ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

ASYNCHRONOUS CONTACT TRACING, FIGHTING PANDEMICS WITH INTERNET OF THINGS

Relatore:
Chiar.mo Prof.
DAVIDE SANGIORGI

Presentata da:
ALESSIO DI DIO

Correlatore:
LUIGI LIQUORI
ENRICO SCARRONE

Sessione II
Anno Accademico 2021/2022

Abstract

La pandemia da COVID-19 ha cambiato le nostre vite obbligandoci a vivere mesi di lockdown, distanziamento sociale ed uso delle mascherine. Il distanziamento sociale e l'uso delle mascherine, anche dopo la prima fase della pandemia, sono state le contromisure principali in quanto permettevano di limitare i contagi permettendo comunque alla gente di uscire di casa. Tutte queste contromisure hanno creato gravi danni all'economia del paese e alla vita personale dei cittadini. Dalla fase iniziale della pandemia si è capito che per gestirla al meglio era necessario effettuare il numero maggiore di tamponi possibili per monitorare al meglio la diffusione del virus ma ciò non era possibile in quanto non esistevano le tecnologie necessarie per testare milioni di persone al giorno. Da questa necessità sono nati i sistemi di Contact Tracing, sistemi che permettono di monitorare in modo anonimo e protetto i contatti sociali delle persone così da capire se sono entrate in contatto con persone infette dal COVID-19 e solo in quel caso effettuare un tampone in modo tale da verificare se sono stati contagiati o meno. Questi sistemi permettono di ridurre il numero di tamponi effettuati perchè il controllo è mirato ai soggetti con rischio maggiore di essere contagiati. Tutti i sistemi di Contact tracing sviluppati ad oggi hanno mostrato problemi relativi alla protezione dei dati, alla scarsa ed inefficace comunicazione e non hanno ridotto al meglio il numero di tamponi effettuati per rilevare realmente coloro che erano stati contagiati avendo quindi uno scarso utilizzo soprattutto a causa della poca fiducia degli utenti riguardo l'utilizzo dei loro dati ed al fatto che dovevano autodichiararsi positivi. Con questa tesi presenterò una nuova tecnica per effettuare il Contact Tracing che combina l'utilizzo del Group Testing [41] all'utilizzo dell'IoT e delle reti per tracciare i contatti tra gli utenti ed il virus chiamata Asynchronous Contact Tracing [43]. La principale differenza tra questa tecnica e le altre è che non c'è un tracciamento dei contatti sociali tra due individui ma tra un individuo ed un luogo che potrebbe risultare contagioso a causa del fatto che sono presenti tracce del virus su oggetti/pavimenti/aria. Mostrerò come è stato progettato e sviluppato e mostrerò le performance grazie a degli esperimenti reali.

Contents

1 General Project Description **9**

1.1 Subject 9
1.2 Motivation 10
1.3 Challenges 11
1.4 Goals 11
1.5 Tasks 12
1.6 Prerequisites 17
1.6.1 oneM2M Standard 17
1.6.2 ICON© Platform by Telecom Italia Mobile (TIM) 19

2 State of the art **21**

2.1 ROBERT 22
2.2 DESIRE 25
2.3 DP-3T 28
2.4 GAEN 30
2.5 PEPP-PT 31
2.6 TCN 33
2.7 BlueTrace 34
2.8 PACT 35

3 Asynchronous Contact Tracing **37**

3.1 ACT Method Architecture and Functionalities 40
3.2 Information broadcasted by ACT Peripheral Service (and listened by ACT Smart Mobile Application) 42

3.3 Information exchanged between ACT Detection Service and ACT Local Service . . . 43
3.4 Information exchanged between ACT Local Service and ACT Control Service . . . 46
3.5 Information exchanged between ACT Smart Mobile Application and ACT Control Service . . . 47
3.6 Information exchanged between ACT Display Application and ACT Control Service . . . 50
3.7 Information exchanged between different ACT Control Services . . . 53
3.8 Indoor localization accuracy using WiFi without GPS . . . 55

4 Design **57**
4.1 Understand the subject, the state of the art, the goals, explore the platform 57
4.2 Finding the best WiFi Analyzer Mobile Application . . . 58
4.3 Study the WiFi Analyzer© and the Kotlin language . . . 58
4.4 Mockups development web and mobile . . . 59
4.5 Mobile application development . . . 59
4.6 oneM2M TIM ICON© interfacing and components development . . . 61
4.7 Web application development . . . 62

5 Implementation **63**
5.1 Mobile application development . . . 63
5.1.1 Description of the functions already present . . . 63
5.1.2 Extensions development . . . 65
5.2 oneM2M TIM ICON© interfacing and components development . . . 71
5.2.1 Detection Service development . . . 71
5.2.2 Local Service development . . . 72
5.2.3 Control Service development . . . 74
5.2.4 National Health Authority development . . . 77
5.3 Web application development . . . 78

6 Results **83**
6.1 Mobile application development 83
6.2 oneM2M TIM ICON© interfacing and components development 87
6.3 Web application development 88

7 Testing and Experiments **93**
7.1 Testing 93
7.2 Fieldwork Experiments 95
7.3 Recommendations 111

8 Conclusions **113**
8.1 Future works 113
8.2 Acknowledgements 114

List of Figures

1.1 Internship Gantt chart 13
1.2 Internship PERT chart 16

2.1 Use case example Courtesy of INRIA [39] 23

3.1 ACT Architecture Courtesy of ETSI Technical Specification TS 103757. SmartM2M; Asynchronous Contact Tracing System [43] 40

6.1 ACT Smart Mobile Application Splash screen 84
6.2 ACT Smart Mobile Application Main screen 85
6.3 ACT Smart Mobile Application Scanning running notification 85
6.4 ACT Smart Mobile Application Settings screen 86
6.5 ACT Smart Mobile Application COVID-19 exposure notification 87
6.6 Local Service TIM ICON© container instance 87
6.7 Detection Service TIM ICON© container instance 88
6.8 Control Service TIM ICON© container instance 88
6.9 ACT Display Application homepage 89
6.10 Filters section 89
6.11 Homepage showing cluster 90
6.12 Homepage showing markers inside cluster 91
6.13 Homepage showing marker's popup 92

7.1 Lagrange building, ground floor 96
7.2 Lagrange building, first floor 97
7.3 Lagrange building, second floor 98

7.4 Number of false positives based on the distance filter with 10 seconds of scan interval on Samsung A52 104
7.5 Number of false positives based on the distance filter with 20 seconds of scan interval on Samsung A52 104
7.6 Number of false positives based on the distance filter with 30 seconds of scan interval on Samsung A52 105
7.7 Number of false positives based on the distance filter with 10 seconds of scan interval on Umidigi F1 Play 106
7.8 Number of false positives based on the distance filter with 20 seconds of scan interval on Umidigi F1 Play 106
7.9 Number of false positives based on the distance filter with 30 seconds of scan interval on Umidigi F1 Play 107
7.10 False positive\Total network ratio based on distance filter with 10 seconds of Scan interval on Samsung A52 108
7.11 False positive\Total network ratio based on distance filter with 20 seconds of Scan interval on Samsung A52 108
7.12 False positive\Total network ratio based on distance filter with 30 seconds of Scan interval on Samsung A52 109
7.13 False positive\Total network ratio based on distance filter with 10 seconds of Scan interval on Umidigi F1 Play 110
7.14 False positive\Total network ratio based on distance filter with 20 seconds of Scan interval on Umidigi F1 Play 110
7.15 False positive\Total network ratio based on distance filter with 30 seconds of Scan interval on Umidigi F1 Play 111

Chapter 1

General Project Description

1.1 Subject

Contact tracing is a technique that allows you to define who a person has come into contact with. This technique has existed since the sixteenth century and has always been used to limit the spread of contagious diseases. In 2019 the world experienced the SARS-CoV-2 virus that led to the COVID-19 pandemic and, also in this case, the use of contact tracing was adopted to better manage the pandemic by limiting its spread as much as possible. Thanks to contact tracing it is therefore possible to trace contacts between people in such a way as to understand if a person has come into contact with the virus, all in a *synchronous* way. By *synchronous* we mean the fact that in order to trace the contact between two subjects it is necessary that the two are close enough for a certain period of time, however this use case does not consider the circumstances in which a subject is in an area in which the virus is also present on objects and therefore there is no real contact with people despite being there with the virus. In these circumstances it is necessary to use an *asynchronous* contact tracing technique where *asynchronous* means that there is no direct interaction between two people even though there is between a subject and the virus. This led to the birth of Asynchronous Contact Tracing (ACT) [43], a contact tracing technique that tracks human/object interactions using IoT-related technologies that understand if a user is in a certain area and if a virus is present in that area.

With regard to ACT, to date the protocol has been created and it has been standardized by the European Telecommunication Standard Institute (ETSI) [43] which technically describes the functioning of the latter. From a practical point of view, some components have been made [42] taking as baseline components the ETSI oneM2M standard for IoT:

- ACT Control Service: is a service that receives informations on users' movements and, interacting with the National Health Authority, informs users if they have gone to places where the virus was present.
- ACT Local Service: is a service that locally manages the status of a place by defining whether it is infected or not. For this, it receives the information from the ACT Detection Service and forwards it to the ACT Control Service.
- ACT Detection Service: is the service that directly obtains information about the presence or absence of the virus on the basis of the checks that are carried out on the dirty water. In this way, in the event that these checks show the presence of the virus, it will inform the ACT Local Service.

1.2 Motivation

The reasons for this project are primarily related to develop a multidisciplinary system that involves telecommunications and IoT, implementing the protocol at its best by exploiting knowledge in software engineering in such a way as to realize a proof of principle partially started in [42] in such a way as to create the components present in the ACT architecture but not yet developed such as the ACT Smart Application and the ACT Web Application. Furthermore, it is necessary to improve the interaction between the ACT infrastructure and the TIM ICON©[1] platform developed according to the IoT oneM2M standard [25] in order to make the system accessible to the public. The last reason is related to the testing of the system, it is in fact necessary to try the system by doing real and simulated tests in order to understand the real effectiveness of the developed infrastructure.

[1] https://icon-lab.tim.it/

The development of the ACT Smart Application implies the creation of a mobile application compatible with Android devices which, interacting with the smart things belonging to the IoT, allows to keep track of the places visited by users.
The development of the ACT Display Application implies the creation of a Web application that allows the user, by visiting its website, to view a map that shows the various areas recommended or not to visit due to the possible presence of the virus. In this way it is possible for the user to know the current pandemic situation and plan his travels doing forecasting.

