TrainAround: Monitoring Athletes Workout Using a BLE Star Topology Network

Tommaso Giorgi, Enrico Nello, Giacomo Pacini, Matteo Pierucci
t.giorgi@studenti.unipi.it , e.nello@studenti.unipi.it, g.pacini14@studenti.unipi.it , m.pierucci5@studenti.unipi.it

ABSTRACT

The use of smartphones and wearables as sensing devices has created innumerable context inference apps including a class of workout tracking apps. Workout data generated by mobile tracking apps can assist users in achieving better health care, rehabilitation, and self-motivation. Previous approaches impose extra burdens on users by requiring users to bring their phones with them while training and during workouts.
In this paper, we propose a practical end-to-end workout tracking application called TrainAround, that simplify the way of monitoring the workout sessions by letting either the coach to gather information about all his/her athletes while they are training and the athletes to be tracked only wearing a smartwatch.
TrainAround leverages a new generation of Android smartwatches such as the TicWatch E2, which benefit from powerful hardware resources, Bluetooth Low Energy radios, and a rich set of sensors. Our application is able to efficiently track cardio activities.

1 Introduction

The most common approach nowadays for workout tracking consists in Wearable apps that run directly on the wearable device and let you access the device's services all on the device itself.
Fitness apps are designed specifically to assist with exercise, other types of physical training, or related fitness topics. The purpose of those fitness app is to provide the user with instructions and examples of one or more types of exercise, physical activity, record statistics on workouts or collect data on walks, or some other fitness topic.
Anyway, all these applications suffer from the lack of possibility for an external trainer / coach to monitor in real time the workout sessions of multiple athletes, instead of just one.
Most of the time, in real life sport-scenarios, it would be interesting to change the perspective from 1:1 (Athlete self-checking his/her own statistics) to One-to-Many (Trainer supervising his/her Athletes' statistics). Also, for those apps that lets users to connect to a personal trainer to share statistics about fitness routine, the delivery of the latter is asynchronous. Thus, they miss an on-the-fly monitoring feature that could let the trainer to track more effectively the session.
In addition to that, existing workout tracking apps running on smartphones cannot accurately sense user activities when the phone is placed away from the user, or just become unhandy if they require the phone to be attached to the body.

We instead developed an application that collects data from multiple Wear OS smartwatches and shows the collected information from all the smartwatches on one central Android device, which is supposed to be the coach / trainer device.
The aim is to find a lightweight solution that does not need the presence of the smartwatches' paired phones, so that the monitored users do not have to bring their phones with them.
Smartwatches are way less intrusive since most users wear them for the majority of the day. Whereas smartphones, typically carried in pockets, can only capture body postures.
Therefore, TrainAround uses a single smartwatch to replace smartphones and other sensors previously used for workout tracking.
The other requirement is to avoid the use of internet, so that the application could work even in remote areas or where there is no Wi-Fi hotspot for the smartwatches.
So, from a technical point of view our work consists in creating a local area network between one android device (acting as a server / central node) and multiple Android Wear OS devices acting as clients / sensors.
TrainAround was thought to be addressed for the track and field world, where the presence of small groups of athletes and their coaches would actually take advantages from using it.
In this paper we will present all the most recent and state of the art approaches we went through while seeking for the most suitable technology for creating the overlay network.

2 Challenges and Design Choices

We start off by considering the preliminary work we made on the available literature to understand which would have been the best choice for building the network infrastructure.
As stated above, our aim was to develop a star topology local network in which the central node is intended to be the trainer's device and the wearables as peripheral devices.
We tried several approaches following a priority list of low-power communication technologies, from the most suitable one to the least:

