CAFFEINE: Collaborative Affordable Framework For Experiments in Interactive Networked Electronics

Scott Eric Petersen
Yale University
scott.petersen@yale.edu

Alejandro Mayagoitia
Yale University
alejandro.mayagoitia@yale.edu

ABSTRACT

CAFFEINE is a mixed hardware/software framework that enables real-time low-latency sonification of sensor data from wearable embedded systems. The framework utilizes a many-to-one-to-many network topology, allowing unlimited pods and clients, constrained only by hardware availability. Wireless, battery-powered pods, built around the ESP32 microcontroller platform, host sensor arrays that capture diverse data, including motion, sound, distance, and light. Data are transmitted to a broker program that routes them to registered clients for sonification or further processing. CAFFEINE prioritizes accessibility and affordability, designed for artists and musicians with minimal technical expertise. Its extensible architecture supports creative applications, from live movement sonification to environmental data-driven compositions, reducing barriers to networked sensor technologies. The framework files and utilities can be found on the GitHub project [1].

1. INTRODUCTION

Numerous commercial motion capture systems have been developed specifically for use in musical contexts. They have various form factors, but most are designed to be worn[2] or attached to an instrument[3]. Some companies offer a variety of devices[4], but the functionality of each device is fixed and the systems are only extendable in software. In contrast, the CAFFEINE framework integrates open-source software with low-cost, commercially available hardware to provide flexible, extensible configurations that support a wide range of applications.

2. DESCRIPTION

CAFFEINE enables real-time, low-latency sonification of sensor input from small, wearable embedded systems called "pods" in a network topology of many-to-one-to-many. Any number of collaborators can register to receive data from any number of pods on the network. The possible number of pods and clients is limited only by hardware. The framework is affordable, reproducible, and accessible for those with little to no hardware and software development knowledge, and the use of the system is simple once it is set up. Collaborators simply need to power on the pods, start the data broker program, and register their computers with the broker to begin receiving sensor data. The simplicity of the system allows musicians to focus on composition and performance, rather than technology.

We anticipate diverse applications of this system. An obvious use case would be the real-time sonification of movement where a pod is being worn by a dancer. However, a pod or pods could be used to train a machine learning model on complex human gestures. Or, in a conceptual space, we imagine long-scale musical forms could be informed by light and temperature tracked over days or weeks by a pod that is installed and stationary, or being transported through a space by an autonomous vehicle or kinetic sculpture. Whatever the application, the CAFFEINE framework has been designed to lower the barrier to electronic sensors and networked music.

3. HARDWARE

3.1 Overview

The minimal hardware configuration for CAFFEINE is an embedded system we refer to as a 'pod', and a laptop to program the pod and run both the broker program and a client program to sonify the data. An accessible network is also required as communication from the pod to the broker machine is done via Wi-Fi.

3.2 Pod

The CAFFEINE pod builds on and advances the embedded system introduced in our previous project, the [project anonymized] [5][6]. It consists of a circuit that connects multiple sensors to an Espressif ESP32 S3 development module [7]. The ESP32 is an ubiquitous microcontroller for embedded applications. Here, we specifically make use of one of its ADC channels and one of its I2C interfaces for sensor input, and its Wi-Fi capabilities for data transmission.

3.3 Power

The power for the ESP32 and the sensor array is supplied by a 3.7v 500 mAh lithium ion flat-pack battery [8] and a boost module to step up the voltage from 3.7v to 5v [9]. The boost module has the added benefit of allowing the battery to be charged in situ. The battery run time for this power configuration is approximately 150 minutes. A battery with a larger capacity can easily be substituted for longer performances.

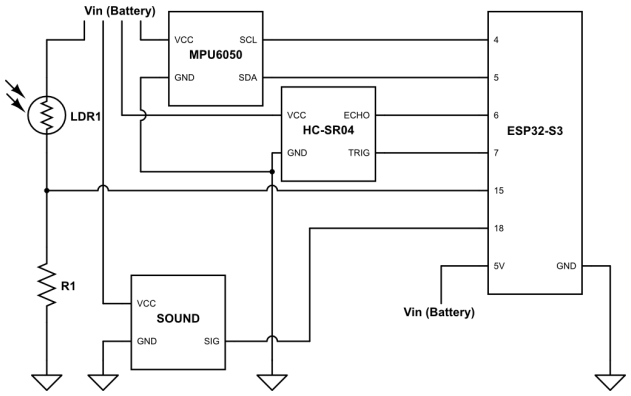

Figure 1: Circuit diagram for one pod.

