




Article

Design of an Open Remote Electrocardiogram (ECG) Service

Augusto Ciuffoletti [†] Department of Computer Science, University of Pisa, 56127 Pisa, Italy; augusto.ciuffoletti@unipi.it[†] Current address: Dipartimento di Informatica, Largo Bruno Pontecorvo, 56122 Pisa, Italy.

Received: 14 March 2019; Accepted: 18 April 2019; Published: 24 April 2019



Abstract: Currently, the deployment of services for real-time delivery of an electrocardiogram to a remote site has a cost that prevents its widespread diffusion, which would contribute to saving lives with prevention, assistance and rescue efficiency. To fill this gap, we introduce the design of a remote electrocardiogram service that privileges open, low-cost options. The architecture is based on the HyperText Transfer Protocol (HTTP) and uses commercial off-the-shelf devices to implement the sensor on the patient's side. The doctor uses a laptop browser to display the tracing, and a cloud computing instance connects the two using WebSockets. A prototype is built to evaluate overall performance, the power consumption of the patient's side device, and the quality of rendering on doctor's browser. The patient's sensor prototype device is portable, and its power consumption is below 1 Watt, thus allowing a daylong autonomy when operated on batteries. Its cost is below 50\$, and the required hardware is commercially available. The whole design is ready for on-field evaluation, and it is available in a public repository.

Keywords: non-diagnostic ECG; WebSocket; Cloud PaaS; ECG acquisition; Arduino; ESP8266

1. Introduction

In 2017, the REHEARSE-AF study [1] investigated the efficacy of electrocardiogram (ECG) screening in a home *health-care* style asking 500 patients to record without assistance a 3-leads ECG. The measurement was run two times a week for the period of one year using commercial devices. Each record was uploaded to a server, using a WiFi Access Point (AP) available in the patient's home. The ECG was automatically screened to detect Atrial Fibrillation (AF) symptoms, and later over-read by a specialist. The outcome of the study was encouraging, since the likelihood of early AF diagnosis, compared with that of a routine care arm, was significantly increased, at a price of little more than 10,000\$ per diagnosis.

There are many other use cases for a remote ECG service, besides the relevant one described above: two stories may give an idea. One is that of an ambulance with paramedical personnel rescuing an injured person after a car accident. It is likely that the Internet is reachable using ground or satellite broadband services, possibly tethering a smartphone. Using the online remote ECG service, we deliver the trace to the medical staff in the hospital, which defines the emergency level. A different home health-care story is that of a cardiac patient that is periodically contacted by her family doctor to check general conditions. In that case, the physicist uses the remote ECG service to receive the trace without leaving his desk, possibly on the phone with the patient.

Remote ECG services have long been recognized as relevant, and have been the target of research interest and investments. There are currently many pilot initiatives going in that direction—similar to

the one referenced above—that are supported by public and private funding, but it is hard to scale up such experiments to cover a whole community, instead of a sample. The concern is not about their proved effectiveness, but about their financial impact.

The hardware and the infrastructure that acquires and manages the trace, which we call the *remote ECG service*, have a significant cost. This was justified twenty years ago when such services emerged, but it is not consistent anymore, given the fall in prices of high-tech devices and services. Exploring today the feasibility of a low-cost, open remote ECG service may break the status quo, and raise the interest of stakeholders—i.e., those that produce, manage or use medical equipment—towards technologies and design options that make affordable a widespread service made of inter-operable devices and networks.

This paper goes in that direction and proves that an open, low-cost service for long-range delivery of a 3-leads ECG is within reach: we show how to implement it using cloud facilities integrated with personal devices. The market offers comparable solutions, like those used in the REHEARSE-AF study, but these rely on proprietary hardware, protocols, software and services. This state of affairs biases high prices, and ultimately limits such lifesaver services to diseases that are sufficiently diffused to promise a reasonable *cost per diagnosis*.

A feature of this project is that its results are reproducible and, if brought to production, they may be useful for large scale screenings to obtain results comparable to those described for the REHEARSE-AE project, but at a significantly lower cost.

The paper is organized as follows:

- the next section is a review of on-topic literature that introduces our methods and results;
- Section 3 applies defined methods in the design of a remote ECG service;
- in Section 4, we describe the materials to implement a functional prototype
- in Section 5, we perform an exhaustive test of all components, to assess the results and guide future improvements;
- finally, we discuss the potential of our prototype in some use cases.

2. Related Works and Results Summary

The electrical functionality of the heart is a piece of fundamental diagnostic information for the medical staff. Its recording dates back to the end of the 19th century. From that time, research addressed first the amplification of the signal, whose amplitude is in the μV range, and next to the design of filters that remove unwanted components (like powerline *hum*, or signals from body activity) [2].

Computers came into play in connection with filters, and today they extract the fundamental features of the ECG signal [3]. Gutierrez-Rivas et al. evaluate the computing resources needed, with a survey of research results in the field [4].

With the advent of the Web, storage and transport of ECG data emerge as a practical perspective. A study that in origin was aimed at using the ECG for personal identification [5] incidentally created a database of ECGs, which later became a precious resource for many. The availability of historical data is relevant for medical research, daily health-care use, and educational purposes.

A primary concern with ECG and other medical data is their confidential nature. Cryptographic systems can be used to secure not only their transport but also their storage when a database aggregates data coming from several distinct sources in a Medical Sensor Network (MSN) [6].

The ubiquitous presence of wireless networks justifies the realization of portable (or even *wearable* [7]) ECG devices that forward the trace to a nearby Personal Computer (PC) or tablet. A Body (or Personal) Data Network (BAN) is a network infrastructure connecting many such devices. The idea dates back to 2001 when Jones et al. [8] introduce the basic principles and concepts. Ten years later, a survey [9] lists nine

BAN projects, five of which include ECG devices. Meanwhile, many research studies produce detailed designs of BANs [10–12] targeting local area networks, and therefore confined in a room or a building.

The availability of cloud servers allows overcoming such limits. In 2013, Xia et al. propose a cloud-based infrastructure [13]: the physician and the patient are clients of a Web server that processes the ECG, evaluates its quality, and provides parameter extraction and visualization. The server is in the AmazonWeb Services (AWS) cloud, while the client uses a personal device that interacts with the server using Secure Hypertext Transfer Protocol (HTTPS). The two edge units communicate in real time, so that doctor and patient may interact during the ECG. In this scenario, end-to-end encryption is a challenge since the server cannot analyze encrypted data. To mitigate the problem, Page et al. [14] propose the use of an encryption algorithm which allows homomorphic operations on encrypted data.