1.3 Challenges

The challenges of this project are also related to the development of systems that interface with the world of the Internet of Thing, exploiting them in such a way as to be able to track users in an innovative way. It will also be necessary to design and develop applications that allow the exchange of information with other systems, including national ones, through TIM's oneM2M ICON© platform, kindly made available by TIM thanks to ETSI TC chairman SmartM2M Enrico Scarrone[2]. Finally, through tests and experiments it is necessary to carry out a study that allows to understand how to better manage the objects belonging to the IoT world that allow us to know the presence or absence of users.

1.4 Goals

The goals envisaged are:

- Familiarization with the topic in general: Contact Tracing, manual, synchronous and asynchronous
- Familiarization with the components of the project already developed
- Develop of the missing ACT components

[2] https://www.gruppotim.it/it/newsroom/notiziario-tecnico-tim/autori/s-t/enrico-scarrone.html

- Allow interfacing of the ACT components with the oneM2M TIM ICON© infrastructure

- Experiments and data analysis

1.5 Tasks

The definition of the tasks and the timing of each of them was carried out through a Gantt chart which shows all the tasks to be carried out and the related deadlines. In the Figure 1.1. I will show the chart made and we will see a brief description of each task.

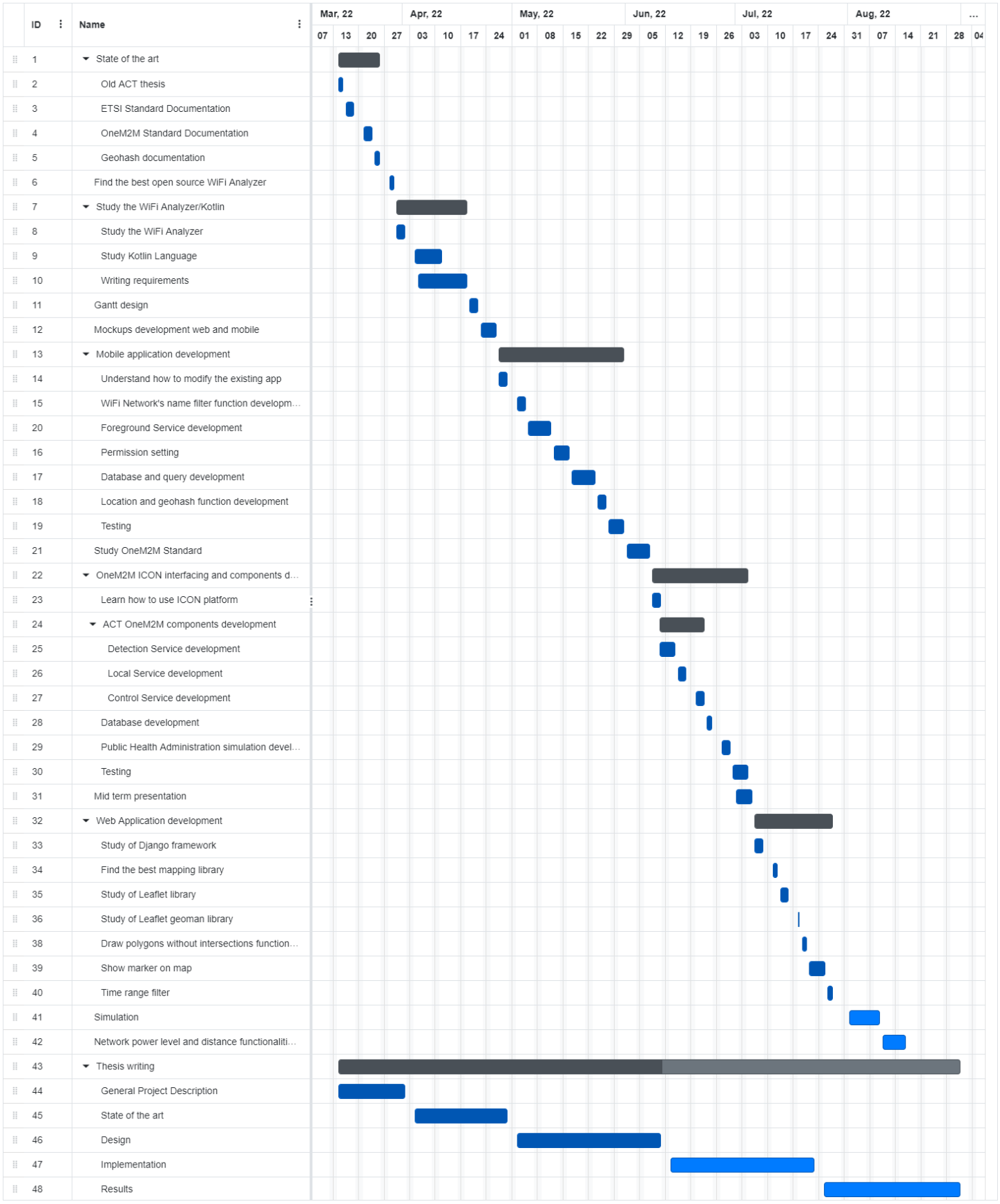

Figure 1.1: Internship Gantt chart

- Understand the subject, the state of the art, the goals, explore the platform (10 days). Learn the background related to contact tracing in general, studying the various contact tracing techniques already developed and comparing them with each other. Learn about Asynchronous Contact Tracing through the description of the technical standard: theoretical model, architecture, infrastructures used, components already built and to be built. Read the oneM2M infrastructure documentation. Read the documentation on how the mathematical public domain geocode system Geohash [11] works.

- Finding the best WiFi Analyzer Mobile Application (2 days): it is necessary to find the best open source application available on the Android Play Store that allows you to analyze WiFi networks so as to be able to create a fork that integrates the functions required by the ACT protocol to the already existing application.

- Study the chosen WiFi Analyzer© [36] features and get familiarity with the new Kotlin language [15] (14 days): understand the code related to the WiFi Analyzer©, study the various features present and understand how to extend the code. Study the Kotlin language necessary for the development and integration of the required functionalities. Finally, define the ACT Smart Application requirements, that is the functionalities that will be subsequently developed and integrated into the project using the Kotlin programming language.

- Gantt chart design (3 days): write the gantt chart containing all the tasks to be carried out and the estimated times in order to organize the work following a well-organized schedule.

- Mockups development for web and mobile application (3 days): designing web and mobile interfaces using the mockup tools in order to offer the best possible browsing experience that is pleasant and easy to users who use the applications.

- Mobile application development (25 days): develop the ACT Smart Mobile Application, an application for Android operating systems, by extending the WiFi Analyzer© previously chosen using the Kotlin programming language. Implement the requirements of the ACT protocol by optimizing the use of smartphone

resources as much as possible and creating a parallel version of the original WiFi Analyzer©.

- Study oneM2M standard (5 days): study of the oneM2M IoT standard, its architecture and its components.

- oneM2M TIM ICON© interfacing and components development (21 days): study of the oneM2M TIM ICON© platform and of the APIs that allow its use and communication. Development of the ACT protocol components that are part of the TIM ICON© platform and interact with it in such a way as to allow communication between the ACT and TIM ICON© components. More specifically: development of the Detection Service, Local Service, Control Service, development of a simulation of the *National Health Authority (NHA)*, set up of a NoSQL Database that stores the information sent by the NHA. Finally, the functions that allow the ACT Smart Mobile Application to communicate with the components that are part of TIM ICON© will be developed.

- Web application development (21 days): develop a web application that allows you to view, through a navigable map, the areas in which there are cases of virus positivity and/or potential outbreaks in such a way as to allow users to know the pandemic situation in real time and to be able to plan their movements on the basis of it.

- Experiments (7 days): set up, implement, and perform carry out software experiments and real hardware emulations in order to test the developed system by verifying the correct communication between the components and the correct functioning of the system in general. Collect real and self-generated data in order to analyze the robustness, accuracy and scalability of the system.

- Documentation (all the stage): I will write a documentation relating to the state of the art of the system up to that point, on the work I have done through the development of the new components, focusing on the design and showing the results. I will show the result of the data collection and analysis done in such a

way as to be able to provide evaluations regarding the robustness and scalability of the system.

In the Figure 1.2 we can see the PERT chart which shows the dependencies between the various tasks, this allows us to understand which tasks need to be carried out first in order to be able to carry out the others.

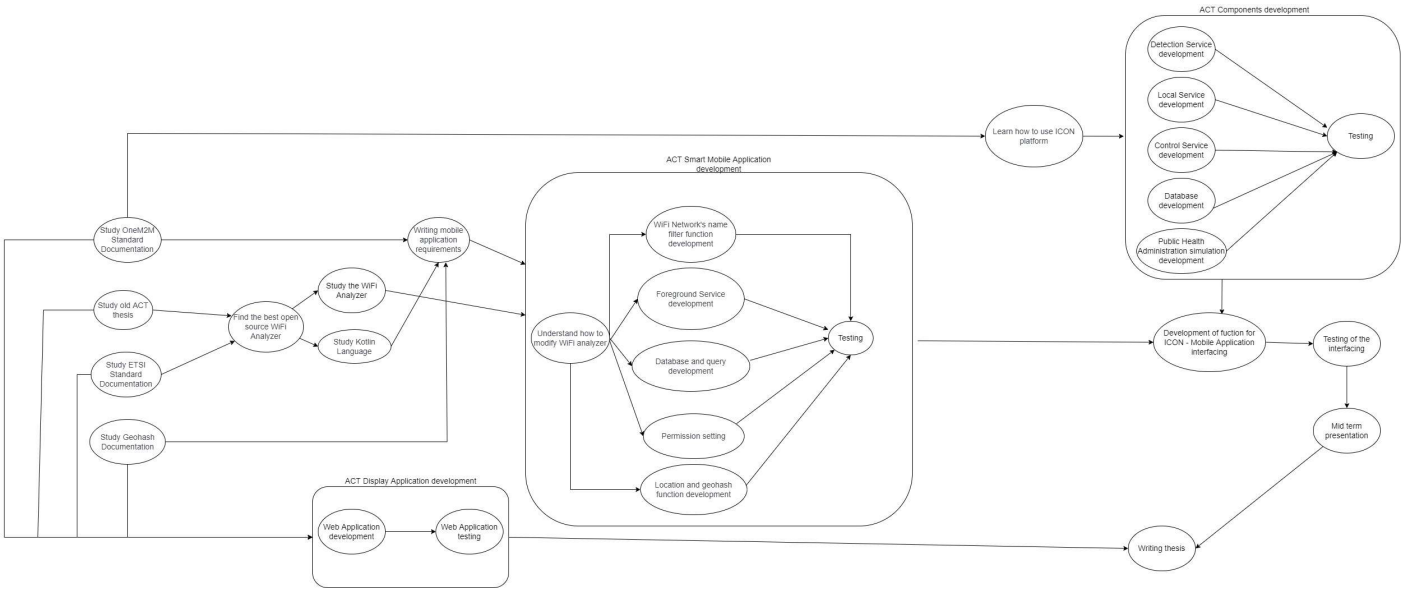

Figure 1.2: Internship PERT chart

From this chart we can define four macro-areas.

