NeuroHub Networking Integration: Time Synchronization Device for Multimodal Brain Imaging and Hyperscanning Research

A Thesis

Submitted to the faculty

of

Drexel University

by

Neha Thomas

in partial fulfillment of the

requirements for the degree

of

Masters of Science in Biomedical Engineering

June 2017

Acknowledgements

I would like to thank my parents and friends for their support and their belief in me. I would also of course like to thank Dr. Ayaz for his advice, guidance, and mentorship throughout the duration of this thesis.

Table of Contents

List of Tables

List of Figures

Abstract

NeuroHub Network Integration: Time Synchronization Device
for Multimodal Brain Imaging and Hyperscanning Research
Neha Thomas
Dr. Hasan Ayaz, PhD

Significant progress has been made over the last decades in understanding the physiological and neural bases of cognitive processes and behavior. The advent of new and improved sensors enables monitoring the human body and brain activity in natural environments, with cost-effective, mobile and wearable form factor systems. As neuroimaging and brain sensing technologies are further developed, there's an expanding interest for using multiple systems concurrently on i) the same brain: multimodal/hybrid measurements for better identification of neurophysiological markers, and ii) multiple brains: hyperscanning for novel investigations of brain functions during social interactions. Particularly for functional neuroimaging, such as Functional Near Infrared Spectroscopy (fNIRS) and Electroencephalography (EEG), precise time synchronization of experimental events with acquired datasets is necessary for proper analysis and interpretation of results. However, there are currently no standards for interoperability and neuroimaging systems have many different designs and interfaces. Furthermore, it is often cumbersome to come up with a custom solution to each new research setup based on the devices involved. The original NeuroHub, a plug-and-play time synchronization device developed at Drexel University, attempted to alleviate some of the complications associated with custom setups and time synchronization. The original NeuroHub relayed any incoming signal to one of its four serial ports, TTL port, and parallel port, to all other

ports on the device and can be connected to multiple sending/listening devices or computers. Although one or more of these legacy ports are present in various neuroimaging systems, modern computing systems require more sophisticated alternatives.

This thesis proposes a solution and improvement to the original NeuroHub, by incorporating time synchronization over a network as an information transfer layer. The network solution enables more flexible experimental configurations and expands the compatible plug-and-play system range. Moreover, this new approach eliminates the need for multiple wires, while still being able to service large number of clients. The new NeuroHub is also able to directly interface with typical RS-232 serial ports and offers the best of both worlds – ability to interface with network and legacy hardware ports for complete customizability, flexibility and backward compatibility.

The new NeuroHub network module consists of a Raspberry Pi Model 1B fitted with a serial port add-on board. The device transmits any event markers received from either networked or serial ports and relays them to the other opened ports. Verification testing confirmed that the device transmits with 100% accuracy and the latency to send a byte from one computer to the other via the network module was minimal, ranging from sub-millisecond speeds to 7 ms depending on the use of serial ports, baud-rate, and configuration order.

The new NeuroHub network module was tested in Brain Compute Interface (BCI) setups using OpenViBE as a stimulus presenter and EEG data recording, with COBI Studio as the fNIRS data recording software to receive markers all through NeuroHub. This simple use case demonstrates the utility of the new NeuroHub for simplification of

complex functional neuroimaging, neuroergonomics and BCI research experimental setups.

1. Introduction

1.1 Motivation

In the field of neuroscience and brain imaging research, there are a growing number of studies that require recording from multiple modalities. For example, researchers may want to use both EEG and fNIRS to extract important features about cognition that may not be otherwise possible with just one modality. Using multiple systems requires that all datasets collected are properly synchronized in time. This is done through event markers, which represent information about important events throughout the course of the study such as the task start/end or onset of stimuli. The difficulty arises when trying to send event markers from multiple devices to other devices. Such setups usually require careful planning and can be burdensome to implement, especially when trying to coordinate devices manufactured by different companies. Each device may use a different data transmission protocol that makes event markers synchronization between different systems complicated and tedious. Additionally, certain devices require very fast and accurate transmission rates, such as EEG, which has a high sampling rate.

1.2 Objective

This thesis aims to develop a portable device that can act as an interface between multiple systems that may have different communication protocols. Such a device would accelerate experimental research setups that utilize multiple recording devices and need fast, reliable timestamping and marker transmission. Such device would implement protocols for transmission of data over a network as well as a physical connection for serially transmitted data. This ensures versatility and practicality for a variety of experimental setups. Essentially, the device would serve as a bridge between older methods of communicating

event markers like the RS-232 and newer methods, such as networking protocols. Finally, the device would be tested for fast and accurate data transmission in a variety of test configurations.

1.3 Outline

In the background section, brain imaging methods will be described to provide context for the device need. Next, common data transmission protocols will be covered that are used for time synchronization. Time synchronization platforms will then be discussed, and some of the shortcomings of these platforms will be addressed in the proposed solution. In the device design section, the design requirements and specifications are reviewed and implemented into a device. The device is then finally tested for its lag time and reliability.

2. Background

2.1 Brain Imaging Methods

2.1.1 Functional Near Infrared Spectroscopy

Functional Near Infrared Spectroscopy (fNIRS) is an optical brain activitymonitoring method that uses near-infrared light to measure concentration changes of deoxygenated and oxygenated hemoglobin (HbR and HbO, respectively) (Quaresima & Ferrari, 2016; Villringer & Chance, 1997; Villringer et al., 1993; Chance et al., 1993; Chance, 1991). It is portable, wearable, cost-effective, and its data is not as susceptible to electrical noise as compared to EEG (Naseer & Hong, 2015). Modern fNIRS systems are miniaturized and can be even built battery operated and wireless to allow untethered monitoring of particpants (Quaresima & Ferrari, 2016; Ayaz et al., 2013). The changes of HbR and HbO are related to the brain activation in the area of measurement through neurovascular coupling theory (Ayaz et al., 2013; Izzetoglu et al., 2005). Light that is transmitted by the fNIRS light source over the scalp penetrates through tissue layers in the brain and a fraction of the scattered photons can reach back to photodetectors strategically placed over the scalp. There are different absorption coefficients for different wavelengths of light for HbR and HbO, which allows calculation of their concentration changes using the modified Beer-Lambert Law (Ayaz et al., 2011; Izzetoglu et al., 2005). The fNIRS signal is sampled at a lower rate compared to EEG (lower temporal resolution) since the underlying hemodynamic response measured by fNIRS is a relatively slow signal (Batula et al., 2017a). On the other hand, fNIRS' spatial resolution is higher than that of EEG and is generally free from artifacts such as eye-blinking and muscle movements (Hirshfield et al., 2009; Sweeney et al 2012; Ayaz et al 2010).

fNIRS has growing range of applications in brain-computer interface that run the gamut from neurofeedback to mental workload and training assessment (Ayaz et al., 2013; Gramman et al., 2017; Izzetoglu et al., 2011; Mckendrick et al., 2016; Mark et al., 2018). For example, Ayaz et al. (2012) used fNIRS to demonstrate the development of expertise in cognitively demanding and complex tasks such as piloting and air traffic control and that hemodynamic changes detected by fNIRS from prefrontal cortex are indicative of task related mental workload. fNIRS has also been shown to be helpful in rehabilitation purposes as a neurofeedback tool for motor imagery. For example, Mihara et al. (2013) showed that it was possible to use fNIRS to enhance the efficacy of imagery-based rehabilitation in hemiplegic stroke patients. Recent reviews of clinical applications are available in Izzetoglu et al (2011) and Teo et al (2016). It is also possible to develop active brain-computer-interfaces (BCI) using fNIRS signals that are captured, processed and classified in real-time during the experiment. For example, Batula et al (2017, 2016) developed an fNIRS based BCI to control a humanoid robot using upper and lower limb motor imagery tasks. And Ayaz et al used prefrontal cortex based fNIRS for control of objects and avatars in virtual environments (Ayaz et al., 2011; Ayaz et al., 2009). For a review of fNIRS based BCI applications, see Naseer & Hong (2015).

2.1.2 Electroencephalography

Electroencephalography (EEG) uses electrodes to measure fluctuations in voltage at the level of the scalp to determine brain activity. These electrodes may sit on the surface of the scalp and can either be dry or stuck with electrode paste. The electrodes are commonly placed using the 10-20 International System of electrode positions (Kennett, 2012). The sub-millivolt voltages recorded from the electrodes are amplified by gains greater than or

equal to 2000, and often digitally sampled from 256 Hz to over 1000 Hz (Schomer & Silva, 2010). EEGs tend to have low signal-to-noise ratios due to the electrodes picking up the activity of millions of cortical neurons. It has low spatial resolution as well, on the order of centimeters, and is generally lower than that of fNIRS (Berka et al., 2004). Therefore, one of the biggest challenges in using EEG is extracting and classifying features that will elicit useful information about the state of the brain. Such features can include event-related desynchronization, event-related potentials like the P300 or steady-state visually evoked potentials (SSVEP) (Amiri et al., 2013).

EEG is a primary brain-imaging mode for BCI (Lebedev & Nicolelis, 2017; Choi et al., 2017; Fazel-Rezai et al., 2012). One study used motor imagery and P300 potential in certain frequencies of brain waves to control the horizontal and vertical movements of a cursor (Li et al., 2010). Similarly, EEG could be used to elicit event-related desynchronization to differentiate right and left imagined movement and use this to control wheelchair direction and movement (Huang et al., 2012). EEG has also been used extensively in epilepsy research. It can be used to diagnose, localize, and detect epilepsy. Diagnosis indicates whether a patient has epilepsy or not, localization refers to the epileptogenic foci, and detection is determining whether a patient is in a seizure state or between seizures (Pouliot et al., 2014). For a review of clinical applications, see Lebedev and Nicolelis (2017).

2.1.3 Multimodal Research

Multimodal research involves the use of more than one sensing mechanism or technique in order to obtain more information than could be possible by using only one modality. One such example is the combination of simultaneous EEG and fNIRS to improve BCI

performance or more accurately discern mental workload states (Liu, Ayaz & Shewokis, 2017; Zich et al., 2016; Aghajani & Omurtag, 2016; Liu et al., 2015; Putze et al., 2014; Liu et al., 2013; Leomy, Collins & Ward, 2011). Combining these two modalities has also been shown to improve the overall BCI performance such as in decoding of motor imagery for BCI use (Yin et al., 2015) and working memory related mental workload classification (Liu, Ayaz & Shewokis, 2017). These hybrid setups provide a more robust way to classify brain activity and reduce false negatives or positives. Since EEG and fNIRS are complimentary and recording seonsors are becoming more and more miniaturized, hybrid systems are realistic and effective ways to improve performance over single modality systems (von Luhmann et al., 2016).

In addition, multimodal research may also include monitoring of other body peripherals such as eye movements and muscle artifacts through electrocoulugram (EOG) and electromyogram (EMG) as well as neurostimulation such as transcranial direct current stimulation (tDCS) and transcranial magnetic stimulation (TMS). Recently wearable fNIRS and tDCS has also been used simultenously over prefrontal cortex (McKendrick, Parasuraman & Ayaz, 2015). Supplementary use of signals is also possible, for example, EOG and EMG can be used to filter these artifacts form the EEG data (Kothe & Makeig, 2013; Cao, Guo & Su, 2015). Among other modalities, eye tracking, pulse oximetry, and galvanic skin response systems are some of the additional body sensors that can be used in a single research setup or together with neuroimaging systems.

2.1.4 Hyperscanning Research

Hyperscanning is a when one modality is used to record the brain activity of multiple subjects that interact with each other as a unique system (Babiloni & Astolfi, 2014). It is predominantly used to investigate social situations, and how one subject's brain activity is related to their behavior as well as that of the partner in the task. EEG and fNIRS are both tools that can be used for hyperscanning research. For example, Liu et. al (2017) investigated brain-to-brain coupling during verbal communication using fNIRS and fMRI. It was found that the brain activity of the listener mirrored that of the speaker but with a delay. Growing number of studies highlight the potential of hyperscanning research for providing new insights for social interaction settings and neuroscience (Liu et al., 2017; Hirsch et al., 2017; Pan et al., 2017; Vanutelli et al., 2016).

2.2 Common Data Transmission Protocols

This section outlines some of the common methods used for communicating data and event markers to other listening devices. Event markers represent information about important events that happen during the duration of the study. They are generated by the stimulus presenter, which is typically the computer that the subject interacts with. Some common events that often need to be marked are the start and end of the task or trial. These markers need to be present on all computers or devices in the research setup for accurate data analysis.

2.2.1 Parallel Port

The standard parallel port (SPP) allows 8 bits to be transferred at the same time., typically meaning they transfer one byte at a time. Most parallel ports use a connector

with 25 pins, termed DB25 connector. With parallel port, speeds of 150 kbps and higher are achievable (IEEE 1284, 2002).

2.2.2 Transistor-Transistor Logic

Transistor-Transistor logic (TTL) is based on an elemental logic block that outputs either a high or low voltage that indicates the logical bit 1 or 0. When the top transistor in the pair of transistor conducts, the output is high. When the bottom transistor conducts, the output is low. In the transition between high and low states, both transistors conduct heavily (Lancaster, 1991). TTL is overall low-cost and has high speeds of up to 20 MHz. Depending on the logic gates used in the TTL circuit, different functions are achieved from counters to data selectors (Lancaster, 1991).

2.2.3 Serial

Serial port transmission or RS-232 communication is done by sending data one bit at a time in sequence (Axelson, 2007). Serial ports are usually bidirectional, meaning that it is possible to receive and send data on the same port. In asynchronous communication, sending and receiving can occur at the same time. The hardware for serial ports is inexpensive and relatively commonplace. Even though native serial ports are rare to find on new models of laptops and computers, it is possible to use USB-Serial converters. Devices with serial ports will generally contain a hardware component called the Universal Asynchronous Transmitter/Receiver (UART). The UART handles low-level details of serial communications such as sending and storing received bits on the serial port (Axelson, 2007).

The UART transmits data in chunks called words. Each word typically contains a Start bit, data bits, an optional parity bit, and at least one Stop bit. A common format for

transmission is to send 1 Start bit, 8 data bits, and 1 Stop bit. Parity bits are used for error detecting. Depending whether the parity is even or odd, the receiving computer can tell whether there was an error if there is an odd or even number of 1s (Axelson, 2007).

Another important feature of the serial port is the baud rate. The baud rate is the number of bits per second (bps) transmitted or received. The highest baud rate seen on most serial ports is 115,200 bps (Axelson, 2007).

As mentioned before, it is possible for computers without native serial ports to communicate via RS-232 through USB-Serial converters. This requires specific drivers to indicate how applications can access the device as a serial port. In most computer's devices and managers, the serial port shows up under a specific COM Port number. The USB-Serial device will also similarly be assigned to a COM Port, and then access to this port is just like accessing any other port (Axelson, 2007).

2.2.4 Transmission Control Protocol

Transmission Control Protocol (TCP) is a network protocol that guarantees data delivery between two locations. It is considered a 4-layer system, consisting of the application, transport, network and link layer. The application is the most high-level and examples include email, Telnet, and File Transfer Protocol. The transport layer provides a flow of data between two hosts; in this case it is TCP. The network layer handles movement of packets around the network. Internet Protocol (IP) provides the network layer in the TCP/IP protocol suite. Finally, the link layer includes the device driver and network interface card within the computer and handles the hardware details of interfacing with the Ethernet cable or other media for transferring packets. When the application sends data, it is sent down the protocol stack where each layer adds information to the data by

prepending headers. Finally, it is sent as a stream of bits across the network. With all headers, the minimum size for a TCP packet is 46 bytes (Stevens and Wright, 1994).

For reliability, every segment that is transmitted has a sequence number assigned to it. For each segment that is sent, the receiver must return an acknowledgement (ACK) to confirm that the bytes are received within a period of time. If the ACK is not received, the data is retransmitted (TCP/IP, 2003).

Finally, the fields in the TCP header are the source port, destination port, sequence number, acknowledgement number, window, and checksum. The window is the TCP buffer on the host to store incoming segments. The checksum exists as a way to verify the bit-level integrity of the header and data (Stevens & Wright, 1994).

2.2.4 User Datagram Protocol

Unlike TCP, User Datagram Protocol (UDP) does not provide reliability in data transmission, and there is no guarantee that data will be received in the correct order that it was sent in. Like TCP, UDP is also a 4-layer system and contains a header, which is only 8 bytes, substantially smaller than TCP (Stevens & Wright, 1994). UDP may provide faster data transmission, but there is a tradeoff between speed and reliability. In situations where accurate transmission is necessary, TCP is the better protocol.

2.3 Current Time Synchronization Methods

This section will cover some of the current flexible methods used to synchronize time markers in research setups. These solutions are generalizable to many custom setups, and focus on purely hardware or purely software solutions to achieve synchronization.

2.3.1 Lab Streaming Layer

Lab Streaming Layer (LSL) is a system to collect measurement time series in research setups using networking with TCP. LSL was developed at the Swartz Center for Computational Neuroscience at University of California, San Diego (Kothe, 2013). It requires programming knowledge to use, and thus is not ideal for researchers who do not have the necessary background. The built-in time synchronization is designed after Network Time Protocol (NTP). A timestamp is sent with each data using the high-resolution clock of the host computer. Clock synchronization information consists of measurements of the momentary offset between the involved clocks that are made every few seconds. A brief sequence of packet exchanges provides information about the estimate of the round-trip time between the two computers and the estimate of the clock offset with round-trip time factored out. This offset is used to remap the local time domain so that data is comparable across all connected computers (Kothe, 2013). Previous research (Grzeczkowski & Ayaz, 2014) indicates that one caveat is that there may be random delays or interruptions which can result in unknown lag times and therefore inaccurate conclusions about the time synchronization using LSL.

2.3.2 Original NeuroHub core module

The original NeuroHub core module is a plug-and-play device built at Drexel University that accepts event markers in the range of 1-255 via four serial ports, 1 parallel port, and 1 BNC cable for TTL communication. Any marker that arrives on a port is replicated and sent to all other ports. This enables different systems that may have different protocols for hardware communication to synchronize event markers easily. The core module accepts and sends bytes at 9600 bps and is built on an Arduino Mega. It was fitted with a custom

shield, or add-on printed circuit board (PCB) to enable interfacing with serial, parallel and BNC ports. It has a consistent latency of 1.02 ms between receiving a byte and sending the byte out (Grzeczkowski & Ayaz, 2014). Fig. 1 displays the core module. The baud rate is not customizable and additionally, newer systems are less likely to have the legacy ports that are present on the core module.

Figure 1 NeuroHub core module with 4 RS232 ports, 1 TTL port, and 1 parallel port

2.4 Problem Definition

The complexity of multimodal and hyperscanning research studies calls for accurate and precise time synchronization across all platforms. The NeuroHub core module allows interfacing with serial port, TTL, and parallel port, but these types of ports are seen less and less on modern computers. To provide more flexibility and customizability, networking ability should be incorporated into the NeuroHub. This type of device would also reduce the number of wires in the setup while still servicing multiple clients and ensuring small lag times.

3. Device Design and Development

3.1 Device Design

3.1.1 Device Requirements

The device should also be easily portable and simplify the event marker synchronization among common communication protocols in multimodal and hyperscanning research studies. The device should easily interface with the NeuroHub core module and have networking capabilities. Therefore, it should be able to send and receive via TCP, UDP, and serial port protocol. It should also be linked to a graphical user interface so that users can select which network protocol, serial port baud rate, and number of networked clients in the setup. The device should have submillisecond latencies since EEG has high temporal resolution (256-1000 Hz) and precision timing is critical in research setups with EEG.

3.1.2 Device Specifications

The computing board that the device is built on should have at least a 16 Mhz processor, and should include a networking component such as Ethernet or Wi-Fi card. Ethernet card is preferred because it is more reliable and faster than Wi-Fi. The board should be programmable in a low-level language like C++ for efficient code and optimized error handling. The board should include at least 1 UART to be able to interface with a standard serial RS-232 port to easily connect to the NeuroHub core module. Finally, the board should be well documented and inexpensive.

3.2 Device Development

3.2.1 Board Selection

There were a few boards to choose from in the initial device planning stage. Since the original core model is constructed from Arduino, it would be possible to add an Arduino

shield with Ethernet networking capabilities. However, this would require a complete redesign of the original NeuroHub to incorporate the additional shield. There is also no way to create a GUI by programming in the Arduino environment. It requires linking to some external GUI developing platform and increases the complexity of setup with the NeuroHub core module. The other option was to build a separate networking module that would be a modular and optional expansion to the core module. Some of the boards that already included network capability were the Raspberry Pi, Minnowboard, and DragonBoard. It was decided that Ethernet should be used for the first generation of the networking module since Ethernet offers speed, minimal latency, and reliability over wireless protocols. This narrowed down the choices to the Raspberry Pi and Minnowboard. The Minnowboard costs more than four times the cheapest Raspberry Pi model; for this reason, the Raspberry Pi 1 Model B was selected. Upgrading to the Raspberry Pi Model 3 B woud also allow more flexible options since it has both Ethernet and Wi-Fi capability. In total, Raspberry Pi was the best documented, and the least expensive and smallest board that met the selection criteria.

3.2.2 Hardware Add-ons and Programming of the NeuroHub Network Module

A DTronix Mini Piio RS232 add on PCB or Raspberry Pi “hat” was chosen as a simple way to add an RS232 interface to the Raspberry Pi. This would enable a straightforward way to connect the NeuroHub network module with the NeuroHub core module. C++ was chosen as the coding language of choice since it is efficient and fast compared to other languages, and easier to perform error handling and maintain than with C. The complete

network module is shown in Fig. 2A. Fig. 2B shows the network module connected to the core module.

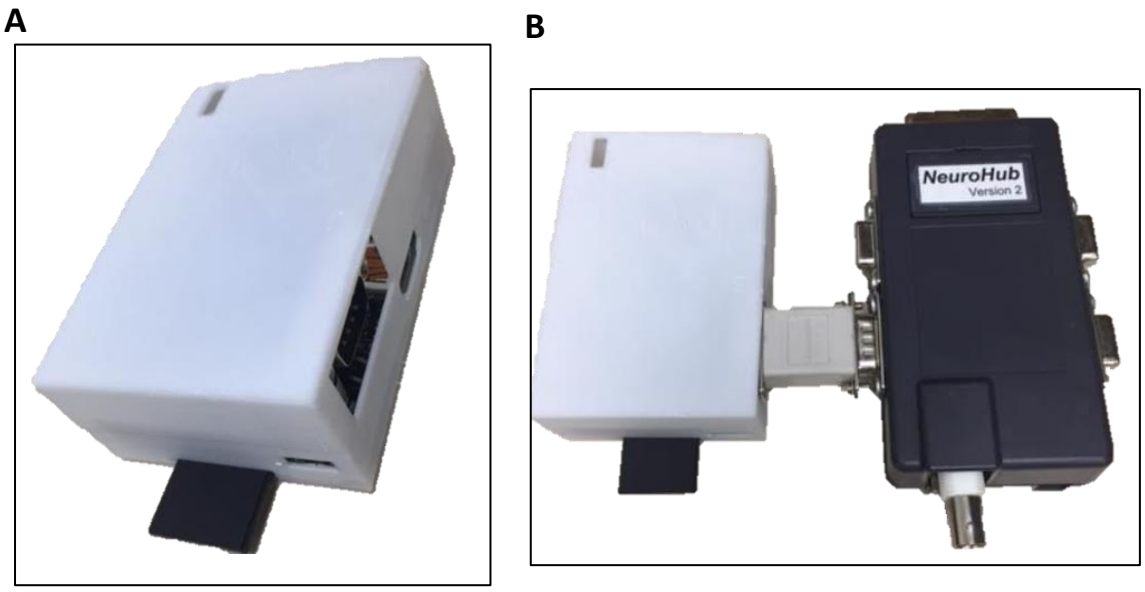


Figure 2 A) NeuroHub network module B) complete NeuroHub: network module connected with core module

In a subsequent revision, the Raspberry Pi 3 Model B was used, since the Pi 3 included both Wi-Fi and Ethernet capability, which made it the most flexible option. A comparison table for the two models is shown in Table 1. The Raspberry Pi 3 boasts superior performance to the older generation in various categories, most importantly, the processor and number of cores. Fig. 3A and 3B show the revised network module using the Raspberry Pi 3.