The presence of a remote ECG service is fundamental for large scale screenings, like the REHEARSE-AF study referenced in the introduction [1]. The study involved 1001 patients divided into two arms to evaluate the improvement of early diagnosis of AF—which is often asymptomatic and implicated in ischemic strokes—using a remote ECG monitoring service named *iECG*. The trial consisted of taking an ECG two times a week, at home and without assistance, using a commercial ECG device attached to a tablet. Using the home Wireless Fidelity (WiFi) AP, the trace was uploaded to a server for automatic and human analysis targeting AF symptoms. The experiment successfully diagnosed 19 AFs, versus five in the control arm, at the cost of slightly more than 10,000\$ for each AF diagnosis, and was supported by public and private funding.

These results are encouraging, but the cost of such trials is a reason of concern [15]: to apply them on a large scale, we need to limit their financial impact by design and build an ecosystem of interoperable devices and services produced by competing enterprises which will encourage the diffusion of the service in a virtuous circle.

The challenge is to finally make the service affordable also to marginal milieux, like developing countries and rural areas [16,17] which are more in need of a qualified remote medical assistance than Western towns. We should apply a cost-aware approach to the solution of the technical challenges we meet in the design of a remote ECG service [18].

To implement such an approach, we address a design philosophy summarized in three key-words:

- **Open Source:** components and protocols are exhaustively documented and freely reproducible;
- **Low Cost:** the less expensive option is always preferable. As a corollary, if one functionality is already available for free, it is not re-implemented;
- **Commercial Devices:** devices must be available on retail (aka Commercial Off-the-shelf (COTS)).

We use the acronym OpLoC to indicate such a design approach. It needs to be applied to all aspects of the design—with a holistic attitude—to avoid its violation on one design aspect as a consequence of another design options, which, taken alone, appears as valid [19].

Put together, the three OpLoC principles ensure that implementations are easily reproducible, which is fundamental for further investigation. When scaling up to production, the OpLoC approach grants the coexistence of different solutions made of interoperable parts, since interfaces are open. Moreover, low cost widens the market, since the design is not selective by scale or wealth.

Thus, given the advantages of an OpLoC solution, we want to investigate whether it is technically possible to obtain a useful solution for a remote ECG service which complies with the OpLoC principles.

The design we describe in this paper uses *open source* devices: an open source board for the sensor, an Arduino for real-time data acquisition, a board based on the ESP8266 chip (NodeMCU) for Internet delivery, a cloud container as a relay point, an HTTP-compliant browser to visualize the trace. The programs that have been written ad hoc are in public domain repositories, and the rest of the code is

open source. The *price* of a single device is of a few tens of dollars and can be significantly reduced using hardware that is not compatible with Arduino pinning; the one we use in the prototype is available on major online shops.

Figure 1 summarizes our results, showing the trace displayed on the doctor’s browser with a latency of less than one second. The patient wears wrist electrodes appropriate for self-measurement. Power consumption allows a day-long operation of the patient’s sensor with a 3.3 V LiPo rechargeable battery.

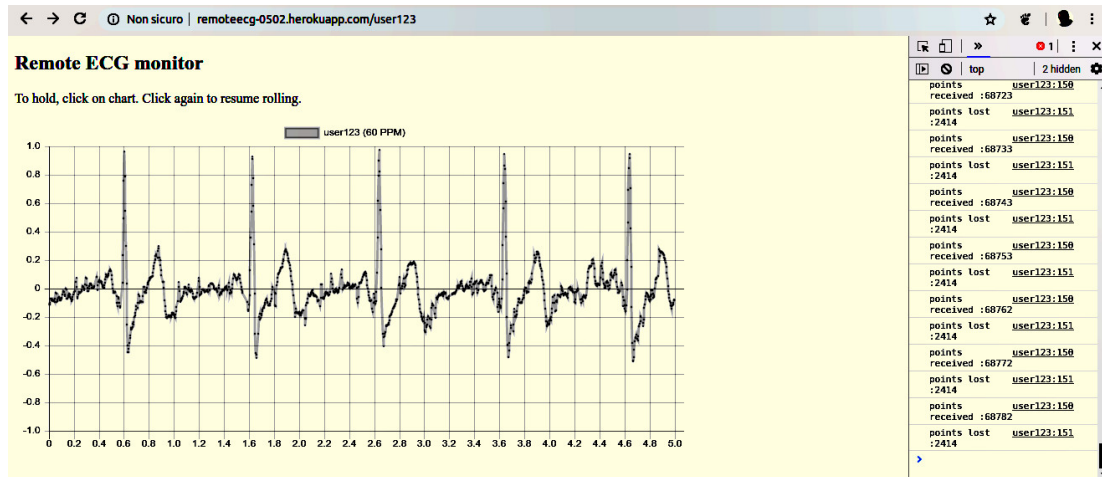


Figure 1. ECG visualization on doctor’s display. Both the laptop and the sensor are in Italy, while the relay server is geolocated in Ireland; on the right side, the report about transfer/loss rates.

In a nutshell, we actualize the ideas of Xia et al. [13], but in a BAN perspective [8], with a financial impact that allows extending nationwide a screening like REHEARSE-AF [1]. One step towards a future ubiquitous ECG monitoring service.

3. An Open Service for Remote ECG

We distinguish three components of a remote ECG system: a display on the doctor’s side, a sensor on patient’s, and a communication infrastructure between them. In this section, we discuss their specific requirements and the design options that are consistent with the OpLoC approach.

3.1. Doctor’s Side Web Interface

On the doctor’s side, we need a generic display: a laptop or a smartphone fit the purpose. To avoid portability issues, we prefer that such a task runs as a JavaScript application in a browser.

As a consequence of the above option, the protocol for the transport of ECG data is the Hypertext Transfer Protocol (HTTP). Such a system design is very convenient, as HTTP implements an effective security policy while avoiding the limits created by firewalls and other security measures. However, HTTP features a rigid request–response scheme which is a problem in our design, since we need to establish a persistent one-way flow for the ECG data. To solve this kind of problem, in 2011, the Internet Engineering Task Force (IETF) introduced the *WebSocket* [20]: in essence after having established an HTTP session, the partners may upgrade it to a bidirectional channel, similar to a Transport Control Protocol (TCP) connection—hence the name—not constrained to follow the request/response protocol.

Given that the doctor’s interface is an HTTP client, we need to find where to run a server for it. It is impractical to implement the server in the patient’s sensor since in that way we should traverse the NAT

which is almost certainly in place on that side. Instead, we consider the use of a relay server in the cloud as a better solution, but we first examine the operation of the patient's side device.

3.2. Patient's Side Sensor Device

Unlike the doctor's device, the sensor on the patient's side has specific features. Although today's smartphones have sensors covering a wide range of applications, they are not yet capable of acquiring heartbeat signals with sufficient precision. Thus, we need to design an ad hoc ECG sensor that receives, amplifies and processes the bio-signal. In this section, we analyze several options for each of its key aspects: ECG acquisition, usability, power consumption, link level connectivity, data transport, and finally hardware design.