- The first macro-area relates to the study of the theory, in this case it is about the studying of the ETSI Asynchronous Contact Tracing standard and its scientific literature [43], the study of the documentation of the oneM2M standard, and of geohash standard. Also part of this category are the tasks related to the research of the WiFi Analyzer© application to fork, the study of its source code and the language used to develop it.
- The second macro-area relates to the development of the ACT Smart Mobile Application. It is possible to carry out the tasks related to this macro area only after completing the tasks of the first macro area. Here we initially have a design

phase necessary to understand which already existing components we need to modify in such a way as to extend them by integrating our functionalities, then we have the tasks related to the development of the functionalities and finally the testing phase.

- The third macro-area relates to the development of oneM2M components that are part of the ACT protocol, specifically: Detection Service, Local Service, Control Service, simulation of the National Health Authority. The completion of these tasks allows the execution of the testing phase relating to the whole developed system.

  Once the second and third macro areas have been completed, it is possible to proceed with the development of the functions that allow the interaction between the oneM2M components and the Smart Application.

- The fourth macro-area relates to the development of the ACT Display Application, the tasks within it relate to the development of this component and, subsequently, to testing.

  Finally we find the tasks related to the writing of the report for the entire project and the mid term presentation.

1.6 Prerequisites

1.6.1 oneM2M Standard

oneM2M [26] is a standard for the Internet of Things founded in 2012 by 8 of the most important organizations that develop standards in the IT field which are: ARIB (Japan), ATIS (United States), CCSA (China), ETSI (Europe), TIA (USA), TSDSI (India), TTA (Korea) and TTC (Japan). Their goal is to create a global technical standard that allows interoperability concerning the architecture, API specifications, security and enrollment solutions for Machine-to-Machine (M2M) and IoT technologies based on requirements contributed by its members. The implementations of this standard

are used in the areas of smart cities, smart grids, connected cars, home automation, public safety, and health. Thanks to oneM2M technology it is possible to eliminate the fragmentation present in the IoT world due to the different communication protocols developed by different manufacturers such that it is possible that two devices of two different brands cannot communicate with each other.
The most famous open source implementations of oneM2M are:

- ACME [2]: it is an easy-to-install and to use implementation, primarily for educational purposes. It has been designed for easy maintenance and use.
- Eclipse OM2M [9]: it is an implementation created to facilitate the development of services that use the oneM2M standard. It provides a horizontal M2M service platform for developing services independently of the underlying network, with the aim to facilitate the deployment of vertical applications and heterogeneous devices.
- OCEAN Mobius [24]: this implementation aims to promote the development and commercialization of IoT services by sharing open source implementations based on IoT standards thanks to a platform developed in accordance with the most widespread and globally accepted IoT standards.
- IoTDM [12]: the IoT Data Management (IoTDM) project is an open source implementation of oneM2M running on OpenDayLight [27]. This project consists in the development of a data-centric middleware that will act as a oneM2M compliant IoT Data Broker and enable authorized applications to retrieve IoT data uploaded by any device.
- OASIS SI [22]: this project is part of the open-source Semantic IoT Service-platform project. It consists of protocol binding, controller and resource handling and database layer for flexibility in such a way as to create techniques that allow to build a communication platform between IoT devices characterized by the possibility of giving semantic rules to the various devices.
- OpenMTC [28]: it is an implementation created to conduct applied research and develop innovative M2M and IoT applications. Thanks to its horizontal services

approach, it allows you to easily integrate IoT devices from different brands, regardless of hardware and network characteristics.

- ATIS OS-IoT [3]: is a library that simplifies the development of small IoT systems allowing you to create small ecosystems that comply with the oneM2M standard.

The main components of oneM2M standard [26] are:

- Application Entity (AE): it is an entity in the application layer that implements an M2M application service logic. This is the highest level component, therefore the closest to the user, for this reason examples of Application Entities are the remote blood sugar measuring application or the power metering application.
- Common Services Entity (CSE): represents an instantiation of a set of "common service functions" of the oneM2M Service Layer. This means that it includes all those resources necessary for the correct execution of the functions offered by the Application Entity. Some examples of CSEs are: data storage and sharing with access control and authorization, event detection and notification, group communication, scheduling of data exchanges, device management, and location services.
- Network Services Entity (NSE): provides services from the underlying network to the CSEs. Examples of such services include location services, device triggering, certain sleep modes like PSM in 3GPP [1] based networks or long sleep cycles.

1.6.2 ICON© Platform by Telecom Italia Mobile (TIM)

TIM ICON© API is an implementation of ETSI oneM2M infrastructure that allows user to deploy their protocol/network on TIM ICON© oneM2M cloud using REST structure to communicate with user or other servers/websites [26]. In TIM ICON© is possible to create containers, container instances, subscription and access control policies for containers. A container (CNT) is created under AE, it can have a specific content type and content. Container whenever receive some new information it

creates a Container Instance (CIN) and store the value in it. In TIM ICON© is also possible to define an access control policy for every container. A subscription (SUB) is used to subscribe to a resource, in this way whenever there is a change in one of the containers in which the user has signed up, a notification is sent. Furthermore TIM ICON© allow to group containers of same resource together using Group (GRP).

Chapter 2

State of the art

In this section we will look at the various contact tracing techniques that have been developed in recent years to better manage the COVID-19 pandemic and future pandemics. Then we will see what Asynchronous Contact Tracing is based on and how it works. Also in this case the goal is to understand the origin of the infection and understand who the infected person came into contact with. In this way it is possible to avoid a global lockdown and isolate only the infected and the cases most at risk of contagion, reducing the economic and social damage generated by the lockdown. In general, we can identify two types of contact tracing: Backward and Forward Contact Tracing [4].
Backward (or reverse or retrospective) Contact Tracing seeks to establish the source of an infection, by looking for contacts before infection. This technique has been used as a way of finding people with tuberculosis who are missed by routine health services. Forward tracing is the process of looking for contacts after infection, so as to prevent further disease spread. For epidemics with high heterogeneity in infectiousness (like COVID-19) it makes sense to apply a mixed methodology in which the use of backward is combined with forward contact tracing so as to more easily find the first infected patient within a cluster of infected people. During the COVID-19 pandemic, the adoption by Japan in early spring 2020 of an approach focusing on backward contact tracing was hailed as successful.
The Digital Contact Tracing tools/protocols uses GPS and Bluetooth® network feature

of the smartphone. The idea was initially discussed back in 2007, where in 2014 the idea was proven for the first time using Bluetooth®. This idea came under the spotlight after the COVID-19 pandemic.
To date, Digital Contact Tracing systems (both forward and backward) use synchronous systems, this means that both parties must be present at the same time. These systems fall into two categories:

- **Centralized**: In this case the contact details and history are stored at the National Health Authority or at main server.

- **Decentralized**: In this case the contact details and history are stored and managed by the clients in the network.

In the next paragraphs we will describe the existing Digital Contact Tracing models.

2.1 ROBERT

ROBERT (ROBust and privacy-presERving proximity Tracing protocol) [39][33] assumes an ecosystem composed of users who install the proximity tracing application (app) and a back-end server (highly secured) under the control of the National Health Authority.

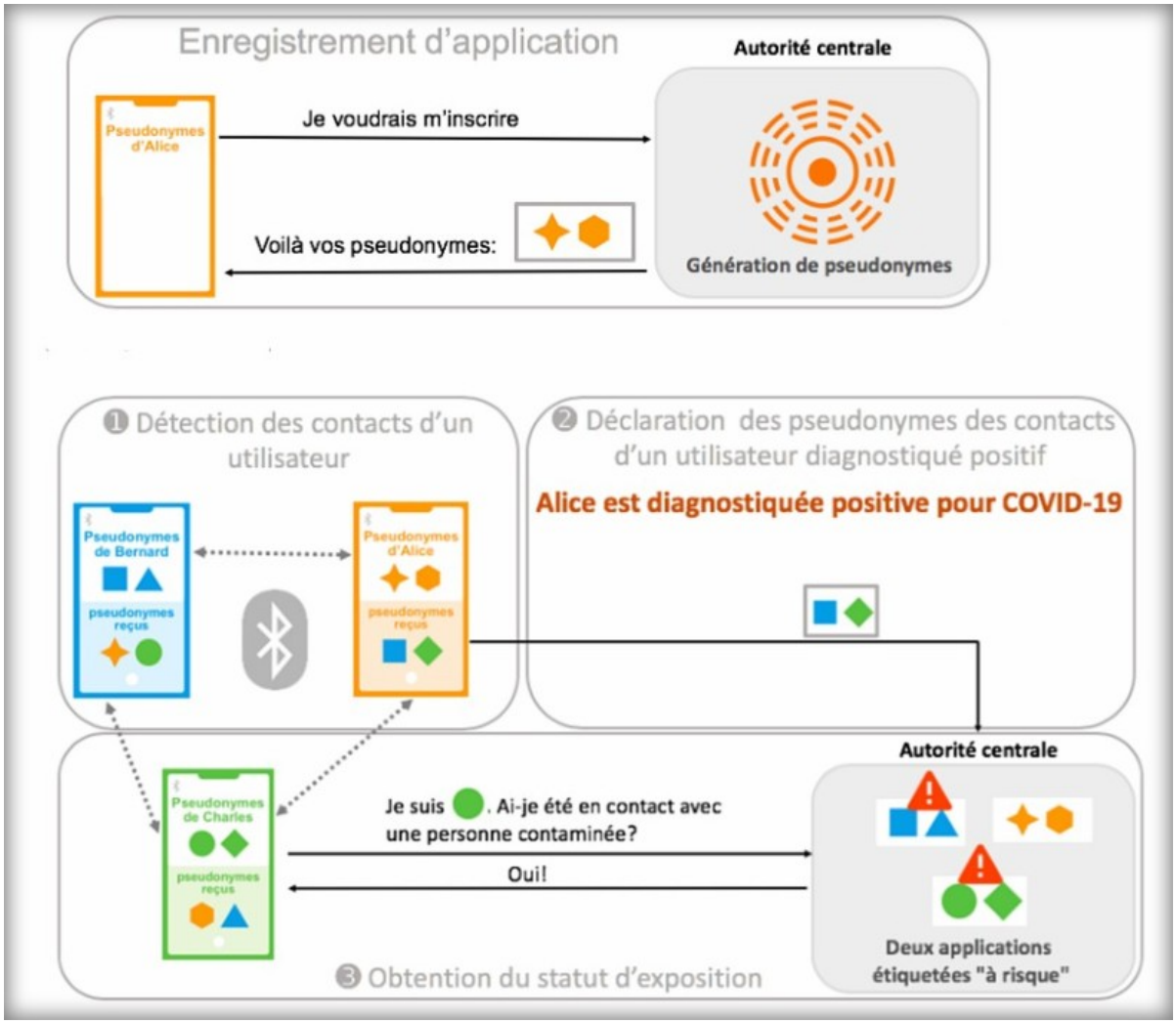

Figure 2.1: Use case example
Courtesy of INRIA [39]