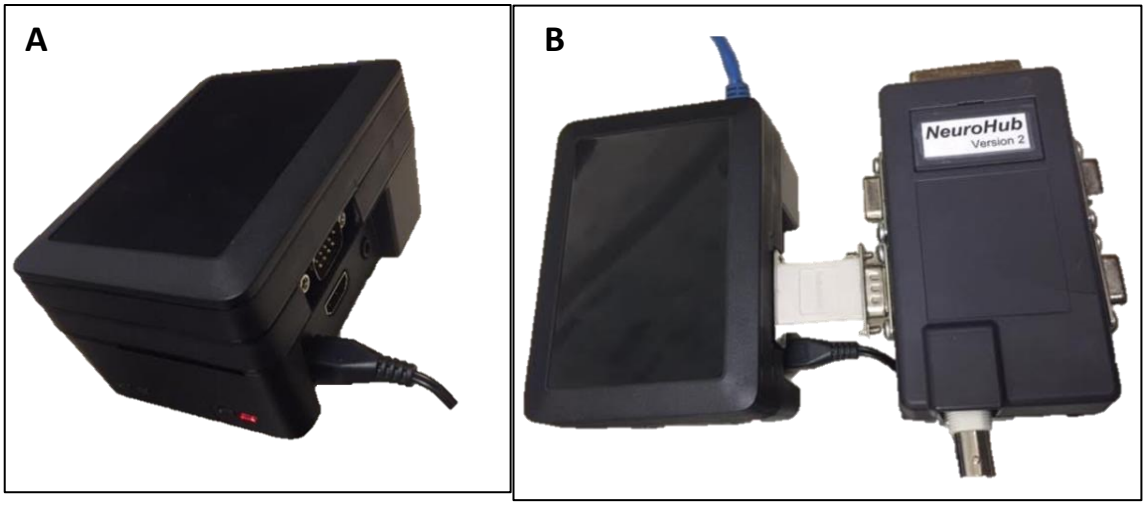


Figure 3 A) NeuroHub Network Module Version 2 B) Network module version 2 connected to core module

Table 1 Comparison chart for Raspberry Pi Model 1 and 3

	Raspberry Pi Model 1 B	**Raspberry Pi Model 3 B**
CPU	700 MHz single-core	1.2 GHz 64-bit quad-core
Ethernet	10/100 Mbit/s	10/100 Mbit/s
SD Card	Full-size SD card	MicroSDHC
GPIO pins	26	40
USB slots	2	4
Wi-Fi	NA	802.11n wireless
Bluetooth	NA	4.1

A GUI was programmed using gtkmm 3.0 libraries and Glade. The GUI allows the user to select the networking protocol, serial port baud rate, and number of networking clients. The underlying code uses standard C++ libraries to connect to, read from, and write to TCP and UDP sockets. WiringPi library was installed on the Raspberry Pi in order to easily write to and read from the serial ports. Once the connect button on the GUI is pressed, the server listens for the number of clients defined by the user. When the number of clients meets the maximum, threads are generated for each client and for the serial port. Each thread contains a blocking read operation. After the read operation returns, the

character message is sent to all other clients and then written to the serial port. At any time, the user may choose to "quit" and close all connections. It is possible to restart connections, at which point the server will again begin listening for clients. A figure of the GUI is shown in Fig. 4. The block diagram of the event marker flow through the network module is shown in Fig. 5. Fig. 6 shows the flow diagram for the network module code.

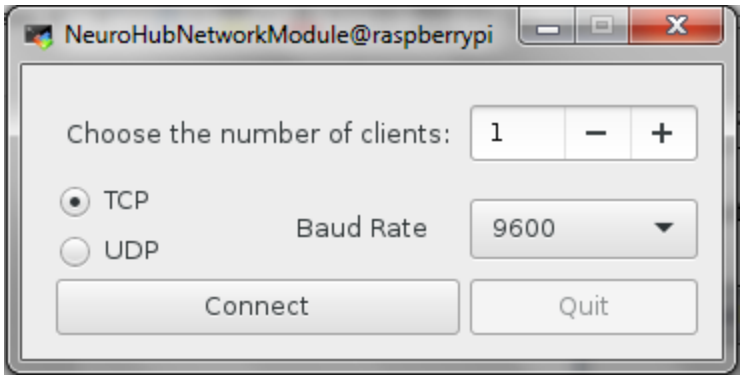


Figure 4 NeuroHub Network Module GUI

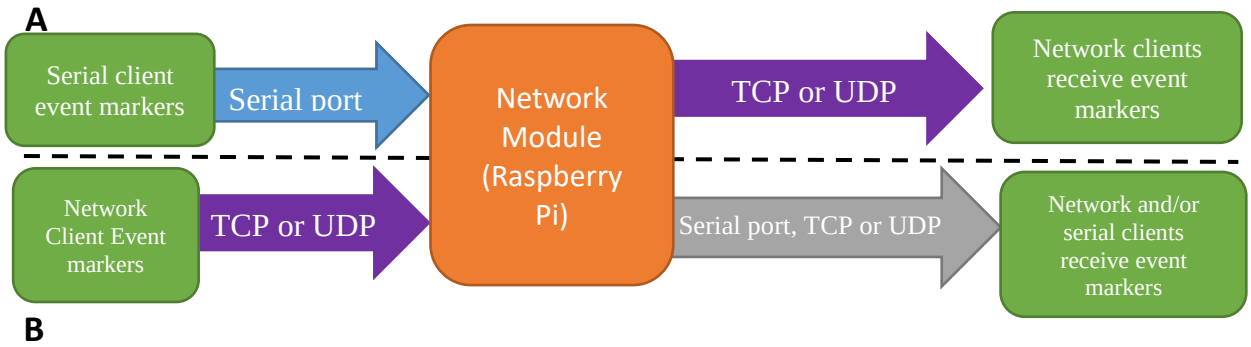


Figure 5 Network Module block diagram. A) and B) reflect different paths for incoming event markers, depending on the initial modality (serial vs networking)

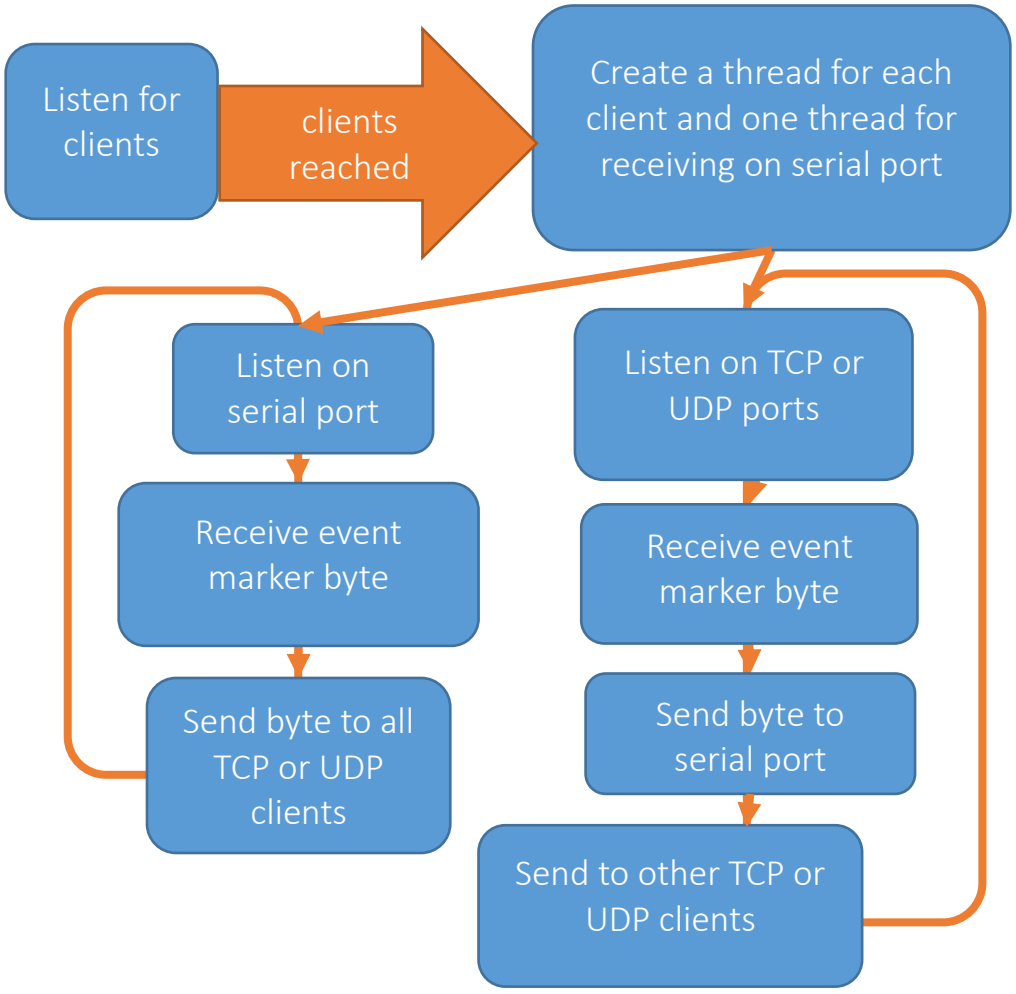


Figure 6 Block diagram for network module programming code

4. Testing and Validation

4.1 Test Configurations

There were four configurations to test the NeuroHub network and core modules connected to a single computer: 1) Receiving via Ethernet and sending via serial, 2) Receiving via serial and sending via Ethernet, 3) Receiving via Ethernet and sending via Ethernet, and 4) Receiving via serial and sending via serial. With the core module, it is possible to add an additional path for byte transmission in tests 1 and 2, shown in Fig. 7A and 8A. The core module can also be used without the network module, as indicated in Fig. 10. In these scenarios, an external computer such as a laptop or desktop would send a byte via Ethernet or Serial to the network module. The network module would then send the same byte back to the sender. Additional tests are to constrain the path of byte transmission to the just host computer for Ethernet to Ethernet and Serial to Serial communication, as indicated in Fig. 9B and 10B. For each of the tests involving Ethernet, both UDP and TCP were tested. For each of the tests involving serial ports, either 9600 or 115200 bps baud rate was used. All tests were done on three different computers – two laptops and one desktop computer. Their specifications are summarized in Table 2. Since the laptops do not have native serial ports, Prolific 2303 USB-to-Serial adapters were used. It was later determined that the type of USB-Serial adapter used can significantly alter the round-trip latency, so the adapter type was another variable that was changed in testing configurations. The adapters used were the Prolific 2303 USB-serial and Sabrent USB-Serial. To characterize the data transmission delay in the networking module, a program was designed in C# to measure the round-trip times for each of the test configurations. The test configurations shown in Figs. 7-10 were completed on version 1 only because of time constraints, as version 2 was

finalized much later. It is important to note that the latencies will be faster on version 2 since the Raspberry Pi 3 Model B is faster than Raspberry Pi 1 Model B.

Table 2 Computer specifications

Computer Type	Lenovo P400 laptop	Dell XPS laptop	Desktop computer
Operating System	Windows 10 Home	Windows 10 Pro	Windows 7 Professional
Processor	Intel core i7-3632QM CPU 2.2GHz	Intel core i7-3612QM CPU 2.1 GHz	Intel core i7 CPU 920 2.67 GHz
Register size	64-bit	64-bit	64-bit
RAM size	8 GB	16 GB	6 GB
Windows Experience Rating	5.9	6.9	7.3

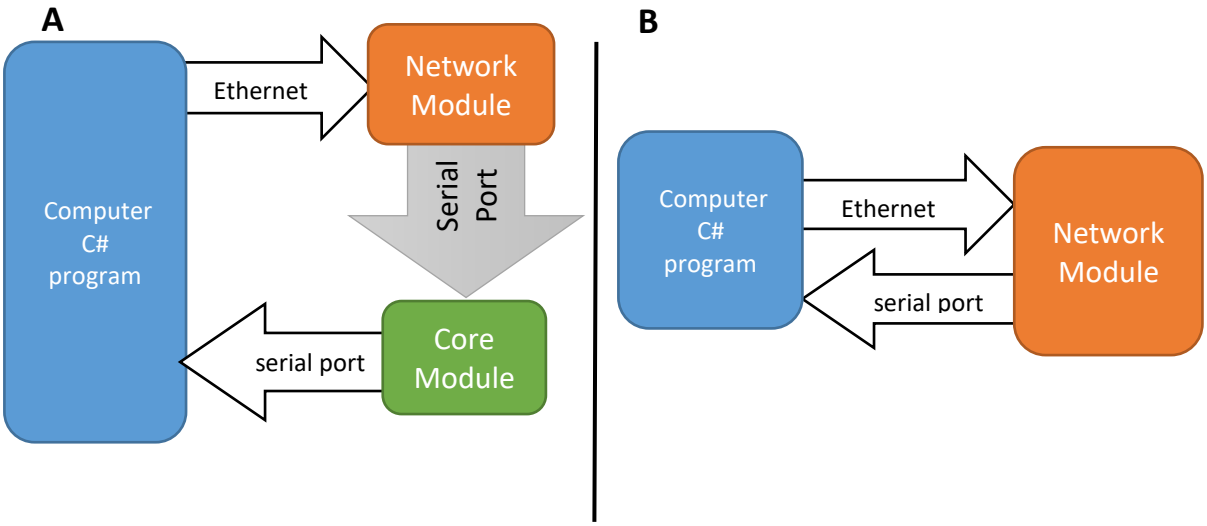


Figure 7 Ethernet to Serial test configurations involving NeuroHub. A) With core module. B) Without core module

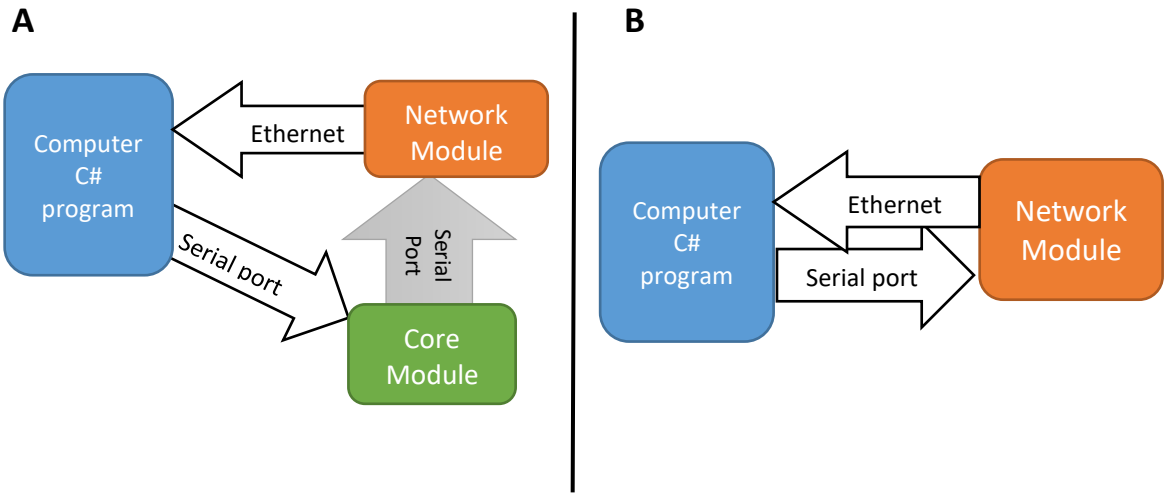


Figure 8 Serial to Ethernet test configurations involving NeuroHub. A) With core module. B) Without core module

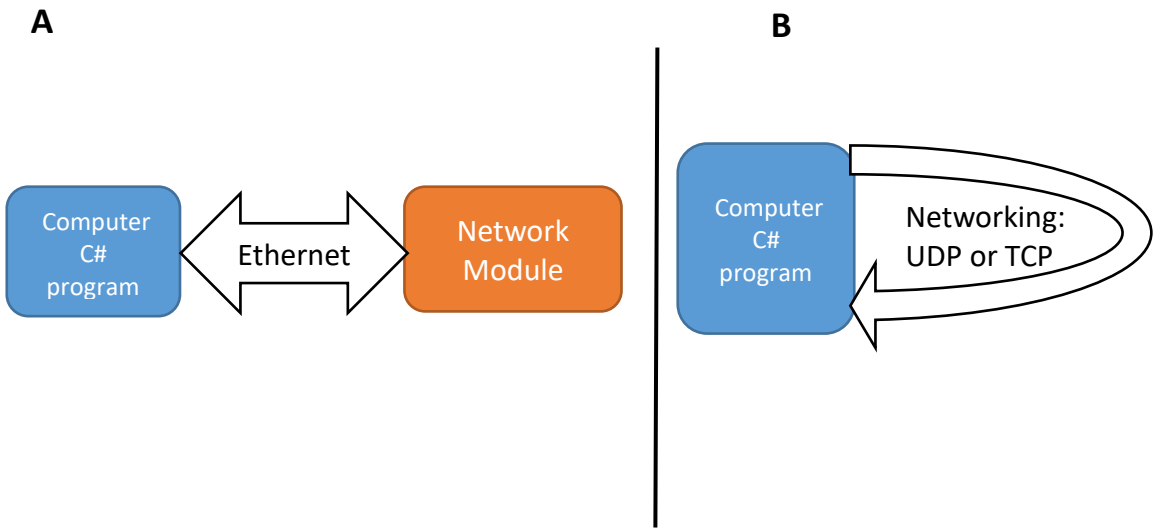


Figure 9 Ethernet to Ethernet test configuration. A) with network module B) without network module

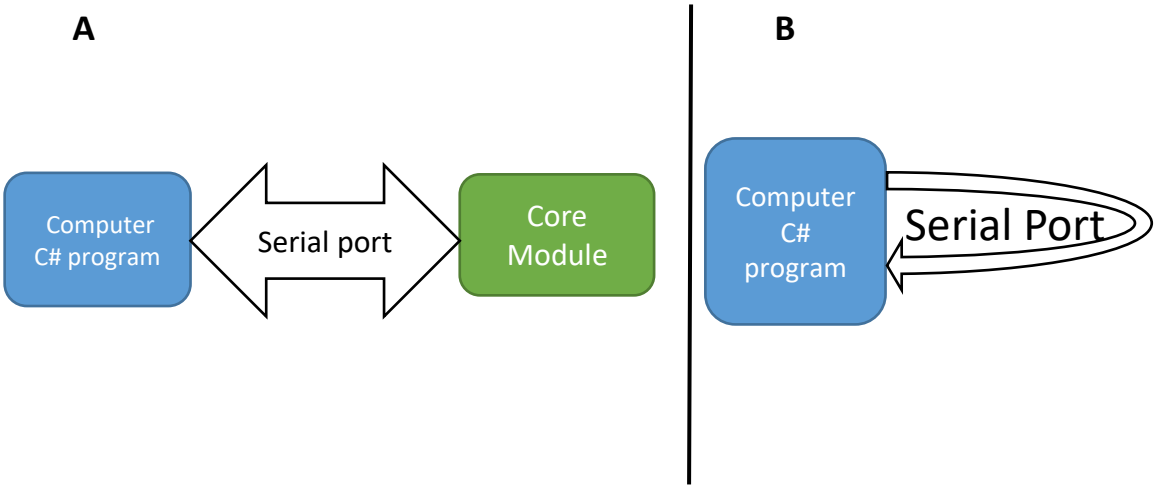


Figure 10 Serial to Serial test configuration. A) with core module B) without core module

Another set of experiments was done to characterize the latencies for marker transmission for several networked clients at the same time for both version 1 and version 2. Version 1 must be connected to the host computer via Ethernet (Ethernet mode). In this setup, it is necessary to connect the host computer to the other clients via an ad-hoc network for multi-client network communication. Version 2 can be connected with Ethernet the same way that Version 1 is, or it can be connected wirelessly via an ad-hoc network initiated by the Raspberry Pi to the clients. The multi-client test configurations are depicted in Figs. 11 and 12.

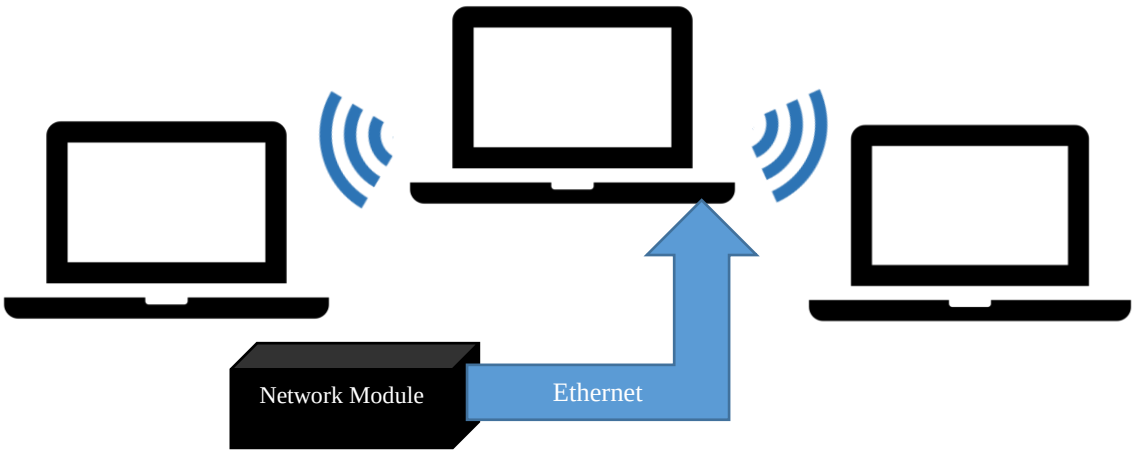


Figure 11 Ethernet mode test configuration with multiple clients

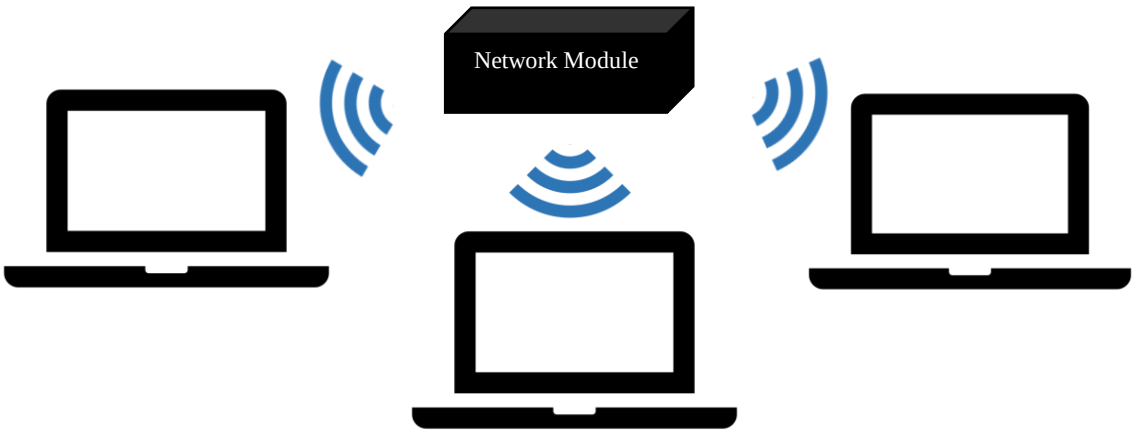


Figure 12 Wireless mode test configuration with multiple clients

4.2 Testing Program Design and Development

Two testing programs were coded in C# as a Windows form application because it is a simple environment to read and write to the serial port. The form enables selection of which COM ports to use, protocol communication selection, and the number of data points. 10 trials of 1000 data points were completed for each test configuration, for a total of 10,000 points per test. The first testing program sends bytes out through either TCP or UDP and receives through the UDP, TCP or serial port. The second testing program sends through the serial port, and receives through either UDP, TCP, or serial port. It should be noted that for the special case of sending through serial port and receiving via Ethernet with TCP, it was necessary to send an "acknowledgement" byte back to the server. Without this acknowledgement byte, the round-trip latency was more than 50 ms for this particular configuration. After adding this byte, the latency dropped to less than 5 ms. The elapsed time between sending and receiving the byte is recorded in milliseconds. The sending and receive methods for these programs operate in separate threads. Once a byte is sent, the program waits to receive the same byte back before sending out a new byte. It is extremely important that the byte received matches the byte sent; 1 is recorded for a matching byte, while 0 is recorded for an incorrectly received byte. The data is saved in a tab delimited text file for further data analysis.

4.3 Data Analysis

A program in MATLAB was created to analyze the .txt files generated by the C# testing program. The program is designed to analyze all data from a specific set, such as all Ethernet – Ethernet tests or all Ethernet – Serial tests. This is because the program checks all data for the minimum and maximum latencies within that test set. It uses this

information to set the x axis limits on the figures so that tests that vary the baud rate or network protocol within the same test configuration are easily comparable. The MATLAB program outputs a .csv file with basic statistics for the overall test as well as each of the ten trials. It generates figures and saves them as .jpeg. The types of figures it generates are a histogram of all 10,000 data points, average values in each trial, histograms for each trial, plot point of all 10,000 data points, the success rate bar graph (percentage of correct bytes returned), and box plot of each trial.

