SharkTooth: A Scalable Real-Time Algorithm for BLE-Based Wireless Body Sensor Networks Synchronization

SharkTooth: A Scalable Real-Time Algorithm for BLE-based Wireless Body Sensor Networks Synchronization

Nicolò Landra, *Graduate Student Member, IEEE*, Danilo Demarchi, *Senior Member, IEEE*, Paolo Motto Ros, *Member, IEEE*

***Abstract*—The rapid expansion of Wireless Body Sensor Networks (WBSNs) in healthcare, rehabilitation, and movement analysis demands precise time synchronization across sensor nodes to ensure reliable multi-modal data fusion. Existing synchronization solutions for Bluetooth Low Energy (BLE)-based WBSNs often rely on hardware-level timestamping or operate in advertising mode, limiting their scalability, interoperability, or usability in real-time interactive applications. In this paper, we present SharkTooth, a novel, scalable, and real-time synchronization algorithm that operates entirely at the application layer of BLE. Unlike prior approaches, SharkTooth does not require any firmware modifications or specialized hardware, and it is compatible with commercial off-the-shelf BLE devices. The algorithm employs an adaptive timestamp correction technique that mitigates both deterministic drift and unpredictable delays caused by packet retransmissions and protocol variability. Extensive experimental validation using up to 12 sensor nodes and 18 BLE network configurations demonstrates sub-millisecond synchronization accuracy, with a median absolute average synchronization error between 0.21 ms and 0.62 ms, even in congested network scenarios. Compared to state-of-the-art solutions, SharkTooth is uniquely capable of delivering robust, long-term synchronization in high-throughput, multi-node systems using only connection-based BLE communication. By prioritizing compatibility, reproducibility, and real-world performance, SharkTooth advances the design of scalable and interoperable WBSNs for next-generation biomedical and wearable IoT applications.**

***Index Terms*—Bluetooth Low Energy (BLE), Synchronization, Digital healthcare, Wireless sensor networks, Internet of Things**

I. Introduction

In recent years, the use of Wireless Body Sensor Networks (WBSNs) in biomedical applications has dramatically increased, representing a significant step toward personalized healthcare [1]. This growth is primarily driven by the need for continuous, non-invasive monitoring of physiological parameters in both clinical and everyday settings. WBSNs consist of multiple sensor nodes distributed across the human body, each capable of acquiring and processing diverse biomedical data, including physiological signals and movement kinematics.

In such systems, the accurate synchronization of different data streams is one of the most critical aspects. Indeed, temporal misalignment among data from different devices can result in inaccurate analyses and unreliable conclusions [2]. This requires a synchronization accuracy that varies depending on the target application. However, a commonly accepted threshold is achieving synchronization accuracy within the sampling interval, ensuring sample-level precision. [3]–[6].

The time de-synchronization in WBSNs can be originated from two main causes. First, the local clocks in devices have limited accuracy in their declared frequency and are prone to drift apart due to frequency differences in their crystal oscillators. These discrepancies are quantified by the tolerance value declared by the manufacturer and expressed in part per million (e.g., $\pm$ 10 ppm). Over time, this drift can worsen due to temperature variations, power fluctuations, mechanical stresses, and aging [7], [8].

Second, the communication protocol used for data exchange within the WBSN introduces additional delays. They occur at various stages of the communication stack, including data sending, accessing, transmitting, and receiving, leading to a time gap between the actual data generation and its final reception [5], [9], [10]. As a result, de-synchronization among multiple acquisition sensors occurs when one or more delay components become unpredictable.

Many wearable systems on the market implement custom communication protocols, offering optimized performance, security, and synchronization accuracy tailored for specific applications [11], [12]. However, these solutions are typically built on proprietary technologies, making them incompatible with external systems unless dedicated drivers or bindings are developed for interoperability. This lack of compatibility hinders the creation of an interoperable ecosystem of WBSNs. In contrast, standardized communication protocols enable wearable devices to seamlessly interact with consumer electronics, such as smartphones, tablets, and PCs. Among these standards, Bluetooth Low Energy (BLE) has emerged as one of the most widely used options for low-power sensor [10], [13]–[15].

BLE communication supports two modes: advertising and connection [16]. While advertising enables fast, unidirectional broadcasts to multiple devices, it lacks security and limits data exchange to simple sensor outputs, making it unsuitable for complex acquisition tasks. In contrast, the connection mode allows secure, bidirectional communication between a central and multiple peripherals, organized in piconets. Data is exchanged at fixed intervals, called Connection Intervals (CIs), using a time-sharing strategy to prevent collisions [17].

Date of current version: July 13, 2025. Corresponding author is Nicolò Landra; email: nicolo.landra@polito.it

All the authors are with Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy.

In the context of the BLE connection, the Generic ATTribute (GATT) profile governs how data is accessed, with read, write, and notify operations enabling configuration, control, and real-time streaming [16]. Although connection mode ensures reliable transmission through acknowledgments and retransmissions [17], this introduces variable latency, posing challenges for accurate time synchronization in real-time applications.

A. Time Synchronization Algorithms

To address the time synchronization issues within WBSNs, or more generally within Wireless Sensor Networks, a wide range of methods have been proposed, aiming to balance precision, energy efficiency, and implementation complexity. These algorithms can be categorized based on their communication protocol, operational layer, and energy-awareness.

One of the broadest categories comprises protocol-agnostic synchronization methods, which are designed for general wireless sensor platforms and are not tied to any specific communication protocol. Within this category, conventional accuracy-oriented methods focus primarily on achieving high synchronization precision, often leveraging MAC-layer access and hardware timestamping. Prominent examples include the Reference Broadcast Synchronization (RBS) method [18], Timing-sync Protocol for Sensor Networks (TPSN) [19], and Flooding Time Synchronization Protocol (FTSP) [9].

RBS adopts a receiver-to-receiver strategy in which nodes synchronize based on the arrival times of broadcast messages from a common reference node, effectively eliminating non-deterministic sender delays. TPSN, in contrast, implements a sender-receiver synchronization scheme organized in a hierarchical structure. It achieves pairwise synchronization through two-way messaging and timestamping at the MAC layer. FTSP builds on this foundation by introducing a flooding mechanism to propagate synchronization messages and estimate clock offset and skew using linear regression. A key contribution of FTSP is the concept of a reference point, composed of a pair of global and local timestamps that refer to the same time instant [9]. This reference is generated at the receiver upon receipt of a timestamped message and serves as the basis for estimating global time across the network.

Another important class of protocol-agnostic methods emphasizes energy efficiency and reduced computational complexity, particularly in scenarios involving resource-constrained sensor nodes [20]–[23]. As a notable example, Huan et al. [23] introduced a Beaconless Asymmetric energy-efficient Time Synchronization (BATS), in which sensor nodes are only responsible for timestamping outgoing data messages, while all synchronization computations are fully delegated to a central head node. Moreover, by leveraging MAC-layer timestamping, this technique achieves sub-microsecond synchronization accuracy while reducing energy consumption at the node level by up to 95 % compared to traditional flooding-based protocols.

Although these protocol-agnostic synchronization methods achieve excellent accuracy and provide foundational architectural principles, they are not tailored to the BLE communication stack and its inherent constraints, such as unidirectional advertising mode and centralized scheduling in connection mode. Furthermore, they depend on MAC-level timestamping mechanisms, which are generally inaccessible in most commercial BLE platforms due to restrictions imposed by the protocol stack and firmware architecture [24]. As a result, these methods are not directly applicable to BLE-based systems.

To address these limitations, a variety of BLE-specific synchronization algorithms have been proposed, each designed to operate within the constraints of the BLE protocol while striving to maintain synchronization accuracy and minimize energy and implementation overhead. These methods can be grouped in two classes according to the employed communication mode.

The first class includes algorithms developed for advertising-based BLE sensor networks [5], [7], [10], [14]. Among these, two notable examples are CheepSync [10] and BlueSync [14]. CheepSync utilizes low-level timestamping at both transmitter and receiver to compensate for the unpredictable delays associated with BLE packet transmission. BlueSync achieves even higher accuracy by exploiting non-connectable advertising packets, single-channel scanning, and a hardware feature known as Programmable Peripheral Interconnect (PPI), available on Nordic Semiconductor microcontrollers.

Despite achieving synchronization accuracy on the order of tens of microseconds for CheepSync and hundreds of nanoseconds for BlueSync, these methods are inherently limited to advertising mode. This restriction is significant because advertising mode supports only unidirectional message exchange, thereby preventing the use of read, write, and notify operations, which are typically required for interactive communication with sensor nodes. As a result, a second class of synchronization approaches has emerged, specifically tailored for connection-oriented BLE networks [6], [8], [13], [24], [25].

The first approach proposed for synchronizing a BLE piconet involved using connection events [13], [24]. Specifically, Dian et al. observed that the time difference between events generated on the master and slave was generally below 800 µs [24]. Although this method achieves submillisecond synchronization, it does not compensate for clock drift and requires repeated connection-disconnection operations.

In contrast, more recent methods have adopted strategies based on packets flooding from sensor nodes to the central device and implemented at the application-layer [6], [8], [25]. This design improves deployability and platform compatibility, as it avoids reliance on lower-layer timestamping and leverages standard BLE stack features.

Specifically, the method proposed by Li et al. [25] pairs timestamps from the central and peripheral nodes and applies a linear least-squares regression algorithm to synchronize data across sensors. This solution appears precise enough for most biomedical applications and is easily transferable to commercial microcontrollers. However, their experiments were limited to two devices, and they did not consider delayed or blocked Bluetooth transmissions, which are relevant in practical scenarios. Balasubramanian et al. [8] proposed a solution using a neural network to process the timestamp pairs

sequence, creating a "virtual clock" for each device. Despite its promising results in mitigating BLE packet transmission unpredictability, the study lacks clarity on how the neural network was trained and its scalability to larger networks. Ultimately, Depolli et al. [6] proposed an offline synchronization method that corrects for random transmission delays and packet loss while relying solely on node-to-central BLE connections. Although their method demonstrated robustness and accuracy for both short- and long-term biomedical applications, their validation was limited to three devices, leaving its scalability untested.

Motivated by the lack of real-time synchronization methods that scale beyond a few nodes and remain robust to packets loss, we present a novel application-layer algorithm that addresses these limitations while maintaining full compatibility with standard BLE devices and suitability for real-time biomedical applications.

B. Contribution

This work aims to propose and validate *SharkTooth*, a scalable real-time synchronization algorithm developed to address the lack of a high-level, easily deployable method for synchronizing sensor networks based on the BLE communication protocol. Initially, our aim was to implement a synchronization algorithm inspired by those already proposed in the literature. However, none of the existing solutions proved effectively applicable to the wearable sensor system developed by our research group (Section II), as they were designed with overly simplistic use-case assumptions.

In contrast, our system was specifically conceived to support challenging rehabilitation applications, such as the real-time control of Functional Electrical Stimulation (FES) [26]–[29], and the acquisition of surface Electromyographic (sEMG) signals [30], [31] and body kinematics. Therefore, we developed our own synchronization algorithm to address the practical challenges associated with our target applications, as presented in Section III. First, we characterized the time delay across our WBSN and identified the sources of desynchronization. Based on these findings, we designed an algorithm aimed at minimizing synchronization error at the application level.

The proposed algorithm was subsequently tested across multiple network configurations and packet lengths to assess its general applicability (Section IV). To the best of our knowledge, no existing synchronization algorithm has undergone a validation process comparable to ours, demonstrating its scalability and broad applicability. The results of the experimental validation, which are discussed in Section V, showed that the proposed algorithm reached an average accuracy between 0.21 ms and 0.62 ms, considering a broad range of network configurations. These performance values not only meet the needs of the target application domain but also demonstrate the robustness and generalizability of the proposed method. This is an aspect that, to our knowledge, has not been demonstrated by any other work in the existing literature. Section V also provides insights into three fundamental aspects necessary to characterize *SharkTooth* in terms of robustness, scalability, and its position within the state of the art. These include a detailed analysis of synchronization robustness under variable communication delays (Section V-A), a comprehensive comparison with existing BLE-based synchronization methods (Section V-B), and an evaluation of the relationship between synchronization error and throughput requirements in the context of the target application (Section V-C).

II. System Description

The synchronization algorithm proposed in this paper was initially developed for a system of wearable sensors designed for surface ElectroMyoGraphic (sEMG) signal acquisition [30]. Specifically, the system used in this study is an updated version of the one described in detail by Rossi et al. [30], now featuring the added capability of acquiring both acceleration and angular velocity data. This system consists of multiple peripheral nodes that establish bidirectional communication with a central unit via a BLE connection, following a piconet scheme.

A. Wireless Sensor Node

The sensor node, depicted in Figure 1, integrates an sEMG analog front-end (AFE), which operates with a standard bipolar electrode setup, using two sensing electrodes and a reference electrode. It employs commercial off-the-shelf components to filter and amplify the difference between the two signals collected by the sensing electrodes. The resulting amplification chain provides a variable gain (×200–1600) within the 30–400 Hz range. The Apollo3 Blue MCU (ARM Cortex-M4F), housed within the Artemis module from SparkFun Electronics [32], serves as the central processing unit (CPU). This module handles sEMG and inertial data processing, wireless BLE communication (v4.2), and standard USB/I²C interfacing. The USB micro-B connector supports firmware flashing, serial communication, and real-time data visualization via UART. Additionally, the device integrates a low-power 6-DoF IMU (LSM6DSO32) [33], which measures 3D acceleration and angular velocity, making it suitable for movement kinematics analysis. The PCB also includes an external 32.768 kHz oscillator (ABS04W-32.768KHZ-K-2-T) with a frequency tolerance of ± 20 ppm. This oscillator serves as the time source for data timestamping in the firmware.

