab-based back-end server software executes in 0.3 ms on the Polywell server compared with 2.56 ms on the Dell laptop server.

5.2. Interpretation of the Results

One question that we are particularly interested in answering is "why does the system need at least a pause of 32 milliseconds to be stable and work appropriately?" To answer this question, an unmodified OpenBCI GUI instantiation that only sends data was configured to log its sending times in a text file. A python script then analyzed this file to calculate the difference between pairs of consecutive transmitted data blocks (the sending delay) as well as the usual statistics related to this difference. As illustrated in Table 3, the mean of sending delay is 33.5 ms with outliers removed (39.7 ms with them included). Taking a more detailed look at the statistics, we estimated the probability mass function (PMF) of sending delay as shown in Figure 9. We see from this plot that sending delays of 33 ms and 34 ms have the highest probabilities (0.22 each) and, in fact, that the probability that the sending delay is greater than 34 ms is very low: less than 0.08. Thus, the sending delay between successive frames streamed from the OpenBCI GUI client to the Matlab server is greater than 34 ms less than 8% of the time. It is certainly not a coincidence that the length of the required Matlab pause (i.e., at least 32 ms) falls into this same time range. It would appear that if the Matlab server responses to the most recently transmitted OpenBCI GUI data frame (containing only flag bits in this experiment) is received by the OpenBCI GUI client *before* it sends its next data frame, reception errors occur, and the stability of the feedback loop is negatively impacted. While our experiments indicate that a 32 ms delay appears to be sufficient, the PMF of the sending delays shown in Figure 9 would lead one to believe that a more conservative delay of 34 ms–or even 35 ms–might be safer.

Table 3: Sending delay's statistics with and without the outliers.

Sending delay	Mean	SD	CI
With Outliers	39.7	13.1	[37.9,41.5]
Without Outliers	33.5	1.0	[33.3,33.6]

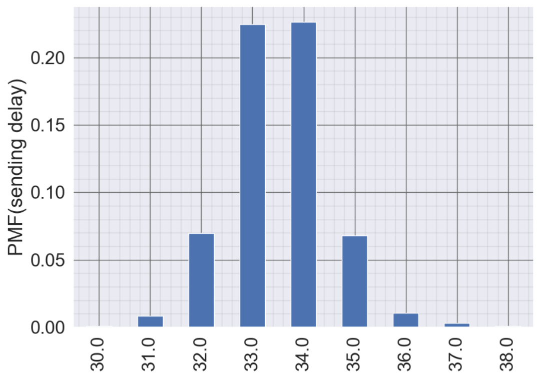


Figure 9: Probability mass function of the sending delay.

6. Conclusion

In this paper, we created and analyzed a distributed framework interfacing the human brain to a networked system of computers using OpenBCI, a low-cost and open-source EEG acquisition platform. The proposed closed-loop system places the participant's brain inside a feedback loop that extends from a client computer hosting the EEG acquisition system through the internet to a remote server which analyzes the real-time EEG stream and feeds control commands back to the client computer. These control commands can be used by the software on the client computer to alter a stimulus or other parameters of a trial. We have also analyzed the prototype system that we developed to better understand the limitations and constraints of the proposed framework.

In particular, we have found that that the back-end server must wait at least 32-34 ms before it sends its reply message to the front-end OpenBCI GUI client in order for the system to maintain its closed-loop stability. One interesting implication of this requirement is that it means that any machine learning algorithm analyzing the real-time EEG data stream has at least 32 ms to do its processing and generate its feedback message without impacting the closed-loop system performance in any way. We have also observed a linear relationship between the loop delay and obligatory Matlab pause with a time offset varying from 0.1 ms to 2.6 ms, depending on the back-end device specifications. We envision that the proposed closed-loop, adaptive brain interface could help neuroscientists conduct their trials more quickly while at the same time also helping clinicians diagnose brain disorders more accurately. Furthermore, it also reduces the cost of performing research trials and diagnostic testing because it relies on a low cost, open source platform. In short, the proposed distributed computing platform for implementing Brain-in-the-Loop neural analysis could open up a myriad of new EEG-based applications, creating a paradigm shift in the way in which the human brain and the digital computer communicate with one another.

References

[1] N. Birbaumer, A. Ramos Murguialday, C. Weber, and P. Montoya, "Chapter 8 Neurofeedback and Brain–Computer Interface," in *International Review of Neurobiology*, vol. 86, pp. 107–117, Elsevier, 2009.

[2] J. D. R. Millán, "Combining brain-computer interfaces and assistive technologies: state-of-the-art and challenges," *Frontiers in Neuroscience*, vol. 1, 2010.