In the Figure 2.1 above we can see an example of a use case where we have 3 users: Alice, Bernard, Charles. Each user initially, through the application, registers for the service by communicating directly with the National Central Authority which assigns him pseudonyms. Subsequently, when two users come into contact, an exchange of pseudonyms takes place via Bluetooth®, the pseudonyms received are saved in the memory so that in the event of a positive outcome of one of the users it is possible to know with whom he/she came into contact. In this use case we assume that

Alice is positive for COVID-19, what she does is to declare her virus positivity, this allows to automatically communicate the pseudonyms present in her memory to the National Central Authority so as to allow it to notify the users associated with those pseudonyms that they have been in contact with a person infected with COVID-19. Apps interact with the system through the four following procedures:

- **Initialization**: In this phase the user install the application from Apple Store or Google Store. Then it registers to the server that generates a permanent ID and several Ephemeral Bluetooth® Identifiers (EBIDs). The back-end maintains a table called IDTable, that keeps an entry for each registered ID. The stored information are "anonymous", it means that there are no associations between personal user's data and the information stored in the IDTable.

- **Proximity Discovery**: After registering for the service, the app sends a HELLO message via the Bluetooth® interface to all nearby devices. Nearby devices will respond in turn with a message from HELLO. These messages contain various information including the Ephemeral Bluetooth® Identifier. The collected HELLO messages are stored, together with the time of reception (and possibly other information such as the strength of the Bluetooth® signal or the user's speed) into local list of the application, the *LocalProximityList*.

- **Infected User Declaration**: When an user is tested and diagnosed COVID-19 positive, and after an explicit user consent and authorisation from the medical services, his smartphone's application will upload its LocalProximityList to the authority server, then flags as "exposed" all IDs of IDTable of which at least one EBID appears in the uploaded LocalProximityList. It's important to note that the server doesn't know the identifiers of the infected user's App but only the EBIDs contained in its LocalProximityList and given any two random identifiers of IDTable e that are flagged as "exposed", the server can not know whether they appeared in the same or in different LocalProximityList lists.

- **Exposure Status Request**: The user checks his virus exposure status on the app by querying the server using his EBIDs. The server then checks how many times the App's EBIDs were flagged as "exposed" and computes a risk score from this

information also considering the exposure duration or the user's speed / acceleration during the contact. If this score is larger than a given threshold, a flag indicating exposure to the virus is sent back to the App and user's account is deactivated, otherwise a neutral flag is sent back. Upon reception of this message, a notification is displayed to the user that indicates the instructions to follow (e.g. go the hospital for a test, call a specific phone number, stay in quarantine, etc.)

About the security of the system they assume that the back-end server is correctly secured, implementing the best state-of-the-art counter-measures and deploying the required security measures to prevent intrusions. Its security will be audited, tested and validated by the competent national bodies.

2.2 DESIRE

DESIRE [38] is an evolution of ROBERT, leveraging the best of the so-called centralized and decentralized systems. DESIRE is based on Private Encounter Tokens (PET), generated from Ephemeral Bluetooth ® Identifiers (EBID). The EBIDs and PETs are generated and computed locally by the mobile device and are nonlinkable. A PET token uniquely identifies an encounter between two nodes and is secret. PET has the advantage over EBID that it reduces the ability of the server to link collocation information coming from different individuals. Furthermore, this method mitigates "replay attack", where a malicious individual collects the EBIDs received by an infected (or potentially infected) individual and replays them in many locations, thus creating a large number of false positives. Being an evolution of ROBERT, DESIRE also follows the same 4 procedures but which have been slightly modified to manage the use of PETs. The procedures are as follows:

- **Initialization**: The user install the application from Google or Apple Store. Then it registers to the server that generates a permanent ID. IDTable keeps an entry for each registered ID. The stored information are "anonymous", it means that there are no associations between personal user's data and the information stored in the IDTable.

- **Proximity Discovery**: The device generates a new and unlinkable EBID at each epoch, it broadcasts this EBID regularly and collects EBIDs of encountered devices. If certain conditions, for example contact length, received signal strength, etc., are satisfied, it generates PET tokens from collected EBIDs. Finally it stores the generated PET tokens in a local list, along with, if necessary, additional metadata (contact length, speed, etc.).

- **Infected User Declaration**: When an individual is tested and diagnosed COVID-19 positive, and after an explicit user consent and authorisation from the medical services, his smartphone's application uploads its local list of generated PET tokens to the authority server, that adds them in a global list of exposed PET tokens.

- **Exposure Status Request**: The user checks his virus exposure status on the app by querying the server using its list of generated PET tokens. The server then checks how many of the App's tokens appears in the global list and computes a risk score from this information also considering the exposure duration or signal strength. If this score is larger than a given threshold, a flag indicating exposure to the virus is sent back to the App, otherwise a neutral flag is sent back. Upon reception of this message, a notification is displayed to the user that indicates the instructions to follow (e.g. go the hospital for a test, call a specific phone number, stay in quarantine, etc.).

About the security of the system various attacks were considered and analyzed:

- **Server Data Breaches**: The server only stores pseudonymous data. In addition, this information is minimized and only used to compute the exposure risk scores. Furthermore, each entries in IDTable of a device is encrypted using a key that is stored only by the device and provided to the server with the Exposure Status queries. As a result, in case of a data breach of the server, all useful information will be encrypted. This risk associated with a data breach is then minimal.

- **Passive Eavesdropping/Tracking (by malicious users and authority)**: Since

PET tokens are unique per encounters and are computed locally, passive eavesdroppers only get the EBIDs, that are changing at every epoch. Furthermore, if the authority deploys some Bluetooth® receivers, it will not be able to relate any EBID to any PET tokens. Passive tracking by the server or users is therefore not possible.

- **Active Eavesdropping/Tracking (by malicious authority)**: If the authority is active and deploys Bluetooth® devices that also broadcast their own EBIDs, the target device, will generate and store the relative PET tokens. The server's devices can also generate the same tokens that the server could use to identify the target's Exposure Status Request messages and possibly track some of his locations. Since the Exposure Status queries of a node are linkable, with enough of these tracking devices, the server could possibly re-identify some users.

- **Reconstructing social interaction graphs (by malicious authority)**: When a user is diagnosed COVID-19 positive, he/she anonymously uploads all elements of its global list independently. Consequently, the server is not able to make any links, neither between the user and the uploaded PET tokens nor between the uploaded PET tokens. When a user queries the server for its exposure status, the server is able to link his ID to all PET tokens contained in the Exposure Status query. However, the PET tokens used in the Exposed Status queries are different from the tokens uploaded to the server if he/she ever gets diagnosed. Consequently, the server is not able to infer any links between exposed tokens in its global list and tokens in requests. Furthermore when two users A and B who exchanged EBIDs and built associated PET tokens, request the server with different tokens. The server is thus not able to link any tokens in different requests.

- **Infected Node Re-identification (by malicious user)**: User can not identify infected contacts since this information is kept on the server and users only get an exposure risk score. However, the *one entry attack*. In this attack, the adversary has only one entry, corresponding to the User, in his local lists (this can easily be achieved by keeping the Bluetooth® interface off, switching it on when the

adversary is next her victim and then switching it off again). When the adversary is notified "at risk", he/she learns that the User was diagnosed COVID-19 positive, that is inherent to all schemes is still possible.

- **Replay attacks**: Not possible since it is assumed that the communication is symmetrical.
- **Relay attacks**: Only possible within, at most, one epoch.
- **False Alert Injection Attacks**: The pollution is an attack where a malicious node colludes with a diagnosed user to include the identifiers of some victims in his contact list. The goal of the malicious adversary is to make the app of a target victim raise false alerts. This attack requires that the colluding user and the victim interact to compute their PETs. Such attacks are therefore only possible via a relay attack, so they will work only within an epoch.

2.3 DP-3T

Decentralized Privacy-Preserving Proximity Tracing[1] (DP-3T) [48] aspires to *"minimize privacy and security risks for individuals and communities and guarantee the highest level of data protection"* while performing Digital Contact Tracing. The system has been deliberately designed to not support tracking positive cases, detect hotspots or trajectories of positive cases and sharing data for epidemiological research. The protocol is proposed to work on participating smartphones which locally generate pseudo-random ephemeral identifiers (EphIDs) and broadcast them using Bluetooth® Low Energy beacons. Simultaneously, the smartphones also listen for these beacons and store the broadcast EphIDs with a timestamp and signal attenuation to estimate exposure. The protocol also requires the support of an honest backend server that is available. The primary role of the server is to distribute anonymous exposure information to the apps of the participating devices. Further, the server is trusted to not add spurious exposure events or remove genuine exposure events. All processing happens locally on the devices. The privacy of the users does not depend on the behaviour of

[1] `https://github.com/DP-3T/documents`

the server. In the event of a positive infection diagnosis the user is authorized by the local health authorities to upload a protocol specific representation of their EphIDs to the backend server for a time window during which they were contagious thus aiding decentralized proximity tracing. About the validation of the infected user it uses a mechanisms to validate upload requests from personal devices in which the backend server aggregates the uploaded EphID representations and distributes them to the participating devices when they query the server. Subsequently the devices can recompute the EphIDs of infected users locally. To facilitate masking the upload traffic of infected users the participating devices generate dummy traffic to provide plausible deniability of real uploads. If any recorded EphID matches those downloaded from the server then the user may have been exposed to the virus. The app estimates the exposure risk by looking at the exposure measurements of the matched beacons.
In line with this framework three protocols are proposed with varying trade-offs between privacy and computation cost: *Low-cost*, *Unlinkable* and *Hybrid*.