B. Firmware and BLE Services

The hardware architecture, combined with dedicated firmware developed using the FreeRTOS real-time operating system [34], ensures low power consumption, efficient data processing, and robust wireless connectivity. These features make the system well-suited for long-duration monitoring applications in WBSNs. The firmware routines establish an interface that allows external devices to access internal data, which is structured using the BLE server based on the GATT profile guidelines. The server organizes device attributes into services and characteristics. Among these, the Device Information and Battery services are standard GATT-defined sets that provide details such as device name, serial number, firmware/hardware version, and battery status. Additionally,

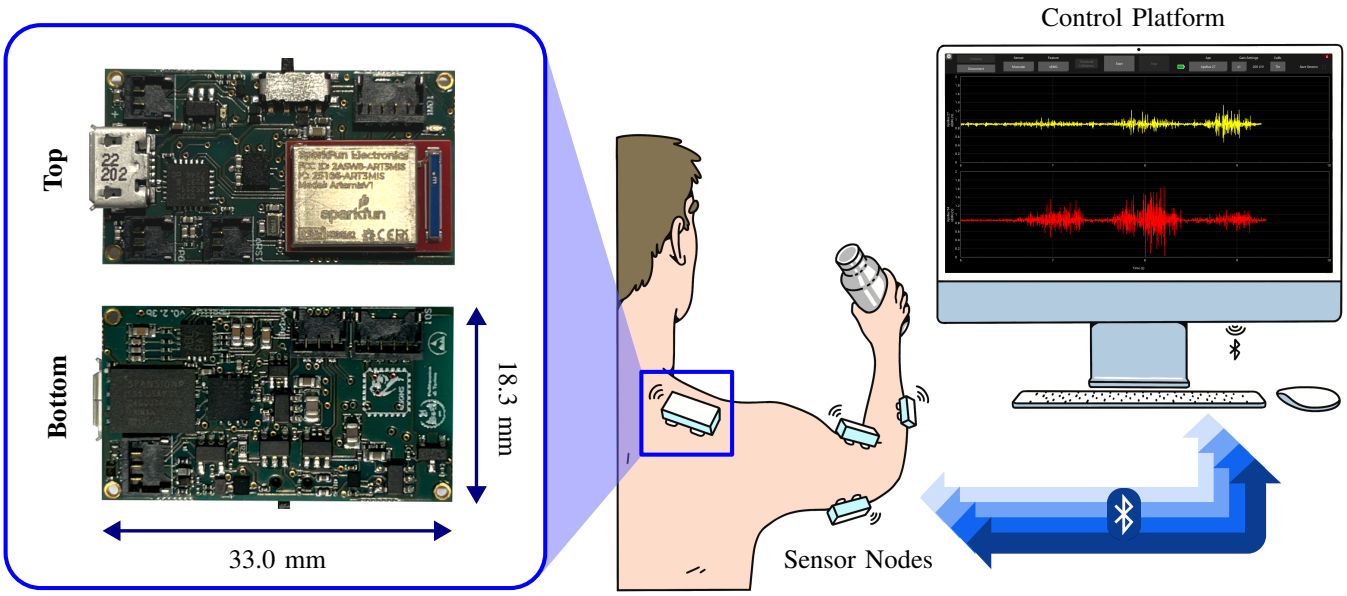


Fig. 1. Overview of the wearable sensing system architecture studied in this work. On the left, top and bottom views of the custom-designed Printed Circuit Board (PCB) of a wireless acquisition node are shown, highlighting its compact size (33.0 × 18.3 mm). The PCB integrates a micro-USB connector, an analog front-end for surface Electromyographic (sEMG) signal conditioning, a processing unit, and a wireless communication module. The power source connector is located at the top left of the top face of the PCB, while the electrodes connector is positioned at the bottom left of the PCB bottom face. In the center, an example of an acquisition setup illustrates how the nodes are applied to the body for motion and muscle activity monitoring. On the right, a control platform receives data from the nodes via Bluetooth Low Energy (BLE) and displays real-time signals for visualization and analysis.

two custom services are designed to handle surface sEMG and IMU data. These services expose Parameters, Command, and Status characteristics, allowing external devices to configure settings, control the sEMG/IMU pipeline behavior, and receive feedback. Data from sEMG/IMU sensors are transmitted as streams of BLE notifications. The sEMG signal is sampled at 1 kHz with a 14-bit resolution, while acceleration and angular velocity data are acquired as 16-bit samples at a maximum sampling frequency of 208 Hz. BLE notifications are sent every 100 ms, storing the sEMG and IMU data into 205-byte and 131-byte packets, respectively, including a 4-byte timestamp and a 1-byte packet number. In the validation process described in Section IV, the sEMG acquisition channel was used to collect a low-frequency reference voltage signal. In that case, the sampling frequency was reduced to 50 Hz for simplicity, allowing us to test different packet lengths while keeping the notification interval and the CI value constant, without compromising generalizability.

C. Control Platform

The control software represents the user access point to the sensor network. It is programmed in Python programming language, and runs on a computer, enabling real-time data processing and saving. It is designed to control the peripheral nodes by sending BLE commands and receiving data notifications through one or more BLE dongles functioning as central units. In this study, we used the Nordic Semiconductor nRF52840 dongles [35], controlled through the set of functions provided by the Blatann library [36].

The decision of developing the *SharkTooth* algorithm in Python was motivated by the widespread adoption of this programming language in WBSNs and Internet of Things platforms [8], [27], [37]–[39]. Python's simplicity, versatility, and extensive ecosystem make it an ideal choice for rapid development and reproducibility. However, we acknowledge that our implementation is not optimized for time-critical applications. Existing low-level synchronization methods in the literature offer excellent performance but often lack scalability. On the other hand, commercial acquisition systems typically rely on custom communication protocols optimized for data transmission, which limits interoperability with other commercial electronic systems. Our approach aims to bridge this gap. Specifically, we prioritized reproducibility and ease of implementation over run-time performance optimization, enabling a general synchronization framework that can be easily scaled across a wide range of WBSN applications.

III. SharkTooth Algorithm

This section is divided into three parts. Section III-A identifies and analyzes three primary factors affecting timestamp accuracy in BLE notifications, establishing the foundation for the SharkTooth synchronization algorithm. Section III-B outlines the implementation of the synchronization protocol, while Section III-C presents a detailed description of the algorithm itself.

Throughout the mathematical equations in this section, superscripts such as y^i denote the value of a variable y at the i-th discrete time step or iteration. These should not be

confused with exponentiation, which is explicitly written using parentheses, e.g., $(y^i)^k$.

A. Timestamp Model

In WBSNs, achieving accurate synchronization is particularly challenging due to the distributed nature of the system and the inherent limitations of wireless communication. The aim of this procedure is to express the time series of data collected by different peripherals in a common sampling time (t_S). This enables precise integration of data from multiple sources, allowing the creation of new information.

The two primary sources of timestamps typically available in BLE networks organized in a piconet scheme are: the peripheral timestamp (t_P), generated locally at the time of data acquisition, and the central timestamp (t_C), assigned upon data reception at the central control platform (Fig. 2). The t_P provides high temporal precision with respect to the sensor's local clock; however, as discussed in Section I, it is subject to clock offset and drift, which result in a linear delay with respect to the reference time t_S:

$$t_S = (\alpha + 1)\ t_P + \beta \tag{1}$$

Conversely, t_C serves as an approximation of t_S and enables temporal comparison across nodes. However, it is affected by unpredictable, time-varying communication delays and packet scheduling uncertainties, which are collectively represented by the term ε:

$$t_C = t_S + \varepsilon(t_S) \tag{2}$$

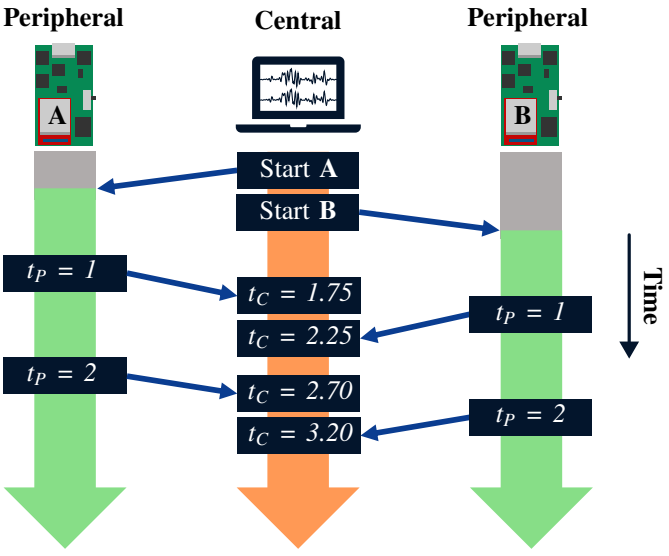


Fig. 2. Illustration of the timestamping process between two peripheral sensor nodes and a central control system. The central system initiates data acquisition by sending start commands to both nodes. Each peripheral device generates local timestamps (t_P), which are transmitted via Bluetooth Low Energy (BLE). Upon reception, the central system assigns corresponding timestamps (t_C), enabling for the estimation of synchronization parameters.

Then, by substituting Equation 1 into Equation 2 and computing the difference between the central and peripheral timestamps, we obtain the parameter δT (Eq. 3), which quantifies the cumulative effect of all delay sources affecting the system (Eq. 4).

$$\delta T = t_C - t_P \tag{3}$$

$$\delta T = \alpha t_P + \beta + \varepsilon(t_S) \tag{4}$$

As shown in Equation 4, δT can be broken down into three main components:

- α is the drift coefficient, which accounts for the frequency difference between the peripheral and central clocks. It can be considered constant over a limited time interval, but may vary due to temperature changes and aging.
- β is the offset coefficient, which accounts for the delayed start of the central and peripheral timers. It contributes to the linear delay along with α.
- ε is the unpredictable component, which is related to all delays caused by the transmission, firmware, and software implementations.

In BLE networks, the selection of connection parameters directly impacts the ε term of the timestamp model. In particular, in the case of packet loss during periodic BLE notification exchanges, data are retransmitted during subsequent CIs, resulting in a reception delay that can be multiple of the CI period. Additionally, even without packet retransmissions, the mismatch between the generation of the peripheral timestamp and the onset of the connection event introduces a fixed uncertainty in the ε that lies within one CI period.

Figure 3 shows the relationship between δT and t_P, taken from an experimental example. In the top plot (Fig. 3A), a linear trend can be observed, consistent with Equation 4. On a smaller scale, the δT signal exhibits a characteristic periodic pattern caused by the mismatch between t_P and the onset of the connection event, which contributes to the unpredictable delay. This behavior is evident in the zoomed-in section (Fig. 3B), where the periodic pattern lies within the range of one CI, which is equal to 30 ms in this specific case. An estimate of the delay variability can be obtained from the inter-packet time ($\delta^2 T$), quantified by computing the differences between adjacent δT samples:

$$\delta^2 T^i = \delta T^i - \delta T^{i-1} \tag{5}$$

$$\delta^2 T^i = \alpha(t_P^i - t_P^{i-1}) + \varepsilon(t_S^i) - \varepsilon(t_S^{i-1}) \tag{6}$$

Assuming the sampling frequency is constant over time, the variability in $\delta^2 T$ primarily reflects the contribution of the unpredictable component ε, since the difference between consecutive peripheral timestamps remains approximately constant (Eq. 6). This variability is quantified by computing the standard deviation of $\delta^2 T$ over a M-samples-long moving window:

$$\sigma_M^i = \sqrt{\frac{1}{M} \sum_{n=i-M}^{i} (\delta^2 T^n - \mu_M^i)^2} \tag{7}$$

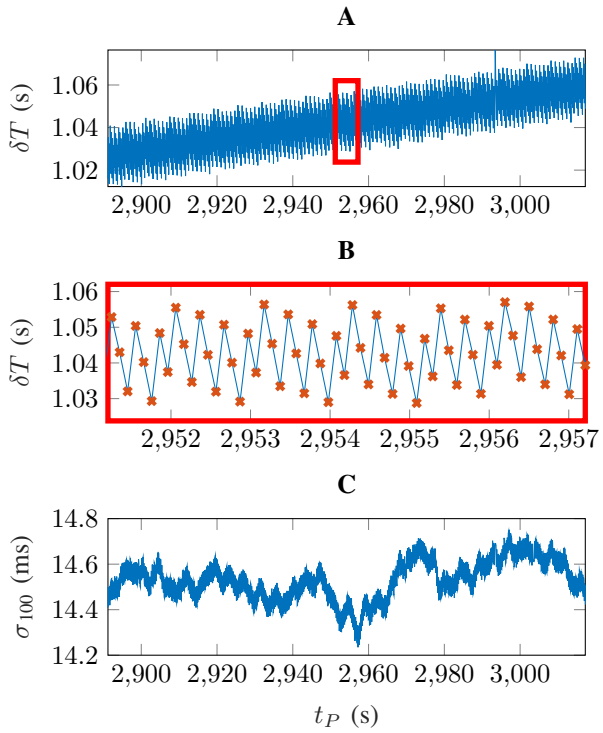


Fig. 3. Experimental evidence of the relationship between δT and t_P under minimum delay conditions. Central and peripheral devices are linked via BLE with a CI of 30 ms. (A) Evolution of δT over peripheral timestamp t_P. The red rectangle marks a region selected for detailed inspection. (B) Magnified view of the selected region in A, showing periodic fluctuations in δT with packet points marked by orange stars. (C) Moving standard deviation σ_{100} of the sequence of δT differences, over a 100-sample window, indicating temporal variations in the timing stability.