4.4 Results

All tests done had a 100% success rate, meaning that the same bytes sent to NeuroHub were received successfully on the C# program. Fig. 13 is an example of the success bar graph for one test configuration. Other test configurations produced the same success graph. Figs. 14 - 17 show the average latencies for round trip times among all three computers and baud rates, with no core module involved.

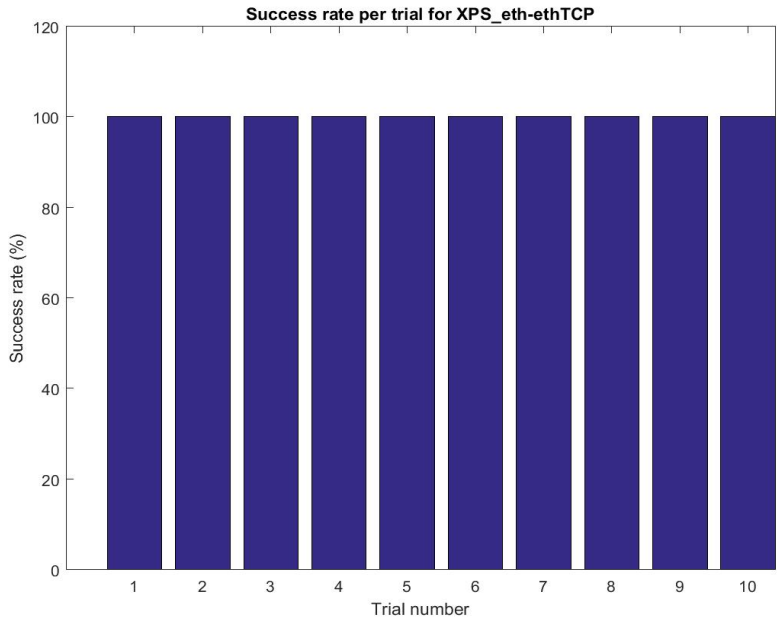


Figure 13 Success rates for 10 trials of test configuration 3 on Dell XPS laptop

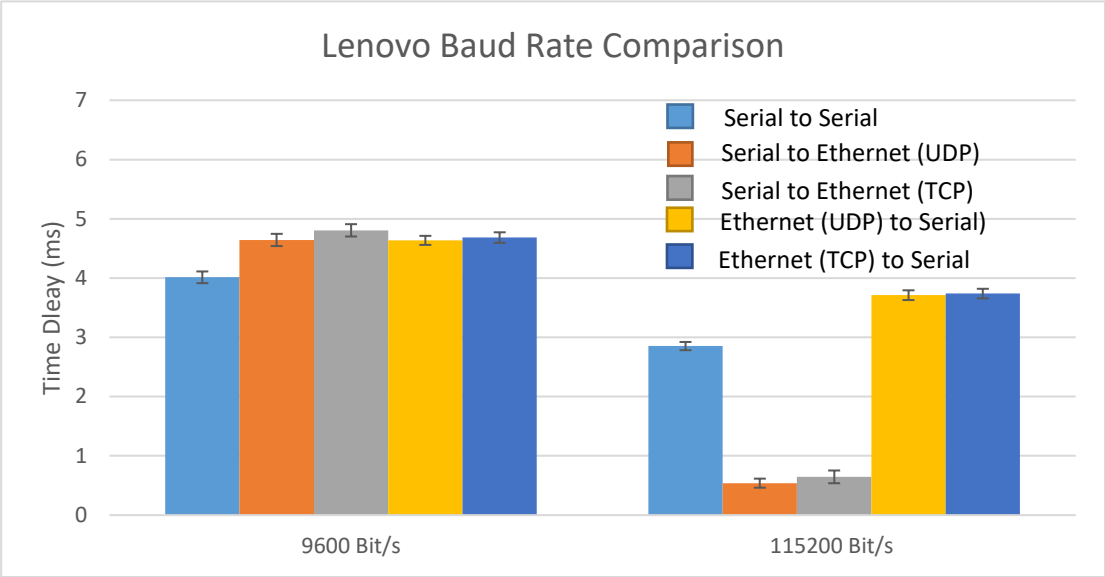


Figure 14 Lenovo average round trip times for all tests

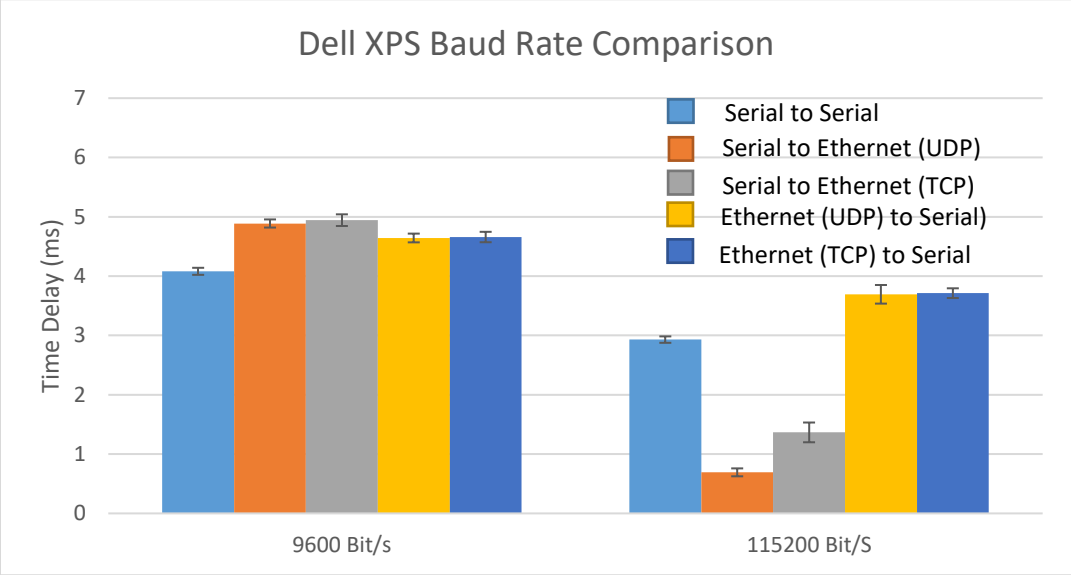


Figure 15 Dell XPS average round trip times for all tests

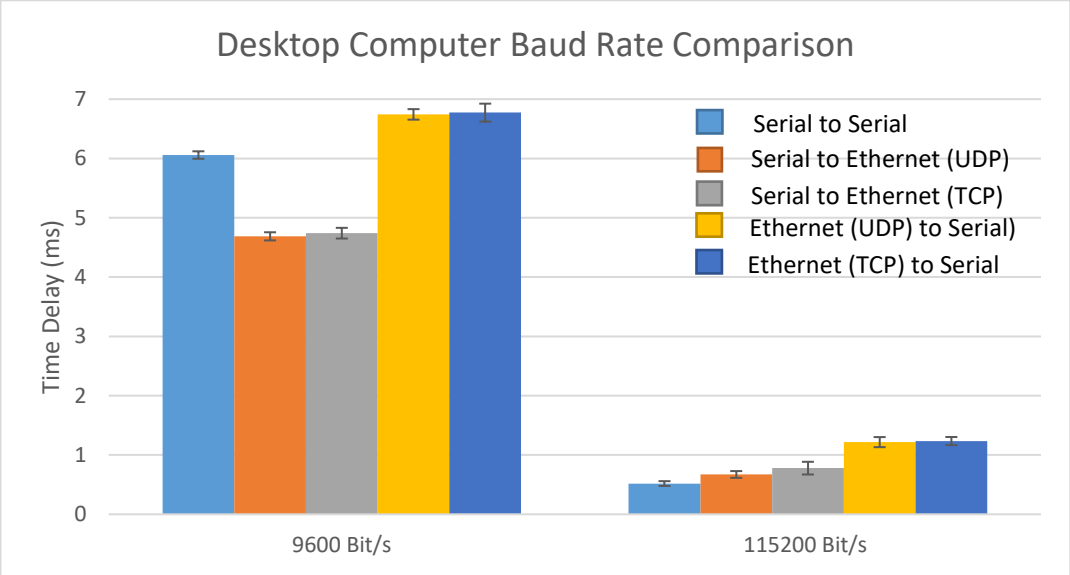


Figure 16 Desktop average round trip times for all tests involving serial ports

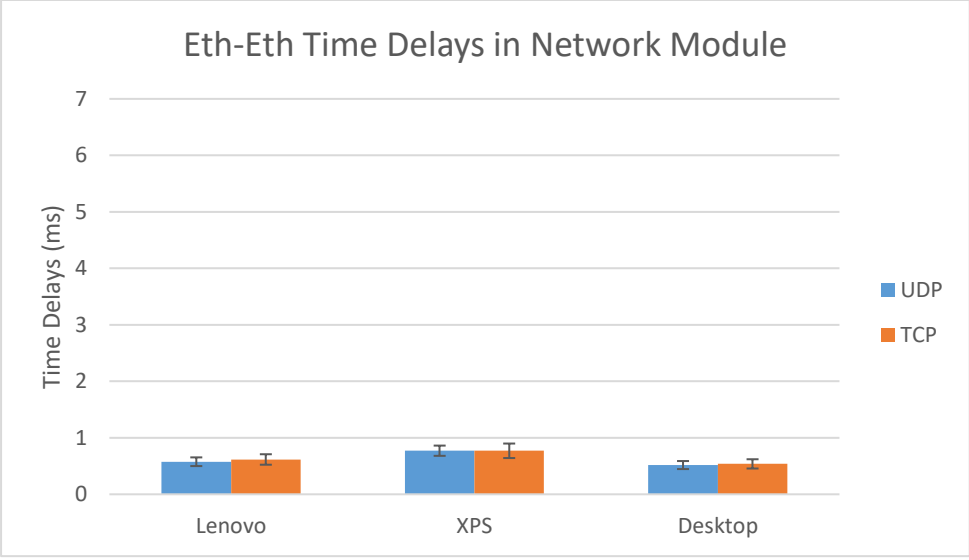


Figure 17 Average round trip times for Ethernet-Ethernet tests on all computers

The round-trip times for the two laptops using USB-serial ports are very similar across all tests and baud rates. The Serial-Ethernet TCP test at 115200 bps on the Dell XPS laptop was slower than its UDP counterpart and the Serial-Ethernet TCP and UDP tests on the Lenovo. This is probably because TCP receiving in the serial to Ethernet configuration is the less efficient on the Dell XPS due to the need for the acknowledgment byte. It seems to be a computer specific issue since this same phenomenon is not present on the other computers. Compared to the desktop computer, increasing the baud rate for the laptop tests that involve the serial port receiving bytes does not improve the latency by much. Since the biggest difference between the desktop computer and laptop computers test configurations is the use of either native or USB-serial port, the lack of improvement in the increased baud rate for the laptops can be attributed to the use of USB-serial converter. Using a USB-Serial converter on the desktop instead of using the native serial ports yielded similar results to the laptops. These results are shown in Fig. 18.

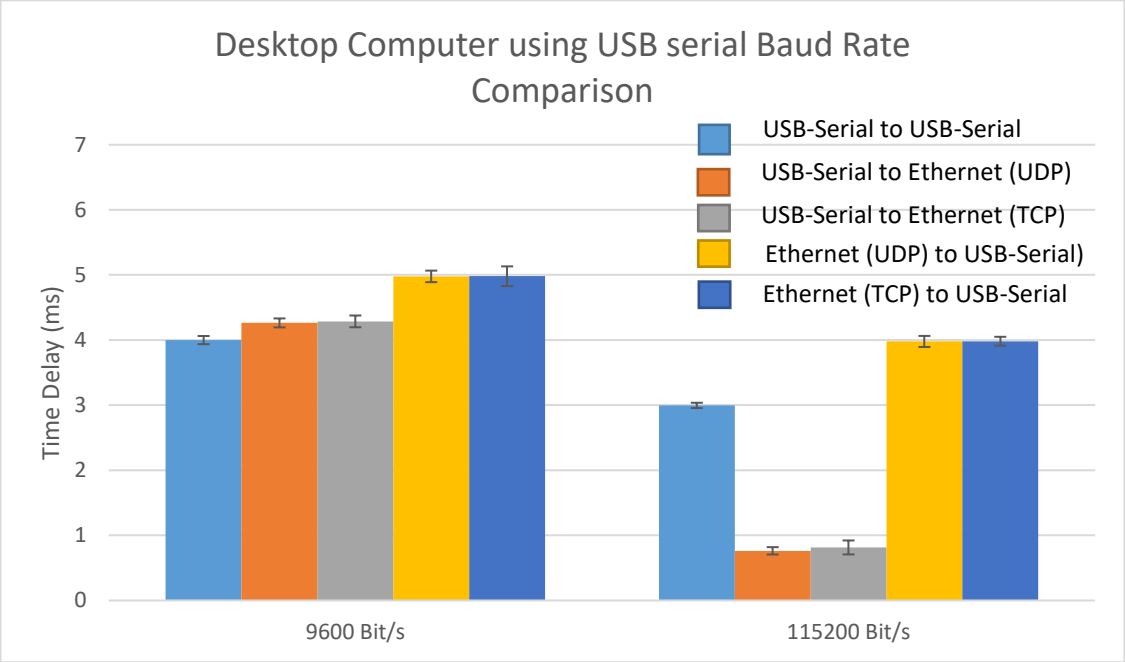


Figure 18 Desktop computer tests using the same USB-Serial converter as for the laptop tests

Wherever the USB-Serial adapter is used to receive event markers, the latency is high and comparable to that of the Lenovo and Dell XPS laptops. When native serial ports are used to receive data, the latency drops below one millisecond at the higher baud rates. This data confirms that the USB-Serial adapter is the likely cause for only a small improvement in latency at the faster baud rate. This is most likely due to the use of software based interrupts in USB-Serial converters. Native serial ports use hardware interrupts, which are more reliable and efficient. This result indicates that the USB-Serial converter has a strong influence on the round-trip time.

All latencies found were less than 7 ms which is more than acceptable for fNIRS since its sampling rate is not very high. EEG is high resolution, and requires at the very least under 2 ms latencies, which was only achieved in Ethernet receiving configurations

at 115,200 bps for the laptops using USB-Serial converters. However, at 115,200 bps on the desktop computer with native serial ports, all round-trip times were sub-millisecond, which is appropriate for EEG sampled at 1000 Hz or fNIRS.

This histogram for all the tests shows a Gaussian distribution for the latencies. An example is shown in Fig. 19. The standard deviation for the tests were all sub millisecond, indicating that the latencies are steady around a certain point. A summary of the results for each of the computers is presented in Tables 3–5 depicting the average latencies with standard deviation.

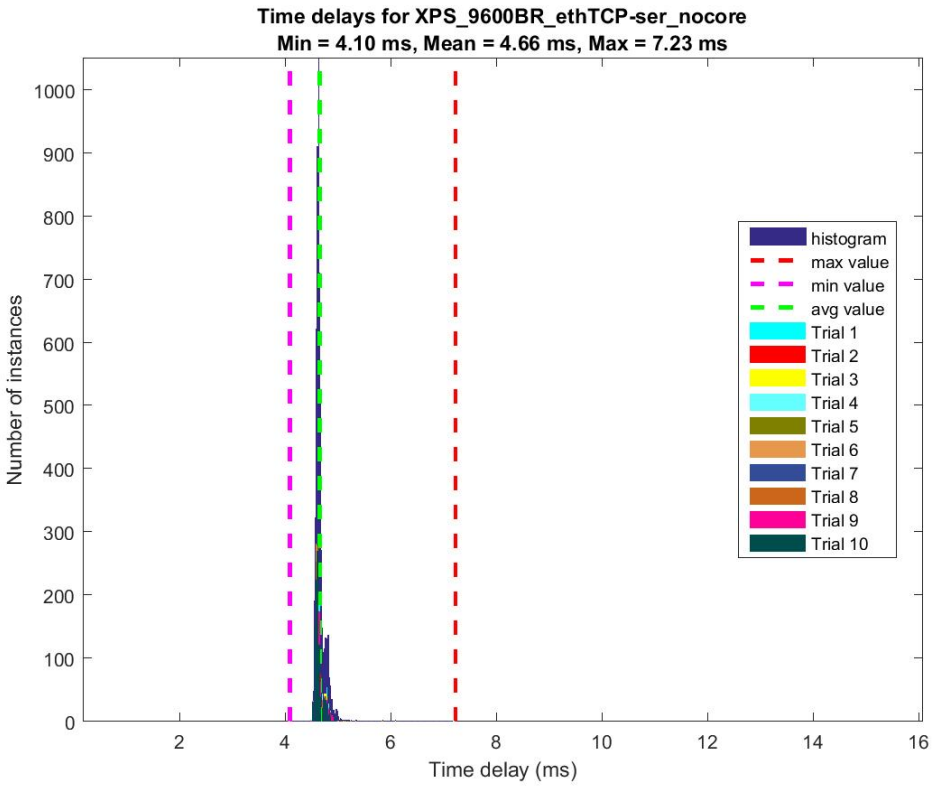


Figure 19 Histogram of latencies for Dell XPS at 9600 bps for test configuration 1

Table 3 Summary of tests for Lenovo laptop

Test type	Lenovo P400 Ideapad (avg ms)		
	Baud Rate (bps)	UDP	TCP
Ethernet to Serial	9600	4.97 ± 0.11	4.73 ± 0.09
	115200	3.75 ± 0.08	3.77 ± 0.08
Serial to Ethernet	9600	4.64 ± 0.10	4.81 ± 0.10
	115200	0.54 ± 0.08	0.65 ± 0.11
Ethernet to Ethernet	NA	0.58 ± 0.08	0.62 ± 0.09
Serial to Serial	9600	4.01 ± 0.10	
	115200	2.85 ± 0.07	

Table 4 Summary of tests for Dell XPS laptop

Test type	Dell XPS laptop (avg ms)		
	Baud Rate (bps)	UDP	TCP
Ethernet to Serial	9600	4.64 ± 0.07	4.66 ± 0.09
	115200	3.69 ± 0.16	3.71 ± 0.08
Serial to Ethernet	9600	4.89 ± 0.07	4.94 ± 0.10
	115200	0.69 ± 0.07	1.37 ± 0.17
Ethernet to Ethernet	NA	0.77 ± 0.09	0.78 ± 0.13
Serial to Serial	9600	4.08 ± 0.06	
	115200	2.93 ± 0.05	

Table 5 Summary of tests for desktop computer

Test type	Desktop computer 9600 bps (avg ms)		
	Baud Rate (bps)	**UDP**	**TCP**
Ethernet to Serial	9600	6.74 ± 0.09	6.77 ± 0.19
	115200	1.22 ± 0.08	1.24 ± 0.07
Ethernet to USB Serial	9600	4.97 ± 0.07	4.98 ± 0.06
	115200	3.98 ± 0.05	3.98 ± 0.04
Serial to Ethernet	9600	4.68 ± 0.07	4.74 ± 0.09
	115200	0.67 ± 0.06	0.78 ± 0.11
USB Serial to Ethernet	9600	4.26 ± 0.29	4.29 ± 0.23
	115200	0.76 ± 0.09	0.81 ± 0.46
Ethernet to Ethernet	NA	0.52 ± 0.08	0.54 ± 0.08
Serial to Serial	9600	6.05 ± 0.06	
	115200	0.52 ± 0.04	
USB Serial to USB Serial	9600	3.99 ± 0.03	
	115200	2.99 ± 0.05	
Serial to USB Serial	9600	4.04 ± 0.29	
	115200	3.94 ± 0.01	
USB serial to serial	9600	5.72 ± 0.28	
	115200	0.61 ± 0.03	

It was additionally found that the type of USB-Serial converter would affect the round-trip times. Changing to the Sabrent USB-Serial converter with the same driver installed as the Prolific 2303 USB-Serial converter reduced the latencies in the serial receiving cases, as shown in Figs. 20 and 21 for the laptops. However, there also appears

to be a higher standard deviation with the Sabrent, as indicated clearly by the error bars in Fig. 21.

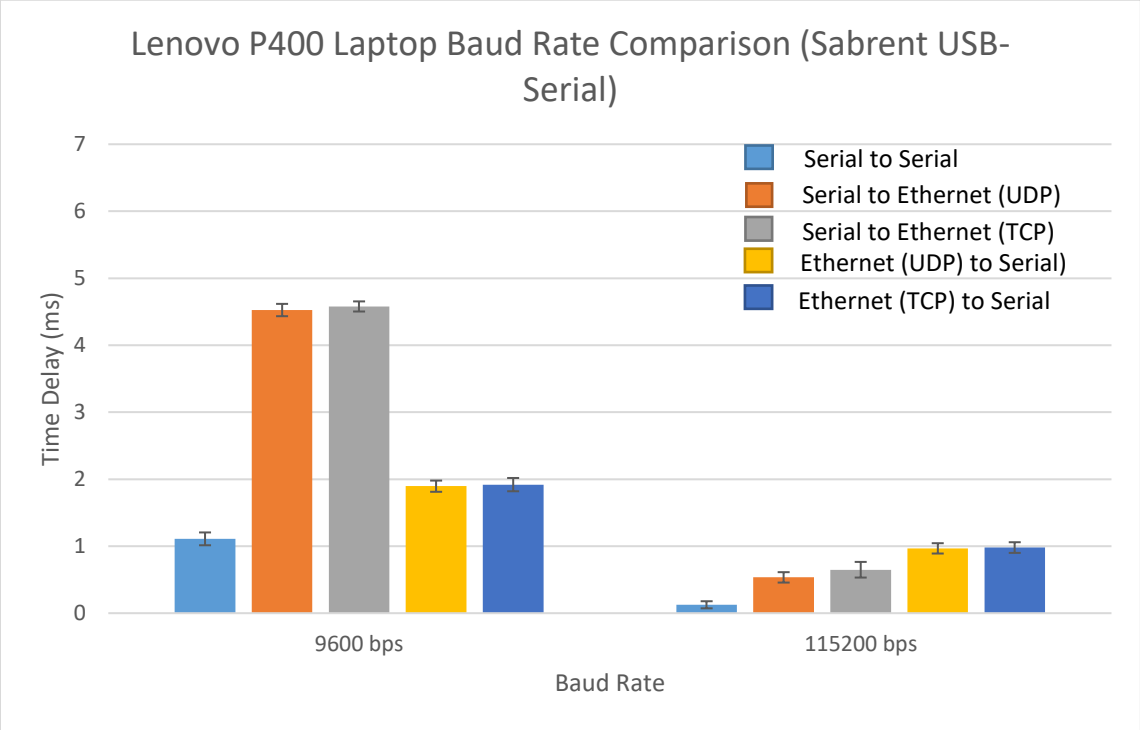


Figure 20 Sabrent USB-Serial Lenovo round trip times without core module

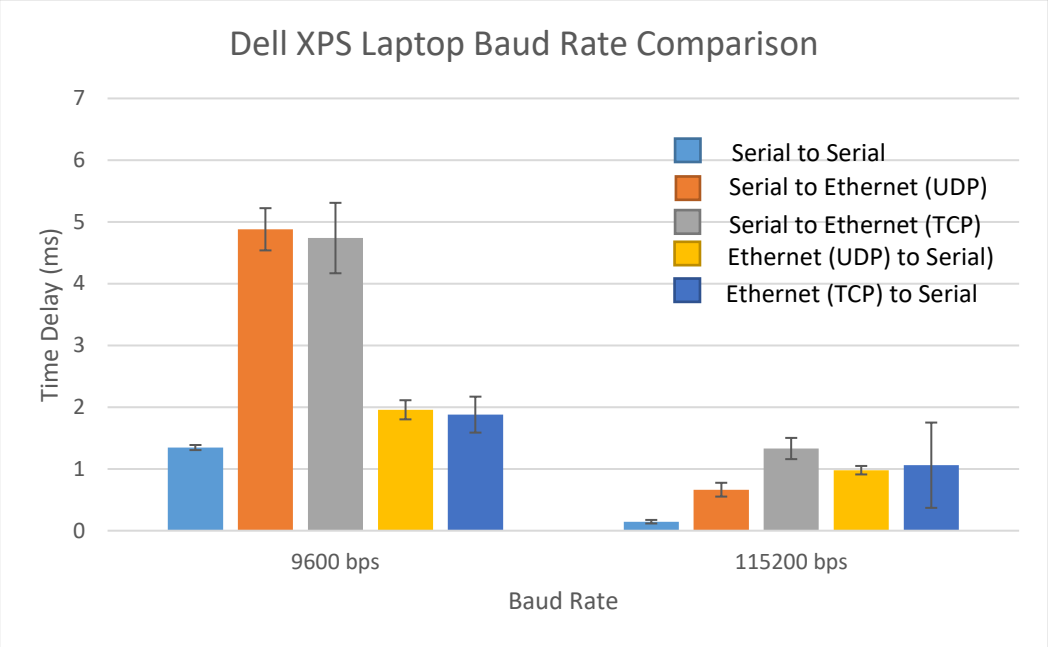


Figure 21 Sabrent USB-Serial Dell XPS round trip times without core module

As shown by figures 18, 20, and 21, the USB-Serial converter does affect the round-trip times, and this should be taken into consideration when using computers without native serial ports.