3.4 Sensor Array

The sensor array consists of an HC-SR04 ultrasonic distance sensor [10], an MPU-6050 accelerometer/gyroscope [11], a Seeed Studio Grove Sound Sensor [12], and a photoresistor. The HC-SR04 distance sensor has an operational range of 2cm to 4m. However, in our tests, the device is unreliable at its extremes and is most stable between 11 cm and distances under 3.5 m. Additionally, noise occurs when the sensor is suddenly moved from the short end of its range to its maximum distance. The proper use of a low-pass filter to avoid noise is important if distances are used as triggers for musical events.

The MPU-6050 has a 3-axis accelerometer, a 3-axis gyroscope sensor, and a temperature sensor. The device is often used in drone construction and anywhere the orientation of a moving object is important [13]. Here, we provide only for the sending of the X, Y, and Z rotational velocity of the pod; however, sending other values is as simple as querying them. There are many libraries available for the MPU-6050 that simplify the acquisition and processing of its various sensor data. For our framework, we chose a library that was as lightweight and specific to the sensor as possible [14].

The sound sensor and the photoresistor are simple devices that return values based on momentary loudness and ambient light levels.

3.5 Cost

All pod components are commercially available. The total cost of the components required to build one pod, including a prototyping breadboard and cables, is approximately $50. To be cost-effective, many of the components we use are sold in packs of three to five. Thus, the cost of constructing subsequent pods decreases significantly. To illustrate, the market cost to obtain all the necessary parts for five pods from reputable online retailers is approximately $158 at the time of writing.

4. NETWORK TOPOLOGY

The network topology of many-to-many via a data broker means that only one IP address needs to be known for the system to function; the IP address of the computer that runs the broker program. All pods must be individually named and programmed with the IP address of the broker, and all clients must use that address to register for data.

Institutional and home networks are compatible with CAFFEINE; however, we recommend using a dedicated, private router that is offline and configured with a static IP address for the computer running the broker program. This setup eliminates the need to reprogram the pods when changing locations and only requires the broker's IP address to be published once, simplifying client registration regardless of where the system is deployed.

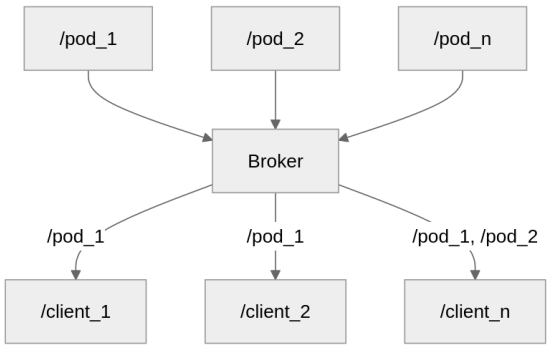

Figure 2: A CAFFEINE configuration where n number of pods are connected in an arbitrary way to n number of clients via the broker.

5. SOFTWARE

5.1 Overview

The software configuration for CAFFEINE consists of two files, an Arduino file that runs on the pod that defines its behavior, and a Python file that runs on the computer acting as the data broker that receives messages from the pod and forwards them to any registered clients on the localhost or the network. The last piece of a complete system is the client software to sonify the data received from the broker. Although that software must be written by the user of the system according to their specific needs, we provide SuperCollider examples that demonstrate functionality and best practices.