Regarding *ECG acquisition*, the processing of bio-signals is an evolving technology, for which sophisticated methods are available. However, the constrained computational capabilities of low-cost devices limit the complexity of a viable solution. Thus, we envision the use of one among the commercial boards that acquire and filter the electrical signals produced by the muscular activity.

The *usability* of the device takes into account that its operation should be simple, without compromising ECG quality, and electrodes placement shouldn't be critical: wearable electrodes like wristbands are better than chest leads. Usability also indicates wireless connectivity, discussed below, as preferable.

Since the device needs to be portable, we need to limit *power consumption* to ensure sufficient autonomy when operated on a battery. To quantify a target power consumption, we consider that the energy density of Lithium-Metal batteries is 0.5 kWh kg^{-1} and introduce a weight limit of 0.1 kg. Under such assumptions, battery capacity cannot exceed 50 Wh. To obtain an autonomy of 10 hours, which fits use cases like a day-long monitoring or a remote nursery, we have that power consumption must be lower than 5 W. Such a limit excludes using a powerful Single Board Computer (SBC) with a power consumption close to such a theoretical limit, like the popular Raspberry PI [21], and addresses constrained devices like Micro-Controller Units (MCU), characterized by a power consumption lower than 1 W.

Another relevant requirement for the ECG sensor is that its use should extend beyond hospitals and private homes so that wireless technologies are preferable to link the sensor to the Internet. Here, we discuss the pros and cons of three widely diffused *link level* wireless technologies: Bluetooth, WiFi, and Mobile Wireless, which includes various generations after GPRS.

Bluetooth is a low-cost, well-established technology for short-range wireless communication. Its introduction in our design would require the presence of a nearby component acting as a gateway between Bluetooth and the Internet, which introduces additional cost and design complexity.

WiFi is another stable technology, with typical coverage that is slightly larger than Bluetooth. WiFi to Internet gateways (or APs) are inexpensive devices very diffused in our towns: houses, shops, and public places frequently provide a private or public AP. A smartphone can be used as a WiFi AP, extending the range of usability of the ECG sensor to any place reached by mobile wireless networks, provided that a tethering smartphone is at hand.

Our third alternative wireless technology is Mobile Wireless, which incurs slightly higher costs compared with WiFi (assuming the AP is already in place). They are due to a more expensive interface and to the need to subscribe with a mobile phone service provider. The operation range depends on the availability of a Point Of Presence (POP). In a word, connectivity compensates subscription costs.

After our analysis, we conclude that the link between the sensor and the Internet could be implemented either with WiFi or with Mobile Wireless, with mixed pros and cons.

Regarding *data transport*, having introduced a web server as an interface on the doctor's side suggests using the same technology for patient's, with the same benefits for security and stability. However, we

come to the challenge of implementing a WebSocket-capable client on a device with constrained capabilities like the ECG sensor.

An appealing alternative to HTTP is the Constrained Application Protocol (CoAP) [22], which follows HTTP guidelines and is specially adapted to operate on constrained devices. In essence, CoAP is based on UDP (instead of TCP) and does not enforce a request/response protocol. It neatly fits the requirements of our design with a marked advantage on bandwidth: comparing the headers involved in HTTP/WebSocket over TCP versus CoAP over UDP, we have 16 bytes for CoAP against 30 bytes for WebSockets. However, the relevance of such an advantage depends on the payload size.

The advantage of HTTP over CoAP is that it is an established and widely supported technology. Such features reduce coding effort and allow using widely diffused hardware devices and services, thus reducing development and deployment costs.

Again, we are facing a critical decision: a more efficient solution, though less supported by experience (CoAP standard is dated 2014), versus a less efficient one, but for which we find well-supported tools.

We finally consider the *hardware design* of the sensor. We address MCUs for cost-related reasons, but we must take into account that both the sampling process and data delivery are demanding tasks. One option is to have two of them running in parallel using one for ECG acquisition, another for network operation. The communication bitrate between the two depends on the sampling frequency: taking 250 Hz as a reference, and assuming that a 3-leads ECG transmits a 16-bit integer for each sample, the net baud-rate is of 4000 baud, which should be supplemented by timing and protocol overhead. Such a data rate is compatible with protocols that are commonly used in embedded systems so that the option of a design based on two specialized MCU is acceptable.

3.3. Relay Server in the Cloud

We consider that the browser and the ECG sensor share a Web server in the cloud that has the role of relaying the ECG data from the patient to the doctor. The Web server provides two kinds of endpoints: one for the patient, another for the doctor. Both of them upgrade to WebSockets.

The cloud server that implements the relay point requires a limited computational capacity and may exist only for the time needed by the doctor to read the ECG. Therefore, it is the ideal candidate for a microservice instance [23] since a Platform as a Service (PaaS) provider can create a new instance of a microservice in seconds, make it accessible on the Internet, and dispose of it after use.

3.4. Design Overview and Security Issues

Figure 2 summarizes the previous discussion with a design overview.

The right box represents the patient's side devices. The portable sensor hosts the analog stage with an operational amplifier and filters for ECG acquisition, a specialized MCU that encodes the analog signal for digital transmission on the internal bus, and another MCU that implements the HTTP User Agent (UA) and the WiFi interface. A nearby AP connects the portable ECG sensor with the Internet.

The left box represents doctor's side devices: another WiFi AP links the smart device (a PC or smartphone) running a browser that displays the ECG.

Regarding security, we highlight that only the routers inside the APs on both doctor's and patient's sides need to expose a public Internet Protocol (IP) address. So that, according to common practice, personal devices are protected by router Network Address Translation (NAT). Communication to and from the server can be encrypted using secure WebSockets.

The cloud provider is responsible for server safety, of which only the doctor and the patient know the Fully Qualified Domain Name (FQDN) or IP address. Using an ephemeral microservice improves security, since the potential intruder has little time to identify and attack it, and cannot reuse gathered information.