- **Low-cost**: The devices start by generating a random initial daily seed for the current day. The secret day seed is rotated daily, and the new seed is set as the hash of the previous day seed. The daily secret seed is used to generate a list of ephemeral ids with the lifetime of a few minutes, known as epoch. If a user is diagnosed as infected, then they can send the secret seed along with the day corresponding to the first day of infection to the server. After this, the phone deletes the secret seed and generates a new one and proceeds as before, this is done to prevent tracking. The server then disseminates the collected secret seeds and the day of infection to all the participating devices which in turn can compute the EphIDs from this information and check whether they have been exposed. If a match is observed, then the beacon's receive time and exposure measurement are taken into account for the exposure risk computation.

- **Unlinkable**: It improves upon Low-cost variant by offering better privacy properties at the cost of increased bandwidth. Instead of distributing the list of seeds of infected users, the backend server includes the hash of EphIDs in a Cuckoo filter [6] which is then broadcasted to the participating devices. A Cuckoo fil-

ter is a space-efficient probabilistic data structure which is used to test the set membership of an element [6]. Due to the probabilistic nature a false positive is possible, however, a false negative is not. In other words, a query returns either ”possibly in set” or ”definitely not in set”. Using a Cuckoo filter precludes the adversaries from linking EphIDs of the infected users. It also allows the infected users to redact EphIDs corresponding to sensitive location or times. In this approach EphIDs are generated each epoch using a random per-epoch seed. In case of positive diagnosis the users upload a chosen set of EphIDs and corresponding epochs. The parameters of the Cuckoo filter are chosen such that it produces about one false positive in a million users over a period of 5 years. Due to hashing of values before placing them in the Cuckoo filter and sparseness in a large set, enumeration attacks are not efficient, hashing also makes reversing the filter of limited use.

- **Hybrid**: It attempts to combine the benefits of both the previous approaches. In this approach the devices generate seeds for each time window and then use the seeds to generate EphIDs as in the Low-cost approach. This allows better unlikability than the Low-cost design as well as the opportunity to redact EphIDs. However, the tracking protection is as not good as the Unlinkable design, though, the bandwidth cost is low.

2.4 GAEN

The Google & Apple® Exposure Notification System (GAEN) [32] is based on the ideas of DP-3T. The API was created with the vision that it would be widely used by COVID-19 contact tracing apps which would be developed independently by National Health Authorities. The API uses Bluetooth® Low Energy to detect proximity between people running the COVID-19 tracing apps. The users decide whether they wish to opt-in to the Exposure Notifications and the system does not collect any location information from the device. In the event of a positive infection diagnosis, it is up to the user to report this in the ad hoc developed application. The protocol works by exchanging randomised tokens between voluntarily participating devices which

are running the same COVID-19 contact tracing app. The tokens, known as Rolling Proximity Identifiers (RPIs), are exchanged when the devices are in close proximity of each other. A 16-byte Temporary Exposure Key is generated locally on the device using cryptographic random number generator, the key is rotated every 24 hours. The Temporary Exposure Key is used to generate RPI Keys and Associated Encrypted Metadata (AEM) Keys. The AEM keys are used to encrypt the Bluetooth® metadata containing the transmit power level which is the measured radiated transmit power of Bluetooth® Advertisement packets and is used to improve distance approximation. RPIs are generated from RPI Keys and are rotated every 10-20 minutes (based on specification), the same frequency at which the Bluetooth® randomised address is changed, to prevent linkability and wireless tracking. Each participating device records a list of RPIs it encounters. In the event of an infection the device can choose to self-report the infection to a Diagnosis Server and upload a limited set of Temporary Exposure Keys known as Diagnosis Keys based on the time window of last 14 days. If the user remains healthy the Temporary Exposure Keys do not leave the device. The Diagnosis Server aggregates the Diagnosis Keys of all infected users, which are then broadcasted to all the devices participating in exposure notification, the participating devices download these keys at least once per day. The Diagnosis Keys are used to generate RPIs of infected users and matched locally on the device to the list of RPIs received. In the event of a match the user is notified and advised on the next steps. In the case of such a notification the system also shares the day the contact with the infected individual occurred, how long it lasted and the Bluetooth® signal strength of that contact.

2.5 PEPP-PT

Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT/PEPP) [31] [30] is a full-stack open protocol that, like the competing Decentralized Privacy-Preserving Proximity Tracing (DP-3T) protocol, makes use of Bluetooth® Low Energy to discover and locally log clients near a user. However, unlike DP-3T, it uses a centralized reporting server to process contact logs and individually notify clients of potential contact

with an infected patient. This protocol is divided into two main phases:

- **Encounter handshake**: When two clients encounter each other, they must exchange and log identifying details. In order to prevent the tracking of clients over time through the use of static identifiers, clients exchange time sensitive temporary IDs issued by the central server. In order to generate these temporary IDs, the central server generates a global secret key which is used to calculate all temporary IDs for a short timeframe. From this an Ephemeral Bluetooth® ID (EBID) is calculated for each user with the AES encryption algorithm. These EBIDs are used by the clients as the temporary IDs in the exchange. EBIDs are fetched in forward dated batches to account for poor internet access. Clients then constantly broadcast their EBID under the PEPP-PT Bluetooth® service identifier, while also scanning for other clients. If another client is found, the two exchange and log EBIDs, along with metadata about the encounter such as the signal strength and a timestamp.

- **Infection reporting**: When a user, out of band, has been confirmed positive for infection the patient is asked to upload their contact logs to the central reporting server. If the user consents, the health authority issues a key authorizing the upload. The user then transmits the contact log over HTTPS to the reporting server to be processed. Once the reporting server has received a contact log, each entry is run through a proximity check algorithm to reduce the likelihood of false positives. The resulting list of contact is manually confirmed and they, along with a random sample of other users, are sent a message containing a random number and message hash. This message serves to wake up the client and have them check the server for new reports. If the client is on the list of confirmed users, the server will confirm potential infection to the client which will in turn warn the user. If a client is in the random sample, it will receive a response with no meaning. The reason a random sample of users is sent a message for every report is so that eavesdroppers are not able to determine who is at risk for infection by listening to communication between the client and server.

There is also an authentication phase that is required to prevent malicious actors from creating a multiple false user accounts, using them to interfere with the system. In order to preserve the anonymity of the users, traditional authentication models using static identifiers such as email addresses or phone numbers could not be employed. Rather, the protocol uses a combination of a proof-of-work challenge and CAPTCHA [5]. The suggested proof-of-work algorithm is a script, as defined in RFC7914 [45], popularized in various blockchain systems such as Dogecoin [8] and Litecoin [20]. Once a user registers with the app, they are issued a unique 128 bit pseudo-random identifier (PUID) by the server. Once completed, OAuth2 [23] credentials are issued to the client to authenticate all future requests.

2.6 TCN

The Temporary Contact Numbers Protocol (TCN) [34] is an open source, decentralized, anonymous exposure alert protocol that uses Bluetooth® Low Energy to track and log encounters with other users. The major distinction between TCN and protocols is the fact the central reporting server never has access to contact logs nor is it responsible for processing and informing clients of contact. Because contact logs are never transmitted to third parties, it has major privacy benefits. This approach however, by its very nature, does not allow for human-in-the-loop reporting, potentially leading to false positives if the reports are not verified by National Health agencies. This protocol works off the basis of Temporary Contact Numbers (TCN), a unique and anonymous 128-bit identifier generated deterministically from a seed value on a client device. When two clients encounter each other, a unique TCN is generated, exchanged, and then locally stored in a contact log. Then, once a user tests positive for infection, a report is sent to a central server. Each client on the network then collects the reports from the server and independently checks their local contact logs for a TCN contained in the report. If a matching TCN is found, then the user has come in close contact with an infected patient, and is warned by the client. Since each device locally verifies contact logs, and thus contact logs are never transmitted to third parties, the central reporting server cannot by itself ascertain the identity or contact

log of any client in the network. In this protocol it's possible to distinguish two main phases:

- **Encounter handshake**: The encounter handshake runs on Bluetooth® Low Energy and defines how two devices acknowledge each other's presence. When two devices come within range of each other, they exchange a handshake containing TCNs. In order to achieve this the encounter handshake operates in two modes (both with two sub-modes), broadcast oriented and connection oriented. Broadcast oriented operates using the modes broadcaster and observer, while connection oriented operates using peripheral and central. The two modes are used to circumvent certain device limitations, particularly in regard to iOS restrictions in place before version 13.4. In broadcast mode, a broadcaster advertises a 16-byte TCN using the service data field of the advertisement data. The observer reads the TCN from this field. In connection-oriented mode, the peripheral advertises using the UUID. The service exposes a read and writeable packet for sharing TCNs. After sharing a TCN, the central disconnects from the peripheral.

- **Infection reporting**: When a user tests positive for infection, they upload a signed report, allowing the past 14 days of encounters to be calculated, to a central server. On a regular basis, client devices download reports from the server and check their local contact logs using the verification algorithm. If there is a matching record, the app notifies the user to potential infection.

2.7 BlueTrace

BlueTrace[2] [37] is the protocol developed by the Singaporean Government for contact tracing of users to stem the spread of the COVID-19 pandemic. The BlueTrace protocol is a centralized one, it has the following features:

- Subscribers provide a phone number for urgent notification in case of positive contact detection. This allows to reach users even if the app is not on, or they

[2] `https://github.com/OpenTrace-community`

have TraceTogether tokens.

- There is a Bluetooth® Low Energy connected process in the protocol where the devices exchange complementary data in a file, containing the device's TempID, device model, health authority identifier, and BlueTrace protocol version.
- The protocol allows devices with transmission only capabilities.