The next round of tests was to investigate the latency that arises when multiple clients are sending event markers at the same time. 2, 3, and 4 clients were tested on version 1 and version 2 network module in Ethernet and wireless mode (see Figs. 11 and 12 for the configuration diagram). The results for each version and mode is shown in Figs. 22 – 24. Each graph has a trendline to show the effect for each additional client.

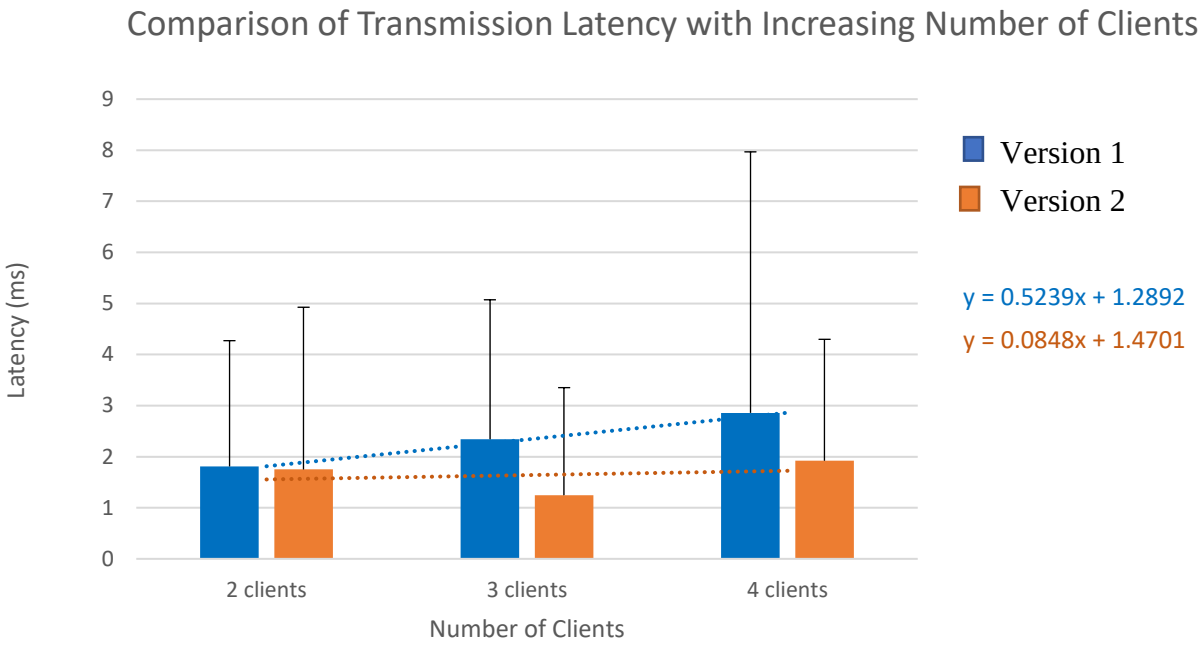


Figure 22 Network module Ethernet mode latencies for simulateanou byte transmission on multiple clients

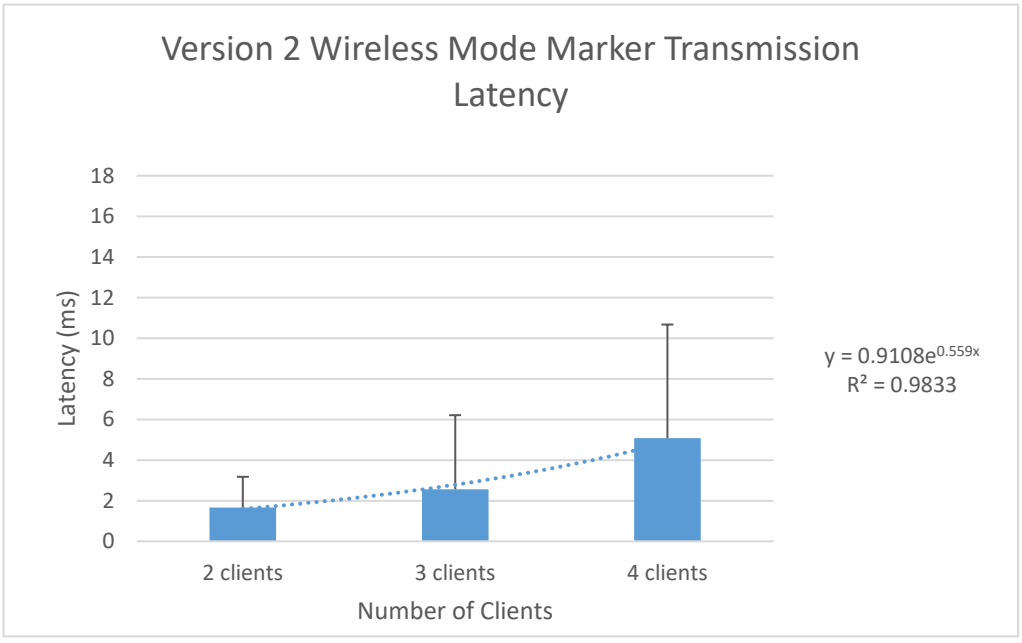


Figure 23 Latency for multiple clients in wireless mode

Out of all the multi-client configurations, Version 2 Ethernet Mode provided the best results with the smallest and most consistent latencies across different numbers of clients, and the lowest standard deviations. Its trendline indicates a slower increase in latency for additional clients compared to Version 1 and Wireless mode. Version 1 Ethernet mode had a worse performance than Version 2 Ethernet mode because of the older Raspberry Pi model. The Raspberry Pi Model 1 B used in Version 1 has a slower CPU and only one core, whereas the Model 3 B has a significantly faster CPU and four cores. The larger number of cores allows the latencies in version 2 Ethernet mode to be stabilized even with an increasing number of clients.

The wireless mode provided similar latencies for 2 and 3 clients as Version 1 Ethernet mode, but the additional fourth client doubles the latency compared to the other configurations, as shown in Fig. 23. Even though the Model 3 B is being used, Wi-Fi connection is inherently slower and less reliable than a wired connection. The wireless connection latency for 3 clients is nearly the same as the version 1 Ethernet mode for 3 clients. This actually indicates that the Model 3 B is still outperforming the Model 1 B, because in Ethernet mode, 1 of the clients is communicating through the wired connection, while the other 2 are through a wireless peer-to-peer connection that relays through Ethernet. On the other hand, in wireless mode, all 3 clients are connected with an ad-hoc network. It should be noted that the Ethernet mode results presented are a worst-case scenario. It is possible to connect the Network Module to a router instead of using ad-hoc networks, which will decrease the latencies. Table 6 discusses the pros and cons for each version and mode.

Table 6 Pros and cons for each network module version and mode

Version and Mode	Pros	Cons
Version 1 Ethernet Mode	• Most tedious to implement in terms of plug-and-play • Smaller latency than Version 2 wireless mode	• Variability increases with more clients • Additional clients show exponential increase in latency • Wired connection required
Version 2 Ethernet Mode	• Most tedious to implement in terms of plug-and-play • Latency under 3 ms for 4 clients or less	• Most reliable (smallest standard deviation) • Additional clients do not cause large increase in latency • Wired connection required
Version 2 Wireless Mode	• Easiest to implement in terms of plug-and-play • No wires needed • Latency under 3 ms for 3 clients or less	• Most unreliable (highest standard deviation) • Large (exponential) increase in latency with additional clients

4.4 Model

The round-trip times for all test configruations was measured, but the individual subcomponents that make up that total round-trip time are not immediately known or easily measured. It is possible to create a system of equations for each test configuration and solve this system to characterize the latencies within just the network and core modules alone. For example, to determine the latency of the core module, round trip measurements were taken with and without the core module. All equations are as follows:

$$NE_r + NE_s + CE_r \quad (1)$$

$$NE_r + NS_s + NC + CS_r \quad (2)$$

$$NC + NS_r + NE_s + CE_r \quad (3)$$

$$NC + CS_r \quad (4)$$

$$CE_r \approx 0 \quad (5)$$

$$NE_r + NS_s + CS_r \quad (6)$$

$$NS_r + NE_s + CE_r \quad (7)$$

$$CS_r \quad (8)$$

NE_r signifies the time it takes to receive a byte through Ethernet on the network module. Similarly, NE_s is the time it takes to send the byte through Ethernet on the network module. CE_r is the time it takes for the C# program to receive the byte via Ethernet. Equation 1 is represented by Fig. 9A. NS_S is the time it takes to send the byte through the serial port on the network module, and CS_r is the time it takes for the C# program to receive the byte on the serial port. NC is the amount of time for the core module to receive and send a byte. Equation 2 is represented by Fig. 7A. NS_r is the amount of time it takes to receive one byte on the serial port in the network module. Equation 3 is visually depicted in Fig. 8A. Equation 4 is depicted by Fig. 10A, and Equation 5 by Fig. 9B. The variables are summarized in Table 7. Equations 6 – 8 are the counterparts to Equations 2 – 4 since they forgo the use of the NeuroHub core module, and are depicted in Figs. 7B, 8B, and 10B.

Table 7 Variable summary

Variable Name	Description
NE_r	Network module receive via Ethernet
NE_s	Network module send via Ethernet
CE_r	C# program receive via Ethernet
NS_s	Network module send via serial
NC	NeuroHub Core module receive and send
NS_r	Network module receive via serial
CS_r	C# program receive via serial

The goal was to determine the latencies of each of the variables to understand the characteristics of the entire system. Finding NC was straightforward, since it is simply a matter of subtracting Equations 6 – 8 from their counterparts in Equations 2 – 4.

As seen in the example for the Lenovo laptop at 9600 bps in Fig. 24 for the Ethernet to serial test configuration, there is about a 1 ms difference between including the core module and forgoing it. Similar results are seen across the different computers and test configurations involving the serial port. The mean times are summarized in Table 8.

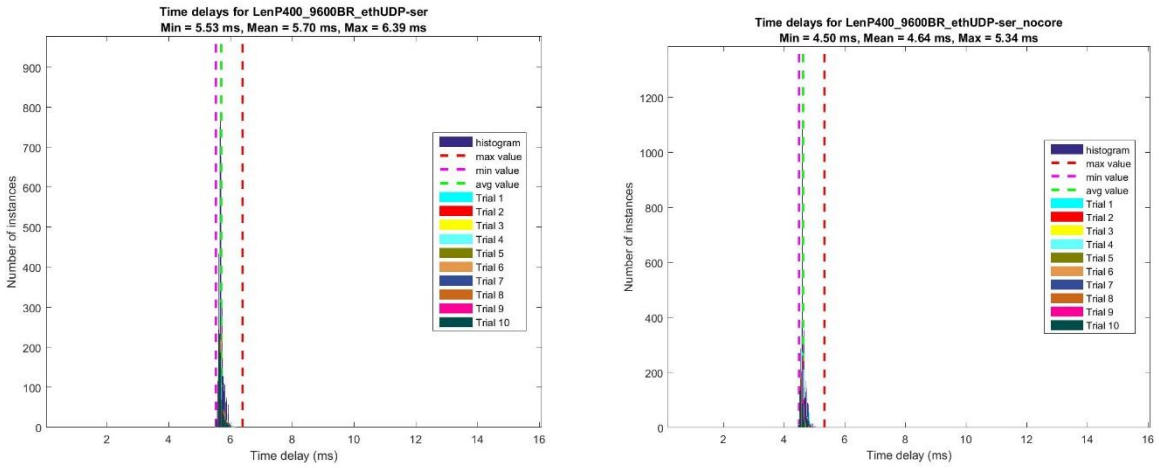


Figure 24 Comparison of latencies A) with and B) without the core module for Lenovo laptop, test configuration #1 at 9600 bps

Table 8 Average NeuroHub core module latencies

Computer Type	NC Mean and standard deviation
Lenovo laptop	1.064 ± 0.11
Dell XPS laptop	1.028 ± 0.02
Desktop	1.044 ± 0.09
Average: 1.05 ± 0.08 ms	

The average NeuroHub core module latency found was 1.05 ms. There is a 2.94% error from the expected value found in Grzeczkowski & Ayaz's previous work of 1.02 ms. This error is most likely attributed to the fact that the measurement done in the previous work was with the oscilloscope, while the 1.05 ms measurement was found with software measurement.

In order to characterize the latencies within just the network module, equations 5 and 8 were subtracted from equations 1, 6, and 7. This isolates the network module terms for each of the test configurations. This was done for all test computers. Figs. 25 - 27 shows the latencies for solely the network module across the three computers.

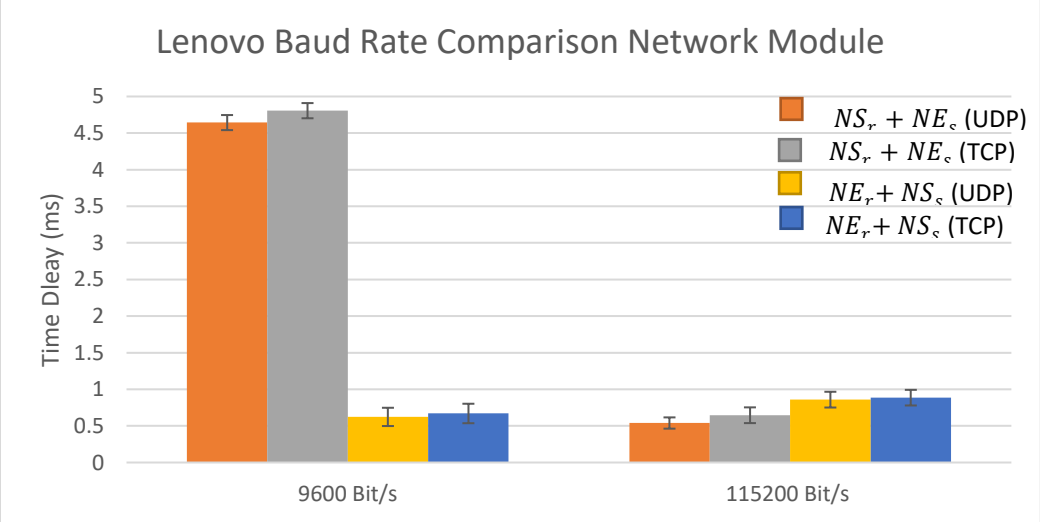


Figure 25 Network Module Latencies for Lenovo laptop

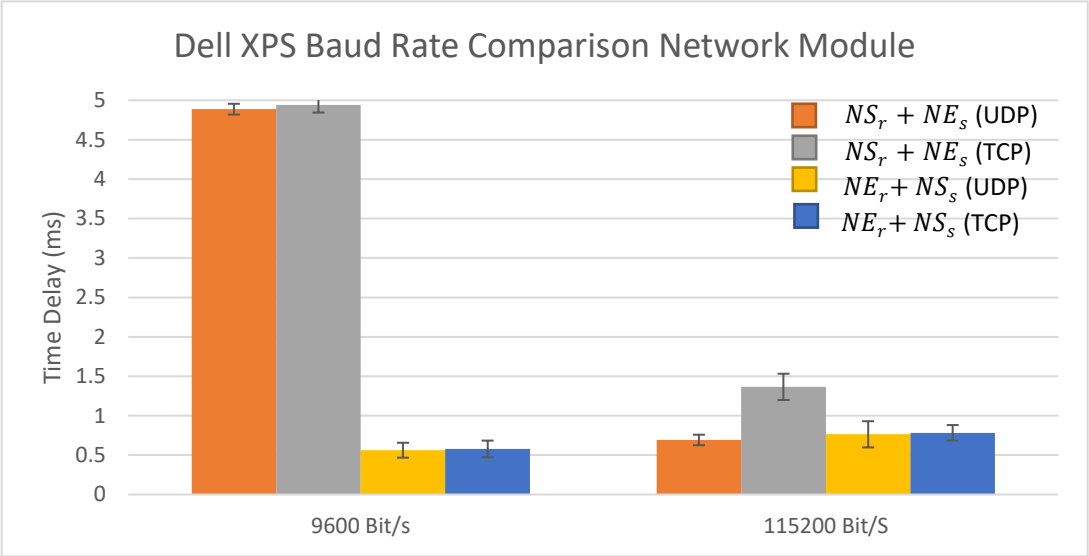


Figure 26 Network Module latencies on Dell XPS laptop

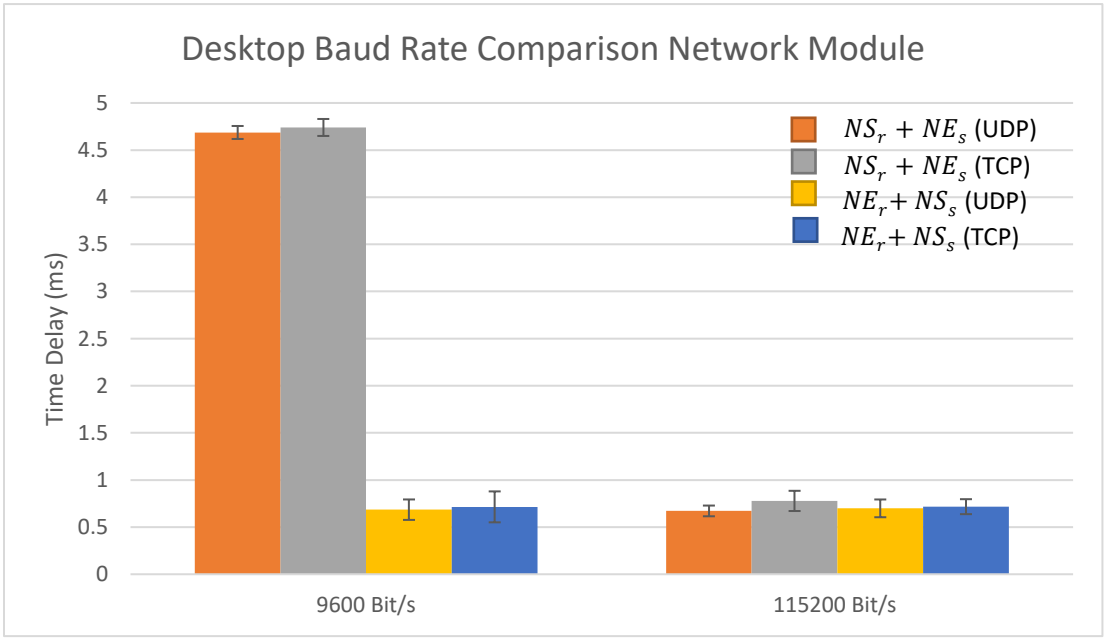


Figure 27 Network Module latencies for Desktop

The NeuroHub network module was consistent across the laptops and the desktop. This verifies that the network module works the same regardless of what computers it is connected to. See Fig. 17 for the network module latencies for Ethernet only tests on all three computers. Figs. 28-29 show the average latencies across all computers, and Table 7 summarizes the variables and their associated average latencies.

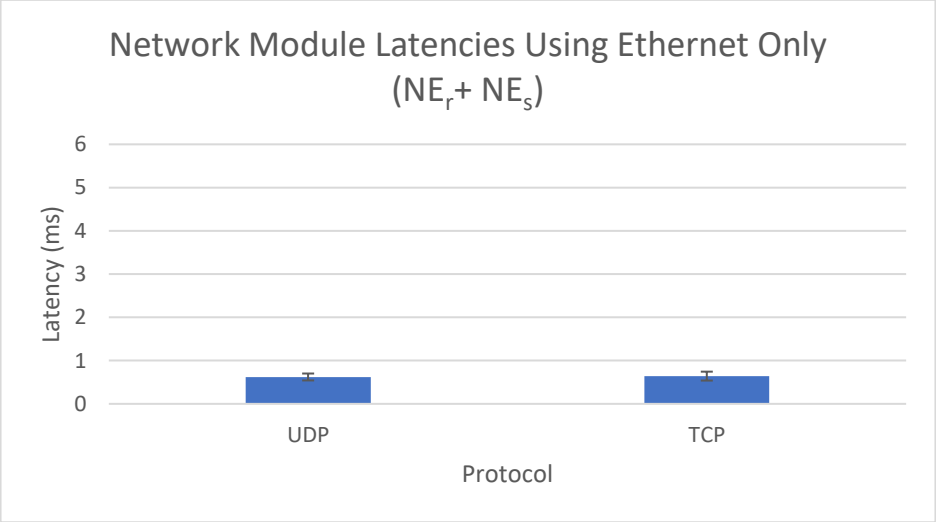


Figure 28 Average Latencies within Network Module for Ethernet only tests

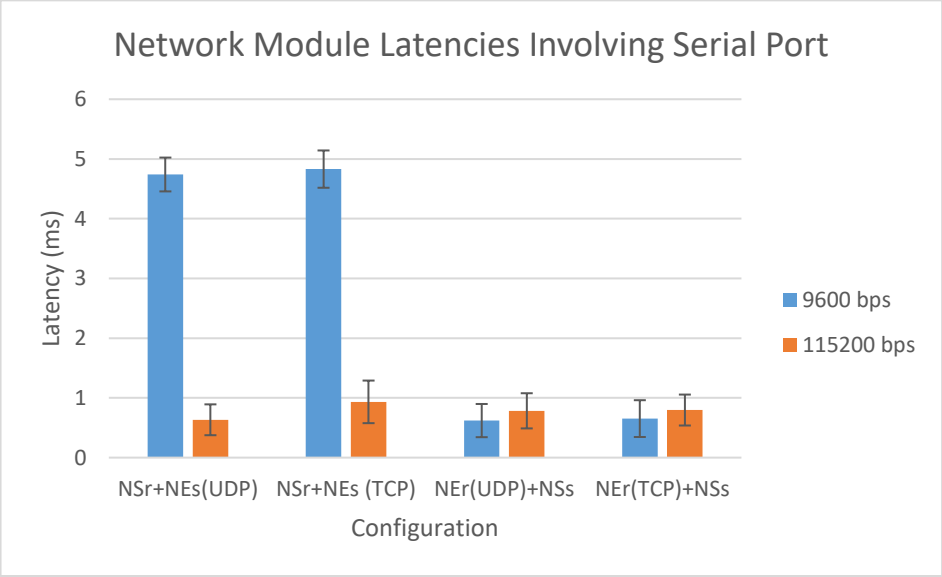


Figure 29 Average Network Module Latencies for All Computers

Table 9 NeuroHub Network Module Average Latencies

Variable	Network Protocol	Baud Rate (bps)	Latency (ms)
$NE_r + NE_s$	TCP	----------	0.64 ± 0.10
	UDP	----------	0.62 ± 0.08
$NS_r + NE_s$	TCP	9600	4.83 ± 0.31
		115200	0.93 ± 0.36
	UDP	9600	4.74 ± 0.28
		115200	0.63 ± 0.26
$NE_r + NS_s$	TCP	9600	0.65 ± 0.31
		115200	0.80 ± 0.26
	UDP	9600	0.62 ± 0.28
		115200	0.78 ± 0.29
NC	-------	9600	1.05 ± 0.08

4.5 Discussion

The NeuroHub network module provides a plug-and-play mechanism to send event markers with minimal latency through TCP, UDP, and serial port. It can be connected to the NeuroHub core module for additional serial ports, TTL, or parallel port. It was found that the latencies across the laptops were very similar due to the use of the USB-serial converters. Additionally, the type of USB-Serial converter will affect the latency. Using native serial ports, however, had much more improved latencies at the higher baud rate for the configurations with receiving on the serial port. It is advisable to use native serial ports over USB-serial converters since there may be unknown lag based on the type of converter. This is because native serial ports use hardware interrupts to receive data, while USB-serial converters are using software interrupts. Hardware-dedicated interrupts are more reliable than software interrupts. Overall, the network module achieved consistent latencies below 7 ms and for certain configurations, below 1 ms.