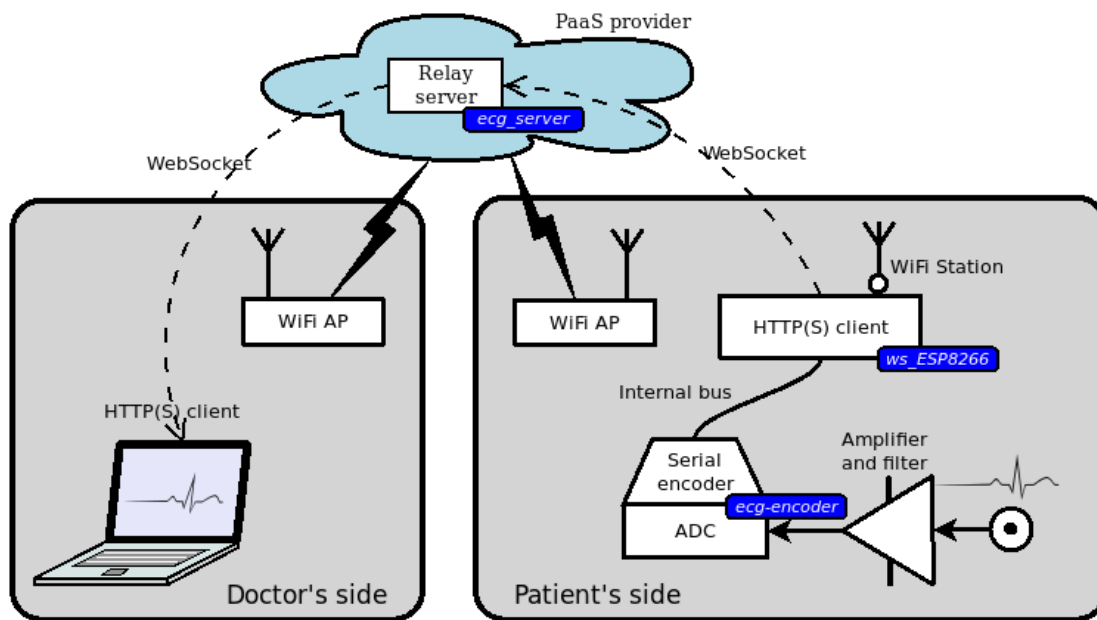


Figure 2. Remote electrocardiogram (ECG) service overview; blue labels refer to code in the repository [24].

4. A Prototype for a Remote ECG Service

The above abstract discussion leaves open some design options that require a trade-off decision. In this section, we describe a prototype realization, thus necessarily resolving such options but without claiming that the ones we have tried are optimal. To have an idea of the final result, in Figure 3, we show the device on patient’s side, while, in Figure 4, we show the doctor’s laptop displaying the ECG.



Figure 3. The patient-side prototype device, with three elastic bands, the battery, and the assembled boards.



Figure 4. The remote ECG display on the doctor's side.

Besides hardware devices, the prototype contains also software that has been written ad hoc: its presence is marked by blue labels in Figure 2. It is publicly available in a BitBucket repository [24]:

- the analog-to-serial encoder in the MCU,
- the HTTP patient-side UA,
- the relay server hosting WebSocket endpoints and doctor's access page.

The organization of this section is similar to that of Section 3 but proceeds by sub-units in a different order: from data production to visualization.

4.1. The Sensor: Amplifier and Filter

The prototype uses the popular *Olimex EKG-EMG* sensor board [25] (OLIMEX Ltd., Plovdiv, Bulgaria): its size is $6 \times 8 \times 3$ cm, it is compatible with the Arduino pinning, and costs around 30\$. There are significantly less expensive boards which are not Arduino compatible: for our prototype, such compatibility has the advantage of electrical, mechanical stability. The board is usually sold with ECG pads and wrist bands. It integrates a 3rd order filter at 40 Hz and two high-pass filters to remove high frequencies and baseline drift.

A single board acquires a non-diagnostic 3-leads ECG through a standard 3-poles jack. Board pinning allows stacking up to six of them for a diagnostic 12-leads ECG but this outside our scope.

4.2. The MCU: ADC and Serial Encoder

The MCU for sensor data acquisition is an Arduino Uno board, the cost of which is around 10\$. The connection with the ECG acquisition board is obtained by stacking them, thus linking the ADC input of the Arduino and the output from the acquisition board: the result is mechanically stable and sufficiently compact (see Figure 3). The analog output of the sensor board is converted into a 10-bits integer by the ADC embedded in the Arduino MCU.

We wrote the Arduino sketch that samples the signal and sends it to the networking MCU using the Arduino Integrated Development Environment (IDE). To have the accurate timing which we need for filtering and analysis purposes, a timer-driven interrupt triggers sampling at 250 Hz [26]. The code snippet

for the interrupt handler is in Listing 1: note that it is prepared to collect six analog data for a standard 12-lead ECG though we use only one in the prototype. The main loop waits for the buffer to be readable and encodes a data frame of 33 8-bit characters:

```
<hh>:<mm>:<ss>.<ddd> <v1><v2><v3><v4><v5><v6>Q
```

The first field is a 1 ms resolution timestamp, and the other six fields are integers in the interval $[0 - 999]$, each of which corresponding to one sample from an ECG channel encoded as a three-characters ASCII string. The Q character serves as frame separator. The redundancy of this format is functional to data integrity and simplifies the development. The encoding compresses (by clipping) the sampled 10-bits value into the $[0-999]$ range. The timestamp is not as accurate as the sampling period: it is used only for rendering and to detect missing data.

We introduce a two positions buffer to avoid interference between data fetch and encoding; a semaphore regulates access to the buffer.

It is worth saying that Olimex provides a different driver [25], but its accuracy is not sufficient for the task. Thus, we implemented a different one.

```
void Timer2_Overflow_ISR() {
  int i;
  unsigned long t;
  digitalWrite(PROBEPIN1,HIGH); // probe
  if (full[b]) {
    Serial.println("fail");
    return;
  }
  for (int Channel=0;Channel<6;Channel++) {
    int x=analogRead(Channel);
    // clip in [0..999]
    x=(x<=12)?0:(x-12);
    Data[Channel][b] = (x>999)?999:x;
  }
  full[b]=true;
  b=b~1; // toggle 0<->1 buffer
  digitalWrite(PROBEPIN1,LOW);
}
```

Listing 1. The function triggered every 4 ms on the Arduino MicroController Unit (MCU). Each element in `Data[i]` is a buffer for six ECG values. There are two such elements to implement a two positions buffer. `full` is a 0/1 switch to rotate the buffers, and to mark when the buffer is available. The probe pin level allows inspection of sketch operation. The complete sketch is available from the repository [24].

4.3. MCU to HTTP User Agent Interface

The unit communicates with the MCU dedicated to networking using the embedded UART running at 115,200 baud. They exchange 250 frames per second, each containing 33 8-bit characters as seen above, so that the resulting data rate is of $33 \times 250 = 8250$ octets per second. Serial protocol overhead is limited to start/stop bits: since communication is reliable and real-time there is no need for parity check and flow control. Thus, the required baudrate is of 82,500 baud, still comparable with that anticipated in Section 3, and the protocol is ready to transfer six samples—more than required—and a timestamp. In theory, such a baudrate is consistent with UART configuration: in Section 5, we verify this statement in practice.

4.4. Patient Side HTTP User-Agent and WiFi Interface

We used a ESP12e board (Espressif Systems, Shanghai, China) to implement the two functions. We plugged the board, which provides a 32-bit MCU with a WiFi interface (ESP8266ex), into a *Protoboard*

Arduino shield stacked on top of the other two boards (see Figure 5). We connected the power and serial lines using jumpers from Arduino pins to the ESP12e ones on the breadboard.