1. Google Nearby Connection API
2. Bluetooth Low Energy
3. Wi-Fi Direct

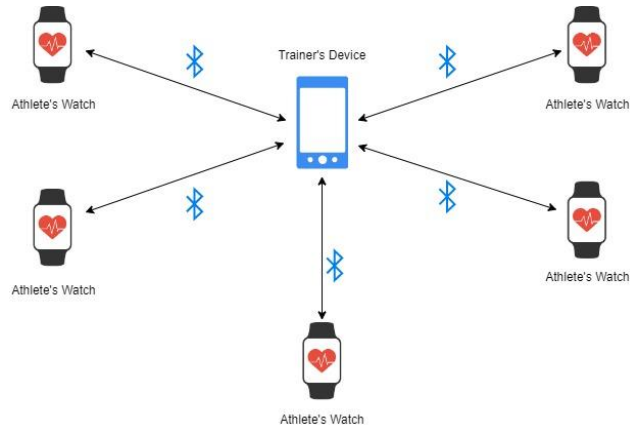
We began considering the Google Nearby Connections API [1] for connecting phone and wearables, but after first attempts we discovered that it is still not supported on Wearable devices.
Then, we tried to seek for solutions concerning BLE. We were able to find an experimental BLE mesh network project by some

students of La Sapienza [2], but it seemed to be not definitive and hard to understand and we could not get it to work with more than two devices, so we discarded that option.
Next, we tried out the BLE Nordic Semiconductors library [3], with which we were able to connect four cell phones to each other and a watch.
Nevertheless, in the example provided by this library, the application allowed communication between devices associated with each other (already connected previously); therefore, it required the need to format the clocks that one wants to connect every time (in fact a watch can only be paired with one smartphone at a time and to change paired smartphone it must be formatted). This was something unfeasible for our application to perform, as the athletes should have format their personal watch to get paired with the trainer's device.
Another flaw with that library was that no documentation appeared to be available, or at least lacking.
Finally, we tried the Wi-Fi Direct option, but like Nearby Connection it was not compatible with the smartwatches.

Finally, we ended up implementing a BLE GATT Server and Client from scratch, following the examples provided by official Android API for BLE [4]. In fact, this was the only way to go in order to build our custom One-to-Many network.

Figure 1: TrainAround is addressed to track & field athletes

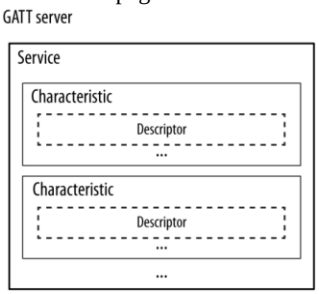
3 System Architecture



Bluetooth Low Energy devices use the Generic Attribute Profile (GATT) to discover and send data. Specifically, on the trainer's android device acting as *Central* entity, we developed the GATT Server. Instead, on the wearable devices of the athletes, acting as *Peripheral* entities we developed the GATT Client.
For BLE-enabled devices (wearables) to transmit data between each other, they must first form a channel of communication.
So, in TrainAround as soon as Bluetooth module is available and permissions are granted, the watches will scan for a nearby GATT server. Once a connection is made, capabilities are discovered, and data can be transferred with the connected device based on the available services and characteristics.

3.1 GATT Server

According to the official documentation, a GATT Server is a device that scans for and connects to BLE peripherals in order to perform some operation.
In the context of our app development, this corresponds to the trainer's Android device.
When TrainAround starts, the GATT server prompts, exposes all the corresponding services and characteristics, starts advertising and waits for incoming connections from clients.
Once a connection is made, the application show the newly connected client in the main page.



3.1.1 Services

A GATT service corresponds to a collection of characteristics (data fields) that describes a feature of a device.
In our case, we choose to implement three primary services:

- **AthleteInformationService**: services that is in charge of letting a client to set and retrieve its name.
- **HeartRateService:** service that oversees the heart rate updates from the clients to the server.
- **MovementService**: service that oversees the speed, pace, step counter, activity recognition and distance updates from the client to the server.

3.1.2 Characteristics

A GATT Characteristic corresponds to an entity containing meaningful data that can typically be read from or written to. In other words, a characteristic can be thought of as a type.

In our case, we implemented the following characteristics:

Service	Characteristics	Type
AthleteInformationService	AthleteNameCharacteristic	Read, Write
HeartRateService	AthleteHeartRateCharacteristic	Write
MovementService	AthleteActivityCharacteristic	Write
	AthleteSpeedCharacteristic	
	AthletePeaceCharacteristic	
	AthleteStepCounterCharacteristic	
	AthleteDistanceCharacteristic	

3.2 GATT Client

According to the official documentation, a GATT Client is a device that advertises its presence and is connected to by a central in order to accomplish some tasks.
In the context of our app development, this corresponds to the wearables. When TrainAround starts, the discovery service begins. As soon as the GATT server is found and the connection established, the application sends small data packages by writing onto one of the exposed BLE characteristic, whenever a new sensor value is registered.