5. Representative Use Case

5.1 Introduction

To test how the device can be used in a real-world research application, a simple test experiment was done using OpenViBE and COBI Studio. OpenViBE is an open-source software platform designed to record, filter, process, classify and visualize brain signals in real time and is customizable through graphical programming. Users who do not have programming experience can still use OpenViBE. Users drag and drop desired boxes into their scenario window and connect the boxes to complete the signal flow. An OpenViBE scenario using a P300 speller was used for the demonstration. The event markers are automatically generated by OpenViBE. These markers were sent to the NeuroHub network module through TCP. To do this, a custom OpenViBE box was coded in C++ to interface with the network module. Markers received from the network module were then sent via serial port at 9,600 bps and 115,200 bps to COBI studio, the fNIRS recording software designed and developed at Drexel University. COBI studio received the markers through an FTDI based USB-Serial converter, because it worked the most reliably with COBI studio. This test setup highlights the ability of the NeuroHub network module to convert one type of data stream into another. In this setup, Version 1 Ethernet Mode is used. The configuration is shown in Fig. 30. The event markers were logged by OpenViBE and COBI studio in separate files. To ascertain the delay between sending the marker from OpenViBE to COBI studio via the network module, the time differential from the first marker in OpenViBE was computed for all the following markers and compared to that of the markers logged in COBI studio.

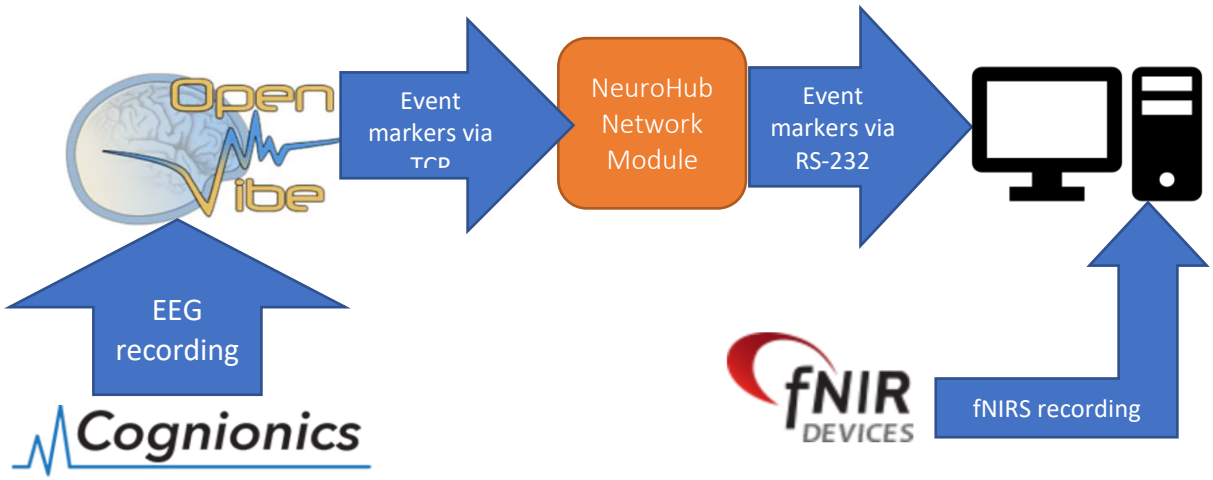


Figure 30 Flow diagram for use case with OpenViBE and COBI Studio

5.2 Results

Fig. 31 shows the histogram of delays between OpenViBE and COBI. The average latency for transmission for both baud rates combined was 0.54 ms ± 0.37 ms. This result confirms that the network module is able to provide submillisecond latencies.

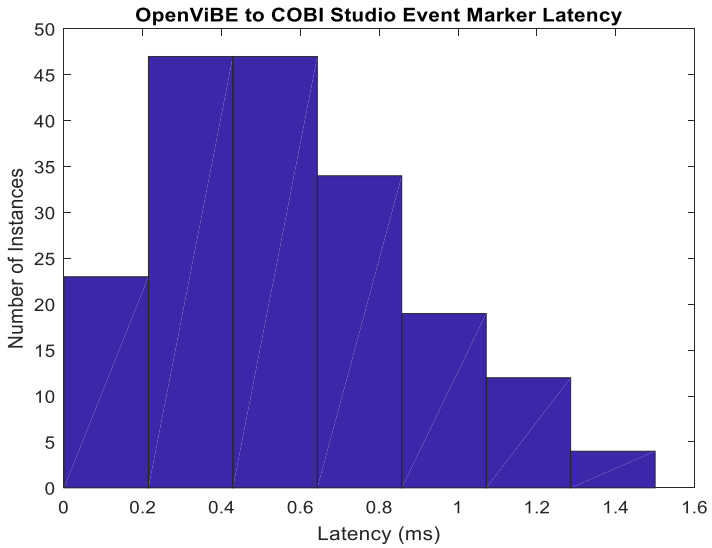


Figure 31 Average time differences between OpenViBE and COBI studio temporal markers at 9.6 and 115.2 kbps

6. Conclusion and Future Work

6.1 Conclusion

The NeuroHub network module supports TCP, UPD, and serial port transmission of event markers values between 1 and 255. The number of networked clients. baud rate, and networking protocol is customizable through a GUI. It has been established that the latency while using the network module version 1 will be less than 7 ms, and in most cases sub millisecond, which is appropriate for the temporal needs of both fNIRS and EEG. Upgrading to Version 2 further improved the latencies due to the faster Raspberry Pi model used. Version 2 Ethernet mode has proven to be the most scalable version since the addition of multiple clients does not substantially change the average latency. In summation, the device met all the desired requirements.

6.2 Future Work

There are various features and approaches that could improve current NeuroHub. One coding improvement for the NeuroHub Networking Module would be to enable use of the USB ports on the Raspberry Pi to send or receive markers through a virtual serial port. This could be used in setups where a USB stick is used to wirelessly send markers to the software, such as with Cognionics EEG acquisition. A dynamic library link (DLL) should be created so that code for existing programs can easily be configured to be able to connect, send, and receive markers to the network module through TCP. The server code on the Raspberry Pi could further be improved by applying a real-time patch to the Linux kernel that the Pi runs on so that latencies can be reduced. Linux is not a real- time operating system, so the time it takes to finish a task is not important and is not strictly overseen as in a real-time operating system. The NeuroHub core module can also be

improved by changing two of the serial port connections from female to male; this will allow more flexible setups based on whatever serial cables the lab has available. The core module's baud rate should also be increased from 9,600 bps to decrease the latency from its current 1 ms setpoint.

The NeuroHub Network Module serves to bridge the divide between older and newer systems. With its serial port interface, it can easily attach to the core module which provides legacy port interface, and with its networking capability, it is able to communicate with newer protocols as well. The future of communication technology is the universal port, a single port that can be used for multiple purposes. An example of this kind of technology is USB-C ports. Many newer computers have USB-C charging and this port can also be used to receive and transmit data. Apple has already attempted to eliminate the use of different ports for different devices by allowing users to connect headphones and charge their Apple products through their patented lightning connector. Technologies like Wi-Fi direct, Bluetooth are also becoming more prevalent as methods to connect multiple devices together. Although neuroimaging systems and their synchronization methods will not change in the short-term, the long-term future goals for the NeuroHub network module is to consider these new technologies and how they can be implemented to serve the needs of both old and emerging brain and body research systems.

List of References

IEEE 1284: Parallel Ports. (2002). Retrieved from http://lavalink.com/wp-content/uploads/white_papers/ieee1284_parallel_ports.pdf

How TCP/IP Works: TCP/IP. (2003). Retrieved from https://technet.microsoft.com/en-us/library/cc786128(v=ws.10).aspx

Aghajani, H., & Omurtag, A. (2016, 16-20 Aug. 2016). *Assessment of mental workload by EEG+FNIRS.* Paper presented at the 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).

Allison, B. Z., Wolpaw, E. W., & Wolpaw, J. R. (2007). Brain-computer interface systems: progress and prospects. *Expert review of medical devices, 4*(4), 463-474. doi:10.1586/17434440.4.4.463

Al-shargie, F., Tang, T. B., Badruddin, N., Dass, S. C., & Kiguchi, M. (2016). *Mental stress assessment based on feature level fusion of fNIRS and EEG signals*.

Axelson, J., & Books24x, I. (2007). *Serial port complete: COM ports, USB virtual COM ports, and ports for embedded systems, second edition* (2nd ed.). Madison, Wis: Lakeview Research.

Ayaz, H., Izzetoglu, M., Shewokis, P. A., & Onaral, B. (2010). Sliding-window motion artifact rejection for Functional Near-Infrared Spectroscopy. *Conf Proc IEEE Eng Med Biol Soc, 2010*, 6567-6570. doi:10.1109/iembs.2010.5627113

Ayaz, H., & Onaral, B. (2009). Assessment of Cognitive Neural Correlates for a Functional Near Infrared-Based Brain Computer Interface System (Vol. 5638, pp. 699-708). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ayaz, H., Onaral, B., Izzetoglu, K., Shewokis, P. A., McKendrick, R., & Parasuraman, R. (2013). Continuous monitoring of brain dynamics with functional near infrared spectroscopy as a tool for neuroergonomic research: empirical examples and a technological development. *Frontiers in human neuroscience, 7*, 871. doi:10.3389/fnhum.2013.00871

Ayaz, H., Shewokis, P. A., Bunce, S., Izzetoglu, K., Willems, B., & Onaral, B. (2012). Optical brain monitoring for operator training and mental workload assessment. *Neuroimage, 59*(1), 36-47. doi:10.1016/j.neuroimage.2011.06.023

Ayaz, H., Shewokis, P. A., Bunce, S., & Onaral, B. (2011). An optical brain computer interface for environmental control. *Conf Proc IEEE Eng Med Biol Soc, 2011*, 6327-6330. doi:10.1109/iembs.2011.6091561

Ayaz, H., Shewokis, P. A., Curtin, A., Izzetoglu, M., Izzetoglu, K., & Onaral, B. (2011). Using MazeSuite and functional near infrared spectroscopy to study learning in spatial navigation. *J Vis Exp*(56). doi:10.3791/3443

Babiloni, F., & Astolfi, L. (2014). Social neuroscience and hyperscanning techniques: past, present and future. *Neuroscience and biobehavioral reviews, 44*, 76-93. doi:10.1016/j.neubiorev.2012.07.006

Batula, A. M., Kim, Y. E. & Ayaz, H. (2017). Virtual and Actual Humanoid Robot Control with Four-Class Imagery Based Optical Brain Computer Interface. *BioMed Research International*.

Batula, A. M., Mark, J., Kim, Y. E., & Ayaz, H. (2016). Developing an Optical Brain-Computer Interface for Humanoid Robot Control. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Foundations of Augmented Cognition: Neuroergonomics and Operational Neuroscience: 10th International Conference, AC 2016, Held as Part of HCI International 2016, Toronto, ON, Canada, July 17-22, 2016, Proceedings, Part I* (pp. 3-13). Cham: Springer International Publishing.

Batula, A. M., Mark, J. A., Kim, Y. E., & Ayaz, H. (2017). Comparison of Brain Activation during Motor Imagery and Motor Movement Using fNIRS. *Comput Intell Neurosci, 2017*, 5491296. doi:10.1155/2017/5491296

Berka, C., Levendowski, D. J., Cvetinovic, M. M., Petrovic, M. M., Davis, G., Lumicao, M. N., . . . Olmstead, R. (2004). Real-Time Analysis of EEG Indexes of Alertness, Cognition, and Memory Acquired With a Wireless EEG Headset. *International Journal of Human-Computer Interaction, 17*(2), 151-170. doi:10.1207/s15327590ijhc1702_3

Cao, K., Guo, Y., & Su, S. W. (2015, 8-10 Dec. 2015). *A review of motion related EEG artifact removal techniques.* Paper presented at the 2015 9th International Conference on Sensing Technology (ICST).

Chance, B. (1991). Optical method. *Annu Rev Biophys Biophys Chem, 20*, 1-28. doi:10.1146/annurev.bb.20.060191.000245

Chance, B., Zhuang, Z., UnAh, C., Alter, C., & Lipton, L. (1993). Cognition-activated low-frequency modulation of light absorption in human brain. *Proceedings of the National Academy of Sciences of the United States of America, 90*(8), 3770-3774.

Choi, I., Rhiu, I., Lee, Y., Yun, M. H., & Nam, C. S. (2017). A systematic review of hybrid brain-computer interfaces: Taxonomy and usability perspectives. *PLOS ONE, 12*(4), e0176674. doi:10.1371/journal.pone.0176674

Fazel-Rezai, R., Allison, B., Guger, C., Sellers, E., Kleih, S., & Kübler, A. (2012). P300 brain computer interface: current challenges and emerging trends. *Frontiers in Neuroengineering, 5*(14). doi:10.3389/fneng.2012.00014

Grzeczkowski, N. V., Ayaz, H., Drexel University. School of Biomedical Engineering, S., & Health, S. (2014). *NeuroHub: Portable and Scalable Time Synchronization Instrument for Brain-Computer Interface and Functional Neuroimaging Research.* (Dissertation/Thesis). Retrieved from http://records.library.drexel.edu/record=b2487497

Harrivel, A. R., Weissman, D. H., Noll, D. C., & Peltier, S. J. (2013). Monitoring attentional state with fNIRS. *Frontiers in human neuroscience, 7*, 861. doi:10.3389/fnhum.2013.00861

Hirsch, J., Zhang, X., Noah, J. A., & Ono, Y. (2017). Frontal temporal and parietal systems synchronize within and across brains during live eye-to-eye contact. *Neuroimage, 157*, 314-330. doi:https://doi.org/10.1016/j.neuroimage.2017.06.018

Hirshfield, L. M., Chauncey, K., Gulotta, R., Girouard, A., Solovey, E. T., Jacob, R. J. K., . . . Fantini, S. (2009). Combining Electroencephalograph and Functional Near Infrared Spectroscopy to Explore Users' Mental Workload. In D. D. Schmorrow, I. V. Estabrooke, & M. Grootjen (Eds.), *Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience: 5th International Conference, FAC 2009 Held as Part of HCI International 2009 San Diego, CA, USA, July 19-24, 2009 Proceedings* (pp. 239-247). Berlin, Heidelberg: Springer Berlin Heidelberg.

Huang, D., Qian, K., Fei, D.-Y., Jia, W., Chen, X., & Bai, O. (2012). Electroencephalography (EEG)-Based Brain-Computer Interface (BCI): A 2-D Virtual Wheelchair Control Based on Event-Related Desynchronization/Synchronization and State Control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, 20*(3), 379-388. doi:10.1109/TNSRE.2012.2190299

Izzetoglu, K., Ayaz, H., Merzagora, A., Izzetoglu, M., Shewokis, P. A., Bunce, S. C., . . . Onaral, B. (2011). The Evolution of Field Deployable fNIR Spectroscopy from Bench to Clinical Settings. *Journal of Innovative Optical Health Sciences, 04*(03), 239-250. doi:10.1142/s1793545811001587

Izzetoglu, M., Izzetoglu, K., Bunce, S., Ayaz, H., Devaraj, A., Onaral, B., & Pourrezaei, K. (2005). Functional near-infrared neuroimaging. *IEEE Trans Neural Syst Rehabil Eng, 13*(2), 153-159. doi:10.1109/tnsre.2005.847377

Jaseja, H. (2015). Intractability in Epilepsy: Role of EEG Desynchronization in Early Identification. *Clinical EEG and Neuroscience, 46*(3), 266-267. doi:10.1177/1550059414528311

Kennett, R. (2012). Modern electroencephalography. *Journal of Neurology, 259*(4), 783-789. doi:10.1007/s00415-012-6425-6

Kothe, C. (2013). Lab Streaming Layer. University of California San Diego.

Kothe, C. A., & Makeig, S. (2013). BCILAB: a platform for brain-computer interface development. *J Neural Eng, 10*(5), 056014. doi:10.1088/1741-2560/10/5/056014

Lalitharatne, T. D., Teramoto, K., Hayashi, Y., & Kiguchi, K. (2013). Towards Hybrid EEG-EMG-Based Control Approaches to be Used in Bio-robotics Applications: Current Status, Challenges and Future Directions. *Paladyn, Journal of Behavioral Robotics, 4*(2). doi:10.2478/pjbr-2013-0009

Lancaster, D. (1974). *TTL Cookbook*. Carmel, Indiana: SAMS.

Leamy, D. J., Collins, R., & Ward, T. E. (2011). *Combining fNIRS and EEG to improve motor cortex activity classification during an imagined movement-based task.* Paper presented at the International Conference on Foundations of Augmented Cognition.

Lebedev, M. A., & Nicolelis, M. A. (2017). Brain-Machine Interfaces: From Basic Science to Neuroprostheses and Neurorehabilitation. *Physiol Rev, 97*(2), 767-837. doi:10.1152/physrev.00027.2016

Li, Y., Long, J., Yu, T., Yu, Z., Wang, C., Zhang, H., & Guan, C. (2010). An EEG-Based BCI System for 2-D Cursor Control by Combining Mu/Beta Rhythm and P300 Potential. *IEEE Transactions on Biomedical Engineering, 57*(10), 2495-2505. doi:10.1109/TBME.2010.2055564

Liu, Y., Ayaz, H., Curtin, A., Onaral, B., & Shewokis, P. A. (2013). Towards a Hybrid P300-Based BCI Using Simultaneous fNIR and EEG. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Foundations of Augmented Cognition: 7th International Conference, AC 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013. Proceedings* (pp. 335-344). Berlin, Heidelberg: Springer Berlin Heidelberg.

Liu, Y., Ayaz, H., Onaral, B., & Shewokis, P. A. (2015). Neural Adaptation to a Working Memory Task: A Concurrent EEG-fNIRS Study. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Foundations of Augmented Cognition: 9th International Conference, AC 2015, Held as Part of HCI International 2015, Los Angeles, CA,*

USA, August 2-7, 2015, Proceedings (pp. 268-280). Cham: Springer International Publishing.

Liu, Y., Ayaz, H., & Shewokis, P. A. (2017). Mental workload classification with concurrent electroencephalography and functional near-infrared spectroscopy. *Brain-Computer Interfaces,* 1-11. doi:10.1080/2326263X.2017.1304020

Liu, Y., Piazza, E. A., Simony, E., Shewokis, P. A., Onaral, B., Hasson, U., & Ayaz, H. (2017). Measuring speaker–listener neural coupling with functional near infrared spectroscopy. *Scientific Reports, 7*, 43293. doi:10.1038/srep43293

Mark, J., Thomas, N., Kraft, A., Casebeer, W. D., Ziegler, M., & Ayaz, H. (2018). Neurofeedback for Personalized Adaptive Training. In C. Baldwin (Ed.), *Advances in Neuroergonomics and Cognitive Engineering: Proceedings of the AHFE 2017 International Conference on Neuroergonomics and Cognitive Engineering, July 17–21, 2017, The Westin Bonaventure Hotel, Los Angeles, California, USA* (pp. 83-94). Cham: Springer International Publishing.

McKendrick, R., Parasuraman, R., & Ayaz, H. (2015). Wearable functional near infrared spectroscopy (fNIRS) and transcranial direct current stimulation (tDCS): expanding vistas for neurocognitive augmentation. *Front Syst Neurosci, 9*, 27. doi:10.3389/fnsys.2015.00027

McKendrick, R., Parasuraman, R., Murtza, R., Formwalt, A., Baccus, W., Paczynski, M., & Ayaz, H. (2016). Into the Wild: Neuroergonomic Differentiation of Hand-Held and Augmented Reality Wearable Displays during Outdoor Navigation with Functional Near Infrared Spectroscopy. *Front Hum Neurosci, 10*, 216. doi:10.3389/fnhum.2016.00216

Mihara, M., Hattori, N., Hatakenaka, M., Yagura, H., Kawano, T., Hino, T., & Miyai, I. (2013). Near-infrared spectroscopy-mediated neurofeedback enhances efficacy of motor imagery-based training in poststroke victims: a pilot study. *Stroke; a journal of cerebral circulation, 44*(4), 1091.

Musselman, M., & Djurdjanovic, D. (2012). Time-frequency distributions in the classification of epilepsy from EEG signals. *Expert Systems With Applications, 39*(13), 11413. doi:10.1016/j.eswa.2012.04.023

Naseer, N., & Hong, K.-S. (2015). fNIRS-based brain-computer interfaces: a review. *Frontiers in human neuroscience, 9*, 3. doi:10.3389/fnhum.2015.00003

Okamoto, M., Dan, H., Dan, I., Sakamoto, K., Takeo, K., Shimizu, K., . . . Kohyama, K. (2004). Three-dimensional probabilistic anatomical cranio-cerebral correlation via the international 10–20 system oriented for transcranial functional brain mapping. *Neuroimage, 21*(1), 99-111. doi:10.1016/j.neuroimage.2003.08.026

Pan, Y., Cheng, X., Zhang, Z., Li, X., & Hu, Y. (2017). Cooperation in lovers: An fNIRS-based hyperscanning study. *Human Brain Mapping, 38*(2), 831-841. doi:10.1002/hbm.23421

Pouliot, P., Tran, T. P. Y., Birca, V., Vannasing, P., Tremblay, J., Lassonde, M., & Nguyen, D. K. (2014). Hemodynamic changes during posterior epilepsies: an EEG-fNIRS study. *Epilepsy research, 108*(5), 883-890. doi:10.1016/j.eplepsyres.2014.03.007

Putze, F., Hesslinger, S., Tse, C.-Y., Huang, Y., Herff, C., Guan, C., & Schultz, T. (2014). Hybrid fNIRS-EEG based classification of auditory and visual perception processes. *Frontiers in Neuroscience, 8*, 373. doi:10.3389/fnins.2014.00373

Quaresima, V., & Ferrari, M. (2016). Functional Near-Infrared Spectroscopy (fNIRS) for Assessing Cerebral Cortex Function During Human Behavior in Natural/Social Situations. *Organizational Research Methods, 0*(0), 1094428116658959. doi:doi:10.1177/1094428116658959

Schomer, D. L., & Lopes da Silva, F. (2010). *Niedermeyer's Electroencephalography: Basic Principles, Clinical Applications, and Related Fields* (Sixth;6;6th; ed.). Philadelphia: Wolters Kluwer Health.

Sweeney, K. T., Ayaz, H., Ward, T. E., Izzetoglu, M., McLoone, S. F., & Onaral, B. (2012). A Methodology for Validating Artifact Removal Techniques for Physiological Signals. *IEEE Transactions on Information Technology in Biomedicine, 16*(5), 918-926. doi:10.1109/TITB.2012.2207400

Teo, W.-P., Muthalib, M., Yamin, S., Hendy, A. M., Bramstedt, K., Kotsopoulos, E., . . . Ayaz, H. (2016). Does a Combination of Virtual Reality, Neuromodulation and Neuroimaging Provide a Comprehensive Platform for Neurorehabilitation? – A Narrative Review of the Literature. *Frontiers in human neuroscience, 10*, 284. doi:10.3389/fnhum.2016.00284

Vanutelli, M. E., Pezard, L., Nandrino, J., & Balconi, M. (2016). Intra and inter-brain connectivity during cooperation: a fNIRS-based connectivity analysis. *Neuropsychological Trends*(20), 154-155.

Villringer, A., & Chance, B. (1997). Non-invasive optical spectroscopy and imaging of human brain function. *Trends in Neurosciences, 20*(10), 435-442. doi:https://doi.org/10.1016/S0166-2236(97)01132-6

Villringer, A., Planck, J., Hock, C., Schleinkofer, L., & Dirnagl, U. (1993). Near infrared spectroscopy (NIRS): A new tool to study hemodynamic changes during activation of brain function in human adults. *Neuroscience Letters, 154*(1–2), 101-104. doi:https://doi.org/10.1016/0304-3940(93)90181-J

von Luhmann, A., Wabnitz, H., Sander, T., & Muller, K. R. (2016). M3BA: A Mobile, Modular, Multimodal Biosignal Acquisition architecture for miniaturized EEG-NIRS based hybrid BCI and monitoring. *IEEE Trans Biomed Eng*. doi:10.1109/tbme.2016.2594127

Yin, X., Xu, B., Jiang, C., Fu, Y., Wang, Z., Li, H., & Shi, G. (2015). A hybrid BCI based on EEG and fNIRS signals improves the performance of decoding motor imagery of both force and speed of hand clenching. *Journal of neural engineering, 12*(3), 036004.