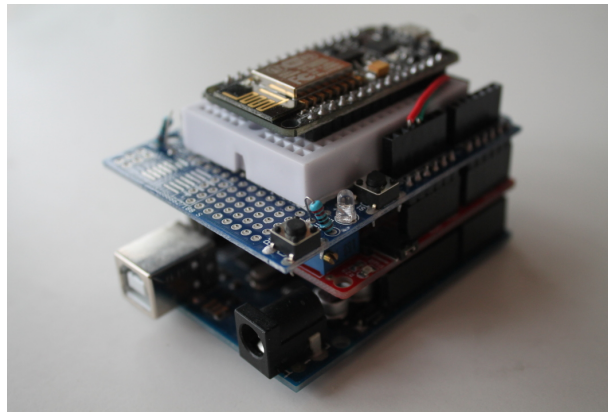


Figure 5. The stack of three Arduino-compliant boards: below the Arduino, the Olimex acquisition board in the middle, the protoboard hosting the ESP12e on the top

Using the Arduino IDE, we wrote a UA that opens a WebSocket connection with the server using *Links2004* WebSocket library functions. When the WebSocket is opened, the UA starts sending frames containing the data received on the serial line. Each frame is encoded in JSON as an array of objects, one for each data frame received from Arduino, with a timestamp and room for six floating point values (only the first one being finally visualized). The snippet in Listing 2 shows the loop that prepares the buffer and delivers it through the WebSocket.

```
while (true) {
  JSONBuffer[0]='['; // Open an array
  ptr=1;
  for (int n=0; n<N; n++) {
    digitalWrite(PROBEPIN2,HIGH); // probe
    if (fillBuffer()!=packetLength) continue;
    digitalWrite(PROBEPIN2,LOW);
    ptr=formatJSON(ptr);
  }
  JSONBuffer[ptr-1]=']'; // close the array
  digitalWrite(PROBEPIN1,HIGH); // probe
  websocket.sendTXT(JSONBuffer);
  digitalWrite(PROBEPIN1,LOW);
}
```

Listing 2. The endless loop delivering a frame to the server. Each frame contains N samples. Probe pins allow to inspect operation. The complete sketch is available from the repository [24].

The JSON array has a variable length in the interval [740,750] bytes. The frame is sent to the server through the WebSocket as a unique TCP segment. Since the length of a single IP packet would exceed the standard length—notably, 576 octets—each frame is divided into two IP packets that sum up from 849 to 857 octets: in theory, the generated Internet traffic is of 21.2 KB/s.

Note that, given a packet size around 850 octets, the 14 octets difference between the length of CoAP and HTTP headers anticipated in Section 3 is negligible (<2%).

4.5. The Server

The server for our experiment is a container (or *dyno*) in the Heroku cloud. For the sake of simplicity, the prototype envisions one single container hosting several ECG services, instead of a hub deploying short-lived micro-services upon request.

The server is implemented using the *Python/Flask* micro-framework and the *gunicorn* WSGI HTTP server. It provides four families of endpoints:

- `/` which returns a introductory page,
- `/in/<id>` used by patient's UA WebSocket,
- `/<id>` used by doctor's browser to open the ECG visualization page,
- `/out/<id>` used as endpoint for doctor's WebSocket,

where `<id>` stands for the unique identifier of the sensor's device.

The server waits for a patient's request at an `/in/<id>` endpoint. When the request arrives, the server upgrades the session to a WebSocket. Then, it records patient's identifier in a dictionary and prepares to receive a request with the corresponding `<id>` from doctor's side. Until the occurrence of such event, all ECG samples from patient's side are discarded. If doctor's request arrives first, the server returns a "Retry later" reply.

When the server receives doctor's request on a `/<id>` path matching patient's identifier, it returns to the doctor's browser a response containing the JavaScript code that opens the WebSocket on `/out/<id>` and displays the ECG. After that point, the server implements a buffer for patient's data frames.

Since each patient UA has a different identifier, a single server may host several ECG services at the same time, each of which is received by only one doctor UA.

The security mechanisms announced in Section 3 are not fully implemented: namely, the server is persistent and communication uses plain HTTP. However, the design is ready for HTTPS security.

We maximized share-ability using a free Heroku plan; this entailed some limitations (e.g., absence of HTTPS support) but was more consistent with our goal.

4.6. The Display UA

The UA on doctor's side is a web browser running on a laptop PC. The doctor opens an HTTP session with the server using an `/<id>` endpoint: if the device of the corresponding patient is already connected, the response contains a JavaScript application that opens a WebSocket on the `/in/<id>` endpoint, and displays the familiar canvas using the `Chart.js` library. The ECG trace is dynamically updated in *roll-mode* using data received across the WebSocket. A hold function is available.

Besides the graphical display, the JavaScript application performs a limited analysis of the ECG trace: it applies a simple numerical filter to remove the 50 Hz power-line noise using a moving average of five values and finds R peaks to compute heartbeat frequency. This demonstrates the possibility of additional filtering and feature extraction on doctor's side, which is compatible with end-to-end encryption.

We used a laptop with a *Core i5 3230M* CPU (2012) using Ubuntu Linux as an operating system and Chromium as browser.

5. Experimental Results

We obtained the trace in Figure 1 with both doctor's and patient's devices in Italy and using a Heroku server in Europe (Ireland, from IP geo-location). Patient's device and doctor's laptop are located in a residential zone of a medium-sized town and are served by a home-grade WiFi router. The ECG is obtained using the simplest placement of electrodes: two bracelets on wrists and the ground electrode held tight on the thumb.

On the left side of Figure 1, we see a timestamp and the corresponding count of lost samples. The loss rate shown in the figure is 3.4% after more than four minutes of operation; such loss rate is stable in the long term.

Regarding autonomy, a patient’s device consumes circa 200 mA (see Figure 6) with short spikes that reach 350 mA during transmission. In fact, we observed a continuous operation of more than six hours with the (nominal) 1.4 Ah 5 V *power bank* shown in Figure 3: a ten hours autonomy requires a capacity of at least 2.4 Ah at 5 V, or 12 Wh, well below the threshold value found in Section 3.

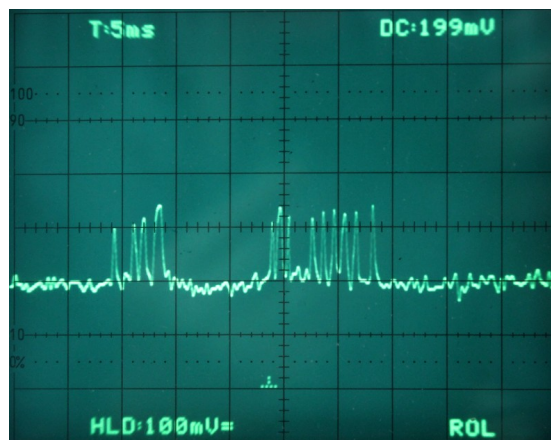


Figure 6. Power consumption of a patient’s device. To obtain the trace, we measured voltage drop on a 1 Ohm shunt resistor in series with batteries: 1 mV on the trace corresponds to 1 mA.