4 Workout Monitoring

4.1 Sensor Choice

The Android platform provides several sensors that has let us monitor the motion of the wearable device.
Most Android-powered devices have built-in sensors that measure motion, orientation, and various other conditions. These sensors are capable of providing raw data with high precision and accuracy and are useful for monitoring three-dimensional device movement or positioning.
For our proposes, we selected the following high-level statistics to monitor the aerobic activity of the athletes:

- Heart Rate
- Step counter
- GPS (speed, distance)
- Recognized Activity

We then perform some computation in order to get the Pace of the athletes. It can be defined as how fast an athlete is running, and it is typically expressed as the average amount of time it takes for the athlete to run one km during a longer run. The heart rate can help both athletes and trainers adjusting the workout intensity to achieve maximum health benefits.

5 Efficient Resource Usage

In contrast to classic Bluetooth, BLE is designed for significantly lower power consumption. This allows apps to communicate with BLE devices that have stricter power requirements, such as proximity sensors, heart rate monitors, and fitness devices like in our case.
It enables long emission range of up to thirty meters, so it covers most of real-life track and field scenarios, where trainer and athletes will not likely be more than a thirty away from each other.
Moreover, it is optimized to sends small amount of data, and this technical specificity makes it possible, among other things, to considerably extend the autonomy of connected objects by reducing their activity time. In this way, Bluetooth Low Energy is highly suitable for our training session monitoring purposes.
Indeed, the majority of our use cases require neither a high emission recurrence, nor the transmission of a large amount of information.
In addition, BLE is supported onto the wearable devices, whereas other technologies we tried were not (e.g., Wi-Fi Direct).

6 Context-Aware Optimization

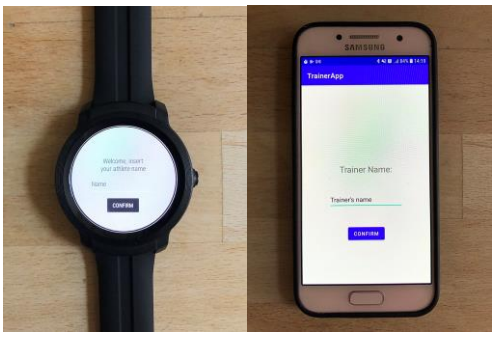
Mobile devices such as smartphones and wearables have become ubiquitous. Therefore, they operate in a dynamically changing context, both in terms of user needs and computational requirements. Context has been defined as:

"*any information that can be used to characterize the situation of an entity*"

and in mobile devices, it is obtained from sensors, or locally available data.
Most of the Android applications do not implement any self-adaptive strategies that react to the battery level, status and context. Thus, the applications continue to consume power even when battery is critically low.
This is an issue that we need to care about as we deal with wearables, that are energy-constrained devices.
We then decided to make TrainAround aware of the context by letting the client-side of the app to stop some energy hungry sensor (like GPS, for estimating speed and distance) whenever the user's recognized activity is set to *Still*. The activity recognition precision strictly depends on the device used for testing, both for timing and accuracy, so we decided to include another context-aware optimization in order to resume sensors even if the activity remains "*Still"* and it is waiting to be updated. First of all, we recorded the step counter value when activity has become "*Still*" and then we resumed these sensors when the new step counter value will reach 50 steps after the recorded value. In this way it is very likely that the user has already started walking or running and so it is important to resume that sensor before the activity recognition update.
So, we basically exploit the information of the user's state provided by the Activity Recognition for triggering the execution of GPS.