5.2 Embedded System Program

The pod program defines the name of the pod that identifies it to the broker. It registers the target computer running the broker software, attaches the pod to a specified network, measures the values from the attached sensor array, and sends those data to the broker program at a user-specified interval via one of two methods, either as a string via UDP or as typed data via OSC. Messages sent as strings via UDP are sent with the name of the pod followed by space-delimited sensor data as a string. Packets sent as OSC messages prepend the pod name as a symbol `/pod1` and then an array of data where each comma denotes a po-

tentially different data type. An example message sent via OSC might be

```
/pod1: [0.030, 0.010, -0.863, 2253,
    22.09, 97]
```

where the first three data points are the X, Y, and Z values of the MPU-6050, the fourth data point is the loudness, the fifth is the distance registered by the HC-SR04 in centimeters, and the last is the light value registered by the photoresistor.

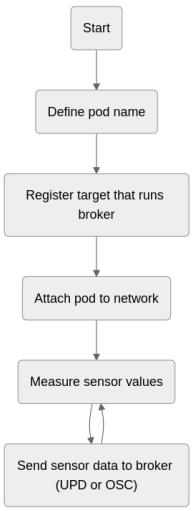

Figure 3: Embedded System Program flow control.

UDP and OSC communication have pros and cons. For UDP, string concatenation is done via a fixed-size character array. Thus, the memory allocation is secure and consistent, and messages are sent and received regularly, with jitter and packet loss typical of any UDP communication. However, this method requires that the messages are parsed once they arrive at the client, as, for example, in SuperCollider, data are received as symbols. The user must convert those symbols to an appropriate data type to be used in sonification, for example, to an integer to be used for frequency control. Languages like SuperCollider and PureData easily accommodate this requirement, but other programs such as digital audio workstations may not.

OSC allows client programs to receive data in the type they were sent and has the added benefit of being universally supported by audio software. This results in data that are usable without conversion. However, the size of the data (e.g. individual sensor readings) in the OSC message must be managed to maintain consistent message size. Our OSC program rounds all values to the second decimal point, which may or may not be best for all use cases, but results in consistent message size. Users must also carefully consider what OSC library they use. Our testing showed that several available libraries resulted in communications that suffered from occasional periods of increased latency. The library used in CAFFEINE and the one that we recommend is ArduinoOSCWiFi.

In either case, the user should choose the method that makes the most sense for their project requirements. If an exact message size is not required, OSC should be preferred for ease of use and maximal compatibility with audio software.

5.3 Data Broker Program

The broker program defines how the data from the pods are received, registers clients who want to receive data and the specific pods from which they want it, and forwards the data as it is sent.

The broker is configured to accept data from any IP address on the network as long as the message being sent is prefixed with the name of a pod registered by the broker. It defines a port to receive data (5001) and a port to send data to clients (9001).

Clients register with the broker by sending a `/register` message with the name of the target pod as the second argument. The broker records the IP address for the client and associates it with the specified pod, for example, `/pod1`. When messages are received from `/pod1`, the broker checks a dictionary for registered clients to receive data from that pod and forwards the data to the appropriate clients. Clients may register to receive data from any and all pods on the network.

As with the pod program, there are two versions of the broker, one that parses strings and one that receives and forwards OSC messages [15].

5.4 Sonification Program

We do not specify a program, but rather a set of considerations that electronic music practitioners will be familiar with when working with abstract, non-musical data. The data from the pod are in arbitrary ranges determined by the sensors themselves. Value ranges may be unipolar or bipolar, and may be small or large. In addition, the data are received at a clocked rate determined by the user, but the exact timing cannot be counted on due to network jitter. The end user of the system must determine how many pods they want to receive data from, what sensors from each array they will use, how often they want to use those values in their program, and how to map the values to musical parameters in their chosen system.

6. DATA TRANSMISSION TESTING

Preliminary observations of the CAFFEINE pod indicate a high degree of responsiveness during sonification, particularly when standard noise mitigation strategies such as low-pass filtering are employed. To precisely measure communication parameters, namely latency, packet loss, and transmission rate, modifications were made to the OSC version of both the pod and broker programs. Furthermore, a supplementary utility, `endpoint_logger.py`, was developed to support end-to-end latency measurement.

6.1 Software Modifications for Testing

To facilitate precise latency measurements, the pod firmware was augmented to synchronize with NTP and include both a timestamp and a sequential message counter in each OSC message. Corresponding modifications were implemented in the broker program to compute pod-to-broker latency by comparing the received timestamp against its own system time. Packet loss was inferred by detecting discontinuities in the message count sequence. The broker was also updated to record latency values in a CSV file and to forward complete messages—including timestamp—to

one or more client machines running the logger program, which computed end-to-end latency by comparing the pod's original timestamp with the current time on the client.