To understand the bottlenecks in our architecture, we individually tested its modules and communication links: the following is a synthesis of the results.

5.1. Data Acquisition

We instrumented the code (see Listing 1) so that the levels of two defined ports indicate respectively data input and serial output: the aim is to visualize every interference between such activities in the Arduino MCU. Figure 7 shows the output of the two pins as displayed on a 2-traces oscilloscope: the lower trace represents the timing of data input from the ADC, while the upper trace shows data output on the serial line. There is no interference between the two operations, and their timing is very regular, driven by the timer-based interrupt. The envelope of acquisition cycles (not shown) is within the 0.1 ms. There is an idle time of 2 ms between the end of data delivery, and the next data acquisition interval. Data preparation, after acquisition and before delivery, takes 0.8 ms. We conclude that there is no interference, and that the Arduino CPU is idle approx. 50% of the time.

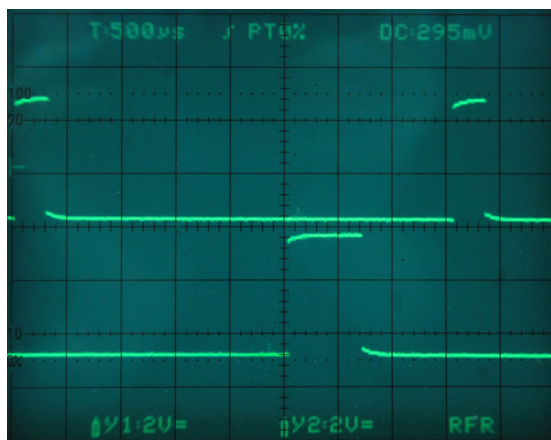


Figure 7. Timing of the acquisition of a single ECG sample (lower trace high period) and of its delivery to the UART (upper trace high period) on the Arduino.

5.2. Data Frame Construction and Delivery

We instrumented the code running on the ESP12e board in order to inspect the preparation of the JSON array containing the samples and its delivery to the WebSocket interface. To this end, one of the GPIO pins is set high during packet delivery (see Listing 2). In Figure 8, the lower trace represents the data acquisition activity of Arduino (the same as the lower trace in Figure 7, with a different time scale), while the upper trace represents packet delivery on the WebSocket interface. The figure shows a cycle of ten data acquisitions on the Arduino (lower trace), the data of which are delivered as a unique JSON object with a single WebSocket operation.

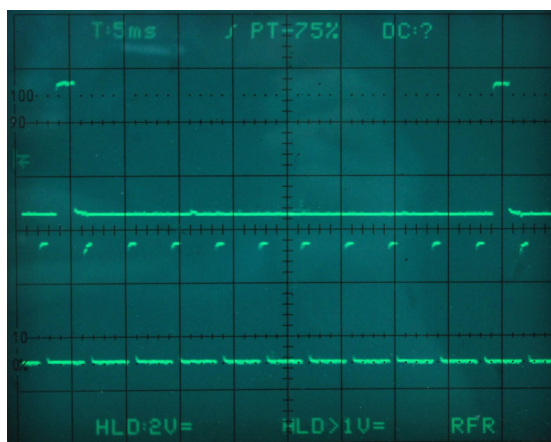


Figure 8. Timing of a series of 10 data acquisition operations (lower trace high periods) and the delivery of the packet through the WebSocket (upper trace high periods). The lower trace is the same as Figure 7, the other is high during network operation of the ESP12e.

The trace focuses on network operation and disregards the activity of the ESP12e serial interface connected with the Arduino: while receiving data, the ESP12e is idle most of the time waiting for data from the Arduino. The relevant events occur just before the pulse in the upper trace when the ESP12e has completed the JSON-encoded frame and delivers it to the WebSocket interface. Figure 9 is an enlarged display of Figure 8 during such transient: it focuses on the last acquisitions before a WebSocket send event. We have evidence—not reported in this paper—that the ESP12e acquires the last data of the frame during

the lapse corresponding to the leftmost pulse in the lower trace in the figure. The successive time before packet delivery (2.3 ms) is spent by the ESP12e receiving the last data item across the serial line. Thus, the data produced by the following acquisition will be temporarily buffered in the UART, and delivered when the WebSocket send call terminates.

The ESP12e is busy almost 100% of the time, but most of the time is spent idling for the next character from the serial line. In this way, it can tolerate WebSocket transmission delays, and there is no interference between operations.

To summarize the timing of data acquisition and transmission, a data frame is *on air* 8 ms after completion, and carries information for 40 ms of ECG, with an overall latency of approximately 50 ms.

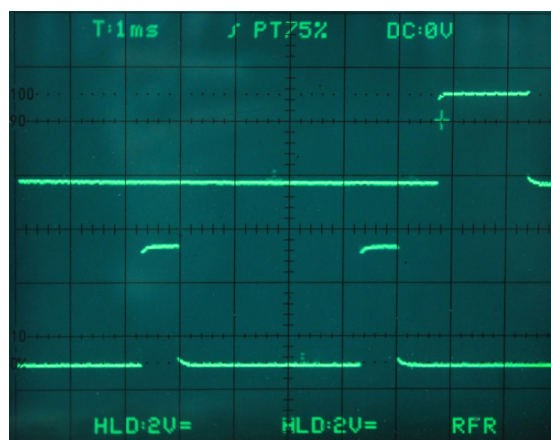


Figure 9. The lower trace shows the last data acquisition operation in Figure 8 (first pulse on the left) in a sequence of ten, the upper one shows the time spent sending a packet on the WebSocket.

5.3. Server and Network Infrastructure

To analyze the timing of data delivery from patient’s sensor to doctor’s display, we split it into two components. One, outside our control, is related to the performance of the Wide Area Network (WAN) between the relay server and the edge devices. The other is related to the timing of the pipeline made by the three components in our design.

To evaluate the latter component, i.e., including the WebSocket-based pipeline but disregarding WAN links, we prepared a laptop hosting both the WebSocket relay server and doctor’s interface. Next, we attached this device to the same WiFi AP of patient’s sensor (see Figure 10). The distance between the ECG sensor and the AP is below ten meters; they do not need to be in their line of sight, but solid walls may reduce the quality of the ECG. To obtain a report about the performance of the whole pipeline, we instrumented the code of the Javascript application to measure transfer/loss rates. We observed a 0% loss rate, meaning that there is no harmful bottleneck; in particular, the ESP12e WebSocket performs timely. An even more relevant result since the pipeline transports the six data needed by a diagnostic ECG, although our test considers only one of them.

The test configuration with WAN communication is shown in Figure 2 and introduces a remote Heroku dyno in Europe that implements the relay server. According to IP geolocation services, the server is 1500 km away from both the sensor and the display. The results are still good for this configuration since the observed loss rate of circa 4% does not affect the readability of the trace, which is the one shown in Figure 1.