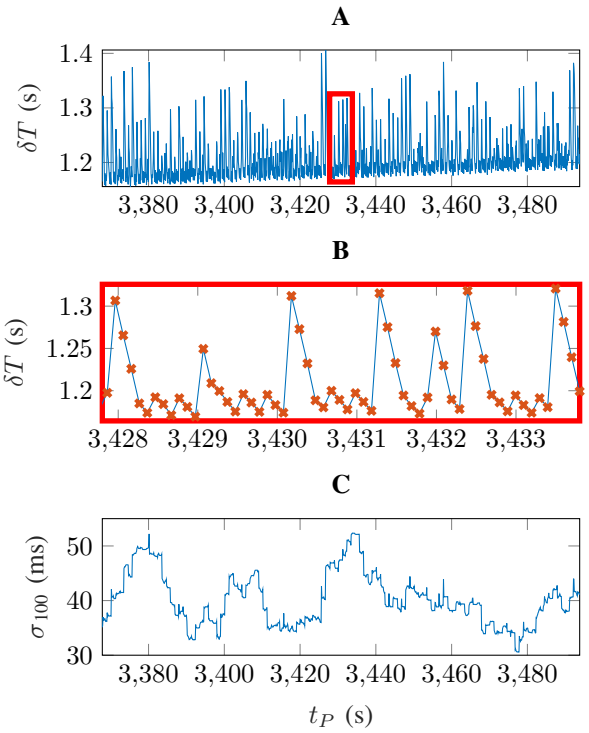


Fig. 4. Relationship between δT and t_P under noisy communication conditions. The central and peripheral devices still communicate with a CI of 30 ms, but increased irregularities are observed. (A) The evolution of δT over time shows a less predictive behavior compared to the previous case (Fig. 3). (B) The magnified view of the highlighted region reveals a higher density of fluctuations in δT, where periodicity is still present but intermittently disrupted by sparse disturbances. These pronounced fluctuations are caused by packet retransmissions and software delays, leading to an increased standard deviation of δT differences, as shown in (C).

where μ is the mean of the last M samples. Figure 3C shows the result of the moving standard deviation of the $\delta^2 T$ signal, computed over a sliding window of 100 samples. Notably, σ_{100} fluctuates around the half of the CI value. This represents a baseline variability that cannot be eliminated and is intrinsically linked to the BLE communication settings. Nevertheless, given its repetitive delay pattern and the approximately constant variance, the δT sequence can be modeled using an Ordinary Least Square (OLS) estimator.

In contrast, packet retransmissions and software delays may disrupt the periodic pattern of the delay, causing spikes that exceed the CI threshold. This scenario is typical of noisy communication conditions, where the assumption of homoscedasticity in the δT distribution is no longer valid. This concept is illustrated in Figure 4, where the regular delay pattern is interrupted by spikes that result in an increased variance of the $\delta^2 T$ signal. This parameter serves as an important metric for estimating the quality of communication between the central and a given peripheral.

B. Synchronization Protocol

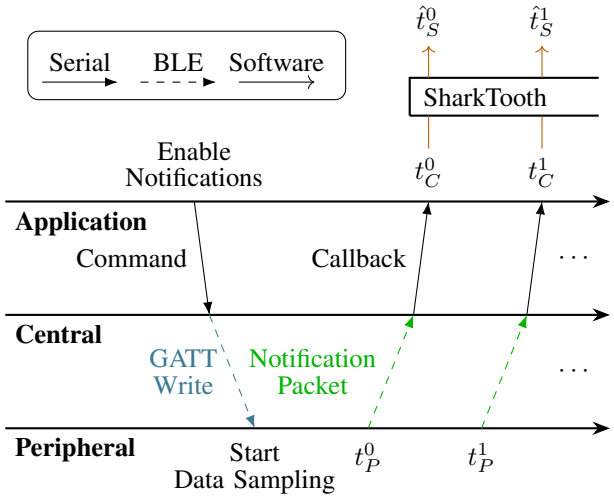
The synchronization protocol is designed to align the clocks of peripheral sensor nodes to a common time reference, t_S, during sensor data acquisition. This is achieved by leveraging BLE notification packets without modifying the standard communication stack or requiring bidirectional timestamping. As illustrated in Figure 5, the synchronization is initiated by the application running on the control platform, which sends a serial command to the central BLE device (e.g., an nRF52840 dongle). The central then performs a GATT write to enable notifications and trigger data acquisition on the peripheral.

Each sensor sample is timestamped immediately after the ADC generates a data-ready interrupt and is stored in a circular buffer. When a predefined number of samples have been collected, the peripheral constructs a notification packet that includes the sensor data, the timestamp of the last acquired sample (i.e., t_P), and a packet ID for sequential identification. As shown in Fig. 5, the maximum BLE payload size is 244 B, which includes two fixed fields: 4 B for the t_P and 1 B for the packet ID respectively. The remaining part of the packet (up to 239 B) can be occupied with sensor data samples.

Upon reception of a notification packet, the central triggers a serial callback in the control application, which immediately timestamps the arrival as t_C. The resulting i-th pair (t_P^i, t_C^i) is collected at runtime and processed by the SharkTooth algorithm (described in detail in Section III-C) to estimate the reference sampling time $\hat{t}_S^i$ associated with each packet. The algorithmis executed independently for each peripheral connection, enabling scalable, real-time, unidirectional synchronization with sub-millisecond accuracy, all while maintaining

full compliance with standard BLE protocols.

In this work, the control application was implemented externally to the central device. This design choice was motivated by the decision to keep the system architecture as high-level and modular as possible, allowing the use of a commercial BLE central running its default firmware. Although the serial communication layer between the central and the host application may introduce additional delays, the objective was to propose and validate a synchronization algorithm capable of minimizing the impact of such intermediate layers.



Notification Packet Payload

Sensor Data Samples	t_P	Packet ID
Up to 239 B	4 B	1 B

Fig. 5. Overview of the synchronization protocol based on the SharkTooth algorithm. The connector legend in the upper left corner indicates the type of operation represented. The notification packet includes sensor data, a peripheral timestamp (t_P), and a packet ID.

C. Synchronization Algorithm

The main issue with previously proposed synchronization methods is their inability to handle the non-deterministic behavior of BLE-based communication. Scaling up the network size implies an increase of the CI value, along with greater variability in δT. This aspect challenges the typical approaches based on the linear regression of synchronization messages flooding, increasing the time transient needed to converge to a stable estimate of the linear trend parameters. Moreover, these methods do not implement any specific strategy to mitigate the errors introduced by packet retransmissions.

In this section, we present the sequence of processing operations that composes *SharkTooth* (Alg. 1). It is designed to be implemented on the control platfrom and executed in parallel for each connected peripheral. Essentially, *SharkTooth* is a robust implementation of a linear regression model that adopts a set of strategies to select the data points less affected by non-deterministic outliers.

First, the newly received BLE packet is processed, extracting the t_P from the packet payload and assigning the t_C timestamp at the control platform level. Then, the δT value is computed (Eq. 3), and the result undergoes a correction step aimed at minimizing the non-linear delays. Eventually, the parameters α and β are estimated through linear regression, and the synchronous timestamp $\hat{t}_S^i$ is computed.

Algorithm 1 SharkTooth Algorithm Iteration

Require: i^{th} Central and peripheral timestamps pair (t_C^i, t_P^i) from node n
Ensure: Synchronous time $\hat{t}_S^i$ for node n
if $i == 0$ **then**
  Initialization (Equations 21-22)
end if
Unpredictable delay correction (Algorithm 2)
if $t_P > 30\,\mathrm{s} + t_{P0}$ **then**
  Update α^i and β^i (Equations 15-17)
else
  Regression matrices reset (Equation 18)
end if
Compute $\hat{t}_S^i$ (Equations 19-20) **return** $\hat{t}_S^i$

The correction of the unpredictable delays represents the key novelty which characterize the synchronization pipeline and enables its implementation at the application level. This procedure is based on the observation that non-linear delays introduce abnormal spikes in the δT term due to packet retransmissions and software processing delays. Building on this consideration, the algorithm (Alg. 2) iteratively selects time points that are less affected by these non-linear delays. As a result, the output sequence of corrected delay values δT_* (Figure 6) represents the linear trend that is tangent to the lower bound of the δT distribution.

Algorithm 2 Unpredictable delay correction

Require: i^{th} Central and peripheral timestamps pair (t_C^i, t_P^i) from node n
Ensure: Corrected peripheral-to-central delay δT_*^i for node n
δT^i and $\delta^2 T^i$ computation (Equations 3 and 5)
σ_M^i computation (Equation 7)
λ_0^i computation (Equation 10)
if $\delta T^i < \delta T_0$ **then**
  Pivot point identification (Equation 11)
else
  if $\delta T^i < \delta T_{thr}^i$ **then**
    δT_*^i update and δT_{thr} reset (Equation 12)
  else
    δT_{thr} slope increment (Equation 14)
  end if
end if
Compute δT_*^i (Equations 8) **return** δT_*^i

The output delay δT_* is computed using a linear equation (Eq. 8), where the slope (γ_0) is iteratively updated. The terms δT_0 and t_{P0} denote the coordinates corresponding to the minimum of the δT time sequence, also referred to as the pivot point.

$$\delta T_*^i = \delta T_0 + \gamma_0 \ (t_P^i - t_{P0}) \tag{8}$$

The update of γ_0 is determined by comparing δT with the probe sequence δT_{thr} (Eq. 9), which acts as a dynamic threshold for selecting the input points eligible for estimating the linear trend.

$$\delta T^i_{thr} = \delta T_1 + \gamma_1 \ (t^i_P - t_{P1}) \tag{9}$$

This probe follows a linear variation over time, controlled by the research coefficient γ_1, which is incremented whenever a non-eligible point is detected. The coordinates t_{P1} and δT_1 denote the peripheral time and the corresponding output trend value resulting from the last eligible point detection. This mechanism ensures the convergence of the algorithm and its adaptability to different trends.

The i^{th} process iteration consists of the following steps:

1) Computation of the peripheral-to-central delay δT^i (Eq. 3) and the inter-packet delay $\delta^2 T^i$ (Eq. 5).

2) Calculation of the standard deviation σ^i_M of the last M $\delta^2 T$ samples (Eq. 7)

3) The parameter λ^i_0 is calculated using the minimum value between half the CI divided by σ^i_M and 1.

$$\lambda^i_0 = min\left(\frac{\text{CI}}{2\sigma^i_M}, 1\right) \tag{10}$$

It quantifies the variability level in the last section of the δT signal. Under normal operating conditions, its value is typically close to 1 and decreases toward 0 in the presence of additional delays.

4) Seeking of the delay point which produce the minimum δT, identified as the pivot point (Eq. 11).

$$\delta T^i < \delta T_0 \rightarrow \begin{cases} \delta T_0 = \delta T^i \\ \gamma_0 = 0 \\ t_{P0} = t^i_P \\ \gamma_1 \leftarrow \infty \end{cases} \tag{11}$$

5) In cases where $\delta T^i \geq \delta T_0$, δT^i is compared with the probe δT^i_{thr} (Eq. 9). If this condition is met ($\delta T^i < \delta T^i_{thr}$), the point is selected for updating the parameters γ_0 and δT_{thr}:

$$\delta T^i < \delta T^i_{thr} \rightarrow \begin{cases} \gamma_0 = (1-\lambda^i)\gamma_0 + \lambda^i \frac{\delta T^i - \delta T_0}{t^i_P - t_{P0}} \\ \gamma_1 = \gamma_0 \\ \delta T_1 = \delta T^i_* \\ t_{P1} = t^i_P \end{cases} \tag{12}$$

This approach ensures that the evolution of γ_0 is determined by a weighted average between its previous value and the slope of the line connecting the pivot point (t_{P0}, δT_0) to the newly selected point (t^i_P, δT^i). This allows for gradual adaptation of the delay estimation model.

The weighting factor λ is calculated to make the update of γ_0 smoother and less sensitive to unpredictable delays. It is computed using the following condition:

$$\lambda^i = \begin{cases} 0.5\lambda^i_0 \ , \ \delta T^i \geq \delta T^{i-1}_* \\ 0.5 \ , \ \delta T^i < \delta T^{i-1}_* \end{cases} \tag{13}$$

This conditional definition ensures that λ^i adapts dynamically based on the delay behavior. Specifically, when $\delta T^i \geq \delta T^{i-1}_*$, the weight is proportional to the variability term λ^i_0, which reflects the recent fluctuation of delay values. In the opposite case ($\delta T^i < \delta T^{i-1}_*$), λ^i is automatically set to its maximum value of 0.5. This condition promotes the rapid convergence of the δT_* sequence toward the lower boundary of the δT distribution, as those points exhibit a stronger alignment with the linear trend.
The maximum weight value of 0.5 was selected based on empirical observations. This value provides an optimal balance between fast convergence of the model and maintaining low output variability.

If $\delta T \geq \delta T_{thr}$, the threshold variation rate γ_1 is incremented, as shown in Equation 14). This adjustment increases the likelihood of a threshold crossing event in the subsequent iterations, ensuring that the model remains responsive to newly eligible points.

$$\delta T^i \geq \delta T^i_{thr} \rightarrow \gamma_1 = \gamma_1 + 10^{-6} \tag{14}$$

The constant factor (i.e., 10^{-6}) used to increment γ_1 is a design parameter that directly controls the sensitivity of the dynamic threshold δT_{thr} to eligible points. This value was empirically determined based on the BLE packet transmission frequency of 10 Hz, ensuring that the threshold can adapt at a rate suitable for the network's communication dynamics. Furthermore, the drift of the peripheral's oscillator must be considered, as this drift approximates the actual value of γ_0. To maintain coherence with this drift, the increment of γ_1 is set to the same order of magnitude.