6.2 Hardware Configurations

Three pod configurations were designed to simulate different use cases:

- **Pod 1** (full configuration): Included MPU-6050 gyroscope, distance sensor, light sensor, and sound sensor.
- **Pod 2**: Included only the MPU-6050 gyroscope.
- **Pod 3**: Included only the light and sound sensors.

6.3 Testing Methodology

Two sets of experiments were conducted:

1. **Single-Pod Tests**: Each configuration was individually evaluated using a set-up consisting of one pod, one computer running the broker, and one client computer running the endpoint logger. This configuration simulated a typical performance scenario involving a single performer and a single data sonifier.
2. **Multi-Pod Tests**: The three pod configurations were evaluated concurrently using one broker computer and three client machines, each assigned to receive data from a distinct pod. This configuration was designed to model a multi-pod performance scenario.

For all tests, a dedicated private network was used. The test devices were approximately 20 feet away from the router with a partial line of sight.

6.4 Results

A pronounced initial spike in latency and packet loss was observed during the startup phase as each pod connected to the network. This behavior is illustrated in Figure 4, which shows the first 500 samples from Pod 1.

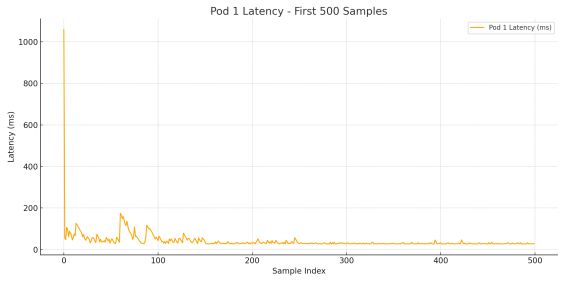

Figure 4: Latency plot of the first 500 samples received from Pod 1 in the single-pod tests.

A recurrent pattern of packet loss was detected in all configurations: Three packets were received, followed by a brief loss interval, after which the transmission stabilized for the remainder of the session (see Table 1).

Although Pods 2 and 3 exhibited similar startup behavior, the magnitude of latency and packet loss was reduced relative to Pod 1, as shown in Figure 5.

Table 1: Packet loss for Pod 1 over the first seven samples.

Latency	Seq number	Dropped	Total
1484.593	0	0	0
480.142	1	0	0
470.851	2	0	0
81.064	41	38	38
71.51	42	0	38
62.811	43	0	38
55.271	44	0	38

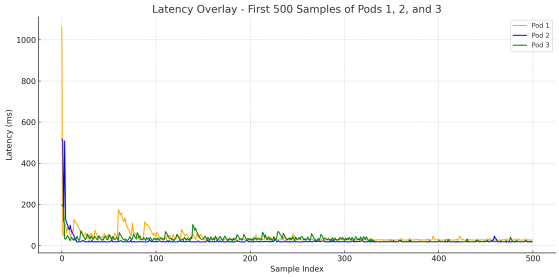

Figure 5: Latency plot of the first 500 samples received from Pods 1, 2, and 3 in the single-pod tests.

Excluding the initial samples, a comparative latency analysis revealed that Pod 1, with the complete hardware configuration, consistently demonstrated a higher average latency (7-9 ms) than the other two configurations, in both individual and simultaneous tests (see Figure 6.)

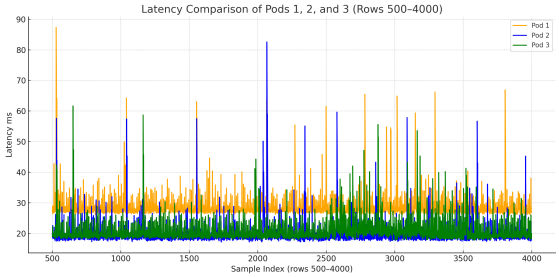

Figure 6: Overlay plot showing latency for Pods 1, 2, and 3 from the single-pod tests.