The results drastically degrade when using a US server. In that case, the loss rate rises to 40% and the ECG is no longer readable.

A similar result is obtained using a tethering smartphone instead of the AP on patient’s side: with the relay server in Europe, the loss rate was 36%.

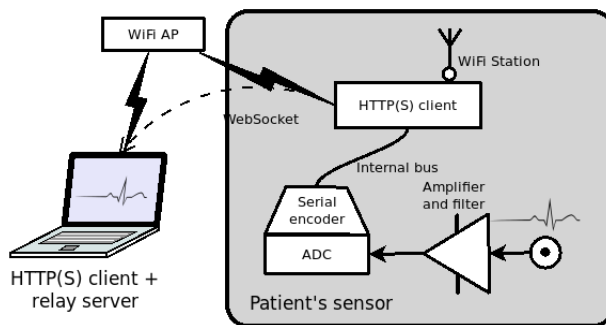


Figure 10. Test setup to measure the processing pipeline in isolation, independent from Wide Area Network (WAN) performance.

5.4. The Display

The display covers five seconds of ECG, and the trace dynamically rolls from left to right. The nominal refresh rate, which corresponds to the frequency with which the sensor generates data frames, is 25 refresh events per second.

The configuration using the local by-pass server is useful to test the JavaScript application: given that there is no data loss, the browser has to display the ECG at full speed. In that case, we observe a limited stuttering that is probably due to the browser.

In the case of moderate (4%) data loss, the display may stop the visualization and later resumes consistently.

When data loss approaches 40% the trace is frequently unreadable.

Regarding transmission delays, they should be limited to allow the doctor to instruct the patient about electrodes positioning, for instance on a phone call while taking the ECG. In normal conditions, latency is unnoticeable, and has been measured around 0.1 s with the European server; however, it strongly depends on the quality of the Internet connection.

We added to the display application three utility functions only to show that our design allows ECG processing on doctor’s side:

- a moving average filter that significantly attenuates the residual 50 Hz noise,
- the dynamic evaluation of the heartbeat rate,
- the possibility to put on hold the screen to analyze the ECG.

6. Conclusions and Future Work

Commercial and research products exist that provide remote ECG services, but the design we present in this paper is distinguished for two tightly related reasons.

One is that we focus on *low cost*: overall, the retail price of the hardware we used to realize a development-oriented prototype is around 50\$. A design that aims at a limited deployment of production-oriented devices can significantly cut such cost, so that the realization of a remote ECG service becomes affordable also for small communities, which are those that may take the most from this kind of service. A low entry cost may also attract the interest of small enterprises.

The other reason is that we privilege *shareability* respect to other technical aspects. For this, all parts of the design are open-source so that there are no royalties to pay to put them into production. Since the interfaces among components are also open, it is possible to produce or improve just one of them and

integrate it into an existing deployment; this opens a market to all those that share the same open design. The name of this is interoperability, and, in the past, it has been a key for the take-off of many technologies.

Focusing on the performance of our prototype, there are markets that might be addressed today with this service. Looking at the REHEARSE-AF study, we see that the results we obtain fit a similar scenario: *home health-care* in a residential area of a developed country. The ubiquitous diffusion of home WiFi routers with wide bandwidth availability eliminates the Internet bottleneck. Local administrations might even provide a local server, or use a local hosting service. Unlike the devices used in REHEARSE-AF, those of our prototype are open and low-cost, which makes a difference.

Alone, the home health-care market addresses millions of potential users. It is much more extended than others that are technically more challenging, like those that follow.

The road rescue service described in the introduction has a much more limited market, but an impact that may be relevant as a life-saving technology. Our solution can't help in similar scenarios since it depends on *WiFi coverage* and experimental results prove that tethering is ineffective. However, there are viable trade-offs for that: for instance, we might cope with bad connectivity by introducing a buffer in the server and accepting a significant latency. A slightly more complex design that we have not investigated.

Assuming sufficient outdoor connectivity, we may consider ambulatory monitoring, like during exercise. We have shown that, as is, the device is sufficiently lightweight and robust to be considered portable, and its autonomy may reach a day of continuous utilization, like that of a Holter monitor.

Even more challenging is the extension of the service to lands covered by constrained wireless networks, like Long Range (LoRa)-based networks [27]. It is the case of least developed countries, where the presence of a remote ECG would be precious to give qualified medical assistance to isolated communities. In such cases, we cannot expect a large market to open, so that experience gained with wealthier environments is precious to enable the design of devices that address challenging use cases with an affordable price.

In the short term, further results extending current work concern with its implementation, and consist of trading off simplicity against other parameters. The *cost* can be improved using non-Arduino compatible devices, at the same time obtaining a more compact and wearable device. *Power consumption* can be reduced avoiding redundant components and making use of power-saving features. Better data compression improves *performance in communication*.

Our point is that when services have a social impact—as in the case of healthcare—it is relevant that their design addresses low-cost, openness and standardization (OpLoC). Such guidelines ensure that the service will smoothly scale extending its benefits to more and more users, as demonstrated every day by the Internet.

The hardware and software design are available in a public BitBucket repository [24].

Funding: There is no funding external to the University of Pisa.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

WAN	Wide Area Network
CoAP	Constrained Application Protocol
ISP	Internet Service Provider
AF	Atrial Fibrillation
OpLoC	Open source, Low cost, Commercial
COTS	Commercial Off-the-shelf
ECG	electrocardiogram

UA	User Agent
FQDN	Fully Qualified Domain Name
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
URL	Uniform Resource Locator
ASCII	American Standard Code for Information Interchange
JSON	JavaScript Object Notation
WSGI	Web Server Gateway Interface
MCU	Micro-Controller Unit
SBC	Single Board Computer
PaaS	Platform as a Service
AP	Access Point
API	Application Programming Interface
WiFi	Wireless Fidelity
BAN	Body (or Personal) Data Network
MSN	Medical Sensor Network
PC	Personal Computer
AWS	Amazon Web Services
POP	Point Of Presence
NAT	Network Address Translation
TLS	Transport Level Security
ADC	Analog Digital Converter
UART	Universal Asynchronous Receiver-Transmitter
IP	Internet Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
LoRa	Long Range
IDE	Integrated Development Environment