6) At the end of each iteration, the output delay δT^i_* is computed through Equation 8.

The estimation of the synchronization coefficients (α, β) is performed through linear regression of output delay δT_* sequence with respect to the t_P values. Given that this algorithm is designed to work in real-time, the regression operation was performed through the iterative computation of the OLS estimator. As shown in Equation 15, the estimator can be efficiently computed through iterative matrix multiplication.

$$\begin{bmatrix} \alpha^i \\ \beta^i \end{bmatrix} = \frac{1}{det(\mathbf{A^i})} \ \mathbf{A^i b^i}, \ \text{if} \ \ det(\mathbf{A^i}) \neq 0 \tag{15}$$

Before the estimation step, the regression matrices $\mathbf{A^i}$ and $\mathbf{b^i}$

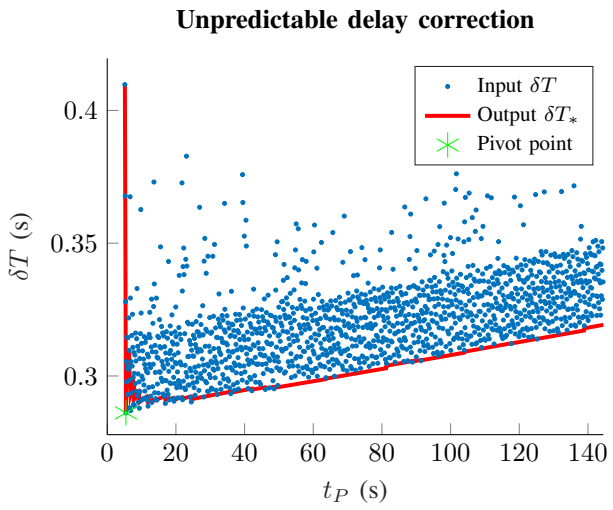


Fig. 6. Correction of unpredictable delay over time. The plot shows the raw input delay values δT (blue dots) as a function of peripheral time (t_P), and the corrected output delay δT_* (red line) obtained through the proposed correction algorithm. The green star denotes the pivot point detected by the algorithm.

are updated using last t_P^i and δT_*^i values.

$$\mathbf{A^i} = \begin{cases} \begin{bmatrix} 1 & -t_P^i \\ -t_P^i & (t_P^i)^2 \end{bmatrix}, & \text{if } \mathbf{A_{0,0}^{i-1}} = 0 \\ \mathbf{A^{i-1}} + \begin{bmatrix} 1 & -t_P^i \\ -t_P^i & (t_P^i)^2 \end{bmatrix}, & \text{if } \mathbf{A_{0,0}^{i-1}} \neq 0 \end{cases} \tag{16}$$

$$\mathbf{b^i} = \begin{cases} \begin{bmatrix} t_P^i \ \delta T_*^i \\ \delta T_*^i \end{bmatrix}, & \text{if } \mathbf{A_{0,0}^{i-1}} = 0 \\ \mathbf{b^{i-1}} + \begin{bmatrix} t_P^i \ \delta T_*^i \\ \delta T_*^i \end{bmatrix}, & \text{if } \mathbf{A_{0,0}^{i-1}} \geq 0 \end{cases} \tag{17}$$

Updating these matrices incrementally allows the parameter estimates to be continuously refined in real time, with constant memory usage and minimal computational cost. As shown in Equations 16 and 17, the matrix $\mathbf{A^i}$ and vector $\mathbf{b^i}$ are updated at each step using rank-one additions involving only the most recent timestamp-delay pair $(t_P^i, \delta T_*^i)$. The model parameters are computed using a closed-form expression (Eq. 15), which requires a matrix-vector product and a determinant evaluation. Both operations have a computational complexity of $\mathcal{O}(1)$ for a two-parameter linear model. This lightweight structure ensures that the algorithm is suitable for real-time execution on embedded platforms, with predictable and minimal computational overhead that scales linearly with the number of peripheral connections.

The stability of t_P^i and δT_*^i values used to update the regression matrices is fundamental for ensuring consistent estimation of the linear synchronization coefficients. Indeed, following the identification of the pivot point, the δT_*^i requires a few seconds to stabilize, as illustrated in Fig. 6. Consequently, each time a new pivot is detected, the elements of $\mathbf{A^i}$ and $\mathbf{b^i}$ are set to zero.

$$t_P^i \leq t_{P0} + 30\,\text{s} \rightarrow \begin{cases} \alpha^i = 0 \\ \beta^i = 0 \\ \mathbf{A^i} \leftarrow \mathbf{0}_{2\times 2} \\ \mathbf{b^i} \leftarrow \mathbf{0}_{2\times 1} \end{cases} \tag{18}$$

A predefined settling period is then observed before performing the first update, allowing the δT_*^i transient response to settle. In our implementation, we set the settling period to 30 s (Alg. 1).

As a final step, the time delay estimate $\delta\hat{T}^i$ is calculated using the linear correction coefficients.

$$\delta\hat{T}^i = \alpha^i t_P^i + \beta^i \tag{19}$$

The delay estimate represents the correction term that can be added to the peripheral timestamp, producing a synchronous timestamp $\hat{t}_S^i$, which approximates the true sampling time t_S.

$$\hat{t}_S^i = t_P^i + \delta\hat{T}^i \tag{20}$$

The synchronization algorithm can be efficiently executed in parallel for different peripherals, making it suitable for both real-time and post-processing applications. The main loop, described above, is preceded by an initialization step that sets the initial configuration of variables required to correct unpredictable delays.

$$\delta T_* \text{ Initialization} \rightarrow \begin{cases} \delta T_0 \leftarrow \infty \\ t_{P0} = 0 \\ \gamma_0 = 0 \end{cases} \tag{21}$$

$$\delta T_{thr} \text{ Initialization} \rightarrow \begin{cases} \delta T_1 \leftarrow \infty \\ t_{P1} = 0 \\ \gamma_1 \leftarrow \infty \end{cases} \tag{22}$$

IV. Validation

The proposed model aims to provide a robust and reliable solution that is resistant to non-deterministic delays. The resulting synchronization accuracy among sensors is within approximately one millisecond. To validate the robustness of the proposed method and its scalability across a broad range of application scenarios, we tested the synchronization algorithm in various use-case configurations, modifying both the number of devices connected to the network and the size of the exchanged packets. These parameters directly impact the probability of packet loss and retransmissions, which, in turn, affects the overall network synchronization.

This section is divided into two parts: (A) Acquisition set-up; (B) Synchrony evaluation. In Part (A), the experimental setup used for validation is described in detail. In Part (B), the synchrony assessment among different units and the validation metrics are presented.

A. Acquisition Set Up

The validation setup was organized following a modular approach to test the synchronization algorithm under different operating conditions, automatically spanning various configurations (Table I). Although time delays can be significantly minimized through appropriate system design choices, such

as low-level firmware and software implementations, the experimental setup was built using off-the-shelf components and a high-level programming language such as Python. As mentioned in Section II-C, this technical choice was made to ensure the reproducibility of results while accounting for as many sources of uncertainty as possible.

A set of twelve acquisition boards, described in Section II, was used for testing. Although these boards were designed for battery operation, a stable common +5 V power source, provided by an Arduino UNO R3 [40], was used in this validation. This setup enabled automated trial repetitions and ensured that the acquisition boards could be properly restarted in case of connection failures.

We evaluated the relative synchronization among different units by collecting a reference sinusoidal voltage signal using their sEMG acquisition circuit (Fig. 7). Since the collected signal results from the amplification of the differential mode between two sensing electrodes, we provided two 5 Hz voltage sinusoids with opposite phase and equal amplitude as input to the amplification stage. The RIGOL MSO5104 oscilloscope [41] was used to generate the reference sinusoids. The differential signal was then filtered and amplified, producing a 1 V_{PP} output signal sampled at a frequency of 50 Hz.

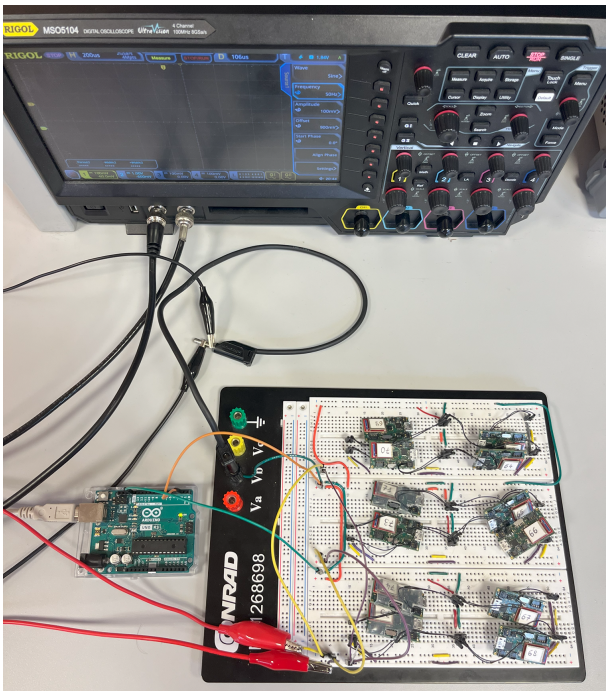


Fig. 7. The validation testbed includes twelve sensor nodes mounted on a breadboard and powered by an Arduino microcontroller. A digital oscilloscope simultaneously provides two sinusoidal reference signals with opposite phase to all acquisition nodes for synchronization validation.

During each acquisition session, a variable number of devices establish a BLE connection with the nRF52840 Dongles, which are connected to the PC (Intel Core i7 vPro 8^{th} Generation) via USB and positioned approximately 3 m from the acquisition boards (Fig. 8). Data collection is managed by a software application written in Python, running on the PC. Once a new BLE packet is received by a dongle, the data payload is extracted and sent to the software application via USB. The arrival of the serial message triggers a callback in the control platform, which assigns the central timestamp t_C and queues the data for processing by background threads. Serial communication between the control platform and the dongles is handled using the Blatann library.

To transmit the maximum BLE packet length of 244 B, the connection event length must be set to 7.5 ms. Consequently, connecting 12 devices to a single receiver would require a CI equal to 90 ms. However, this value is incompatible with the real-time transmission of many biomedical signals, such as EMG and inertial sensor data.

In order to allow successful retransmission of lost BLE packets, the packet transmission period must be at least twice the connection interval. Based on these considerations, selecting a CI of 90 ms would imply a packet sending periodicity of at least 180 ms, limiting the maximum data throughput per unit to less than 1.355 kB/s. In contrast, a typical EMG sensor (1 kHz, 1 channel, 16 b/sample) and IMU sensor (208 Hz, 6 channels, 16 b/sample) each exceed 2 kB/s. Therefore, the acquisition units were organized into three groups of four, each communicating with a dedicated dongle. This setup enabled the use of a fixed CI value across all the validation sessions. Indeed, splitting the connections across three dongles reduces the minimum required connection interval to 30 ms and increases the feasible data throughput per unit to 4.066 kB/s.

However, using three separate dongles does not reduce the complexity of managing twelve peripherals to that of a single dongle handling four connections. The control platform must still manage three concurrent data streams, and the coexistence of three independent networks operating in close proximity increases the likelihood of packet collisions.

TABLE I
SUMMARY OF THE KEY PARAMETERS USED IN THE SYNCHRONIZATION PERFORMANCE EVALUATION

Parameter	**Value**
# Tested configurations	18
# Peripherals	2-4-6-8-10-12
Packet length	17-127-244 B
Acquisition duration per repetition	3600 s
# Repetitions per combination	20
Connection Interval	30 ms
Reference Signal	$0.9 + 0.5\sin(10\pi t)$ V
Sampling rate	50 Hz

The BLE notification time period was set to 100 ms, a typical value suitable for transmitting both EMG and IMU data. As a result, the voltage samples collected during the reference signal acquisition were quantized at 2 B and packed into a 17 B payload, along with a 4 B timestamp for the last sample (t_P), 2 B indicating the time difference between the first and last sample, and 1 B for the packet number.