2.8 PACT

The Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing (PACT) [40] specification advocates for a third-party-free approach to assisted mobile contact tracing. The objective of PACT is to set forth transparent privacy and anonymity standards, which permit adoption of mobile contact tracing efforts while upholding civil liberties. PACT assumes that communication between devices and server is protected using the Transport Layer Security (TLS) protocol. Privacy and integrity properties of the protocol are guaranteed by the pseudo randomness. PACT describes a series of mobile functionalities beyond digital contact tracing. Such functionalities seek to augment he/she services provided by National Health Authorities and are the follows:

- **Mobile-assisted contact tracing interviews**: a way to speed up manual contact tracing's interview processes by filling in on a device much of a contact interview form before the contact interview process even starts, reducing the burden on National Health authorities.
- **Narrowcast messages**: capability that allows National Health authorities to quickly warn specific, relevant subsets of citizens through custom-tailored messages.
- **Privacy-sensitive, mobile tracing**: an actual digital contact tracing protocol, based on the basic idea of having users broadcasting signals ("pseudonyms"), while also recording the signals they receive. A co-location approach that avoids the need to collect and share absolute location information.

Chapter 3

Asynchronous Contact Tracing

Asynchronous Contact Tracing (ACT) [43] is based on a novel testing idea, invented by Professor Dorfman during World War II, called *Group Testing* [41]. This method consists of testing positivity for a virus by testing clusters of patients rather than testing one patient at a time. Samples are then taken from the patients, combined and tested at once. If the test result is negative it means that all patients are test negative, if it is positive it means that there will be at least one of the patients who tested positive. In this case, the test must then be repeated individually to understand who is the positive among the members of the group. It has been shown that statistically this technique allows to save an excellent percentage of tests because it has been seen that at a probabilistic level, by testing a good number of patients, the probability that there is an infected inside the cluster is low compared to the probability that there is not. This leads to savings in terms of tests to be carried out, which therefore leads to savings in economic resources and human efforts. This methodology has inspired ACT because ACT process is a subset of the Dorfman Group Testing because people are interested only in the presence or absence of the SARS-CoV-2 virus in order to make the ACT process work. The reason is due to the fact that people cannot identify the *infectors*[1] of waste-water, or air-filtering or other materials or goods. The use of this type of group test combined with the use of IoT technologies therefore allows for a better control on a large scale which reduces costs and management times, avoiding

[1]Infectors denotes either humans or things hosting the virus.

current global lockdowns but favoring selective lockdowns. In this way whenever people test an infected location, the ACT technique will communicate to the population a precise geographical area which can be defined immediately as safe or unsafe. The main difference between ACT and Synchronous Contact Tracing (SCT) is that ACT do not require two persons to be present at the same time and at a given distance in order to trace a risk of infection. ACT traces objects and surfaces or locations that have been contaminated by the virus. The SARS-CoV-2 virus is known to stay contagious for some time in an aerosol, or on an object or a hard surface, depending on the nature of the surface, the virus concentration, temperature, humidity conditions, exposition to sun light, so it is the perfect example to use and apply ACT.
ACT is a digital contact tracing protocol that works as follows:

- Many commercial WiFi ordinary access points, called in the ACT jargon ACT Peripheral Services, are installed in specific locations, such as a supermarket corridor, a public toilet, a metro station, etc. Each peripheral transmits continuously two information:
  - its unique WiFi identifier (BSSID),
  - its unique WiFi human-readable network name (SSID). In the ACT standard, the SSID could start with the string "ACT-" and continue with a string set according to the deployment needs.
- An app called ACT Smart Mobile Application is installed in each smartphone which using the WiFi sensor listens and registers in its local and protected memory, all the BSSIDs having as SSID network name prefix the "ACT-" prefix string. The ACT Smart Mobile Application can ask the ACT Control Service, directly connected with the National Health Authority, to be informed on the forecast of the presence of COVID-19 in the geographical places visited by the smartphone owner. The geographical locations visited by the owner of the smartphone will never be shared outside the ACT Smart Mobile Application to comply with the GDPR.
- Samples of potentially infected material are tested locally using ad-hoc and in-

place (manual, automatic, or semi-automatic) ACT Detection Services, or outsourced, sending a sample of the material to the closest Biological Laboratory.

- The results of tests are sent to the ACT Local and ACT Control Service in association with the BSSID of the WiFi access points close by to the point of taking of the sample. This information is interpreted by the National Health Authority into a precise virus forecast, which is geo-localized and finally widely communicated by the ACT Control Service to the ACT users using well-established Web techniques.

- It is also possible to establish a direct communication channel between the user and the National Health Authority to suggest the behavior (e.g. to go fast for a RT-PCR test) in case that the ACT Smart Mobile application identify a risk of infection.

- The ACT Smart Mobile Application match the BSSIDs retrieved from the ACT Control Service, each of one associated with a precise forecast, with the BSSIDs recorded in the smartphone local protected memory; in case of non-empty intersection, the ACT Smart Mobile Application notify immediately a potential contamination to the owner of the smartphone. At that point, he/she may decide to go to a Biological Laboratory for an individual unitary test.

- Through the ACT Display Application it is possible to see the evolution of the situation in real time on the map and access the forecasts available by the ACT Control Service.

- Additionally, the ACT Smart Mobile Application, in case the owner of the smartphone is tested positive to the virus, may send to the ACT Control Service the time of the test together with the list of pairs composed by the BSSIDs and the related timestamps.

3.1 ACT Method Architecture and Functionalities

The ACT architecture is made up of various components that communicate with each other in a one-way or two-way manner as shown in the following Figure 3.1.

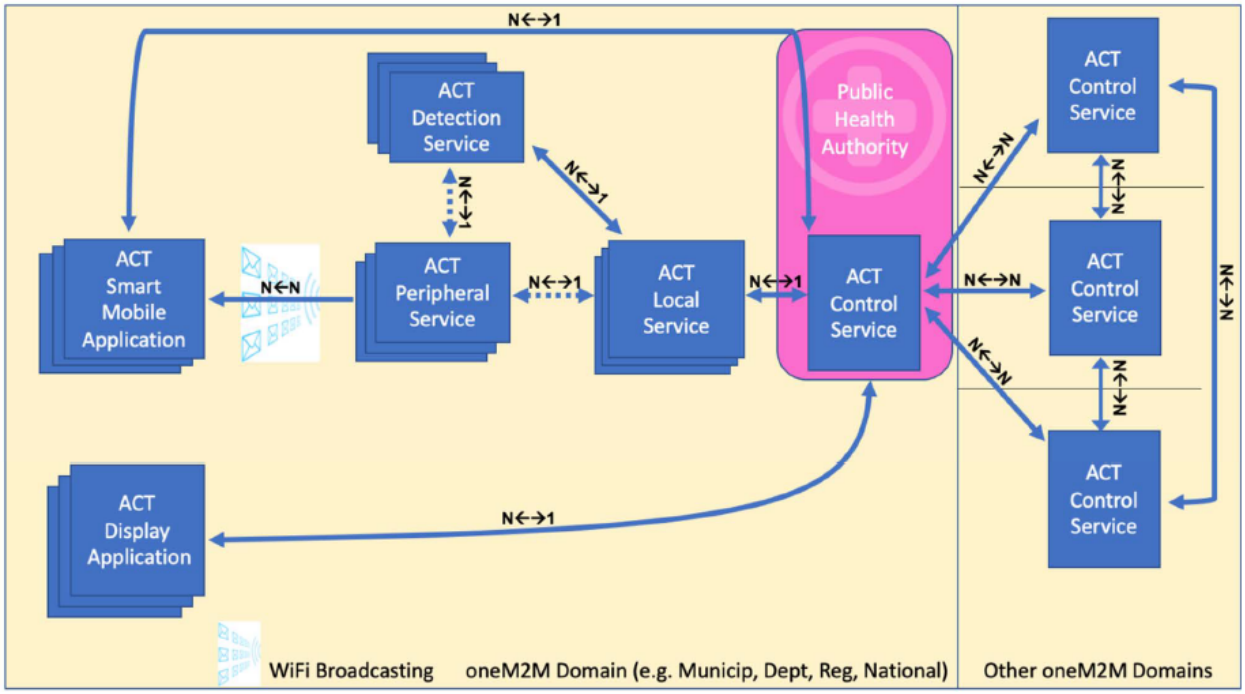

Figure 3.1: ACT Architecture
Courtesy of ETSI Technical Specification TS 103757. SmartM2M; Asynchronous Contact Tracing System [43]

In the Figure 3.1 it is possible to see the relationships between the various ACT components with their respective multiplicities. The ACT Peripheral Service interacts with the ACT Detection Service with an N to N relationship as in this way it is possible to allow the correlation of the detection of the contamination with the proximity area where the ACT Peripheral Service is deployed. It also interacts with the ACT Local Service with an N to 1 relationship as each area of the map is covered by a group of ACT Peripheral Services. In this way, in case of detection of the virus, the ACT Local Service will communicate to the ACT Control Service the BSSID identifier of the ACT Peripheral Service involved in all test detections, together with their location. Finally,

the ACT Peripheral Service interacts with the ACT Smart Mobile Application with a one-way N to N relationship as the purpose of this relationship is to ensure that the ACT Smart Mobile Application is constantly listening in order to record the user's movements on the base of the detection of the various ACT Peripheral Services.
The ACT Detection Service has an N to N relationship with ACT Peripheral Services as multiple ACT Detection Services can be associated with multiple ACT Peripheral Services. The ACT Detection Service has a N to 1 relationship with the ACT Local Service because its purpose is to inform the ACT Local Service about the detection of the contamination in one of the proximity areas associated to one of the ACT Peripheral Service that belongs to it.
The ACT Local Service has a 1 to N relationship with the ACT Detection Service as it receives the information from one or many Detection Services belonging to it and forwards this information to the Control Service. It also has a N to 1 relationship with the ACT Control Service because it sends information about the area for which it checks for the presence of the virus. Finally it has a 1 to N relationship with ACT Peripheral Services because it transmits to the ACT Control Service the BSSID identifier of the ACT Peripheral Service involved in all test detections, together with their location.
The ACT Control Service has a 1 to N relationship with the ACT Local Service because it receives the information from all the Local Services related to Detection Services located in his area of pertinence and provides to it information, according to the National Health Authorities policies. It also has a 1 to N relationship with the ACT Smart Mobile Application and the ACT Display Application because through these entities it informs end users of the presence or absence of the virus in the areas they have visited.
The ACT Smart Mobile Application has an N to N relationship with the ACT Peripheral Service because it collects the identifier from the one or many Peripheral Services and periodically compares it with the identifier-related published by the Control Service. It also has a N to 1 relationship with the ACT Control Service because it receives information about the presence or absence of the virus from the ACT Control Service.
The ACT Display Application has a N to 1 relationship with the ACT Control Service because it aims to query and receive feedback about the status of the situation in spe-

cific geographical areas. By delegation, it can also query other Inter-Control Services. In the next sections we will see what information is exchanged between the various components of the architecture.