References

- Halcox, J.P.; Wareham, K.; Cardew, A.; Gilmore, M.; Barry, J.P.; Phillips, C.; Gravenor, M.B. Assessment of Remote Heart Rhythm Sampling Using the AliveCor Heart Monitor to Screen for Atrial Fibrillation. *Circulation* **2017**, *136*, 1784–1794. [[CrossRef](#)] [[PubMed](#)]
- Haritha, C.; Ganesan, M.; Sumesh, E.P. A survey on modern trends in ECG noise removal techniques. In Proceedings of the 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Nagercoil, India, 18–19 March 2016; pp. 1–7. [[CrossRef](#)]
- Naik, G.R.; Reddy, K.A. A new model for ECG signal filtering and feature extraction. In Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 14–17 October 2016; pp. 765–768. [[CrossRef](#)]
- Gutierrez-Rivas, R.; Garcia, J.J.; Marnane, W.P.; Hernandez, A. Novel Real-Time Low-Complexity QRS Complex Detector Based on Adaptive Thresholding. *IEEE Sens. J.* **2015**, *15*, 6036–6043. [[CrossRef](#)]
- Lugovaya, T.S. Biometric human identification based on electrocardiogram. Master’s Thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University “LETI”, Saint-Petersburg, Russian, 2005. Available online: <https://physionet.org/physiobank/database/ecgiddb/biometric.shtml> (accessed on 23 April 2019).
- Sathya, D.; Kumar, P.G. Secured remote health monitoring system. *Healthc. Technol. Lett.* **2017**, *4*, 228–232. Available online: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5761311/> (accessed on 23 April 2019). [[CrossRef](#)] [[PubMed](#)]

7. Baig, M.M.; GholamHosseini, H.; Moqem, A.A.; Mirza, F.; Lindén, M. A Systematic Review of Wearable Patient Monitoring Systems—Current Challenges and Opportunities for Clinical Adoption. *J. Med. Syst.* **2017**, *41*, 115. [CrossRef] [PubMed]
8. Jones, V.; Bults, R.; Konstantas, D.; Vierhout, P. Healthcare PANs: Personal Area Networks for trauma care and home care. In Proceedings of the Fourth International Symposium on Wireless Personal Multimedia Communications, Aalborg, Denmark, 9–12 September 2001; Volume 3, pp. 1369–1374. Available online: <https://research.utwente.nl/files/6141897/paper51withheaderpage.pdf> (accessed on 23 April 2019).
9. Chen, M.; Gonzalez, S.; Vasilakos, A.; Cao, H.; Leung, V.C.M. Body Area Networks: A Survey. *Mob. Netw. Appl.* **2011**, *16*, 171–193. [CrossRef]
10. Shnayder, V.; Chen, B.R.; Lorincz, K.; Fulford-Jones, T.R.F.; Welsh, M. *Sensor Networks for Medical Care*; Technical Report TR-08-05; Harvard Computer Science Group: Cambridge, MA, USA, 2005. Available online: <http://ftp.deas.harvard.edu/techreports/tr-08-05.pdf> (accessed on 23 April 2019).
11. Plesnik, E.; Malgina, O.; Tasič, J.F.; Zajc, M. ECG signal acquisition and analysis for telemonitoring. In Proceedings of the Melecon 2010—2010 15th IEEE Mediterranean Electrotechnical Conference, Valletta, Malta, 26–28 April 2010; pp. 1350–1355. [CrossRef]
12. Cristea, C.; Pasarica, A.; Andrusac, G.; Dionisie, B.; Rotariu, C. A wireless ECG acquisition device for remote monitoring of heart rate and arrhythmia detection. In Proceedings of the 2015 E-Health and Bioengineering Conference (EHB), Iasi, Romania, 19–21 November 2015; pp. 1–4. [CrossRef]
13. Xia, H.; Asif, I.; Zhao, X. Cloud-ECG for real time ECG monitoring and analysis. *Comput. Methods Programs Biomed.* **2013**, *110*, 253–259. [CrossRef] [PubMed]
14. Page, A.; Kocabas, O.; Soyata, T.; Aktas, M.; Couderc, J.P. Cloud-Based Privacy-Preserving Remote ECG Monitoring and Surveillance. *Ann. Noninvasive Electrocardiol.* **2014**, *20*, 328–337. [CrossRef] [PubMed]
15. Welton, N.J.; McAleenan, A.; Thom, H.H.; Davies, P.; Hollingworth, W.; Higgins, J.P.; Okoli, G.; Sterne, J.A.; Feder, G.; Eaton, D.; et al. Screening strategies for atrial fibrillation: A systematic review and cost-effectiveness analysis. *Health Technol. Assess.* **2017**, *21*, 1–236. [CrossRef] [PubMed]
16. Walker, B.A.; Khandoker, A.H.; Black, J. Low cost ECG monitor for developing countries. In Proceedings of the 2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, 7–10 December 2009; pp. 195–199. [CrossRef]
17. Deb, S.; Islam, S.M.R.; RobaiatMou, J.; Islam, M.T. Design and implementation of low cost ECG monitoring system for the patient using smart device. In Proceedings of the 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE), Cox’s Bazar, Bangladesh, 16–18 February 2017; pp. 774–778. [CrossRef]
18. Guo, S.L.; Han, L.N.; Liu, H.W.; Si, Q.J.; Kong, D.F.; Guo1, F.S. The future of remote ECG monitoring systems. *J. Geriatric Cardiol.* **2016**. [CrossRef]
19. Ciuffoletti, A. Low-cost IoT: A holistic approach. *J. Sens. Actuator Netw.* **2018**, *7*, 19. [CrossRef]
20. Fette, I.; Melnikov, A. The WebSocket Protocol. RFC 6455, RFC Editor, 2011. Available online: <http://www.rfc-editor.org/rfc/rfc6455.txt> (accessed on 23 April 2019).
21. Ciuffoletti, A. On-line Remote EKG as a Web Service. *arXiv* **2019**, arXiv:1901.00724. Available online: <https://arxiv.org/abs/1901.00724> (accessed on 23 April 2019).
22. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor, 2014. Available online: <http://www.rfc-editor.org/rfc/rfc7252.txt> (accessed on 23 April 2019).
23. Newman, S. *Building Microservices: Designing Fine-Grained Systems*; O’Reilly Media: Sebastopol, CA, USA, 2015. Available online: <http://shop.oreilly.com/product/0636920033158.do> (accessed on 23 April 2019).
24. Ciuffoletti, A. Prototype of an Open Remote ECG Service. 2018. Available online: https://bitbucket.org/augusto_ciuffoletti/prototype-of-an-open-remote-ecg-service (accessed on 23 April 2019).
25. SHIELD EKG-EMG REV.B. Available online: <https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/resources/SHIELD-EKG-EMG-REV-B-SCHEMATIC.pdf> (accessed on 23 April 2019).

26. Pizzuti, G.; Cifaldi, S.; Nolfo, G. Digital sampling rate and ECG analysis. *J. Biomed. Eng.* **1985**, *7*, 247–250. [[CrossRef](#)]
27. Mdhaffar, A.; Chaari, T.; Larbi, K.; Jmaiel, M.; Freisleben, B. IoT-based health monitoring via LoRaWAN. In Proceedings of the IEEE EUROCON 2017-17th International Conference on Smart Technologies, Ohrid, Macedonia, 6–8 July 2017; pp. 519–524, [[CrossRef](#)]



© 2019 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).