In addition to the basic payload, a variable-length overhead can be appended to extend the total packet size. To evaluate different transmission conditions, we tested overhead values of 0 B, 110 B, and 227 B, corresponding to three representative use cases. The effect of the number of connected devices on

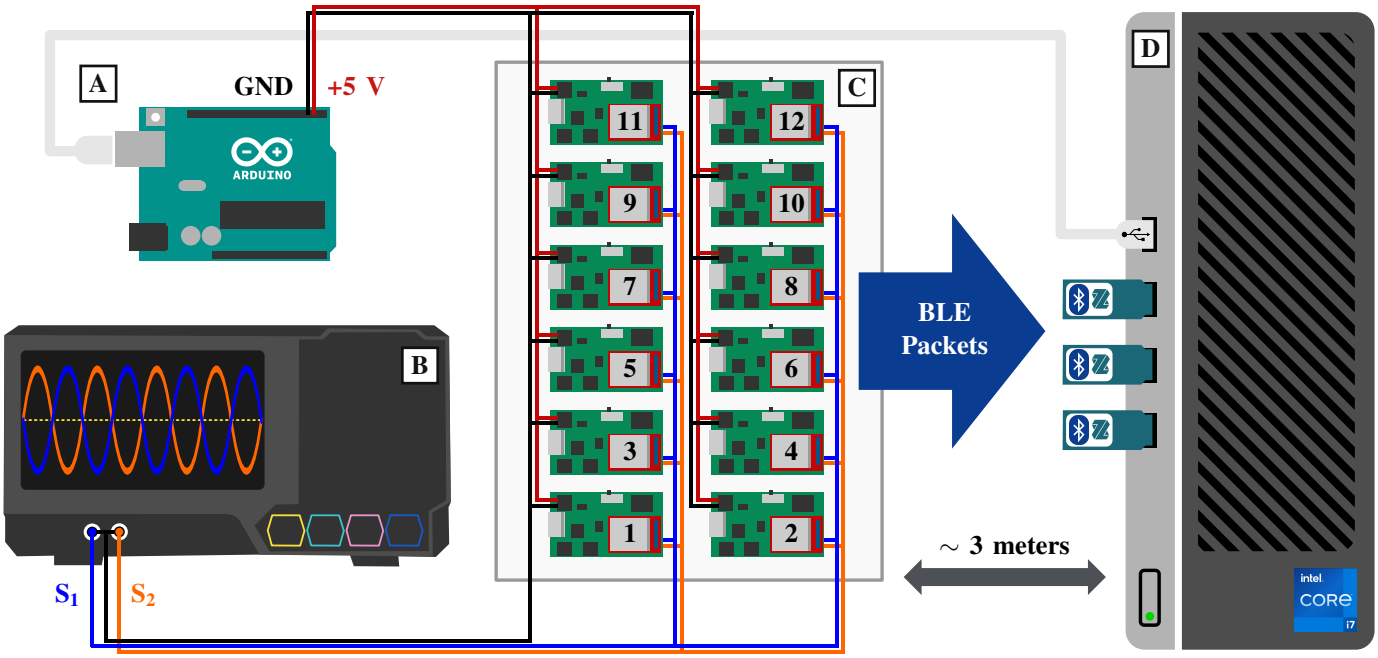


Fig. 8. Schematic representation of the experimental setup for validating the proposed synchronization algorithm. Two reference signals with opposite phase (S_1 and S_2), generated by the oscilloscope (B), serve as inputs to the analog front-end of 12 Apollux sensor nodes (C). Each node samples the resulting differential signal at 50 Hz and transmits the data via BLE packets to three USB dongles connected to a workstation (D) over a distance of approximately 3 m. An Arduino UNO board (A) provides a stable +5 V power supply and ground reference (GND).

synchronization performance was evaluated testing 2, 4, 6, 8, 10, 12 devices connected in parallel. Each of the eighteen configurations, obtained by varying the packet length and the number of connections, was tested through twenty one-hour-long acquisitions to assess the convergence and long-term stability of the synchronization algorithm.

B. Synchronization Metrics Evaluation

The aim of the proposed synchronization algorithm, described in Section III, is to ensure seamless time alignment between signals collected by different sensors, thereby enabling consistent multi-sensor data fusion. As illustrated in Figure 9, the reference signal copies acquired by ten peripheral devices appear misaligned when plotted against the peripheral time t_P. In contrast, when represented in the estimated synchronous time domain $\hat{t}_S$, the signals exhibit clear alignment. This visual comparison highlights the effectiveness of the synchronization process.

To quantify synchronization performance, we compute the difference between the estimated and actual inter-device delays for each sensor pair. The estimated delay is derived from the difference in synchronous timestamps $\hat{t}_S$ associated to the same t_P sample. The ground-truth delay is obtained by performing cross-correlation between signal pairs in the unsynchronized peripheral time domain t_P. The full procedure for estimating both quantities is described in detail below.

After each acquisition session, the signal samples (s_n^i) from the n-th sensor are associated with two sequences of timestamps: the peripheral timestamps t_{Pn}^i and the estimated

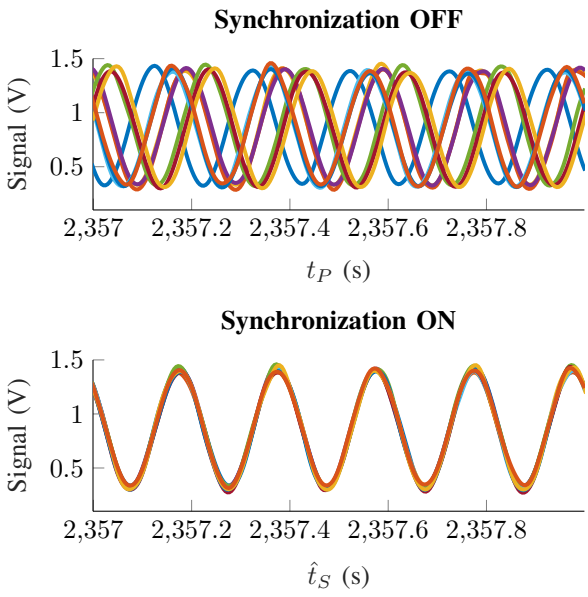


Fig. 9. Example of the effect of the synchronization algorithm. In the top plot, data from ten different peripherals collecting the same reference sinusoidal signal exhibit noticeable phase misalignment. In the bottom plot, the same signals are shown after applying synchronization.

timestamps $\hat{t}_{Sn}^i$. Before evaluating synchronization performance, both timestamp sequences are resampled to produce a new set of equally spaced samples t_P^j, common across all sensors, along with their corresponding resampled synchronous timestamps $\hat{t}_{Sn}^j$. This resampling step preserves the original sampling frequency of 50 Hz, while enabling consistent delay estimation across sensors.

The estimated delay ($\hat{\tau}^j_{nm}$) between two sensors n and m at each sample j is then computed as:

$$\hat{\tau}^j_{nm} = \hat{t}^j_{Sn} - \hat{t}^j_{Sm} = \delta\hat{T}^j_n - \delta\hat{T}^j_m \tag{23}$$

then, $\hat{\tau}^j_{nm}$ is averaged over one-second epochs:

$$\hat{\tau}^k_{nm} = \frac{1}{f_s} \sum_{j=kf_s}^{(k+1)f_s-1} \hat{\tau}^j_{nm} \tag{24}$$

where k indexes epoch and f_s represents the sensor sampling rate (i.e., 50 Hz).

The ground-truth delay (τ^k_{nm}) between two sensors n and m at each one-second epoch k is computed as:

$$\tau^k_{nm} = \tau^k_n - \tau^k_m \tag{25}$$

where τ^k_n represents the delay of the signal recorded by sensor n and a reference 5 Hz sinusoidal waveform. The procedure to compute the τ^k_n sequence for each peripheral involves the following steps:

- Up-sampling the voltage signal s^i_n from 50 Hz to 10 kHz, generating the signal s^u_n.
- Generating a software-based 5 Hz sinusoidal function ($\tilde{s}^u$) as a common reference for all peripherals.
- Segmenting peripheral and reference signals into contiguous one-second epochs.
- Computing the Cross-Correlation (CC) between peripheral and reference signals for each k-th epoch.
- Defining τ^k_n as the time offset that maximizes the CC between s^u_n and $\tilde{s}^u$

Cross-correlation is widely used to estimate delays between signals [6], [25]. However, its resolution and range are constrained by signal characteristics: the sampling frequency determines the resolution, while sinusoidal signals limit the maximum detectable delay to half of the signal's period. Up-sampling enhances sub-millisecond resolution but increases processing time. To balance accuracy and computational efficiency, a resampling frequency of 10 kHz was selected.

Preliminary tests indicated that synchronization estimation errors typically fall within a range of several hundred microseconds. Therefore, the reference signal frequency was selected so that its period (200 ms) exceeds typical synchronization errors. This choice ensures reliable delay estimation and avoids ambiguity. Additionally, testing variable packet lengths while preserving information content required a reduction in the number of collected samples. To maintain reference signal integrity and reproducibility after resampling, the sinusoidal signal period was set accordingly.

Since the goal of synchronization is to minimize the relative delay between peripherals, the Relative Synchronization Error (RSE) was computed for each pair of peripherals n and m as follow:

$$\mathrm{RSE}^k_{nm} = \hat{\tau}^k_{nm} - \tau^k_{nm} \tag{26}$$

As shown in Figure 10, the RSE sequences computed for all pairs of acquisition nodes typically exhibit an initial transient phase, which gradually dissipates as the acquisition progresses. Additionally, different devices may display distinct

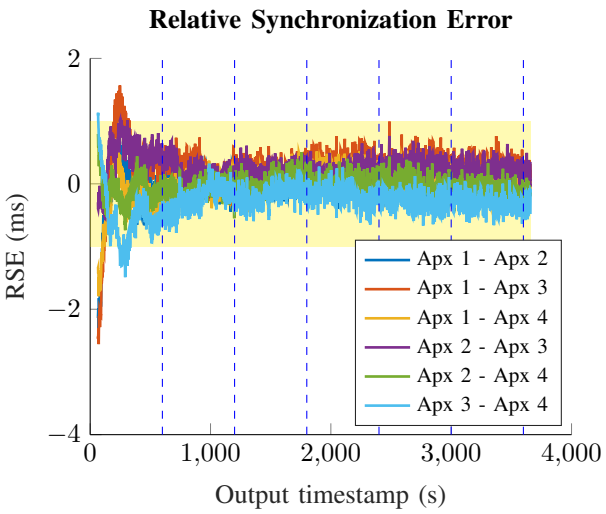


Fig. 10. Example of Relative Synchronization Error (RSE) over time for different pairs of four acquisition nodes. The plot shows the delay (in milliseconds) between node pairs as a function of the output timestamp. The acquisition lasts more than 3600 s and is divided into consecutive 600 s sections, marked by the dashed blue lines. The shaded yellow band highlights the ±1 ms target accuracy region.

behaviors within the same acquisition. To compare repetitions with varying validation parameter configurations, we selected a set of metrics to synthetically characterize the synchronization performance of the *SharkTooth* algorithm in each acquisition.

Specifically, we first divided the entire acquisition period into six consecutive segments of 600 seconds each, in order to compute the validation metrics within each segment. In each section, we identified the worst-case RSE sequence (RSE^k_{max}), defined as the one associated with the sensor pair exhibiting the highest synchronization error. This was determined using the 95th percentile of the absolute RSE values, as expressed in the following equation:

$$\begin{aligned} \mathrm{RSE}^k_{max} &= \mathrm{RSE}^k_{xy} \\ \text{where} \quad (x,y) &= \arg\max_{(n,m)} \left\{ \mathrm{P}_{95} \left(\left| \mathrm{RSE}^k_{nm} \right| \right) \right\} \end{aligned} \tag{27}$$

The resulting RSE^k_{max} sequence is then analyzed to extract three main validation metrics: the absolute average value, the standard deviation, and the 95$^{\text{th}}$ percentile. This process yields three metrics per time segment, computed for each of the twenty acquisition sessions across all eighteen system configurations.

V. Results and Discussion

The validation protocol, described in Section IV, produced a set of experimental results summarized in Table II. The table reports the median and InterQuartile Range (IQR) of the absolute average and standard deviation of the RSE across different configurations and acquisition sections. To enhance visualization, these validation metrics are color-coded, with increasing cell color intensity corresponding to higher median values. The median and IQR are computed for each metric based on 20 repetitions.

TABLE II
MEDIAN AND INTERQUARTILE RANGE (IQR) OF THE RSE METRICS ACROSS DIFFERENT CONFIGURATIONS AND ACQUISITION SECTIONS