[3] S. K. Ehrlich, K. R. Agres, C. Guan, and G. Cheng, "A closed-loop, music-based brain-computer interface for emotion mediation," *PLOS ONE*, vol. 14, p. e0213516, Mar. 2019.

[4] R. H. Wilkins, "Neurosurgical classic—xvii," *Journal of Neurosurgery*, vol. 21, no. 3, pp. 240 – 244, 01 Jan. 1964.

[5] S. Sandrone, M. Bacigaluppi, M. R. Galloni, and G. Martino, "Angelo Mosso (1846–1910)," *Journal of Neurology*, vol. 259, pp. 2513–2514, Nov. 2012.

[6] M. X. Cohen, *Analyzing neural time series data: theory and practice*. Issues in clinical and cognitive neuropsychology, Cambridge, Massachusetts: The MIT Press, 2014.

[7] J. H. Gruzelier, "EEG-neurofeedback for optimising performance. I: A review of cognitive and affective outcome in healthy participants," *Neuroscience & Biobehavioral Reviews*, vol. 44, pp. 124–141, July 2014.

[8] M. Reiner, R. Rozengurt, and A. Barnea, "Better than sleep: Theta neurofeedback training accelerates memory consolidation," *Biological Psychology*, vol. 95, pp. 45–53, Jan. 2014.

[9] J. Faller, J. Cummings, S. Saproo, and P. Sajda, "Regulation of arousal via online neurofeedback improves human performance in a demanding sensory-motor task," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 6482–6490, Mar. 2019.

[10] N. Birbaumer, "slow Cortical Potentials: Plasticity, Operant Control, and Behavioral Effects," *The Neuroscientist*, vol. 5, pp. 74–78, Mar. 1999.

[11] F. Pichiorri, G. Morone, M. Petti, J. Toppi, I. Pisotta, M. Molinari, S. Paolucci, M. Inghilleri, L. Astolfi, F. Cincotti, and D. Mattia, "Brain-computer interface boosts motor imagery practice during stroke recovery: BCI and Motor Imagery," *Annals of Neurology*, vol. 77, pp. 851–865, May 2015.

[12] T. O. Bergmann, "Brain State-Dependent Brain Stimulation," *Frontiers in Psychology*, vol. 9, p. 2108, Nov. 2018.

[13] O. Jensen, A. Bahramisharif, R. Oostenveld, S. Klanke, A. Hadjipapas, Y. O. Okazaki, and M. A. J. van Gerven, "Using Brain–Computer Interfaces and Brain-State Dependent Stimulation as Tools in Cognitive Neuroscience," *Frontiers in Psychology*, vol. 2, 2011.

[14] M. Thies, C. Zrenner, U. Ziemann, and T. O. Bergmann, "Sensorimotor mu-alpha power is positively related to corticospinal excitability," *Brain Stimulation*, vol. 11, no. 5, pp. 1119–1122, 2018.

[15] J. Wang, Z. Feng, and N. Lu, "Feature extraction by common spatial pattern in frequency domain for motor imagery tasks classification," in *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 5883–5888, 2017.

[16] A. al Qerem, F. Kharbat, S. Nashwan, S. Ashraf, and k. blaou, "General model for best feature extraction of EEG using discrete wavelet transform wavelet family and differential evolution," *International Journal of Distributed Sensor Networks*, vol. 16, p. 155014772091100, Mar. 2020.

[17] Y. Liu and O. Sourina, "Eeg-based subject-dependent emotion recognition algorithm using fractal dimension," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3166–3171, 2014.

[18] J. Liu, G. Wu, Y. Luo, S. Qiu, S. Yang, W. Li, and Y. Bi, "Eeg-based emotion classification using a deep neural network and sparse autoencoder," *Frontiers in Systems Neuroscience*, vol. 14, p. 43, 2020.

[19] R. Bauer and A. Gharabaghi, "Reinforcement learning for adaptive threshold control of restorative brain-computer interfaces: a Bayesian simulation," *Frontiers in Neuroscience*, vol. 9, Feb. 2015.

[20] J. DiGiovanna, B. Mahmoudi, J. Fortes, J. Principe, and J. Sanchez, "Coadaptive Brain–Machine Interface via Reinforcement Learning," *IEEE Transactions on Biomedical Engineering*, vol. 56, pp. 54–64, Jan. 2009.

[21] "TCP/IP Five-Layer Software Model Overview - Developer Help."

[22] M. Galarnyk, "Understanding Boxplots," July 2020.