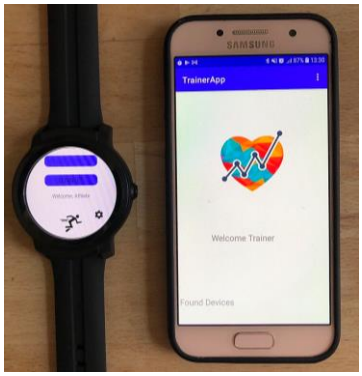
7 Application Demo

1. When application first start, it asks for the device name both on the client and server side:

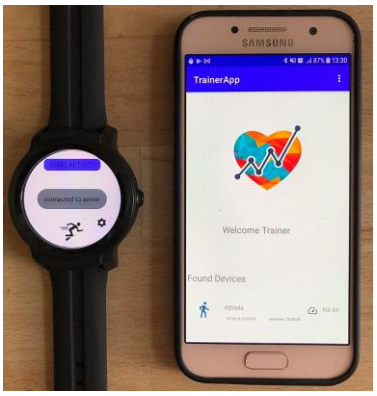


2. The client application then shows the *Start Scanning* button, to start the discovery for the GATT server:

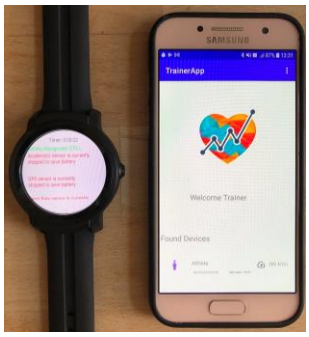
3. Once the server has been successfully discovered, onto the client side will appear the *Connect* button:



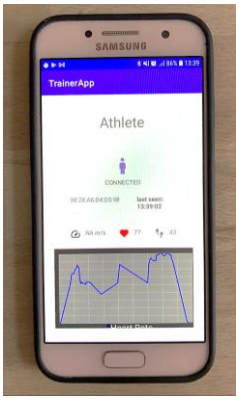
4. Once the connection between GATT server and GATT client is successful, the Found Device list on the server side will update accordingly:



5. Now the client can start the training session real time monitoring by simply clicking onto the *Start Activity* button:



6. The trainer on his / her device can select one specific athlete that wants to monitor, and the application will show a dedicated page with all the real-time statistics:

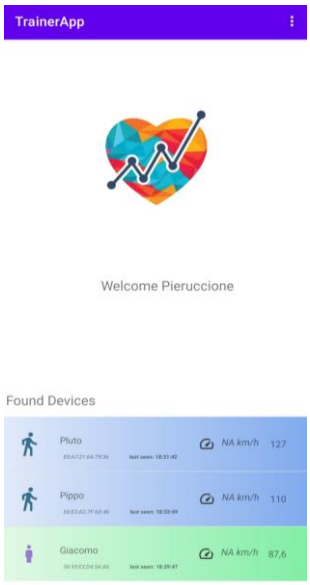


8 Experimental Results

We performed several tests with our wearable devices:

- Galaxy Watch 4
- TicWatch Pro S
- TicWatch E2

We were able to track each three of them simultaneously with a Samsung Galaxy A3 device running the server instance:



The athletes list on the trainer's application are sorted following the heart rate measures.

Regarding the maximum distance, from the tests we were able to monitor the clients up to 30 meters in open field.

9 Conclusion

TrainAround is a workout tracking application that changes the perspective of the training sessions, both for the athletes and the trainers. The biggest innovation consists of letting the monitoring of multiple devices simultaneously, instead of having to cope with several 1:1 connection.

Our project is mainly focused on structuring the network infrastructure for letting the devices to communicate between each other, and the final goal is to provide a way to monitor in real-time some parameters. That is the main reason why we were not able to find other significant aggregate data to provide to the trainer's side.

It also is important to mention that the accuracy of the chosen sensors is not reliable a hundred percent of the time, and sometimes could lead to some evaluation errors.

Discussing about future possible enhancements, we could get deeper into the monitoring phase, improving the techniques to achieve more usable data aggregation and statistics about the sessions. In addiction to that, it would be useful for the trainer to have a summary of the past training sessions.

The ultimate goal would be to transform our network into a BLE mesh; doing so, we would overcome the issue related to the maximum number of connectable devices, that at the moment depends on the hardware of the device acting as GATT server.

Another aspect to consider is about the data privacy; using smartwatches as sensing devices for context inference apps can create new sources of privacy leakage.

Personal workout history detected by TrainAround is normally considered as sensitive information of a user. Without proper access control, such information could be exploited maliciously by third-party apps.

REFERENCES

[1] https://developers.google.com/nearby/connections/android/get-started
[2] https://github.com/netlab-sapienza/android-ble-mesh
[3] https://github.com/NordicSemiconductor/Android-BLE-Library/
[4] https://github.com/android/connectivity-samples