RSE metrics (ms)	Configuration ID	Packet length (Byte)	#Peripherals	Acquisition section 1	2	3	4	5	6
				median (**interquartile range**) ms					
Absolute Average	A2	17	2	0.30 (0.08)	0.22 (0.03)	0.22 (0.03)	0.21 (0.03)	0.21 (0.02)	0.21 (0.02)
	A4	17	4	0.39 (0.21)	0.30 (0.07)	0.29 (0.05)	0.29 (0.07)	0.31 (0.09)	0.28 (0.07)
	A6	17	6	0.45 (0.13)	0.40 (0.19)	0.35 (0.13)	0.33 (0.09)	0.33 (0.12)	0.34 (0.10)
	A8	17	8	0.52 (0.08)	0.44 (0.14)	0.42 (0.12)	0.38 (0.14)	0.42 (0.12)	0.40 (0.18)
	A10	17	10	0.61 (0.24)	0.52 (0.24)	0.48 (0.16)	0.42 (0.17)	0.40 (0.19)	0.39 (0.19)
	A12	17	12	0.57 (0.22)	0.49 (0.20)	0.45 (0.12)	0.46 (0.15)	0.44 (0.17)	0.41 (0.15)
	B2	127	2	0.39 (0.12)	0.26 (0.07)	0.24 (0.05)	0.22 (0.05)	0.23 (0.05)	0.22 (0.02)
	B4	127	4	0.55 (0.21)	0.50 (0.23)	0.37 (0.24)	0.39 (0.11)	0.36 (0.09)	0.32 (0.10)
	B6	127	6	0.56 (0.23)	0.44 (0.17)	0.40 (0.18)	0.41 (0.15)	0.36 (0.15)	0.36 (0.12)
	B8	127	8	0.58 (0.21)	0.47 (0.14)	0.42 (0.12)	0.43 (0.14)	0.42 (0.09)	0.41 (0.09)
	B10	127	10	0.91 (0.55)	0.51 (0.28)	0.49 (0.19)	0.51 (0.31)	0.47 (0.24)	0.44 (0.12)
	B12	127	12	0.82 (0.43)	0.56 (0.27)	0.53 (0.19)	0.50 (0.15)	0.45 (0.22)	0.50 (0.21)
	C2	244	2	0.43 (0.12)	0.32 (0.11)	0.27 (0.09)	0.26 (0.03)	0.24 (0.03)	0.25 (0.04)
	C4	244	4	0.68 (0.28)	0.46 (0.11)	0.41 (0.07)	0.39 (0.08)	0.35 (0.06)	0.35 (0.08)
	C6	244	6	0.66 (0.29)	0.58 (0.25)	0.46 (0.16)	0.48 (0.24)	0.47 (0.15)	0.46 (0.11)
	C8	244	8	0.87 (0.48)	0.70 (0.33)	0.58 (0.21)	0.46 (0.26)	0.47 (0.27)	0.49 (0.15)
	C10	244	10	0.98 (0.44)	0.72 (0.23)	0.64 (0.23)	0.61 (0.22)	0.57 (0.21)	0.58 (0.30)
	C12	244	12	1.27 (0.46)	1.16 (0.79)	0.68 (0.47)	0.72 (0.35)	0.64 (0.29)	0.62 (0.46)
Standard Deviation	A2	17	2	0.33 (0.11)	0.26 (0.01)	0.25 (0.01)	0.25 (0.01)	0.25 (0.01)	0.25 (0.01)
	A4	17	4	0.43 (0.13)	0.28 (0.02)	0.25 (0.01)	0.26 (0.02)	0.26 (0.02)	0.25 (0.02)
	A6	17	6	0.50 (0.15)	0.27 (0.02)	0.25 (0.01)	0.25 (0.01)	0.25 (0.01)	0.25 (0.02)
	A8	17	8	0.55 (0.22)	0.26 (0.03)	0.26 (0.01)	0.24 (0.02)	0.24 (0.02)	0.24 (0.02)
	A10	17	10	0.73 (0.15)	0.27 (0.02)	0.26 (0.03)	0.26 (0.02)	0.25 (0.02)	0.25 (0.02)
	A12	17	12	0.59 (0.19)	0.27 (0.04)	0.26 (0.03)	0.25 (0.01)	0.25 (0.02)	0.25 (0.02)
	B2	127	2	0.45 (0.17)	0.27 (0.02)	0.27 (0.01)	0.26 (0.01)	0.26 (0.01)	0.26 (0.01)
	B4	127	4	0.59 (0.24)	0.32 (0.04)	0.30 (0.03)	0.29 (0.02)	0.28 (0.02)	0.29 (0.02)
	B6	127	6	0.66 (0.31)	0.32 (0.05)	0.29 (0.03)	0.29 (0.02)	0.29 (0.02)	0.28 (0.02)
	B8	127	8	0.70 (0.27)	0.33 (0.06)	0.30 (0.01)	0.30 (0.02)	0.29 (0.02)	0.30 (0.01)
	B10	127	10	1.06 (0.68)	0.33 (0.05)	0.30 (0.03)	0.30 (0.02)	0.30 (0.02)	0.29 (0.01)
	B12	127	12	0.94 (0.37)	0.37 (0.14)	0.32 (0.03)	0.30 (0.03)	0.30 (0.02)	0.30 (0.02)
	C2	244	2	0.55 (0.20)	0.31 (0.02)	0.30 (0.02)	0.30 (0.02)	0.30 (0.01)	0.29 (0.01)
	C4	244	4	0.69 (0.22)	0.37 (0.06)	0.38 (0.03)	0.37 (0.03)	0.37 (0.03)	0.37 (0.02)
	C6	244	6	0.80 (0.45)	0.40 (0.07)	0.38 (0.03)	0.38 (0.04)	0.38 (0.04)	0.37 (0.03)
	C8	244	8	1.04 (0.43)	0.43 (0.11)	0.39 (0.04)	0.39 (0.04)	0.40 (0.05)	0.38 (0.04)
	C10	244	10	0.96 (0.51)	0.43 (0.08)	0.43 (0.11)	0.39 (0.07)	0.39 (0.04)	0.39 (0.06)
	C12	244	12	1.57 (0.82)	0.54 (0.37)	0.43 (0.06)	0.42 (0.07)	0.41 (0.08)	0.41 (0.07)

Interesting observations emerge from the analysis of the absolute average: configurations with a higher number of connected devices and increased packet length exhibit higher RSE values. Specifically, configuration C12 (244 bytes, 12 peripherals) shows the highest median absolute average values across all acquisition sections, reaching 1.27 ms in section 1. In contrast, configurations with fewer devices, such as A2 (17 bytes, 2 peripherals), consistently show the lowest values, with medians remaining below 0.5 ms across all sections. This suggests that network congestion and increased data load significantly impact synchronization performance, leading to higher delay estimation errors.

Configurations with a packet length of 17 B (A2-12) maintain a median absolute average value consistently below 1 ms across all acquisition sections. Similarly, configurations with no more than 4 connected peripherals (A2, A4, B2, B4, C2, C4) exhibit lower delays, reinforcing that smaller networks ensure more stable synchronization. Examining the evolution of the absolute average overtime (Fig. 11), we observe a general decreasing trend. In high-complexity configurations such as C12, the median drops from 1.27 ms in section 1 to 0.62 ms in section 6, indicating a stabilization effect over time. Simpler configurations such as A2, however, maintain relatively stable values across all sections, reinforcing the robustness of less complex setups.

The lower block of Table II reports the standard deviation of the RSE across different configurations and acquisition sections. Higher standard deviation values generally correspond to configurations with a greater number of devices, indicating increased variability in synchronization accuracy. Configuration C12 again exhibits the highest median variability, reaching 1.57 ms in section 1, further reinforcing the idea that as network complexity grows, so does the variability of RSE. Conversely, simpler configurations, such as A2 and A4, exhibit lower variability, suggesting more stable synchronization performance when fewer devices are involved. Notably, the standard deviation drops substantially from section 2 onward in all configurations, indicating that the system stabilizes over time. This effect is particularly evident in configurations with higher peripheral counts, where the standard deviation decreases by more than 50 % between sections 1 and 6.

From the analysis of the IQR values, it is possible to evalu-

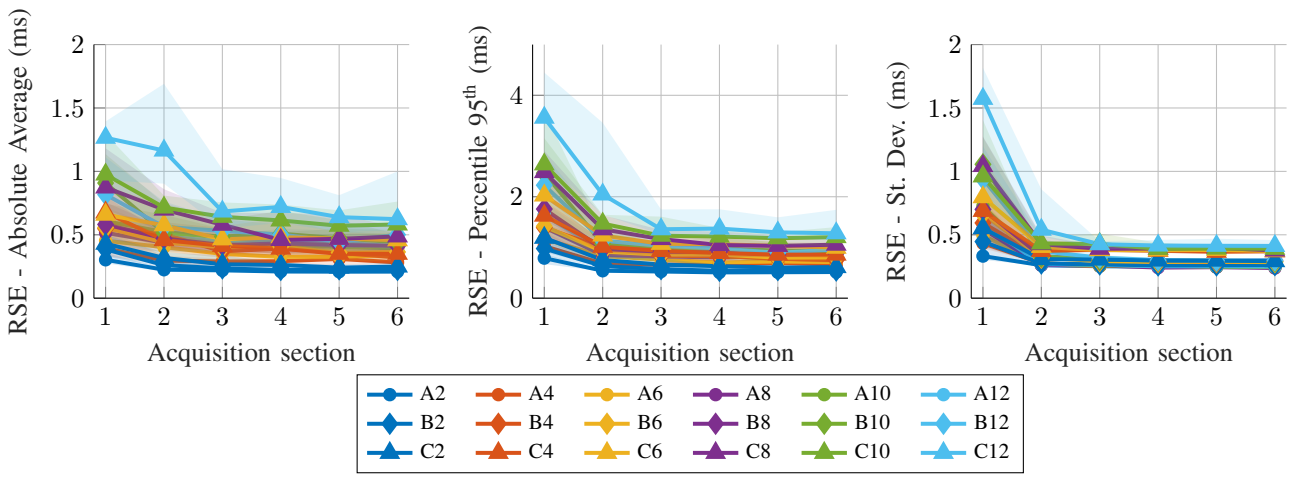


Fig. 11. Relative Synchronization Error (RSE) trends across six time sections for various system configurations. The subplots report three validation metrics: the absolute average, the absolute 95^{th} percentile, and the standard deviation of the RSE. Each line represents the median value of a given metric across experimental repetitions for a specific configuration. Configurations are color-coded by the number of connected devices (2–12) and use different marker shapes according to the packet length category (A, B, or C) used in the trials. Shaded areas represent the interquartile range across repetitions.

ate the consistency of each metric across different acquisitions. In all presented metrics, the IQR follows the same trend as the median values, peaking in section 1 and decreasing across subsequent sections. This behavior demonstrates the overall stability of the presented algorithm, which converges toward intra- and inter-acquisition stability.

A. Correlation Between Synchronization Accuracy and Variance

Figure 12 explores the relationship between the 95th percentile of RSE and the variance of inter-packet time (denoted as σ^2_{100}) across different acquisition sections. Each scatter plot corresponds to a distinct section, with data points color-coded according to connection configurations. The absolute 95th percentile values follow a pattern similar to the absolute average RSE, providing further insight into the worst-case synchronization performance.

Configuration C12 records the highest median absolute 95th percentile value in section 1, peaking at 3.56 ms, before gradually decreasing to 1.28 ms in section 6. This behavior suggests that the system adapts to sustained operation, mitigating extreme synchronization errors over time. Configurations with smaller device counts, such as fewer than 6 devices, maintain significantly lower absolute 95th percentile values across all sections, consistently staying below 1.7 ms. This confirms that simpler network setups provide more reliable worst-case synchronization performance. A similar behavior is observed in configurations with a packet length of 17 B (A2-12), reinforcing the notion that lower data loads result in improved synchronization stability.

Looking at the trend of the 95th percentile across all sections, we observe as variance increases, RSE also increases, indicated by the positive slope of the fitted black regression lines. This suggests that increased fluctuations in inter-packet timing contribute to greater synchronization errors and represent a quality measure of the ongoing synchronization process.

Notably, the red dashed line in each subplot marks the threshold value of σ^2_{100} equal to 225 ms^2, approximating the minimum variance achievable under optimal communication conditions. In all sections, data points below this threshold exhibit relatively low errors, while points exceeding this threshold tend to show significantly higher errors, confirming the impact of variance on synchronization performance. Additionally, configurations with higher packet lengths and more connected devices tend to cluster in the high-error, high-variance region, reinforcing the trends observed in Table II.

Across acquisition sections, the impact of variance on RSE appears to diminish progressively: while section 1 exhibits a strong correlation between variance and error, later sections, such as sections 5 and 6, show a more moderate relationship, with data points becoming more clustered and regression slopes decreasing. This may suggest that over time, the network stabilizes, potentially due to the convergence of the synchronization algorithm, where iterative updates to the regression matrices (Eq. 16 and 17) result in reduced sensitivity to packet timing fluctuations.

These findings highlight a critical trade-off in wireless synchronization: increasing the number of connected devices and packet length improves data throughput but significantly raises synchronization error and its variability. This may be primarily attributed to the network congestion produced by high throughput applications: the likelihood of packet collisions is higher in system configurations involving more than two piconets operating in parallel (#Peripherals > 4). Additionally, the thread-based control software may introduce unpredictable delays in central timestamping as the number of concurrent threads increases. Nevertheless, even the results obtained under worst-case configurations remain excellent for many biomedical applications. The linear trends observed in Figure 12 reinforce that minimizing inter-packet time variance is essential for improving synchronization accuracy, making it a key factor in optimizing network performance.

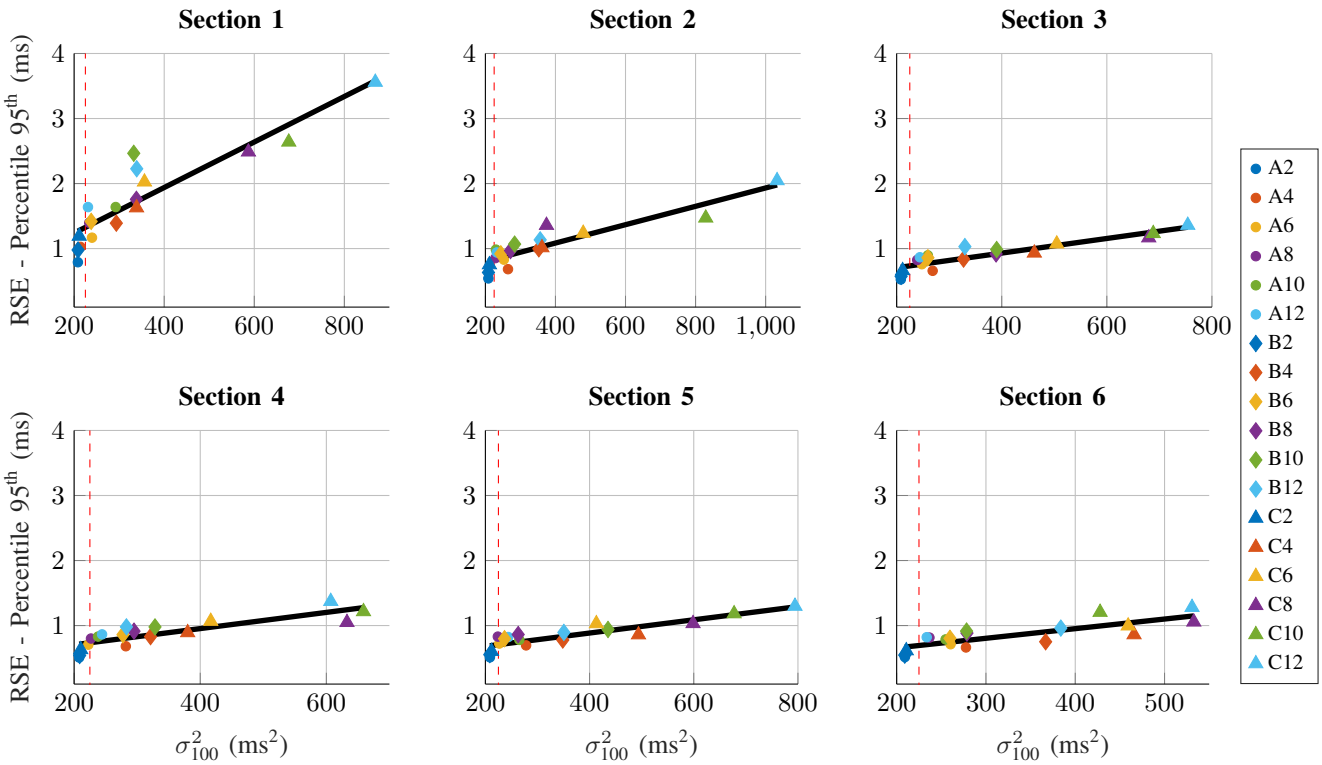


Fig. 12. Relationship between the 95th percentile of Relative Synchronization Error (RSE) and inter-packet time variance (σ^2_{100}) across six different acquisition sections. In each scatter plot, data points represents the median value of the all repetitions. They are color-coded by number of connected devices and have different shapes according to the packet length used in the experimental trial. Notably, in each acquisition section a linear trend can be highlighted (solid black line). The red dashed lines denote the value of half CI squared (225 ms^2).