Zich, C., Debener, S., Thoene, A.-K., Chen, L.-C., & Kranczioch, C. Simultaneous EEG-fNIRS reveals how age and feedback affect motor imagery signatures. *Neurobiology of Aging, 49*, 183-197. doi:10.1016/j.neurobiolaging.2016.10.011

Appendix A: NeuroHub Network Module Development and Setup

Materials List

1. Raspberry Pi Model 1 B
2. Raspberry Pi Model 3 B
3. SD card adapter x2
4. 8 – 32 GB Micro SDHC x3
5. Raspberry PIIO – MiniPiio RS232 add-on board https://www.tindie.com/products/DTronixs/raspberry-piio-minipiio-rs232-add-on-board/?pt=full_prod_search
6. Raspberry Pi GPIO to DB9M RS232 Serial Board
7. ModMyPi Modular (variable height) case for Raspberry Pi 2/3 complete set https://www.amazon.com/Modular-variable-height-case-Raspberry/dp/B01LYARY5D/ref=sr_1_1?ie=UTF8&qid=1494270226&sr=8-1&keywords=modmypi+case
8. Ableconn PI232DB9K GPIO to DB9M RS232 Serial Stackable Board for Raspberry Pi https://www.amazon.com/Ableconn-PI232DB9K-Serial-Stackable-Raspberry/dp/B01LZO0K2Y/ref=sr_1_2?ie=UTF8&qid=1496342691&sr=8-2&keywords=raspberry+pi+gpio+to+db9m+rs232+serial+board

Setup OS, programs, and files

The initial version uses the Raspberry Pi Model 1 B. This model uses a regular size SD card. Install the Raspbian Jesse OS image onto the SD card, which can be obtained for free from the Raspberry Pi website. The distribution used here was released on March 18,

2016. Other distributions are found here: http://downloads.raspberrypi.org/raspbian/images/ . It is possible to write the image file with Win32DiskImager if using Windows. 3 micro SD cards and 2 full-size SD adapters should ideally be used for each of the configurations – Version 1 Ethernet mode, Version 2 Ethernet mode, and Version 2 wireless mode. Make sure to save the SD card image after making any substantial changes. At the end of development, there will be three different image versions for the SD card.

In the cmdline.txt file of the boot directory on the SD card, append the line ip=192.168.137.99. Plug in the Ethernet cable from the Raspberry Pi to the computer. Enable sharing Internet connection on the Local Area Network (LAN) in the Wi-Fi adapter settings. Go to the properties of the LAN, right click on ipv4, select properties, and select use the following address. Type in 192.168.137.1 with subnet mask 255.255.2525.0 and click ok. Download the free version of MobaXTerm to SSH into the Pi. Create a new session with IP address as 192.168.137.99. To login, user name is pi and password is raspberrypi.

Update pi with

```
sudo-apt get update
sudo-pat get upgrade
```

Then install Wiring Pi, with the following commands:

```
cd
git clone git://git.drogon.net/wiringPi
cd ~/wiringPi
git pull origin
./build
```

Check installation with:

```
gpio -v
gpio readall
```

Next, install gtkmm 3.0 for the GUI using

```
sudo apt-get install libgtkmm-3.0-dev
```

Stop the console from using the serial port by typing

```
sudo systemctl stop serial-getty@ttyAMA0.service
sudo systemctl disable serial-getty@ttyAMA0.service
```

If using Raspberry Pi 3 also do

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

Then edit cmdline with
`sudo nano /boot/cmdline.txt`

Delete the line `console=serial0,115200`

For Raspberry Pi 3
`sudo nano /boot/config.txt`
Add the following line to the end of the file
`dtoverlay=pi3-disable-bt`

Bluetooth uses the high performance GPIO pins that were previously used on the

Raspberry Pi 1 for the serial port.

To configure a static IP address:

```
sudo nano /etc/network/interfaces
```

After `eth0 inet manual` place `dns-nameservers 8.8.8.8 8.8.4.4`

In `/etc/dhcpcd.conf` insert following lines at the very end:

```
interface eth0
static ip address = 192.168.137.99
static routers=192.168.137.1
static domain_name_servers=192.168.137.1
```

Save the new updated image on the SD card to an external hard drive.

Wireless Mode Configuration (Raspberry Pi 3 only)

If trying to configure Version 2 (Raspberry Pi 3) wireless mode perform the following

steps:

```
cd /etc/network
sudo cp interfaces interfaces-wifi
sudo nano interfaces-adhoc
```

Copy the following information into the interfaces-adhoc file

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
address 192.168.137.99
netmask 255.255.255.0
wireless-channel 1
wireless-essid RPIWireless
wireless-mode ad-hoc
```

Ctrl+X, yes and enter to save the file.

Type in the following command to switch to adhoc mode:

```
sudo cp /etc/network/interfaces-adhoc interfaces
```

Switch back to Wi-Fi with

```
sudo cp /etc/network/interfaces-wifi interfaces.
```

Install a package to allow the Pi to assign any connected device an appropriate IP address:

```
sudo apt-get install isc-dhcp-server.
```

Edit the dhcpd.conf file with

```
sudo nano /etc/dhcp/dhcpd.conf.
```

Comment out all other lines and ensure that only these are not commented:

```
ddns-update-style interim;
default-lease-time 600;
max-lease-time 7200;
authoritative;
log-facility local7;
subnet 192.168.137.0 netmask 255.255.255.0 {
range 192.168.137.5 192.168.137.150;
}
```

Save the config file and reboot. From another computer, you will be able to connect to the new adhoc network called RPIWireless. Then it is possible to SSH into the Pi with the address 192.168.137.99. Save this new image to a different file name on an external hard drive.

Writing and saving the network module server code

Create a folder in the home directory with

```
mkdir foldername
```

Navigate inside the folder with

```
cd foldername
```

Drag and drop the NeuroHubNetworkModule.cpp.

Drag and drop the NeuroHubGui3.glade file from the host computer to the SFTP panel in MobaXTerm inside the newly created folder. NeuroHubGui3.glade can be found at https://github.com/nehatk17/NeuroHubNetworkModule/tree/master/Codes/Network%20Module%20Programming. Alternatively, it is possible to create a new .glade file by downloading the Glade GUI program from https://glade.gnome.org/. Check the default preferences and begin making the GUI by dragging and dropping the desired elements. For radio buttons, ensure only one of the ones you wish to group is checked. In the General tab of the other buttons, write the name of the active button in the Group attribute. To populate the combobox, Click on ComboBox model ellipses in the General tab, and select New. Change column type to gchararray. Scroll down in the same panel and click add new row. For sequential editing, select horizontal. Type in the desired row names. Click on the combobox in the widget panel at the right of the screen and hit the edit button that appears at the taskbar at the top. Click the hierarchy tab and select Add. In the Properties and Attributes section next to Text, hit the drop-down menu that says unset and select gchararray1-gchararray. Exit, and when you click on the drop down in the window, the items you initially wrote show up. See https://www.youtube.com/watch?v=Z5_F-rW2cL8 for a visual tutorial.

Compile the code with

```
g++ NeuroHubNetworkModule.cpp -o NeuroHubNetworkModule -std=c++11 `pkg-
config gtkmm-3.0 --cflags --libs` -pthread -lwiringPi -lrt
```

To run the code, type

```
export XAUTHORITY = /home/pi/.Xauthority
```

Then

```
sudo ./NeuroHubNetworkModule
```

The program should be running. First select number of networking clients and press connect. The code will listen for clients to connect and once the maximum number of clients are reached, threads for each client are established to distribute the event markers.

To automate running the program once login is complete, do the following from the home directory:

```
sudo nano start_script.sh
```

In the empty file that opens, type:

```
export XAUTHORITY=/home/pi/Xauthority
cd foldername
sudo ./NeuroHubNetworkModule
```

Then exit and save.

Enter the following command:

```
sudo chmod u+x start_script.sh
```

Then enter:

```
sudo nano .bashrc
```

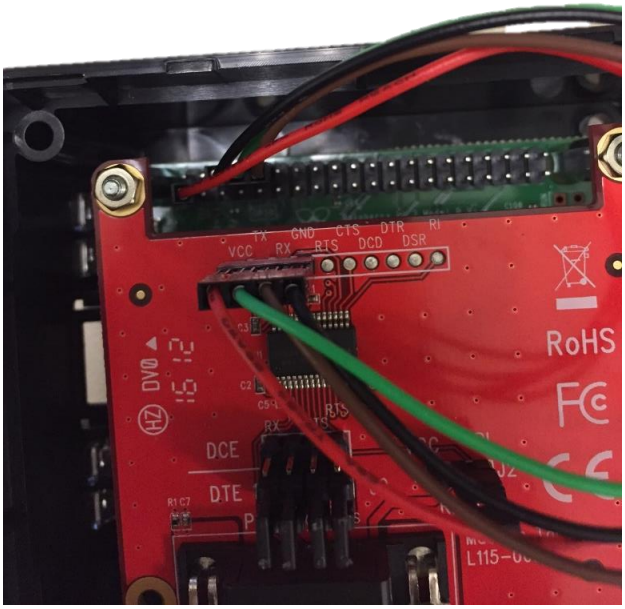
At the very end of this file, add

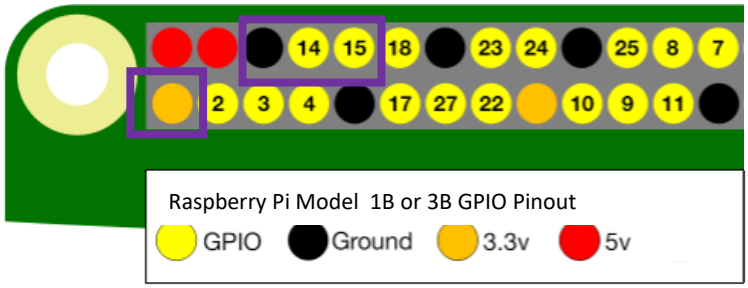
```
sudo ./start_script.sh
```

Exit and save. Reboot (`sudo reboot`) to see changes take effect. Save the working image to a new file on an external hard drive.

Attaching Add-On Boards and Case

The RS232 boards used are attached mechanically to the pins on either Raspberry Pi Model. The DTronix board is suitable for the Raspberry Pi Model 1 B, while Ableconn is used for the Raspberry Pi Model 3. The Ableconn board ships with mounting accessories for stabilization. The wires should be attached to the pins as shown in the picture below. The red wire goes to the leftmost bottom row of pins. The black wire goes to the 3rd pin from the left in the top row. The green wire connects to the pin next to the black wire's pin, and brown connects to the neighboring pin to the right of the green. The following image shows the pinout of the Raspberry Pi board. Pins 14 and 15 are the transmit (Tx) and receive (Rx) pins for the UART, which will be connected to the corresponding pins on the expansion board.





The modular case will fit the Raspberry Pi 3 and there are screws to hold the spacers in place. One of the spacers needs to be cut so that the serial port can be visible. Mark the places where it needs to be cut. Cut through the marked parts with large scissors. Finally, place a

second spacer and the cover on. Insert medium sized screws on the bottom of the case (provided in the modular kit) to secure the additional case components. Using a small extender attached to the serial port on the Raspberry Pi will give easier access to the port.

The case for the Raspberry Pi 1 was 3D printed at the Innovation Studio at Drexel University (located at Drexel One Plaza, 2nd floor). The .prt files used to design the case in CREO can be accessed at https://github.com/nehatk17/NeuroHubNetworkModule/tree/master/3D%20Printing%20Files . To ensure the baseplate and cover fit properly, create an assembly and add the necessary references. To generate the appropriate 3D print files, save the .prt file as .stl, select apply and hit ok for the default settings. The width dimensions of the cover and base may need to be adjusted slightly to fit the audio port better. The final 3D printed cover was adhered to the base with Superglue.

Connecting multiple network clients in Ethernet or wireless mode

For Ethernet mode, it will be necessary to use a Windows 7 computer as it proved difficult and unreliable to try and create the hosted network from a Windows 10.

1. In Windows 7, go to the network and sharing center, and click “set up a new connection or network”.
2. Scroll down in the dialogue box and select “Set up a wireless ad hoc network”.
3. Enter a name and if desired, a password and hit next.
4. If using other Windows 7 computers, simply connect to the ad-hoc network that shows up in Wi-Fi connections icon at the bottom right corner of the taskbar.
5. If using Windows 10 computers, perform the following steps:
    a. Go to the network and sharing center and select “Set up a new connection or network”.
    b. Select “Manually connect to wireless network”
    c. Enter the SSID of the adhoc network created on the Windows 7 computer as well as the password, if any.
    d. Uncheck start this connection automatically, click next and close.
    e. In the command line, type the following: `netsh wlan set profileparameter <ssid> connectiontype=ibss connectionmode=manual`
    f. Then type: `netsh wlan connect <ssid>`. Now the Windows 10 computer should be connected to the adhoc network started on the Windows 7.
    g. SSH into the Raspberry Pi from the Windows 7 computer as described at the beginning of this appendix. Make sure to assign different IP addresses to the Windows 10 computers in the range 192.168.137.X, where X is any number from 2 – 254. Do this by changing the TCP/IP properties in the Ad-hoc Wi-Fi in the Network Connections panel.
6. For wireless mode, repeat the same steps for the Windows 10 computer, but instead using the SSID generated by the Raspberry Pi 3. Windows 7 computers will be able

to connect without any command line input. There is no need to assign the connected computers new IP addresses in the range of 192.168.137.X as appropriate addresses are automatically assigned by the Raspberry Pi 3 to the connected computers.

Appendix B: Source Code

Network Module Server

```
/*

compile with g++ NeuroHubNetworkModule.cpp -o NeuroHubNetworkModule -
std=c++11  `pkg-config gtkmm-3.0 --cflags --libs` -pthread -lwiringPi


if necessary use -g after g++ in compilation and link to use with gdb.

to run:
export XAUTHORITY=/home/pi/.Xauthority
sudo ./NeuroHubNetworkModule
or gdb ./NeuroHubNetworkModule

*/

#include <gtkmm.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <cstdlib>
#include <iostream>
#include <chrono>
#include <cstring>
#include <string.h> // memset
#include <pthread.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <vector>
#include <string>
#include <fstream>
#include <algorithm>
#include <sys/time.h>
#include <sched.h>
#include <time.h>
#include <sys/mman.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
using namespace std;


// Gtk widgets
Gtk::SpinButton *spinbutton1 = 0;
Gtk::ComboBox *combobox1 = 0;
Gtk::RadioButton *radiotcp, *radioudp = 0;
```

```
Gtk::Button *button_stop, *button_start = 0;


//variables for network connection
#define PORT "8888"
#define IP_ADDR "192.168.137.99"
#define MAXLEN 1
int BACKLOG =0;
int serfd;
static unsigned int cli_count = 0;
size_t size = sizeof(struct sockaddr_in);
struct sockaddr_in their_addr;
vector<sockaddr_in> udparray;
vector<int> tcparray;
bool ard = false;
bool wiringpisetup = false;
bool exitard = false;
int sock;
int baudrate; //9600 or 115200;
int baudint = 0;

pthread_t servbegin;
pthread_mutex_t sendcli;

bool tcpbool;
bool looprun = true;
bool ardrun = true;
bool enterard=false;


//network functions functions
void send_message_all_udp(char *s, int mysock);
void send_message_all_tcp(char *s);
void send_message_udp(char *s, int uid, int port);
void send_message_tcp(char *s, int uid);
void send_client_udp(char *s, int mysock, int port);
void send_client_tcp(char *s, int sock);
void *from_ard_udp(void *);
void *from_ard_tcp(void *);
void *handle_conn_udp(void *);
void *handle_conn_tcp(void *);
void *begin_server(void *);



/*send message to original sender*/
void send_client_udp( char *s, int mysock, int myport){
      int i;
      for(i=0;i<BACKLOG;i++){

                  if(htons(udparray[i].sin_port) == myport){
                        sendto(mysock,s,1,0,(struct
sockaddr*)&udparray[i], sizeof udparray[i]);
                  }

      }
}
```

```
//send message to all clients except original sender
void send_message_udp(char *s, int mysock, int myport){
      int i;
      for(i=0;i<BACKLOG;i++){

                  if(htons(udparray[i].sin_port) != myport){
                        sendto(mysock,s,1,0,(struct
sockaddr*)&udparray[i], sizeof udparray[i]);
                  }

      }
}

/* Send message to all clients */
void send_message_all_udp(char *s, int mysock){
      int i;
      for(i=0;i<BACKLOG;i++){

                  sendto(mysock,s,1,0,(struct    sockaddr*)&udparray[i],
sizeof udparray[i]);

      }
}

/* /////    TCP send functions //// */
/*send message to original sender*/
void send_client_tcp( char *s, int mysock){
      int i;
      for(i=0;i<BACKLOG;i++){
            if(tcparray[i]){
                  if(tcparray[i] == mysock){
                        send(tcparray[i], s, 1,0);
                  }
            }
      }
}

//send message to all clients except original sender
void send_message_tcp(char *s, int mysock){
      int i;
      for(i=0;i<BACKLOG;i++){
            if(tcparray[i]){
                  if(tcparray[i] != mysock){
                        send(tcparray[i], s, 1,0);
                  }
            }
      }
}

/* Send message to all clients */
void send_message_all_tcp(char *s){
      int i;
      for(i=0;i<BACKLOG;i++){
            if(tcparray[i]){
                  send(tcparray[i], s, 1,0);
            }
```

```
      }
}
/*receive messages from Arduino*/
void *from_ard_udp(void *thissock)
{
  int mysock = *(int*)thissock;
  int counter = 1;
  int ardint;
  char ardchar;
  char *mesg;
  while (ardrun)

  {
    if (serialDataAvail (serfd) == -1)
    {
          fprintf(stdout, "(Serial) No data able to be received: %s\n",
strerror (errno));
          exit(EXIT_FAILURE);
    }

   ardint = serialGetchar(serfd) ;

    //cout << ardint << endl;
    if (ardint > 0)
    {
    //cout << ardint << endl;
  // usleep(400);
     ardchar = char(ardint);
     mesg = &ardchar;

 pthread_mutex_lock(&sendcli);
    send_message_all_udp(mesg, mysock);
    pthread_mutex_unlock(&sendcli);



    }

  }
  cout << "exit from_ard" << endl;
  exitard= true;
}

/* handle the connections from client */
void *handle_conn_udp(void *pnewsock)
{
  int mysock = *(int*)pnewsock;


   char client_msg[MAXLEN];


  int read_size;
  struct timeval tv;

  int clientint;
  int myport;
```

```
while(looprun){

 read_size    =    recvfrom(mysock,    client_msg,    1,    0,    (struct
sockaddr*)&their_addr, &size);


     clientint = int(*client_msg);



     myport = htons(their_addr.sin_port);
    if (clientint >0)
    {
    //usleep(1000);


    pthread_mutex_lock(&sendcli);
    send_client_udp(client_msg,mysock,myport); //was send_message



    serialPuts (serfd, &(*client_msg)) ;

    pthread_mutex_unlock(&sendcli);
    }


  }
  cout << "exit handle -conn " << endl;


}

/* TCP data handling */
void *from_ard_tcp(void *)
//void *from_ard(int sock)
{
;
/* Declare ourself as a real time task */



  int counter = 1;
  int ardint;
  char ardchar;
  char *mesg;
  while (ardrun)

  {
    if (serialDataAvail (serfd) == -1)
    {
          fprintf(stdout, "(Serial) No data able to be received: %s\n",
strerror (errno));
          exit(EXIT_FAILURE);
    }
```

```
     ardint = serialGetchar (serfd) ;

    if (ardint > 0)
    {
    //usleep(1000);
  //  cout << ardint << endl;
     ardchar = char(ardint);
     mesg = &ardchar;


pthread_mutex_lock(&sendcli);
send_message_all_tcp(mesg);
pthread_mutex_unlock(&sendcli);


    }

  }
  cout << "exit from_ard" << endl;
  exitard=true;
}

/* handle the connections from client */
void *handle_conn_tcp(void *pnewsock)
{
  int mysock = *(int*)pnewsock;


   char client_msg[MAXLEN];

  int read_size;

  int clientint;
  int mesgcount = 0;
  int myport;


  while(looprun){


     read_size = recv(mysock, client_msg, 1, 0);



     clientint = int(*client_msg);


     if (clientint > 0)
     {
    // usleep(1000);
    client_msg[read_size] = '\0';

     /*  cout << "length of client message: " << strlen(client_msg) <<
endl;
       cout << "# bytes is : " << read_size << endl;     */
```

```
 pthread_mutex_lock(&sendcli);
   send_client_tcp(client_msg,mysock); //was send_message



     serialPuts(serfd, &(*client_msg)) ;
 pthread_mutex_unlock(&sendcli);


    }


  }
  cout << "exit handle -conn " << endl;



}

void *begin_server(void *)
{
  //pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
  looprun = true;
  ardrun = true;
  BACKLOG = spinbutton1->get_value_as_int();
  baudint = combobox1->get_active_row_number();
  if (baudint == 0)
  {
     baudrate = 9600;
  }
  else
  {
    baudrate = 115200;
  }
  tcpbool = radiotcp->get_active();


    struct addrinfo hints, *res;
    int reuseaddr = 1; // True

    // Get the address info
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    if (tcpbool==1)
    {
    hints.ai_socktype = SOCK_STREAM;
    }
    else{

    hints.ai_socktype = SOCK_DGRAM;} //TCP = SOCK_STREAM, SOCK_DGRAM =
UDP)
    if (getaddrinfo(IP_ADDR, PORT, &hints, &res) != 0) {
        perror("getaddrinfo");
        exit (EXIT_FAILURE);
        //return 1;
    }
```

```
    // Create the socket
    sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (sock == -1) {
        perror("socket");
        exit (EXIT_FAILURE);
       // return 1;
    }

    // Enable the socket to reuse the address
    if    (setsockopt(sock,    SOL_SOCKET,    SO_REUSEADDR,    &reuseaddr,
sizeof(int)) == -1) {
        perror("setsockopt");
        ::close(sock);
        exit (EXIT_FAILURE);
        //shutdown(sock,2);
       // return 1;
    }

    // Bind to the address
    if (bind(sock, res->ai_addr, res->ai_addrlen) == -1) {
        perror("bind");
        ::close(sock);
        exit (EXIT_FAILURE);
        //shutdown(sock,2);
        //return 0;
    }

    freeaddrinfo(res);
    if (tcpbool ==1)
    {
    if (listen(sock, BACKLOG) == -1) {
        perror("listen");
        exit (EXIT_FAILURE);

    }
    }

      if( (serfd= serialOpen("/dev/ttyAMA0", baudrate))<0) //opens on-
board serial port, baud rate 9600
      {
      fprintf(stderr,   "unable   to   open   serial   device:   %s\n",
strerror(errno));
      exit(EXIT_FAILURE);
      }

    if (!wiringpisetup)
    {
      if (wiringPiSetup() == -1)
      {
        fprintf (stdout, "Unable to start wiring Pi: %s\n", strerror
(errno));
        exit(EXIT_FAILURE);
      }
      else
      {
        wiringpisetup=true;
      }
```

```
    }
    cout << "listening for connections" << endl;
    // Main loop - accepting initial connections from the # clients
specified.