The average message throughput for both sets of tests was measured at approximately 98 messages per second (see Table 2), with rates slightly faster for the multi-pod arrangement. Both sets of tests show a somewhat broad minimum to maximum latency range, although the mean latency remained within acceptable limits for real-time performance scenarios (see Tables 3 and 4).

6.5 Interpretation of Testing Data

Somewhat counterintuitively, the tests show that overall performance did not degrade in the multi-pod tests (see Table 4), and in some instances are actually better, suggesting that the framework's many-to-one-to-many communication model scales effectively under load (Figure 7). It should also be noted that the inclusion of diagnostic and logging routines introduced additional computational overhead. In live contexts, actual performance may improve

Table 2: Message throughput for all pods in both single and multi-pod tests. Numbers are messages per second.

Pod	Avg Messages
Pod 1 (ind)	96.78
Pod 2 (ind)	98.17
Pod 3 (ind)	98.33
Pod 1 (grp)	99.14
Pod 2 (grp)	99.33
Pod 3 (grp)	98.29

Table 3: Latency Statistics for single-pod tests (Rows 500–4000) where all times are in milliseconds

Pod	Min Latency	Max Latency	Avg Latency
Pod 1	26	87.33	28.63
Pod 2	17.21	82.58	19.95
Pod 3	17.82	61.69	20.7

modestly in the absence of such instrumentation.

Initial tests were performed on an enterprise network where significantly poorer results were obtained. The presence of an unknown number of additional network devices and the lack of proximity to the wireless access point (WAP) resulted in higher latency and packet loss, causing us to abandon the tests. For optimal performance, users are strongly advised to use a dedicated router.

7. WORKFLOW

Users of the CAFFEINE framework must purchase the components and assemble the circuit described here to complete their pod. Because the ESP32 has many additional pins unused in our configuration, the pod can be easily extended with additional sensors. Alternately, given fewer sensors, inputs may be omitted. In either case, only small changes to the pod program are necessary.

The Arduino development software and Python 3 must also be installed if they are not already on the development system. The pods must initially be programmed by the Arduino IDE. The broker software must be run on either the same or a different computer whose IP address is known and entered into both the pod program and the client program. Finally, using an audio program such as SuperCollider, the user can register with the broker to begin receiving real-time sensor data from the pod.

8. FUTURE DIRECTIONS

8.1 Enclosures

CAFFEINE describes a system and configuration, but does not specify the end-use of the system; therefore, no enclosures have been described. Because applications will vary considerably, housings for the pod will take diverse forms. For example, if a live coder wants to incorporate the dancing of members of their audience into their performance, the enclosure for the pod should be as comfortable and as streamlined as possible. In some cases, for example, to track the movements of hikers as they traverse some remote location, it would be better to prioritize ruggedness. Such an application could see the system augmented with a photovoltaic cell when electricity is unavailable.

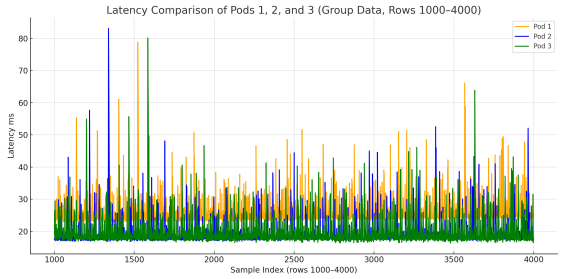

Figure 7: Overlay plot showing latency for Pods 1, 2, and 3 from the multi-pod test.

Table 4: Latency Statistics for the multi-pod test (Rows 1000–4000) where all times are in milliseconds.

Pod	Min Latency	Max Latency	Avg Latency
Pod 1	22.79	78.79	26.12
Pod 2	16.91	83.09	19.47
Pod 3	16.33	80.08	19.69

8.2 Data processing, Profiles, and Classes

The data returned from the pod are not processed. While this may be preferable to advanced users, others might prefer pre-processed data that can be readily and intuitively applied in a musical context. For convenience, processing could be linked to a specific profile that also determines which sensor data are received. For example, a profile named `/gyro` might exclusively return data from the MPU-6050 sensor, with an n-dimensional Kalman filter applied to stabilize the X, Y, and Z values. These stabilized values could then be assigned to commonly used musical ranges, for example, -1 to 1 for panning, 0 to 1 for amplitude, and 100 to 5000 for filter cut-offs or other frequency-related parameters. To streamline these processes, a program class or object could be developed for popular audio programming languages, simplifying integration and use.