B. Comparison with the State of the Art

The proposed synchronization method is evaluated against existing state-of-the-art approaches, summarized in Table III. The comparison highlights key advancements in terms of scalability, synchronization accuracy, and online processing capabilities.

One of the primary contributions of this work is its ability to scale from 2 to 12 peripherals, representing a significant improvement over previous works that typically focus on small-scale networks (2–4 peripherals). The few exceptions, such as [10], which synchronizes 8 peripherals, rely on advertising-based communication, which is inherently less stable and reliable than connection-oriented communication [17]. The ability to maintain synchronization across an increasing number of peripherals further supports the robustness of the proposed approach in practical, multi-node wireless sensing applications.

Compared to the reported average accuracy of previous methods, the proposed synchronization approach achieves a median accuracy of 0.21 ms in case of two connected peripherals, increasing up to 0.62 ms in the worst case (12 connections). While some highly specialized solutions, such as [7], [10], report ultra-low synchronization errors, they operate in significantly different conditions, leveraging advertising-based synchronization with fewer timing constraints. As mentioned in Section I, advertising mode allows peripherals node to periodically broadcast data to a central scanner without requiring connection establishment, scheduled communication intervals, or packet retransmission mechanisms. While the absence of these features reduces unpredictable communication delays, such as those introduced by connection scheduling or retransmission backoff, it also makes the data exchange inherently unreliable.

Similarly, the approach in [14] achieves remarkable drift correction (330 ns per 60 s), but its applicability to multi-device systems remains untested. In addition, this method relies on broadcasting synchronization packets from a central device to multiple peripherals. With this communication mode, the simultaneous transmission of synchronization information and sensor data is not possible.

Among the connection-based approaches, our work achieves accuracy values comparable with recent BLE-based implementations (e.g., [6], [8], [25]), while significantly extending the number of connected peripherals. Notably, studies such as [8] and [25] demonstrate synchronization in 4 and 2-device networks, respectively, with average errors around 0.356 ms and 0.477 ms, comparable to our worst-case performance despite operating in simpler network conditions. The results further confirm that the proposed approach can maintain competitive accuracy even as the system scales.

The method proposed by Li et al. [25] is the only one that exploits the continuous stream of sensor data from the peripherals to the central device to perform synchronization concurrently. However, their approach was demonstrated on a simple system configuration consisting of only two peripher-

TABLE III
STATE OF THE ART IN SYNCHRONIZATION ALGORITHMS FOR BLE-BASED WIRELESS SENSOR NETWORKS

Work	Year	#Peripherals	Communication mode	Connection Interval	Device Platform	Online algorithm	Sync via Data Packets	Average accuracy
[13]	2015	2	Connection	ND	Bluegiga BLE121	✓	✗	0.03 ms
[10]	2016	8	Advertising	-	RL78-G14	✓	✗	0.01 ms
[24]	2017	2	Connection	ND	Bluegiga BLE121LR	✓	✗	0.75 ms
[7]	2020	4	Advertising	-	nRF52832	✓	✗	0.03 ms
[14]	2021	2	Advertising	-	nRF52	✓	✗	330 ns per 60 s
[8]	2023	4	Connection	7.5 ms	STM32L476	✓	✗	0.356 ms
[5]	2023	2	Advertising	-	nRF52832	✓	✗	0.2 ms
[25]	2023	2	Connection	15 ms 10 ms	CC2640R2f nRF52840	✓	✓	0.069 ms 0.477 ms
[6]	2024	3	Connection	ND	ND	✗	✗	$\leq$1 ms
This work	2025	2 $\rightarrow$ 12	Connection	30 ms	Apollo3 Blue	✓	✓	0.21 ms $\rightarrow$ 0.62 ms

ND = Not Declared

als, which is unrealistic in the context of sEMG acquisition. Moreover, it does not address the issue of unpredictable delays across the system stack.

Another major advantage of the proposed approach is the implementation as an online synchronization algorithm, ensuring real-time processing. This is a fundamental distinction from works such as [6], which achieve synchronization only in post-processing, limiting their use in real-time applications. Additionally, connection-based synchronization requires careful handling of network congestion, which our method successfully addresses while preserving online capabilities.

This work employs a 30 ms CI, striking a balance between data throughput and synchronization stability. Many prior studies rely on shorter intervals (7.5 - 15 ms) to improve accuracy [8], [25], but at the cost of limiting scalability. In fact, increasing the number of communication channels requires to set the CI value accordingly. However, too high CI value limit the maximum data throughput that can be achieved. Our approach successfully synchronizes up to 12 peripherals without severe degradation in performance, proving its robustness in complex networks. To do it, we used a maximum of three central devices (BLE dongles) to control 12 peripherals, without compromising the data throughput needed by our target application (i.e., sEMG signal acquisition).

The ease of implementation represents another key feature that distinguishes our method from the state of the art. Among existing works, only Balasubramanian et al. [8] employed a high-level data acquisition system designed for high-performance motion tracking, built on a Python-based control platform. They tackled the challenge of unpredictable delays across the system stack using a neural network trained to identify patterns in time-stamped data series. However, this solution appears highly tailored to their specific system and would require retraining to be generalized to different platforms. Additional limitations are found in methods that rely on the connection establishment event for synchronization between central and peripheral devices [13], [24]. These approaches lack continuous clock drift compensation and require periodic disconnection and reconnection, which hinders real-time performance.

C. SharkTooth in Biomedical Applications

The proposed synchronization method was initially developed to address time alignment challenges in a custom wearable system designed by our research group (Section II). However, recognizing the broader demand for a robust and easily implementable BLE-based synchronization technique at the application layer, we extended our approach to ensure scalability across diverse use cases. This resulted in a solution capable of addressing synchronization requirements in various real-world scenarios, particularly those demanding millisecond-level timing accuracy.

As part of the validation process, we analyzed the relationship between RSE and the global data throughput of the BLE network. As illustrated in Figure 13, higher data throughput generally leads to increased RSE, with the most pronounced growth observed in section 1. This suggests that specific system configurations or environmental conditions may amplify delay estimation errors under high data load. The observed linear trend provides a practical tool for estimating the expected RSE of a given network setup based solely on its data throughput, enabling informed design choices for BLE-based wireless sensor networks in biomedical applications.

For example, in our system, a single-channel sEMG signal sampled at 1 kHz with 16-bit resolution results in a data throughput of 2 kB/s per sensor. In multi-sensor configurations, the network throughput reaches 12 kB/s and 24 kB/s for 6 and 12 sensors, respectively. These configurations approach the upper end of the validated throughput range and are relevant for applications such as high-density muscle activation mapping or neuromuscular assessments. Notably, in the case of sEMG, the 95$^{\text{th}}$ percentile of RSE reaches median values higher than the signal's sampling period (i.e., 1 ms), potentially affecting short-term signal alignment. However, a key observation is that longer acquisitions tend to reduce RSE values, due to the averaging and convergence phenomena described earlier.

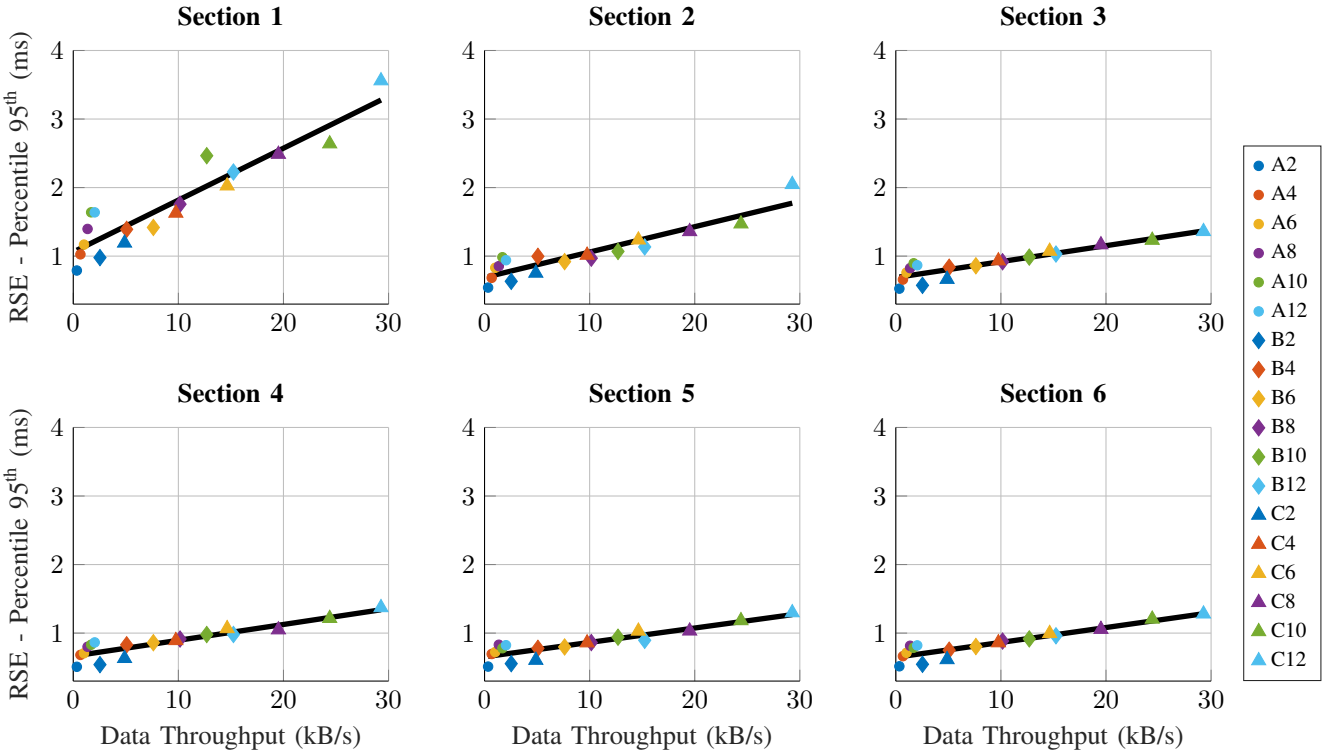


Fig. 13. The median Relative Synchronization Error (RSE) 95th percentile as a function of the global data throughput of the BLE network across the six acquisition sections. Data points correspond to 18 different configurations (A2–C12), each with unique marker styles and colors. A linear regression line (in black) is fitted to the data in each section to illustrate the trend of RSE increase with throughput.

As a result, for these high-demand applications, *SharkTooth* is particularly suitable for mid- to long-term acquisitions, where time alignment precision improves over time.

In contrast, an inertial measurement unit (IMU) acquiring 3-axis accelerometer and 3-axis gyroscope data (6 channels total) at 208 Hz with 16-bit resolution generates approximately 2.5 kB/s per sensor. In this case as well, considering multi-sensor scenarios—commonly employed in full-body motion capture systems for rehabilitation or biomechanical analysis—the total network throughput may reach the upper limit of the validation domain. Nevertheless, despite the higher total throughput in these configurations, the 95$^{\text{th}}$ percentile of RSE remains consistently lower than the corresponding sampling period (approximately 4.8 ms), ensuring reliable synchronization even during short acquisitions. This highlights the versatility and robustness of the proposed method across a wide range of application scenarios, with specific strengths depending on the signal type and duration of data collection.

VI. Conclusion

In this work, we introduced *SharkTooth*, a scalable, real-time synchronization algorithm designed for BLE-based WBSNs. The algorithm addresses the critical challenge of multi-node synchronization by mitigating clock drift, transmission latencies, and unpredictable delays without requiring specialized hardware modifications or low-level timestamping. Instead, *SharkTooth* operates entirely at the application layer, making it highly adaptable to a wide range of off-the-shelf BLE devices. Compared to the state of the art, *SharkTooth* offers a unique combination of high accuracy, scalability, and ease of integration.