    // Main loop
    bool running = true;
    // Initialize clients
    while (running)
    {

      char client_msg[MAXLEN];
      if (tcpbool ==1)
      {
      size_t size = sizeof(struct sockaddr_in);
      struct sockaddr_in their_addr;
      int clilen = sizeof(their_addr);
      int newsock = accept(sock, (struct sockaddr*)&their_addr, &size);
      if (newsock == -1)
      {
        perror("accept");
        exit (EXIT_FAILURE);
       // return -1;
      }


      cli_count++;
      printf("Got    a    connection    from    %s    on    port    %d\n",
inet_ntoa(their_addr.sin_addr), htons(their_addr.sin_port));
      tcparray.push_back(newsock);
      if (cli_count == BACKLOG)
      {
         cout << "Max clients reached" << endl;
        running = false;
        break;
      }

      }
      else{
      int  byte_count  =  recvfrom(sock,  client_msg,  1,  0,  (struct
sockaddr*)&their_addr, &size);


      cli_count++;
      printf("Got    a    connection    from    %s    on    port    %d\n",
inet_ntoa(their_addr.sin_addr), htons(their_addr.sin_port));
      udparray.push_back(their_addr);
      if (cli_count == BACKLOG)
      {
         cout << "Max clients reached" << endl;
        running = false;
        break;
      }
    }

    }
    /* Send message to all clients that server is ready to accept data */
```

```
    char r = (char)(cli_count);

     char *mesg = &r;

    if (tcpbool == 1)
    {
      send_message_all_tcp(mesg);
    }
    else{

    send_message_all_udp(mesg,sock);
   }

    pthread_t *ptr, from_ard_t;
    ptr =static_cast<pthread_t*>(malloc(sizeof(pthread_t)*(cli_count)));


    int i;
    if (tcpbool==1)
    {
      for (i=0;i<(BACKLOG);i++)
    {
      if    (pthread_create(&ptr[i],    NULL,    handle_conn_tcp,    (void
*)&tcparray[i]) != 0)//was newsock
                {
        fprintf(stderr, "Failed to create thread\n");
        exit (EXIT_FAILURE);
            }

    }


      if (pthread_create(&from_ard_t, NULL, from_ard_tcp, NULL)!=0)
      {
      fprintf(stderr, "Failed to create thread\n");
      }
      enterard=true;
    }
    else{
     for (i=0;i<(BACKLOG);i++)
      {
        if (pthread_create(&ptr[i], NULL, handle_conn_udp, (void *)&sock)
!= 0)//was newsock
                {
          fprintf(stderr, "Failed to create thread\n");
        exit (EXIT_FAILURE);
              }

      }


      if    (pthread_create(&from_ard_t,    NULL,    from_ard_udp,    (void
*)&sock)!=0)
      {
      fprintf(stderr, "Failed to create thread\n");
      }
```

```
      enterard=true;
    }
      cout << "Created threads with arduino" << endl;
      pthread_join(from_ard_t, NULL);

      cout << "joined arduino thread" << endl;



    for(i = 0; i < (BACKLOG); i++)
    {
       pthread_join(ptr[i], NULL);
    }
    cout << "joined send/recv threads" << endl;



    close(sock);
    serialClose(serfd);
    udparray.clear();
    tcparray.clear();
    cli_count = 0;
    pthread_exit(NULL);
    button_start->set_sensitive(true);
    button_stop->set_sensitive(false);


}

//GUI functions
void buttonstart_clicked()
{


  if (pthread_create(&servbegin, NULL, begin_server, NULL)!=0)
      {
      fprintf(stderr, "Failed to create thread\n");
      }


  button_start->set_sensitive(false);
  button_stop->set_sensitive(true);

}

void buttonstop_clicked()
{
  cout << "End" << endl;
 looprun= false;
 ardrun = false;
  pthread_cancel(servbegin);
  pthread_join(servbegin, NULL);

  serialClose(serfd);
  ::close(sock);
 udparray.clear();
 tcparray.clear();
```

```
  cli_count = 0;


  button_stop->set_sensitive(false);
  if (enterard)
  {
      while (!exitard)
    {
    }
      button_start->set_sensitive(true);
  }
  else
  {
    button_start->set_sensitive(true);
  }
enterard = false;

exitard = false;
}

int main(int argc, char **argv)
{

  Glib::RefPtr<Gtk::Application>  app  =  Gtk::Application::create(argc,
argv, "org.gtkmm.example");

  //Load the GtkBuilder file and instantiate its widgets:
  Glib::RefPtr<Gtk::Builder> refBuilder = Gtk::Builder::create();
  try
  {
    refBuilder->add_from_file("NeuroHubGui3.glade");
  }
  catch(const Glib::FileError& ex)
  {
    std::cerr << "FileError: " << ex.what() << std::endl;
    return 1;
  }
  catch(const Glib::MarkupError& ex)
  {
    std::cerr << "MarkupError: " << ex.what() << std::endl;
    return 1;
  }
  catch(const Gtk::BuilderError& ex)
  {
    std::cerr << "BuilderError: " << ex.what() << std::endl;
    return 1;
  }



  Gtk::Window *window1 = 0;

  refBuilder->get_widget("window1", window1);
  refBuilder->get_widget("button_stop", button_stop);
  refBuilder->get_widget("button_start", button_start);
  refBuilder->get_widget("spinbutton1", spinbutton1);
  refBuilder->get_widget("combobox1", combobox1);
```

```
  refBuilder->get_widget("radiotcp",radiotcp);
  refBuilder->get_widget("radioudp",radioudp);



  Gtk::RadioButton::Group group = radiotcp->get_group();


  Glib::RefPtr<Gtk::Adjustment>              m_adjustment              =
Gtk::Adjustment::create(1.0, 1.0, 9.0, 1.0, 9.0, 0.0);
  spinbutton1->set_adjustment(m_adjustment);

  // connect more signals
  combobox1->set_active(0);
  button_start-
>signal_clicked().connect(sigc::ptr_fun(buttonstart_clicked));
  button_stop-
>signal_clicked().connect(sigc::ptr_fun(buttonstop_clicked));


  app->run(*window1);

  return 0;
}
```

(Starting Serial) C# Testing Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using System.IO.Ports;
using System.Threading;
using System.Net.Sockets;
using System.Net;
using HighResTimer;


namespace OneClient_StartSerial
{
    public partial class Form1 : Form
    {
        Stopwatch stopwatch = new Stopwatch();
        Decimal datapoints;

        StringBuilder sb = new StringBuilder();
        SerialPort serialPortOut = new SerialPort();
        SerialPort serialPortIn = new SerialPort();
        TcpClient tcpclnt;
```

```
        UdpClient udpclnt;
        IPEndPoint ep;
        Timing mytimer = new Timing();

        int baudrate;
        Thread sendThread;
        Thread recvThread;
        ThreadStart sending;
        ThreadStart receiving;
        NetworkStream stream;
        public Form1()
        {
            InitializeComponent();

        }

        private void startbutton_Click(object sender, EventArgs e)
        {
            if (radiotcp.Checked)
            {
                radioudp.Enabled = false;
                radioserial.Enabled = false;
            }
            else if (radioudp.Checked)
            {
                radioserial.Enabled = false;
                radiotcp.Enabled = false;
            }
            else
            {
                radiotcp.Enabled = false;
                radioudp.Enabled = false;
            }
            comPortOut.Enabled = false;
            startbutton.Enabled = false;

            if (serialPortOut.IsOpen) { serialPortOut.Close(); }
            if (serialPortIn.IsOpen) { serialPortIn.Close(); }

            baudrate                                                =
Int32.Parse(this.baudratebox.SelectedItem.ToString());
              serialPortOut.PortName                                =
comPortOut.SelectedItem.ToString();
            serialPortOut.BaudRate =  baudrate;//
            serialPortOut.Open();
                serialPortOut.DiscardOutBuffer();
            if (radioserial.Checked)
            {
                serialPortIn.PortName                               =
comPortIn.SelectedItem.ToString();
                serialPortIn.BaudRate = baudrate;//
                serialPortIn.Open();
                serialPortIn.DiscardInBuffer();

            }
```

```
            //set priority to high
            //        Process.GetCurrentProcess().PriorityClass       =
ProcessPriorityClass.High;
            //Thread.CurrentThread.Priority = ThreadPriority.Highest;



            //clear stringbuilder
            sb.Clear();

            Byte[] bb = new byte[1]; //1 byte of data coming in


            //serialPortOut.DiscardOutBuffer();

            datapoints = numericUpDown1.Value;

            //non loop format - for cppserv


            if (radiotcp.Checked)
            {
                tcpclnt = new TcpClient();
                textBox1.AppendText("TCP Connecting... \n");
                tcpclnt.Connect("192.168.137.99",  8888);  //address  of
RPi on arbitrary non privileged port
                textBox1.AppendText("TCP Connected \n");
                stream = tcpclnt.GetStream();
                int bytes = stream.Read(bb, 0, 1);




            }
            else if (radioudp.Checked)
            {
                try
                {
                    udpclnt = new UdpClient();
                    ep                          =                       new
IPEndPoint(IPAddress.Parse("192.168.137.99"), 8888);
                    textBox1.AppendText("UDP Connecting... \n");

                    udpclnt.Connect(ep);
                    textBox1.AppendText("UDP Connected \n");
                    byte[] writebyte = BitConverter.GetBytes(1);



                    udpclnt.Send(writebyte, writebyte.Length);
                    bb = udpclnt.Receive(ref ep);



                }
                catch
```

```
                {
                    return;
                }

            }
            else
            {
            }
            //intialize network connections

            /*Receive the welcome from server */


            mytimer.Start();

            int numback = bb[0];
            textBox1.AppendText("Received initial message from server: "
+ bb[0] + "\n");

            textBox1.AppendText("Warmup \n");
            if (baudrate == 9600)
            {
                while (mytimer.Duration * 1000 < 1500)
                {

                }
            }
          /*  stopwatch.Restart();
           // while (stopwatch.ElapsedMilliseconds < 1500)
            {
            }
            stopwatch.Stop();*/

            textBox1.AppendText("Beginning Testing");
            textBox1.AppendText(Environment.NewLine);


            ThreadProgram clientObject = new ThreadProgram(udpclnt, ep,
stream, mytimer, datapoints, sb, serialPortOut, serialPortIn, radiotcp);

            if (radiotcp.Checked)
            {

                if (checkser1.Checked)
                {

                    //            receiving            =            new
ThreadStart(clientObject.ser1portrecvData);
                    receiving                    =                    new
ThreadStart(clientObject.sersendrecvData);
                }
                else
                {
                    sending = new ThreadStart(clientObject.sersendData);
                    receiving                    =                    new
ThreadStart(clientObject.tcprecvData);
```

```
                    sendThread = new Thread(sending);
                }



            }
            else if (radioudp.Checked)
            {

                if (checkser1.Checked)
                {
                    sending = new ThreadStart(clientObject.sersendData);
                    receiving                   =                   new
ThreadStart(clientObject.ser1portrecvData);
                    //receiving                  =                   new
ThreadStart(clientObject.sersendrecvData);
                    sendThread = new Thread(sending);
                }
                else
                {
                    sending = new ThreadStart(clientObject.sersendData);
                    receiving                   =                   new
ThreadStart(clientObject.udprecvData);
                    sendThread = new Thread(sending);
                }

            }
            else if (radioserial.Checked)

            {

                    sending = new ThreadStart(clientObject.sersendData);
                    receiving                   =                   new
ThreadStart(clientObject.serrecvData);
                    sendThread = new Thread(sending);

            }



            recvThread = new Thread(receiving);
            recvThread.Start();
            if (!checkser1.Checked || radioserial.Checked)
            {  }
            sendThread.Start();

            if (!checkser1.Checked || radioserial.Checked)
            {

            }
            sendThread.Join();
            recvThread.Join();
            if (radiotcp.Checked)
            { tcpclnt.Close(); }
            else if (radioudp.Checked)
            {
                udpclnt.Close();
```

```
            }
            else
            {

                serialPortIn.Close();
            }
            comPortOut.Enabled = true;
            startbutton.Enabled = true;
            radiotcp.Enabled = true;
            radioudp.Enabled = true;
            radioserial.Enabled = true;
            textBox1.Clear();
            // Close Com ports
            if (serialPortOut.IsOpen) { serialPortOut.Close(); }
            this.Invoke(new EventHandler(SaveDialog));
        }

        private void radioserial_CheckedChanged(object sender, EventArgs
e)
        {
            if (radioserial.Checked)
            {
                comPortIn.Enabled = true;
                List<String> tList = new List<String>();
                comPortIn.Items.Clear();
                foreach (string s in SerialPort.GetPortNames())
                {
                    tList.Add(s);
                }
                tList.Sort();

                comPortIn.Items.AddRange(tList.ToArray());
                comPortIn.SelectedIndex = 0;



            }
            else
            {
                comPortIn.Enabled = false;
            }
        }
        public void SaveDialog(object sender, EventArgs e)
        { /// When the timer runs out or STOP is pressed, a Save Dialog
appears
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            saveFileDialog1.FilterIndex = 1;
            saveFileDialog1.RestoreDirectory = true;
            if (radiotcp.Checked)
            {
                saveFileDialog1.FileName = "DeviceTest_ser-ethTCP_10";
            }
            else
            {
                saveFileDialog1.FileName = "DeviceTest_ser-ethUDP_10";
```

```
            }

            if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
            {                File.WriteAllText(saveFileDialog1.FileName,
sb.ToString()); }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            List<String> tList = new List<String>();
            comPortOut.Items.Clear();
            foreach (string s in SerialPort.GetPortNames())
            {
                tList.Add(s);
            }
            tList.Sort();

            comPortOut.Items.AddRange(tList.ToArray());
            comPortOut.SelectedIndex = 0;
            comPortIn.Enabled = false;
            comPortOut.Enabled = true;

            baudratebox.Items.Add("9600");
            baudratebox.Items.Add("115200");
            baudratebox.SelectedIndex = 0;

        }

        public class ThreadProgram
        {
            public System.Object lockThis = new System.Object();
            public NetworkStream stream;
            public UdpClient udpin;
            public IPEndPoint ep;
            private static Mutex mut = new Mutex();
            Decimal datapoints;
            bool received = true;
            bool sent = false;
            int num2send=1;
            int numback;
            StringBuilder sb;
            int correctbyte = 0;
            int counter = 0;
            SerialPort serialPortOut;
            SerialPort serialPortIn;
            byte[] sentbyte = new byte[1];
            byte[] recvbyte = new byte[64];
            byte[] ackbyte = new byte[64];

            RadioButton radiotcp;
            Timing mytimer = new Timing();
            double sent1;
            double sentelapsmil;
            double recv1;
            double recvelapsmil;
            double elapsmil;
```

```
            public ThreadProgram(UdpClient udp, IPEndPoint epin,
NetworkStream streamer, Timing timer, Decimal thedatapoints,
StringBuilder stringb, SerialPort serialPort, SerialPort serialPort2,
RadioButton radio_tcp)
            {
                stream = streamer;

                sb = stringb;
                datapoints = thedatapoints;

                serialPortOut = serialPort;
                radiotcp = radio_tcp;
                mytimer = timer;
                udpin = udp;
                ep = epin;
                serialPortIn = serialPort2;

            }

            public void sersendData()
            {
               /* if (serialPortOut.BaudRate == 115200)
                {
                    while (mytimer.Duration * 1000 < 1500)
                    {

                    }
                }*/
                for (int repeat = 0; repeat < datapoints; repeat++)
                {

                    while (!received)
                    {

                    }
                    received = false;

                    mut.WaitOne();
                    sentbyte = BitConverter.GetBytes(num2send);
                    mut.ReleaseMutex();



                        if (radiotcp.Checked) {  }
                        //serialPortOut.DiscardOutBuffer();

                        serialPortOut.Write(sentbyte, 0, 1);


                     //Send the byte

//stopwatch.Reset();



                    sentelapsmil = mytimer.Duration;
                    sent = true;
```

```
            }

        }

        public void sersendrecvData()
        {


            if (serialPortOut.BaudRate == 115200)
            {
                while (mytimer.Duration * 1000 < 1500)
                {

                }
            }
            for (int repeat = 0; repeat < datapoints; repeat++)
            {



                sentbyte = BitConverter.GetBytes(num2send);


                if               (radiotcp.Checked)               {
serialPortOut.Write(sentbyte, 0, 1); }
                else
                {
                    serialPortOut.Write(sentbyte, 0, 1);
                    //serialPortOut.DiscardOutBuffer();


                }
                //Send the byte
                //stopwatch.Reset();



                sentelapsmil = mytimer.Duration;

                numback = serialPortOut.ReadByte();
                recvelapsmil = mytimer.Duration;
                serialPortOut.DiscardInBuffer();



                if (numback == num2send)
                {
                    correctbyte = 1;
                }
                else
                {
                    correctbyte = 0;
                }
                // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");
```

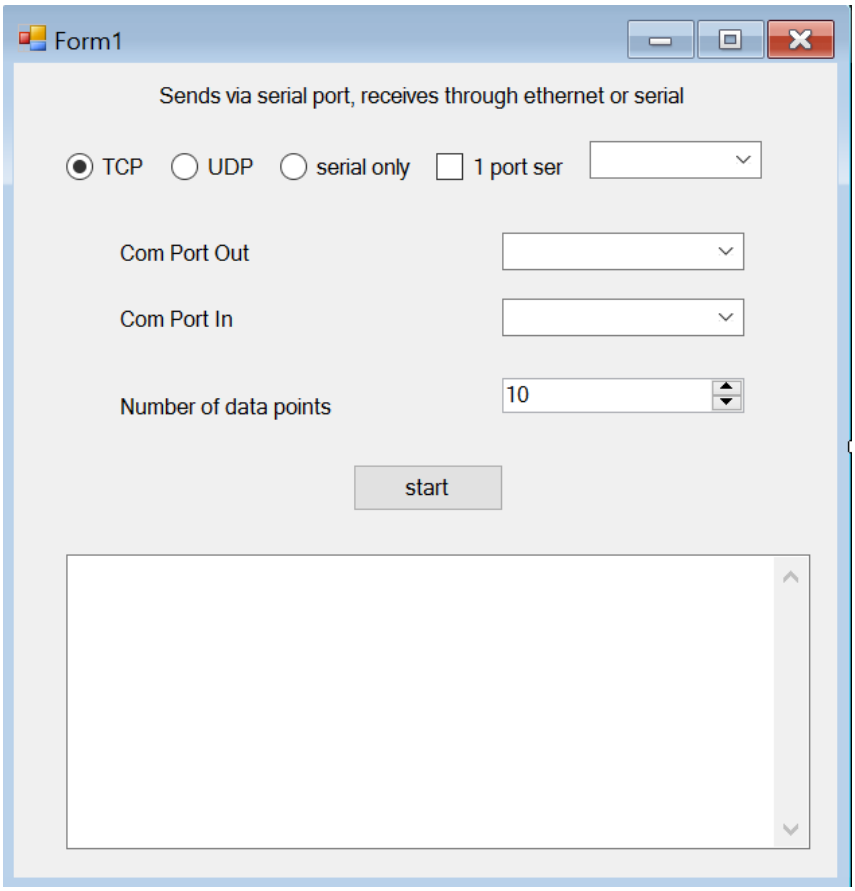
```
                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);
                    //textbox1.AppendText("Received      after:"      +
elapsedmilli + "\t" + correctbyte+"\n");

                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }









                }

            }

            public void tcprecvData()
            {
                ackbyte = BitConverter.GetBytes(0);

                //ackbyte = BitConverter.GetBytes(0);
                for (int repeat = 0; repeat < datapoints; repeat++)
                {



                    //stream.ReadAsync(recvbyte, 0, 1);

                    stream.Read(recvbyte, 0, 1);
                    stream.Write(ackbyte, 0, 1);
                    recvelapsmil = mytimer.Duration;

                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil*1000+ "\t" +
correctbyte);

                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
```

```
            }
        }
        public void serrecvData()
        {

            for (int repeat = 0; repeat < datapoints; repeat++)
            {

                while (sent==false)
                { }

                recv1 = mytimer.Duration;

                    numback = serialPortIn.ReadByte();



                recvelapsmil = mytimer.Duration;
                sent = false;


                if (numback == num2send)
                {
                    correctbyte = 1;
                }
                else
                {
                    correctbyte = 0;
                }
                // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");
                elapsmil = recvelapsmil - sentelapsmil;
                counter++;
                sb.AppendLine(counter + "\t" + elapsmil*1000 + "\t"+
correctbyte);

                //textbox1.AppendText("Received      after:"      +
elapsedmilli + "\t" + correctbyte+"\n");
                mut.WaitOne();
                num2send += 1;
                if (num2send > 255) { num2send = 1; }
                mut.ReleaseMutex();
                received = true;
            }
            //Thread.Sleep(100);
            // recvloop = false;
        }
        public void serrecv2Data()
        {

            for (int repeat = 0; repeat < datapoints; repeat++)
            {

                for (int i = 0; i < 2; i++)
                {
                numback = serialPortIn.ReadByte();
```

```
                    }



                    recvelapsmil = mytimer.Duration;
                    numback = recvbyte[0];


                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }
                    // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");
                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);
                    //textbox1.AppendText("Received      after:"      +
elapsedmilli + "\t" + correctbyte+"\n");
                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }
                //Thread.Sleep(100);
                // recvloop = false;
            }
            public void ser1portrecvData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {

                    serialPortOut.DiscardInBuffer();


                    numback = serialPortOut.ReadByte();
                    recvelapsmil = mytimer.Duration;
                 //   serialPortOut.DiscardInBuffer();
                 //  serialPortOut.DiscardOutBuffer();







                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
```

```
                    {
                        correctbyte = 0;
                    }
                    // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");
                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);
                    //textbox1.AppendText("Received      after:"      +
elapsedmilli + "\t" + correctbyte+"\n");
                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }

                    mut.ReleaseMutex();
                    Thread.Sleep(10);
                    received = true;
                }
                //Thread.Sleep(100);
                // recvloop = false;
            }
            public void udprecvData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {



                   recvbyte = udpin.Receive(ref ep);

                    recvelapsmil = mytimer.Duration;

                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }
                    // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");
                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);
                    //textbox1.AppendText("Received      after:"      +
elapsedmilli + "\t" + correctbyte+"\n");
                    received = true;
                }
            }
        }
```

```
    }
}
```



(Starting Ethernet) C# Testing Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using System.IO.Ports;
using System.Threading;
using System.Net.Sockets;
using System.Net;
```

```
using HighResTimer;


namespace OneClient_NetworkTiming_TCP
{


    public partial class Form1 : Form
    {

        Decimal datapoints;
        StringBuilder sb = new StringBuilder();
        SerialPort serialPortIn = new SerialPort();
        TcpClient tcpclnt;
        UdpClient udpclnt;
        IPEndPoint ep;
        Timing mytimer = new Timing();

        int baudrate; //9600;

        Thread sendThread;
        Thread recvThread;
        ThreadStart sending;
        ThreadStart receiving;
        NetworkStream stream;

        public Form1()
        {
            InitializeComponent();
        }



        private void startbutton_Click(object sender, EventArgs e)
        {
            comPortIn.Enabled = false;
            startbutton.Enabled = false;


            if (serialPortIn.IsOpen) { serialPortIn.Close(); }
            if (checkserial.Checked)
            {

                serialPortIn.PortName                                   =
comPortIn.SelectedItem.ToString();
                baudrate                                                =
Int32.Parse(this.baudratebox.SelectedItem.ToString());
                serialPortIn.BaudRate = baudrate; //was 9600
                serialPortIn.Open();
                serialPortIn.DiscardInBuffer();
            }


            //set priority to high
            //        Process.GetCurrentProcess().PriorityClass       =
ProcessPriorityClass.High;
            //Thread.CurrentThread.Priority = ThreadPriority.Highest;
```

```
            //clear stringbuilder
            sb.Clear();
            Random   seed   =   new   Random((int)DateTime.Now.Ticks   &
0x0000FFFF);
            Byte[] bb = new byte[1]; //1 byte of data coming in


            //serialPortIn.DiscardOutBuffer();

            datapoints = numericUpDown1.Value;


            if (radiotcp.Checked)
            {
                tcpclnt = new TcpClient();
                textBox1.AppendText("TCP Connecting... \n");
                tcpclnt.Connect("192.168.137.99",  8888);  //address  of
RPi on arbitrary non privileged port
                textBox1.AppendText("TCP Connected \n");
                stream = tcpclnt.GetStream();
                int bytes = stream.Read(bb, 0, bb.Length);

            }
            else
            {
                try
                {
                    udpclnt = new UdpClient();
                    ep                        =                        new
IPEndPoint(IPAddress.Parse("192.168.137.99"), 8888);
                    textBox1.AppendText("UDP Connecting... \n");

                    udpclnt.Connect(ep);
                    textBox1.AppendText("UDP Connected \n");
                    byte[] writebyte = BitConverter.GetBytes(99);



                    udpclnt.Send(writebyte, writebyte.Length);
                    bb = udpclnt.Receive(ref ep);

                }
                catch

                {
                    return;
                }

            }
            //intialize network connections

            /*Receive the welcome from server */
```

```
            mytimer.Start();


            int numback = bb[0];
            textBox1.AppendText("Received initial message from server: "
+ bb[0] + "\n");

            textBox1.AppendText("Warmup \n");
            while (mytimer.Duration *1000 < 1500)
            {

            }


            textBox1.AppendText("Beginning Testing");
            textBox1.AppendText(Environment.NewLine);


            ThreadProgram clientObject = new ThreadProgram(udpclnt, ep,
stream, mytimer, datapoints, sb, serialPortIn);

            if (radiotcp.Checked)
            {
                sending = new ThreadStart(clientObject.tcpsendData);
                if (checkserial.Checked)
                {

                    receiving                   =                   new
ThreadStart(clientObject.tcprecvserData);
                }
                else
                {
                    receiving                   =                   new
ThreadStart(clientObject.tcprecvData);
                }
            }
            else
            {
                sending = new ThreadStart(clientObject.udpsendData);
                if (checkserial.Checked)
                {
                    receiving                   =                   new
ThreadStart(clientObject.udprecvserData);
                }
                else
                {
                    receiving                   =                   new
ThreadStart(clientObject.udprecvData);
                }
            }

            sendThread = new Thread(sending);
            recvThread = new Thread(receiving);
            recvThread.Start();
            sendThread.Start();
```