8.3 Alternate Network Topologies

It may be the case that with small numbers of pods and clients, a peer-to-peer network topology may be better. Without an intermediary broker, communication should be faster. A list of the static IP addresses of the individual pods running servers could be distributed to users, and the data sent directly to their connected clients. If the P2P topology actually resulted in lower latency, testing would be required to determine at what scale, if any, that configuration failed to improve communication rates.

9. PARTS LIST

1. Espressif ESP32 S3 Dev Module [7]
2. HC-SR04 Ultrasonic Distance Sensor [10]
3. MPU-6050 module [11]

4. Seeed Studio Grove Sound module [12]

5. Photoresistor - resistance values vary and should be chosen based on overall light levels in the anticipated performance space. For lower light levels, prefer lower resistance values.

6. 10k Ohm resistor

7. 3.7V Lipo Battery 500mAh Rechargeable Lithium ion Polymer Battery 502535 Lithium Polymer ion Battery with JST Connector [8]

8. Seeed Studio Lipo Rider Plus (Charger/Booster) - 5V/2.4A USB Type C [9]

9. micro USB cable

10. WB-102 Solderless Breadboard

11. 20 solderless breadboard cables

References

[1] *CAFFEINE*, Yale Open Music Initiative. [Online]. Available: https://github.com/scottericpetersen/CAFFEINE

[2] *MiMU Gloves*, MI.MU GLOVES LIMITED. [Online]. Available: https://mimugloves.com/about/

[3] *MUGIC® Motion Sensor 2.0*, MUGIC MOTION. [Online]. Available: https://www.mugicmotion.com/

[4] *Instruments of Things Product Page*, Instruments of Things GmbH. [Online]. Available: https://instrumentsofthings.com/collections/frontpage

[5] S. Petersen, M. Santolucito, and K. Kaczmarek, "Studio Report: Yale Open Music Initiative," in *Proceedings of the 2019 International Computer Music Conference*, 2019, pp. 530–532.

[6] *OMI-Pod*, Yale Open Music Initiative. [Online]. Available: https://github.com/Yale-OMI/OMI-Pod

[7] *ESP32-S3-DevKitC-1*, Espressif System Co. [Online]. Available: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-devkitc-1/index.html

[8] *Li-Polymer Battery Technology Specification*, Shenzhen PKCELL Battery Co. [Online]. Available: https://cdn-shop.adafruit.com/product-files/1578/C1854+PKCell+Datasheet+Li-Polymer+503035+500mAh+3.7V+with+PCM.pdf

[9] *Lipo Rider Plus(Charger/Booster) - 5V/2.4A*, Seeed Technology Co. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/338/106990290_Web.pdf

[10] *Ultrasonic Ranging Module HC-SR04*, Elecfreaks, Inc. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[11] *MPU-6000 and MPU-6050 Product Specification*, InvenSense Inc., August 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[12] *Grove - Sound Sensor User Manual*, Seeed Technology Co., September 2015. [Online]. Available: https://www.mouser.com/catalog/specsheets/Seeed_101020023.pdf

[13] S. Sing Kang, N. T. Singh, and R. Grover, "Improving Drone Agility and Stability with Advanced Flight Control Systems Using MPU-6050 IMU and High-Fidelity Simulation," in *2024 International Conference on Cybernation and Computation (CYBERCOM)*, 2024, pp. 121–127.

[14] R. J. Fetick and tockn, "MPU6050_light," accessed January 12, 2025. [Online]. Available: https://github.com/rfetick/MPU6050_light?tab=readme-ov-file

[15] H. Tai, F. Bruggisser, and S. Goodchild, "ArduinoOSC," accessed January 13, 2025. [Online]. Available: https://github.com/hideakitai/ArduinoOSC/tree/main