Through rigorous validation involving up to 12 peripherals across 18 network configurations, we demonstrated that *SharkTooth* achieves a median absolute average synchronization error between 0.21 ms and 0.62 ms even under high-congestion BLE conditions. The algorithm exhibits:

- **Scalability**: The algorithm successfully synchronizes up to 12 peripheral devices, a significant improvement over existing methods. Additionally, many of the tested configurations show promise for effective extension to support a higher number of simultaneous connections.
- **Robustness**: By mitigating the effects of packet retransmissions and communication-induced delays, *SharkTooth* ensures reliable synchronization across diverse operating conditions.
- **Real-time operation**: This method enables continuous online synchronization, making it well-suited for real-time biomedical and motion-tracking applications.

While our experiments confirm the effectiveness of *SharkTooth* in BLE-based systems, future work will focus on enhancing its efficiency and adaptability. Although beyond the scope of this study, low-level technical refinements could further improve the already strong synchronization performance of our implementation. A key area of investigation will be the impact of varying the CI values and packet sending frequency on synchronization accuracy, data throughput, and network reliability. Additionally, we plan to validate the algorithm in real-world biomedical scenarios, such as wearable rehabilitation

devices and remote patient monitoring systems. By providing a high-accuracy, scalable, and easily deployable synchronization solution, *SharkTooth* paves the way for the next generation of wireless body sensor networks.

ACKNOWLEDGMENT

The authors would like to thank Dr. Andrea Prestia for his contribution to the development of the hardware used in the experimental validation. They also gratefully acknowledge Dr. Fabio Rossi and Dr. Andrea Mongardi for their support in the conceptualization of the experimental validation and in the discussion of the preliminary results.

REFERENCES

[1] S. Shajari, K. Kuruvinashetti, A. Komeili, and U. Sundararaj, "The emergence of ai-based wearable sensors for digital health technology: a review," *Sensors*, vol. 23, no. 23, p. 9498, 2023.

[2] S. Shabani, A. K. Bourke, A. Muaremi, J. Praestgaard, K. O'Keeffe, R. Argent, M. Brom, C. Scotti, B. Caulfield, and L. C. Walsh, "An automatic foot and shank IMU synchronization algorithm: proof-of-concept," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2022, pp. 4210–4213.

[3] R. V. Schulte, E. C. Prinsen, L. Schaake, and J. H. Buurke, "Synchronization of wearable motion capture and EMG measurement systems," in *2022 International Conference on Rehabilitation Robotics (ICORR)*. IEEE, 2022, pp. 1–6.

[4] G. Cerone, A. Giangrande, M. Ghislieri, M. Gazzoni, H. Piitulainen, and A. Botter, "Design and validation of a wireless body sensor network for integrated EEG and HD-sEMG acquisitions," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 61–71, 2022.

[5] J. Cappelle, S. Goossens, L. De Strycker, and L. Van der Perre, "Low-Power Synchronization for Multi-IMU WSNs," *IEEE Embedded Systems Letters*, 2023.

[6] M. Depolli, N. Verdel, and G. Kosec, "Offline synchronization of signals from multiple wireless sensors," *IEEE Sensors Journal*, 2024.

[7] G. Coviello, G. Avitabile, and A. Florio, "A synchronized multi-unit wireless platform for long-term activity monitoring," *Electronics*, vol. 9, no. 7, p. 1118, 2020.

[8] K. K. Balasubramanian, A. Merello, G. Zini, N. C. Foster, A. Cavallo, C. Becchio, and M. Crepaldi, "Neural network-based Bluetooth synchronization of multiple wearable devices," *Nature Communications*, vol. 14, no. 1, p. 4472, 2023.

[9] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 39–49.

[10] S. Sridhar, P. Misra, G. S. Gill, and J. Warrior, "Cheepsync: a time synchronization service for resource constrained bluetooth le advertisers," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 136–143, 2016.

[11] Delsys, "Trigno avanti sensor," accessed: 2025-04-19. [Online]. Available: https://delsyseurope.com/trigno-avanti/

[12] Movella, "Xsens MTw Awinda," accessed: 2025-04-19. [Online]. Available: https://www.movella.com/products/wearables/xsens-mtw-awinda

[13] A. Bideaux, B. Zimmermann, S. Hey, and W. Stork, "Synchronization in wireless biomedical-sensor networks with Bluetooth Low Energy," *Current Directions in Biomedical Engineering*, vol. 1, no. 1, pp. 73–76, 2015.

[14] F. Asgarian and K. Najafi, "BlueSync: time synchronization in Bluetooth low energy with energy-efficient calculations," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8633–8645, 2021.

[15] L. Kong, W. Li, T. Zhang, H. Ma, Y. Cao, K. Wang, Y. Zhou, A. Shamim, L. Zheng, X. Wang *et al.*, "Wireless technologies in flexible and wearable sensing: from materials design, system integration to applications," *Advanced Materials*, vol. 36, no. 27, p. 2400333, 2024.

[16] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica, "Performance evaluation of bluetooth low energy: A systematic review," *Sensors*, vol. 17, no. 12, p. 2898, 2017.

[17] M. Woolley, Jan. 2021. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/understanding-reliability-in-bluetooth-technology/

[18] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, 2002.

[19] S. Ganeriwal, "Timing-sync Protocol for sensor networks," in *ACM SenSys*, 2003.

[20] K. S. Yıldırım and Ö. Gürcan, "Efficient time synchronization in a wireless sensor network by adaptive value tracking," *IEEE Transactions on wireless communications*, vol. 13, no. 7, pp. 3650–3664, 2014.

[21] F. Gong and M. L. Sichitiu, "Cesp: A low-power high-accuracy time synchronization protocol," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2387–2396, 2015.

[22] K. S. Kim, S. Lee, and E. G. Lim, "Energy-efficient time synchronization based on asynchronous source clock frequency recovery and reverse two-way message exchanges in wireless sensor networks," *IEEE Transactions on Communications*, vol. 65, no. 1, pp. 347–359, 2016.

[23] X. Huan, K. S. Kim, S. Lee, E. G. Lim, and A. Marshall, "A beaconless asymmetric energy-efficient time synchronization scheme for resource-constrained multi-hop wireless sensor networks," *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1716–1730, 2019.

[24] F. J. Dian, A. Yousefi, and K. Somaratne, "A study in accuracy of time synchronization of BLE devices using connection-based event," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2017, pp. 595–601.

[25] J. Li, E. Quintin, H. Wang, B. E. McDonald, T. R. Farrell, X. Huang, and E. A. Clancy, "Application-Layer Time Synchronization and Data Alignment Method for Multichannel Biosignal Sensors Using BLE Protocol," *Sensors*, vol. 23, no. 8, p. 3954, 2023.

[26] A. Prestia, F. Rossi, A. Mongardi, P. Motto Ros, M. Ruo Roch, M. Martina, and D. Demarchi, "Motion Analysis for Experimental Evaluation of an Event-Driven FES System," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, no. 1, pp. 3–14, 2022.

[27] A. Prestia, F. Rossi, A. Mongardi, D. Demarchi, and P. Motto Ros, "Raspberry Pi based Modular System for Multichannel Event-Driven Functional Electrical Stimulation Control," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2022, pp. 2592–2597.

[28] N. Landra, A. Prestia, A. Mongardi, F. Rossi, D. Demarchi, and P. Motto Ros, "A Biomimetic Multichannel Synergistic Calibration for Event-Driven Functional Electrical Stimulation," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2022, pp. 85–89.

[29] F. Rossi, A. Prestia, A. Mongardi, N. Landra, P. Motto Ros, and D. Demarchi, "Live Demonstration: A Real-Time Bio-Mimetic System for Multichannel FES Control," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2022, pp. 248–248.

[30] F. Rossi, A. Mongardi, P. Motto Ros, M. Ruo Roch, M. Martina, and D. Demarchi, "Tutorial: A Versatile Bio-Inspired System for Processing and Transmission of Muscular Information," *IEEE Sensors Journal*, vol. 21, no. 20, pp. 22 285–22 303, 2021.

[31] A. Prestia, A. Mongardi, D. Demarchi, F. Rossi, and P. Mott Ros, "A Low-Power Low-Complexity Circuit for Event-Based Feature Extraction from sEMG," in *2024 IEEE SENSORS*, 2024, pp. 1–4.

[32] Sparkfun Electronics, "Artemis," accessed: 2025-04-19. [Online]. Available: https://www.sparkfun.com/artemis

[33] *LSM6DSO32*, ST Microelectronics, 3 2020, rev. 1.

[34] FreeRTOS, accessed: 2025-04-19. [Online]. Available: https://www.freertos.org/

[35] Nordic Semiconductor, "nRF52840 Dongle," accessed: 2025-04-19. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle

[36] T. Gerstenberg, "Blatann," accessed: 2025-04-19. [Online]. Available: https://blatann.readthedocs.io/en/latest/

[37] A. Baldini, R. Garofalo, E. P. Scilingo, and A. Greco, "A real-time, open-source, IoT-like, wearable monitoring platform," *Electronics*, vol. 12, no. 6, p. 1498, 2023.

[38] H. Li, J. Wang, S. Zhao, F. Tian, J. Yang, and M. Sawan, "Real-time biosignal recording and machine-learning analysis system," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 427–430.

[39] J. Joshi, K. Wang, and Y. Cho, "Physiokit: An open-source, low-cost physiological computing toolkit for single-and multi-user studies," *Sensors*, vol. 23, no. 19, p. 8244, 2023.

[40] Arduino, "Arduino UNO Rev3," accessed: 2025-04-19. [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3?srsltid=AfmBOoqYawxFmI3b9nl4S2dBN6jGRz8q-NL6dwMSBQ7UdFpRcCF0Wb9R

[41] RIGOL, "Oscilloscopio Rigol MSO5104 100MHz mixed signals," accessed: 2025-04-19. [Online]. Available: https://www.rigolitalia.it/products/https://www.rigolitalia.it/products/oscilloscopio-rigol-mso5104-100mhz-mixed-signals?srsltid=AfmBOop-hfIM_PuoPGRIumEHhuaV5e2WQce2IfqMfHahy6RqoOI5sRQc

Nicolò Landra (GSM'22) received both his Bachelor and Master degrees in biomedical engineering from Politecnico di Torino, Turin, Italy, in 2019 and 2021, respectively.

Since November 2022 he is a PhD student at Politecnico di Torino, Department of Electronics and Telecommunications. He is a member of the eLiONS (formerly MiNES) Laboratory and his interests include the design and development of wearable electronic systems for biomedical applications, mainly focusing on the rehabilitation field.

Danilo Demarchi (M'10-SM'13) received Engineering Degree and Ph.D. in electronics engineering from Politecnico di Torino, Italy, in 1991 and 1995, respectively.

Full Professor at Politecnico di Torino, Department of Electronics and Telecommunications. Working on Smart System Integration and IoTs for the AgriFood Value Chain and for BioMedical Devices. Pioneer in the research on Wearable Plant Sensors. Visiting Professor at EPFL Lausanne (2019) and at Tel Aviv University (2018-2021). Visiting Scientist (2018) at MIT and Harvard Medical School for the project SISTER (Smart electronic IoT SysTEms for Rehabilitation sciences). Author and co-author of 5 patents and more than 350 scientific publications in international journals and peer-reviewed conference proceedings. Leading the eLiONS (electronic Life-Oriented iNtelligent Systems - http://elions.polito.it) Laboratory of Politecnico di Torino and coordinating the Italian Institute of Technology Microelectronics group at Politecnico di Torino (IIT@DET).

Founder and Editor in Chief of the IEEE Transactions on AgriFood Electronics -TAFE. Founder and General-Co-Chair of the IEEE Conference on AgriFood Electronics - CAFE. Founder and Chair of the IEEE CAS Special Interest Group on AgriFood Electronics. 2023-2024 Distinguished Lecturer for the IEEE CAS Society with the Lecture "Let the Plants Do the Talking: Smart Agriculture by the messages received from Plants and Soil". Founder at Politecnico di Torino of the Master's Degree in AgriTech Engineering, where he is teaching the course of "IoT for Agriculture". General Chair of IEEE BioCAS (Biomedical Circuits and Systems) Conference in 2017 in Torino and founder of IEEE FoodCAS Workshop (Circuits and Systems for the FoodChain). TPC Co-Chair of IEEE ICECS 2019, IEEE BioCAS 2021 and 2022 conferences. General Co-Chair of IEEE BioCAS 2023.

Paolo Motto Ros (M'16) received the M.Sc. and Ph.D. degrees in electronic engineering from the Politecnico di Torino, Italy, in 2005 and 2009, respectively.

Assistant Professor at Politecnico di Torino, Department of Electronics and Telecommunications, Turin, Italy. Post-doc researcher at Politecnico di Torino (2009-2012), senior (since 2014) post-doc researcher at Istituto Italiano di Tecnologia (2012-2019), senior post-doc researcher and adjunct professor (since 2017) at Politecnico di Torino (2019-2022). Author and co-author of more than 90 international scientific publications.

Dr. Motto Ros was program co-chair of ApplePies 2024, member of the organizing committee of IEEE ICECS 2019, the FoodCAS Satellite Event at IEEE ISCAS 2021, and IEEE CAFE 2023; review committee member of IEEE BioCAS 2021–2024, special session organizer at IEEE MeMeA 2021, program committee member of IEEE LASCAS 2022 and 2023, guest editor of MDPI Sensors and guest associate editor of Frontiers in Neurorobotics. He is Associate Editor of IEEE Transactions on Biomedical Circuits and Systems and IEEE Transactions on AgriFood Electronics.