```
            sendThread.Join();
            recvThread.Join();
            if (radiotcp.Checked)
            { tcpclnt.Close(); }
            else
            {
                udpclnt.Close();
            }
                comPortIn.Enabled = true;
            startbutton.Enabled = true; // Diable the stop button

            textBox1.Clear();
            // Close Com ports
            if (serialPortIn.IsOpen) { serialPortIn.Close(); }
            this.Invoke(new EventHandler(SaveDialog));


        }

        private void Form1_Load(object sender, EventArgs e)
        {
            baudratebox.Items.Add("9600");
            baudratebox.Items.Add("115200");
            baudratebox.SelectedIndex = 0;


            comPortIn.Enabled = false;

        }
        public void SaveDialog(object sender, EventArgs e)
        { /// When the timer runs out or STOP is pressed, a Save Dialog
appears
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            saveFileDialog1.FilterIndex = 1;
            saveFileDialog1.RestoreDirectory = true;
            if (radiotcp.Checked)
            {
                if (checkserial.Checked)
                {
                    saveFileDialog1.FileName     =     "DeviceTest_eth-
serTCP_10";
                }
                else
                {
                    saveFileDialog1.FileName     =     "DeviceTest_eth-
ethTCP_10";
                }

            }
            else
            {
                if (checkserial.Checked)
                {
                    saveFileDialog1.FileName     =     "DeviceTest_eth-
serUDP_10";
```

```
                }
                else
                {
                    saveFileDialog1.FileName     =     "DeviceTest_eth-
ethUDP_10";
                }

            }

            if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
            {                File.WriteAllText(saveFileDialog1.FileName,
sb.ToString()); }
        }

        private void checkserial_CheckedChanged(object sender, EventArgs
e)
        {
            if (checkserial.Checked)
            {
                List<String> tList = new List<String>();
                comPortIn.Items.Clear();
                foreach (string s in SerialPort.GetPortNames())
                {
                    tList.Add(s);
                }
                tList.Sort();

                comPortIn.Items.AddRange(tList.ToArray());
                comPortIn.SelectedIndex = 0;
                comPortIn.Enabled = false;
                comPortIn.Enabled = true;
            }
            else
            {
                comPortIn.Enabled = false;
            }
        }
        public class ThreadProgram
        {
            public System.Object lockThis = new System.Object();
            public NetworkStream stream;
            public UdpClient udpin;
            public IPEndPoint ep;
            private static Mutex mut = new Mutex();
            int num2send = 1;
            Decimal datapoints;
            int numback;
            bool received = true;
            byte[] recvbyte = new byte[1];
            byte[] sentbyte = new byte[1];
            int counter = 0;

            StringBuilder sb;
            int correctbyte=0;

            SerialPort serialPortIn;
```

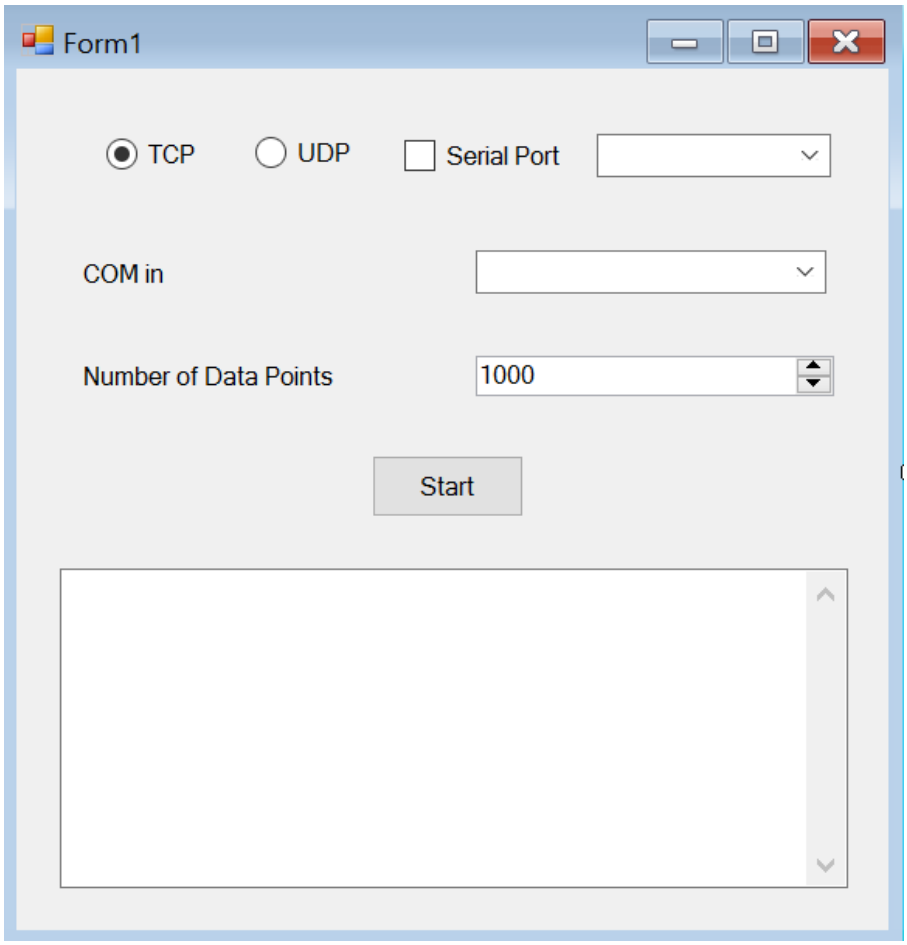
```
            Timing mytimer = new Timing();
            double sentelapsmil;
            double recvelapsmil;
            double elapsmil;

            public   ThreadProgram(UdpClient   udp,   IPEndPoint   epin,
NetworkStream   streamer,   Timing   timer,   Decimal   thedatapoints,
StringBuilder stringb, SerialPort serialPort)
            {
                stream = streamer;

                sb = stringb;
                datapoints = thedatapoints;

                serialPortIn = serialPort;


                mytimer = timer;
                udpin = udp;
                ep = epin;

            }

            public void tcpsendData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {


                        while (!received)
                        {

                        }
                        received = false;



                    mut.WaitOne();

                    sentbyte = BitConverter.GetBytes(num2send);
                    mut.ReleaseMutex();



                        stream.Write(sentbyte, 0, 1);

                         sentelapsmil = mytimer.Duration;



                }

            }

            public void tcprecvserData()
```

```
            {


                for (int repeat = 0; repeat < datapoints; repeat++)
                {


                    //1 byte of data coming in

                    numback = serialPortIn.ReadByte();

                    recvelapsmil = mytimer.Duration;



                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;

                     sb.AppendLine(counter+"\t"+elapsmil*1000  +  "\t"  +
correctbyte);

                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;

                }
            }
            public void tcprecvData()
            {


                for (int repeat = 0; repeat < datapoints; repeat++)
                {


                    stream.Read(recvbyte, 0, 1);

                    recvelapsmil = mytimer.Duration;

                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
```

```
                    {
                        correctbyte = 0;
                    }

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);

                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }
            }
            public void udpsendData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {


                    while (!received)
                    {

                    }
                    received = false;


                    mut.WaitOne();

                    sentbyte =  BitConverter.GetBytes(num2send);
                    mut.ReleaseMutex();


                    udpin.Send(sentbyte, 1);


                    sentelapsmil = mytimer.Duration;



                }

            }

            public void udprecvserData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {

                    numback = serialPortIn.ReadByte();

                   recvelapsmil = mytimer.Duration;
```

```
                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;

                   sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);

                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }
            }
            public void udprecvData()
            {

                for (int repeat = 0; repeat < datapoints; repeat++)
                {

                    recvbyte= udpin.Receive(ref ep);

                    recvelapsmil = mytimer.Duration;

                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }


                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;

                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);


                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }
```

```
            }
        }

    }
}
```



(One computer networking round trip) C# Testing Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using HighResTimer;
```

```
using System.IO;

namespace Server
{
    public partial class Form1 : Form
    {
        Decimal datapoints;
        NetworkStream stream;
        Thread sendThread;
        Thread recvThread;
        ThreadStart sending;
        ThreadStart receiving;
        TcpClient tcpclnt;
        ThreadProgram clientObject;
        StringBuilder sb = new StringBuilder();
        TcpListener myList;
        UdpClient listener;
        IPEndPoint groupep;

        IPEndPoint ep;
        Timing mytimer = new Timing();

        public Form1()
        {
            InitializeComponent();
        }

        private void startbutton_Click(object sender, EventArgs e)
        {
            datapoints = datapts.Value;
            sb.Clear();
            mytimer.Start();
            if (tcpradio.Checked)
            {
                IPAddress ipAd = IPAddress.Parse("192.168.137.1");
                // use local m/c IP address, and
                // use the same in the client

                /* Initializes the Listener */
                TcpListener myList = new TcpListener(ipAd, 8001);

                /* Start Listeneting at the specified port */
                myList.Start();
                tcpclnt = new TcpClient();
                Console.WriteLine("Connecting.....");

                tcpclnt.Connect("192.168.137.1", 8001);

                Socket s = myList.AcceptSocket();
                stream = tcpclnt.GetStream();
                clientObject = new ThreadProgram(listener, s, myList,
tcpclnt, groupep, ep, stream, mytimer, datapoints, sb);
            }
            else
            {
                IPEndPoint groupep = new IPEndPoint(IPAddress.Any,
11000);
```

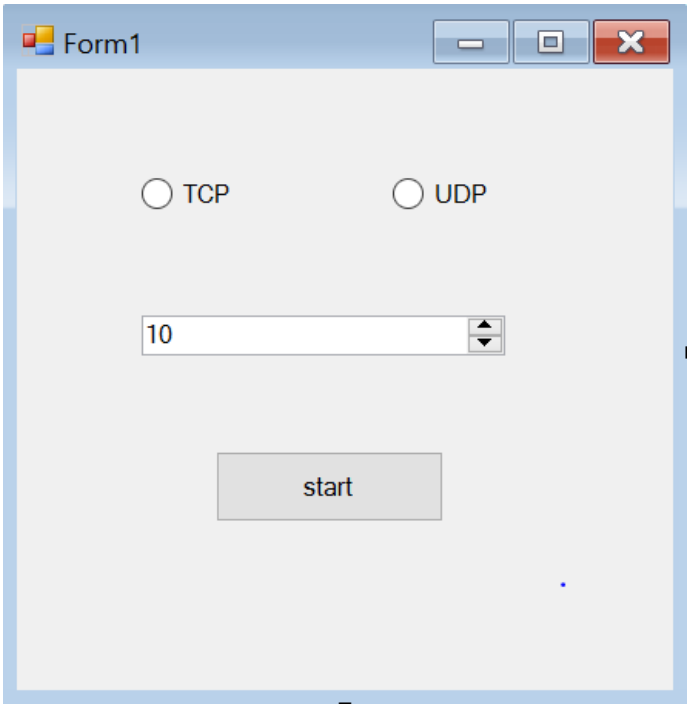
```
                UdpClient listener = new UdpClient();
                // udpclient.ExclusiveAddressUse = false;
                //
udpclient.Client.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReuseAddress, true);
                //listener.Client.Bind(listener);

                listener.Client.Bind(groupep);
                Socket   s   =   new   Socket(AddressFamily.InterNetwork,
SocketType.Dgram,
            ProtocolType.Udp);

                IPAddress broadcast = IPAddress.Parse("127.0.0.1");
                IPEndPoint ep = new IPEndPoint(broadcast, 11000);

                //  udpserver.ExclusiveAddressUse = false;
                //
udpserver.Client.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReuseAddress, true);
                //IPEndPoint           epin2           =           new
IPEndPoint(IPAddress.Parse("192.168.0.101"), 5678);
                //udpserver.Client.Bind(epin2);// was epin and ipaddress
parse 192.168.137.1, 1234 (after change in network adapter settings)
                clientObject  =  new  ThreadProgram(listener,  s,  myList,
tcpclnt, groupep, ep, stream, mytimer, datapoints, sb);
            }


            if (tcpradio.Checked)
            {
                sending = new ThreadStart(clientObject.tcpsendData);
                receiving = new ThreadStart(clientObject.tcprecvData);

            }
            else
            {
                //while (mytimer.Duration * 1000 < 1500)
               // { }
                sending = new ThreadStart(clientObject.udpsendData);
                receiving = new ThreadStart(clientObject.udprecvData);
            }

            sendThread = new Thread(sending);
            recvThread = new Thread(receiving);
            recvThread.Start();
            sendThread.Start();


            sendThread.Join();
            recvThread.Join();
            this.Invoke(new EventHandler(SaveDialog));

        }
        public void SaveDialog(object sender, EventArgs e)
        { /// When the timer runs out or STOP is pressed, a Save Dialog
appears
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
```

```
            saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            saveFileDialog1.FilterIndex = 1;
            saveFileDialog1.RestoreDirectory = true;
            if (tcpradio.Checked)
            {

                saveFileDialog1.FileName = "Comp_hosteth-ethTCP_10_";



            }
            else
            {
                saveFileDialog1.FileName = "Comp_eth-ethUDP_10_";



            }

            if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
            {                File.WriteAllText(saveFileDialog1.FileName,
sb.ToString()); }
        }

        public class ThreadProgram
        {
            public System.Object lockThis = new System.Object();
            public NetworkStream stream;
            Socket s;
            private static Mutex mut = new Mutex();
            int num2send = 1;
            Decimal datapoints;
            int numback;
            bool received = true;
            byte[] recvbyte = new byte[1];
            byte[] sentbyte = new byte[1];
            int counter = 0;
            IPEndPoint listenep;
            IPEndPoint epbroad;
            UdpClient listener;
            TcpClient tcpclnt;
            TcpListener myList;
            StringBuilder sb;
            int correctbyte = 0;

            Timing mytimer = new Timing();
            double sentelapsmil;
            double recvelapsmil;
            double elapsmil;

            public   ThreadProgram(UdpClient   client,   Socket   sender,
TcpListener tcplist, TcpClient tcpclient, IPEndPoint ep1, IPEndPoint ep2,
NetworkStream   streamer,   Timing   timer,   Decimal   thedatapoints,
StringBuilder sbin)
            {
                stream = streamer;
```

```
    datapoints = thedatapoints;
    listenep = ep1;
    epbroad = ep2;
    listener = client;
    s = sender;
    myList = tcplist;
    tcpclnt = tcpclient;
    sb = sbin;
    mytimer = timer;


}

public void tcpsendData()
{


    for (int repeat = 0; repeat < datapoints; repeat++)
    {


        while (!received)
        {

        }
        received = false;



        mut.WaitOne();

        sentbyte = BitConverter.GetBytes(num2send);
        mut.ReleaseMutex();



        stream.Write(sentbyte, 0, 1);

        sentelapsmil = mytimer.Duration;


    }
    tcpclnt.Close();
}
public void tcprecvData()
{


    for (int repeat = 0; repeat < datapoints; repeat++)
    {


        s.Receive(recvbyte);

        recvelapsmil = mytimer.Duration;
```

```
                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);

                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }
                s.Close();
                myList.Stop();
            }
            public void udpsendData()
            {


                for (int repeat = 0; repeat < datapoints; repeat++)
                {
                   =

                    while (!received)
                    {

                    }
                    received = false;


                    mut.WaitOne();

                    sentbyte = BitConverter.GetBytes(num2send);
                    mut.ReleaseMutex();
                    // stopwatch.Reset();

                    s.SendTo(sentbyte, epbroad);



                    sentelapsmil = mytimer.Duration;
                    /


                }
                s.Close();
```

```
            }
            public void udprecvData()
            {


                for (int repeat = 0; repeat < datapoints; repeat++)
                {


                    recvbyte = listener.Receive(ref listenep);


                    recvelapsmil = mytimer.Duration;

                    numback = recvbyte[0];

                    if (numback == num2send)
                    {
                        correctbyte = 1;
                    }
                    else
                    {
                        correctbyte = 0;
                    }
                    // sb.AppendLine(elapsedmilli + "\t" + correctbyte +
"\n");

                    elapsmil = recvelapsmil - sentelapsmil;
                    counter++;
                    //sb.Append("\t" + recvmil);
                    sb.AppendLine(counter + "\t" + elapsmil * 1000 + "\t"
+ correctbyte);


                    mut.WaitOne();
                    num2send += 1;
                    if (num2send > 255) { num2send = 1; }
                    mut.ReleaseMutex();
                    received = true;
                }

                listener.Close();
            }
        }
    }
}
```



MATLAB Data Analysis Code

```
close all
[FileName,PathName] = uigetfile('*.txt','Select a multiple of 10 trials
.txt files','MultiSelect','on');

nFile = size(FileName,2);
nrows = 1000;
ncols = 20;

datadim = nFile/10;
datamat = zeros(nrows,ncols,datadim);
dim = 1;
count = 1;
maintitle = {};
maintitle{1}=FileName{1}(1:end-11);
if iscell(FileName)
    for i=1:nFile

    data=dlmread(strcat(PathName,FileName{i}),'\t');
    datamat(:,count*2-1:count*2,dim)=data(:,2:3);

        if i == nFile
            break;
        elseif count == 10

            maintitle{end+1} = FileName{i+1}(1:end-11);
            count = 1;
            dim = dim +1;
        else
            count=count+1;
```

```
        end
    end
end

timevalues = datamat(:,1:2:end,:);
boolvalues = datamat(:,2:2:end,:);
mintime = min(min(min(timevalues)));
maxtime = max(max(max(timevalues)));
maxavg = max(max(max(mean(timevalues))))+max(max(max(std(timevalues))));
minavg = min(min(min(mean(timevalues))))-max(max(max(std(timevalues))));
colors = {[0 1 1],[1 0 0],[1 1 0],[.4 1 1],[.5 .5 0],[.9 .6 .3],[.2 .3
.6],[.8 .4 .1],[1 0 .6],[0 .3 .3]};
for k = 1:datadim
time_test = timevalues(:,:,k);
bool_test = boolvalues(:,:,k);
avgvalues = mean(time_test);
stdev = std(time_test);

successrate = (sum(bool_test)/nrows)*100;
if successrate < 100
    display('Failed bool test')
    break;
end
nbins = round(nrows*nFile/datadim/4);

maxval = max(max(time_test));
minval = min(min(time_test));
avgval = mean(avgvalues);
medval = median(median(time_test));
range = maxval - minval;
h1=figure;


[N, edges] = histcounts(time_test);
maxy = max(N);
bar(edges(1:end-1),N,'EdgeColor','None')
%histogram(time_test,'EdgeColor','None')
LW=2;
axis([mintime maxtime 0 maxy])
ax=gca;
hold on;
line([maxval maxval],[ylim],'Color','r','LineStyle','--','LineWidth',LW)
line([minval minval],[ylim],'Color','m','LineStyle','--','LineWidth',LW)
line([avgval avgval],[ylim],'Color','g','LineStyle','--','LineWidth',LW)
%line([medval medval],[ylim],'Color','g','LineStyle','--')

%y=ylim;
strstats = sprintf('Min = %.2f ms, Mean = %.2f ms, Max = %.2f
ms',minval,avgval,maxval);
legend(ax,'histogram','max            value','min            value','avg
value','Location','Best')


xlabel('Time delay (ms)')
ylabel('Number of instances')
str = {sprintf('Time delays for %s', maintitle{k}),strstats};
title(str,'Interpreter','None')
```

```
file_hist = strcat(PathName,maintitle{k},'_hist.jpg');
file_hist2 = strcat(PathName,maintitle{k},'_histoverlay.jpg');
saveas(h1,file_hist)



h2 = figure;
bar(avgvalues);
axis([0 inf 0 maxavg])
hold on
errorbar(avgvalues,stdev,'.')
str2 = sprintf('Average time delays per trial for %s', maintitle{k});
title(str2, 'Interpreter','None');
xlabel('Trial number')
ylabel('Average time delay (ms)')


file_avg = strcat(PathName,maintitle{k},'_avgvals.jpg');
saveas(h2,file_avg)

h3 = figure;
bar(successrate);
xlabel('Trial number')
ylabel('Success rate (%)')
str3 = sprintf('Success rate per trial for %s', maintitle{k});
title(str3,'Interpreter','None');
axis([0 inf 0 120])

file_success = strcat(PathName,maintitle{k},'_successrate.jpg');
saveas(h3,file_success)

totarray = reshape(time_test,nrows*nFile/datadim,1);
h4 = figure;
plot(totarray,'o')
axis([0 10000 mintime maxtime])
xlabel('Data point')
ylabel('Time delay (ms)')
str4 = sprintf('Total data points for %s', maintitle{k});
title(str4,'Interpreter','None');

file_alldata = strcat(PathName,maintitle{k},'_alldata.jpg');
saveas(h4,file_alldata)

trialstats = zeros(ncols/2,5);
for j=1:ncols/2
    trial = time_test(:,j);
    mintrial = min(trial);
    maxtrial = max(trial);
    avgtrial = avgvalues(j);
    medtrial = median(trial);
    stdtrial = stdev(j);
    ranget = maxtrial - mintrial;
    trialstats(j,:)=[mintrial maxtrial avgtrial medtrial stdtrial];
    h5 = figure;
 [N, edges] = histcounts(trial);
 maxy = max(N);
```

```
 bar(ax,edges(1:end-1),N,'EdgeColor','None','FaceColor',colors{j})
histogram(trial,'EdgeColor','None')
axis([mintime maxtime 0 maxy])
hold on
line([maxtrial              maxtrial],[ylim],'Color','r','LineStyle','--
','LineWidth',LW)
line([mintrial              mintrial],[ylim],'Color','m','LineStyle','--
','LineWidth',LW)
line([avgtrial              avgtrial],[ylim],'Color','g','LineStyle','--
','LineWidth',LW)
% line([medtrial medtrial],[ylim],'Color','g','LineStyle','--')
strstats  =  sprintf('Min  =  %.2f  ms,  Mean  =  %.2f  ms,  Max  =  %.2f
ms',mintrial,avgtrial,maxtrial);


legend('histogram','max             value','min             value','avg
value','Location','Best');
 xlabel('Time delay (ms)')
ylabel('Number of instances')
str5  =  {sprintf('Time  delays  for  %s,  trial  %d',  maintitle{k},
j),strstats};
title(str5,'Interpreter','None')

file_hist_trial                                                          =
strcat(PathName,maintitle{k},'trial_',num2str(j),'_hist.jpg');
saveas(h5,file_hist_trial)



end
legend(ax,'histogram','max   value','min   value','avg   value','Trial
1','Trial 2','Trial 3','Trial 4','Trial 5','Trial 6','Trial 7','Trial
8','Trial 9','Trial 10','Location','Best')
saveas(h1,file_hist2)

h6 = figure;
boxplot(time_test)
axis([0 inf mintime maxtime])
xlabel('Trial number')
ylabel('Time delay (ms)')
str6 = sprintf('Boxplot for %s',maintitle{k});
title(str6,'Interpreter','None')
file_boxplot = strcat(PathName,maintitle{k},'_boxplot.jpg');
saveas(h6,file_boxplot)

fid = fopen(strcat(PathName,maintitle{k},'_stats.csv','w'));

for i = 1:1000:9001
    newtime(i:i+999)=time_test(:,(i-1)/1000+1);
end
stdall = std(newtime);
Stats        =        {'Minimum','Maximum','Average','Median','Standard
Deviation';minval,maxval,avgval,medval,stdall};
fid = fopen(fullfile(PathName,strcat(maintitle{k},'_stats.csv')),'w');

fprintf(fid,'%s,%s,%s,%s,%s\n',Stats{1,:});
fprintf(fid,'%f,%f,%f,%f,%f\n',Stats{2,:});
```

```
fprintf(fid,'%s\n','Trials');
dlmwrite(fullfile(PathName,strcat(maintitle{k},'_stats.csv')),trialstat
s,'-append');
for j=1:ncols/2
    fprintf(fid,'%f,%f,%f,%f,%f\n',trialstats(j,:));
end
fclose(fid);

end
```

Appendix C: Creating Custom Box in OpenViBE

Follow the instructions located here: http://openvibe.inria.fr/build-instructions/ to acquire the OpenViBE source code. This is necessary to add custom boxes. Once the program has been successfully built, add both ovpCBoxAlgorithm codes from the OpenVibe folder on https://github.com/nehatk17/NeuroHubNetworkModule/tree/master/Codes/OpenViBE%20custom%20box to the following director: openvibe/plugins/processing/network-io/src/box-algorithms. Copy the ovp_main file to the following directory: openvibe/plugins/processing/network-io/src. Finally, rebuild the program. Windows IDE build can be launched from openvibe/scripts using win32-launch-vc. Launch the program using the openvibe-designer application located in openvibe/dist. Make sure that the "show unstable" is checked in the boxes panel on the right in the OpenViBE Designer GUI. The Event Marker to TCP should show up under the Network IO category.