3.2 Information broadcasted by ACT Peripheral Service (and listened by ACT Smart Mobile Application)

The ACT Peripheral Service shall broadcast the information specified in the following table.

Table 2.9.2

Parameter name	**Type**	**Description**
PERIPHERAL-SERVICE-ID	STRING	The identifier depends on the communication technology used. The identifier of the supported technology is defined in this clause below the table.

The transmission technology supported by the ACT Peripheral Service is WLAN (WiFi) according to IEEE 802.11. The ACT Peripheral Service is broadcasted by a commercial WiFi Access Point, and the string of the PERIPHERAL-SERVICE-ID shall be the BSSID. In this case, the SSID of the Access Point configured by the ACT Local Service shall contain the prefix "ACT-", and it shall be composed by the concatenation of the string "ACT-" with a string set according to the deployment needs. This allows the ACT Smart Mobile Application to decide (by filtering among the different detected SSID) which BSSID would be memorized among the many ones of the different WiFi Access Points detected. When detecting the BSSID/SSID, the ACT Smart Mobile Application shall memorize in its internal memory a record containing:

- the BSSID;

- the time TIMEIN (calculated internally by the mobile device and representing the time of when the BSSID/SSID is detected);

- the time TIMEOUT (calculated internally by the mobile device and representing the time of when the BSSID/SSID signal is lost).

A short temporary loss of the ACT Peripheral Service signal should not be considered as an interruption of the presence of the mobile device under the coverage of the ACT Peripheral Service.

3.3 Information exchanged between ACT Detection Service and ACT Local Service

The ACT Detection Service shall transmit to the ACT Local Service the information specified in the following table.

Table 2.9.3-1

Parameter name	**Type**	**Description**
STATUS	It is defined by one of the following STRING values: • ACTIVE • SLEEPING • OUT-OF-SERVICE • RESTARTING • MAINTENANCE-REQUESTED • FAULT	This parameter shall be mapped according to the following: • **ACTIVE**: the Detection Service is operative; • **SLEEPING**: the Detection Service has the energy saving status activated (e.g. when the associated shop is currently closed); • **OUT-OF-SERVICE**: the Detection Service is still connected but it cannot become operative (e.g. it is subject to a maintenance procedure); • **RESTARTING**: the Detection Service is in the process of becoming operative; • **MAINTENANCE-REQUESTED**: the Detection Service require a maintenance (e.g. for the refilling of the test reagents, for the calibration, or for other tuning reasons); • **FAULT**: the Detection Service has found itself to be faulty.
TEST-TIME	Time as defined in ISO 8601.	The time of the collection of the test sample shall be reported in the TEST-TIME parameter. This parameter is absent if the Detection Service answers to a STATUS command and there is no TEST-RESULT to be reported.
TEST-RESULT	It is defined by one of the following NATURAL values: 0 ... 100.	This parameter shall be mapped according to the following: • 0: No Virus detected; • 1 to 100: Virus detected with an indication of the level of contamination of the sample. This parameter may be absent when the Detection Service answers to a STATUS command and there are not TEST-RESULTS to be reported.

The ACT Local Service shall transmit to the ACT Detection Service the information specified in the following table.

Table 2.9.3-2

Parameter name	**Type**	**Description**
COMMAND	It is defined by one of the following STRING values: • RESTART • SHUTDOWN • SLEEP • STATUS-REQUEST • TEST-START • TEST-STOP	This parameter shall be mapped according to the following: • **RESTART**: the Detection Service shall restart; • **SHUTDOWN**: the Detection Service shall shutdown; • **SLEEP**: the Detection Service shall activate the energy saving status; • **STATUS-REQUEST**: the detection service shall respond providing the STATUS Parameter; • **TEST-START**: the Detection Service shall initiate performing the tests, according to the given TEST-INTERVAL indication; • **TEST-STOP**: the Detection Service shall stop performing the tests.
TEST-INTERVAL	NATURAL	This parameter shall be mapped according to the following: • 0: the test shall be executed continuously; • 1 to N: time interval, expressed in seconds, from the time of the last test if available; in case the time from the last test is not available, the test shall be executed immediately.

3.4 Information exchanged between ACT Local Service and ACT Control Service

The ACT Local Service shall transmit to the ACT Control Service the information specified in the following table.

Table 2.9.4

Parameter name	**Type**	**Description**
PERIPHERAL-SERVICE-ID	Defined in Table 2.9.2	Defined in Table 2.9.2
LOCAL-SERVICE-INFO	URL	URL pointing to customized information provided by the Local Service.
LOCATION	STRING	It shall contain the geographical area, represented using the Geohash Geocode System [11] of the place where the sample reported in the provided TEST-RESULT has been taken.
LOCAL-SERVICE-INFO	URL	URL pointing to customized information provided by the Local Service (valid for that specific Peripheral Service).
TEST-RESULT	Defined in Table 2.9.3-1	Defined in Table 2.9.3-1
TEST-TIME	Defined in Table 2.9.3-1	Defined in Table 2.9.3-1
DISINFECTION	BOOLEAN	TRUE indicates that: • active virus is not expected to be present; • traces of not active virus have been reasonably removed.
DISINFECTION-TIME	Time as defined in ISO 8601.	Time of completion of the disinfection.

The LOCATION of a specific PERIPHERAL-SERVICE-ID shall be sent only at the initialization and at any change of LOCATION parameter. The LOCAL-SERVICE-INFO optional parameter may be present in the case that the ACT Local Service intends to make available to the ACT Smart Mobile Application additional customized information via a dedicated URL. It may be sent at the initialization and at any change of the LOCAL-SERVICE-INFO parameter. The DISINFECTION/DISINFECTION-TIME parameter couple may be associated (or not) to a TEST-RESULT/TEST-TIME parameter couple.

3.5 Information exchanged between ACT Smart Mobile Application and ACT Control Service

The ACT Smart Mobile Application shall periodically send a query the ACT Control Service for providing the information specified in the following table.

Table 2.9.5-1

Parameter name	**Type**	**Description**
QUERY	STRING	It shall contain the geographical area represented using the Geohash Geocode System [11]. It includes the area visited by the device hosting the Smart Mobile Application.

The periodicity of the request is configured in the application, and can be modified by the ACT Control Service by means of application software version upgrades. The precise geographical position of the smartphone shall not be transmitted to the ACT Control Service. In case of absence of underlay network connectivity, the ACT Smart Mobile Application will keep track of all visited geographical areas, and, as soon as the network connectivity will be available, the ACT Smart Mobile Application will issue a sequence of queries related with such areas. The CONTROL-SERVICE-IDs are configured in the ACT Smart Mobile Application: many CONTROL-SERVICEIDs can be then stored in the ACT Smart Mobile Application; this allow to correlate the

ACT Control Service with the given Municipality/Department/Region/Country it is currently depending in.
The ACT Control Service shall respond to the ACT Smart Mobile Application with a list of replies specified in the following table.

Table 2.9.5-2

Parameter name	**Type**	**Description**
REPLIES	LIST-OF-REPLY	List of replies indicating the Peripheral Services with the related RED forecasts. Defined in Table 2.9.5-3.
MESSAGE	STRING	When present, it shall be displayed by the Smart Mobile Application to provide guidance to the users.

Each element corresponds to an ACT Peripheral Service that:

- geographically belongs to the geographical area indicated in the received QUERY;
- is associated to a positive virus detection (the period under consideration is defined by the ACT Control Service).

The response also includes the time periods of RED-FORECASTS associated to each of the reported PERIPHERAL-SERVICE-ID. The relevance of the contamination is assessed by the ACT Control Services according to National Health Authority policies; only RED cases are reported. The MESSAGE parameter may be included as an optional text message by the ACT Control Service and, when present, shall be displayed to the user of the ACT Smart Mobile Application. The LOCAL-SERVICE-INFO parameter (described in section 2.9.4) is sent to the ACT Control Service that relays it to the ACT Smart Mobile Applications within the REPLIES information parameter. The CONTROL-SERVICE-INFO optional parameter may be present in the case that the ACT Control Service intends to make available to the ACT Smart Mobile Application additional customized information via a dedicated URL.

In the above table the REPLY record-type is composed by a PERIPHERAL-SERVICE-ID, RED-FORECASTS, LOCAL-SERVICE-INFO, and a CONTROL-SERVICE-INFO, and it is defined in the following table.

Table 2.9.5-3

Type	**Field name**	**Type**	**Description**
REPLY	PERIPHERAL-SERVICE-ID	Defined in Table 2.9.2	Defined in Table 2.9.2
	RED-FORECASTS	LIST of RED-FORECAST	List of RED forecast frames. Defined in Table 2.9.5-4.
	LOCAL-SERVICE-INFO	URL	URL pointing to customized information provided by the Local Service (valid for that specific Peripheral Service).
	CONTROL-SERVICE-INFO	URL	URL pointing to customized information provided by the Control Service (valid for that specific Peripheral Service).

In the above table the RED-FORECAST record-type is composed by a time FRAME, and it is defined in the following table.

Table 2.9.5-4

Type	**Field name**	**Type**	**Description**
RED-FORECAST	FRAME	(Time,Time) as defined in ISO 8601.	This parameter indicates the RED time frame of the corresponding Peripheral Service.

Additionally, user tested positive to the virus may instruct the ACT Smart Mobile Application, to inform the ACT Control Service of the potential contamination of

visited locations, by sending the information described in the following table.

Table 2.9.5-5

Parameter name	**Type**	**Description**
PERSONAL-TEST-TIME	Time as defined in ISO 8601.	The time when the smartphone user has been tested positive to the virus (i.e. the time when the sample was taken).
PERSONAL-TEST-CODE	STRING	Reference ensuring that the test is recognized by the National Health Authority. The format is defined by the National Health Authority.
VISITED-BSSIDS	LIST of PERIPHERAL-SERVICE-ID	The PERIPHERAL-SERVICE-ID is defined in Table 2.9.2.

3.6 Information exchanged between ACT Display Application and ACT Control Service

The web-based ACT Display Application (Personal Computer, Tablet, Smart TV, etc.) sends queries to the ACT Control Service for providing information. The parameters are specified in the following table.

Table 2.9.6-1

Parameter name	Type	Description
QUERY	STRING	Geographical area of interest represented using the Geohash Geocode System [11].
FORECAST-FRAME	((Time, Time), STRING) as defined in ISO 8601.	It shall contain two Times, i.e. the time-interval for which a forecast is queried and a STRING that can be: • RED in case the query is asking only for notifying PERIPHERAL-ID with RED forecast; • ALL in case the query is asking for notifying PERIPHERAL-ID with all the kind of forecast.

The ACT Control Service shall respond to the ACT Display Application with a list of replies specified in the following Table 2.9.6-2, where each reply is defined in Table 2.9.6-3. The relevance of the contamination is assessed by the ACT Control Services according to National Health Authorities policies.

Table 2.9.6-2

Parameter name	Type	Description
REPLIES	LIST of REPLY	List of replies indicating the Peripheral Services with the related forecasts. Defined in Table 2.9.6-3.
MESSAGE	STRING	When present, it shall be displayed by the Display Application to provide guidance to the users.

In the above table the REPLY record-type is composed by a DETECTION-AREA and a FORECASTS, and it is defined in the following table.

Table 2.9.6-3

Type	Field name	Type	Description
REPLY	DETECTION-AREA	STRING	Geographical area of application of associated forecast, represented using the Geohash Geocode System [11].
	FORECASTS	LIST of FORECAST	List of FORECAST associated to the DETECTION-AREA.

In the following table we can see that the FORECAST record-type is composed by a FORECAST-COLOR and a time FRAME.

Table 2.9.6-4

Type	Field name	Type	Description
FORECAST	FORECAST-COLOR	It is defined by one of the following STRING values: • GREY • GREEN • YELLOW • RED	The meaning of each color code is defined by the National Health Authority.
	FRAME	(Time,Time) as defined in ISO 8601.	This parameter indicates the time frame of the FORECAST.

The UX/UI of the ACT Display Application should be able to draw some ”animation” of the evolving situation between the required time frame. Optionally, the following parameter may be added to the list, as specified in the following table.

Table 2.9.6-5

Parameter name	Type	Description
MESSAGE	STRING	Used by the Control Service to provide information and guidance to the users of the Display Application.

3.7 Information exchanged between different ACT Control Services

The ACT Control Services should be able to communicate each other's to share the ACT related information across the domains of the different National Health Authorities. The ACT Control Service may query each other's, as specified in the following table.

Table 2.9.7-1

Parameter name	Type	Description
QUERY	STRING	Geographical area of application of query, represented using the Geohash Geocode System [11].

The target ACT Control Service(s) is/are selected according to the geographical area of competence, and the query is then sent on the corresponding communication framework interface (i.e. the oneM2M Mcc' API). The ACT Control Service shall respond with a list of replies specified in the following table (Table 2.9.7-2), where each reply is defined in Table 2.9.7-3. The relevance of the contamination is assessed by the relevant ACT Control Services according to relevant National Health Authorities policies.

Table 2.9.7-2

Parameter name	Type	Description
REPLIES	LIST of REPLY	List of replies indicating the Peripheral Services with the related forecasts. Defined in Table 2.9.6-3.
MESSAGE	STRING	When present, it provide guidance between Control Services.

In the following table the REPLY record-type is composed by a PERIPHERAL-SERVICE-ID, DETECTION-AREA, and a FORECASTS.

Table 2.9.7-3

Type	Field name	Type	Description
REPLY	PERIPHERAL-SERVICE-ID	Defined in Table 2.9.2	Defined in Table 2.9.2
	DETECTION-AREA	STRING	Geographical area of application of associated forecast, represented using the Geohash Geocode System [11].
	FORECASTS	LIST of FORE-CAST	List of FORECAST associated to the DETECTION-AREA.

In the following table the FORECAST record-type is composed by a FORECAST-COLOR and a time FRAME.

Table 2.9.7-4

Type	Field name	Type	Description
FORECAST	FORECAST-COLOR	It is defined by one of the following STRING values: • GREY • GREEN • YELLOW • RED	The meaning of each color code is defined by the National Health Authority.
	FRAME	(Time,Time) as defined in ISO 8601.	This parameter indicates the time frame of the FORECAST.

3.8 Indoor localization accuracy using WiFi without GPS

As we have seen the ACT uses WiFi networks to identify the area the user is in, this is an implementation of indoor localization. For indoor localization we mean those techniques that allow to identify the position of objects or users inside a room. These techniques differ from outdoor ones because outdoors it is easier to obtain the user's position thanks to the use of GPS which, through the use of satellites, allows the exact location of the user. Furthermore, the outdoor environment is larger, undergoes less changes and is described with a lower level of detail than the indoor environment. On the other hand, to carry out the localization in the indoor environment it is possible to use the GPS but it would have a low level of accuracy, so today many systems are tested that use the WiFi infrastructure already present in the buildings, in this way it is possible to do indoor localization using WiFi. This technique, however, generates new problems due to various factors [44]: in the first instance it is necessary to calibrate the WiFi signal strengths according to the room in which we are, also we must take into account the fact that buildings have a lot of interference due to for example the very high amount of WiFi networks present or the use of other wireless

technologies such as Bluetooth. Furthermore, the signal strength received by the WiFi transmitter changes radically according to the obstacles it encounters, for example there is a large difference in power if the signal encounters an open or closed door, if it encounters a wall made of a certain material rather than another and according to the arrangement of the furniture. There are techniques that partially solve these problems by increasing the percentage of accuracy of the system using supervised machine learning techniques, especially K-nearest neighbors [14] that allow you to classify the positions of users by saying whether or not they are in that particular room [46] [47]. These techniques first require a training phase where the user visits the building in which he/she is located and makes surveys at various points in the building where he/she detects the signal strength of the surrounding WiFi networks. In this way, when it is located in a point of the building that has not been previously tagged, the algorithm will classify it based on the similarity between the signal strength of the newly detected WiFi networks and those already present in the training set. As far as the ACT standard is concerned, this high level of accuracy is not necessary because the aim is not to precisely locate the user but rather to define an area within which he/she is located. In fact, we use the coverage range of the WiFi network as a demarcation area so that if the user detects the network, and therefore is within its coverage area, we can define the area in which it is located.

Chapter 4

Design

4.1 Understand the subject, the state of the art, the goals, explore the platform

The objective of this task is to study the state of the art and the components already developed in order to understand the context and the state of development of the project in general. To do this, it was initially necessary to study the ETSI Technical Specification TS 103 757 [43] which best describes the protocol created, describing in detail what each component does and with whom it communicates. In this way it was possible to outline the subsequent tasks related to the development of the missing components. Due to the fact that, from the point of view of implementation, Asynchronous Contact Tracing uses the TIM ICON© oneM2M platform to develop many of the components required by the protocol, it was first necessary to study the ETSI oneM2M standard [26] and then the platform TIM ICON© itself. In this way it was also possible to become familiar with the platform in such a way as to be able to use it for testing, component development, experiments. Subsequently it was necessary to study the documentation relating to the Geohash mechanism so as to be able to subsequently exploit it during the software development phases.

4.2 Finding the best WiFi Analyzer Mobile Application

The search for the best WiFi Analyzer Mobile Application already present on the Play Store was made taking into consideration various factors:

- Number of app downloads
- Open source code or not
- Readability of the code
- Application weight
- Application design level
- Programming language used

This research led to the choice of the WiFi Analyzer© app developed by VREM Software Development [36]. The latter is an Open Source application available for smartphones with Android operating system developed using the Kotlin programming language that allows you to scan the WiFi networks around the user indicating SSID and BSSID, power in decibels, signal quality, distance estimation from the source in meters, graph relating to the quality of the channels and graph relating to the signal strength over time. It is also possible to choose the scan interval and export the information obtained by scanning the networks. The original project of the application was then forked using the relative GitHub, studied and subsequently modified.

4.3 Study the WiFi Analyzer© and the Kotlin language

As mentioned above, it was necessary to study the WiFi Analyzer© chosen so that it could be modified to our liking, satisfying the requirements of the ACT protocol. To do this it was first of all necessary to study the Kotlin programming language that was used for the development of this application, once this was done it was necessary to study the internal architecture of the application. This involved the study of

all the activities, packages, classes and interfaces present in the application and interconnected with each other in such a way as to understand their responsibilities, dependencies, characteristics and behaviors in general. Once this is done, I moved on to the definition of the components to be modified and created on the basis of the requirements imposed by the ACT protocol. Regarding the development of the graphic user interface, it was decided to keep the interface already present because, showing all the data necessary for the management of WiFi networks, it has all the features required for the development of the new functions required by the ACT protocol.

4.4 Mockups development web and mobile

Regarding the development of the user interface related to the mobile application, as mentioned above, the native user interface of the forked mobile application has been maintained. Regarding the development of the user interface relating to the web application, it was decided to implement a map that shows the user where the virus outbreaks currently present are located so as to allow him to know the places where there is a greater risk of infection. In the future, the web application will also allow for predictions of future outbreaks and will do data analysis in general.

4.5 Mobile application development

The objective of this task is to develop the component which in the Asynchronous Contact Tracing protocol is called ACT Smart Mobile Application. This is a mobile application developed for smartphones with Android operating system that allows you to scan the WiFi networks around the user, filter them based on the SSID considering only the networks that are part of the ACT infrastructure, subsequently these networks are stored within the smartphone using a local database taking into account the timestamp of when the network has been detected and the timestamp of when the network is no longer detected. In this way, a time interval is created that indicates exactly the time in which the user has been within the coverage range of that particular repeater and therefore, if traces of viruses are subsequently detected within that

area that could having previously infected the users who have passed through that zone, a notification is sent to those users so as to inform them of the risk of infection and invite them to take a test. To find out if traces of viruses have been detected in the places frequented by a particular user, the latter sends through the app his current position in the form of Geohash to the Control Service, in this way the real exact position of the user will never be provided but a different position located within the rectangle created by the Geohashing function. Subsequently, the Control Service will select all the infected Peripheral Services with the relative positive time intervals and will communicate them to the user. The user will compare the information received from the Control Service with that present in his local database and, in the case of matching, will be informed of the possibility of having been infected by the virus.
A fork of the previously chosen WiFi Analyzer© application was used for the development of this application. Initially, it was first of all necessary to understand how the app is structured in such a way as to understand which classes to modify and which to create from scratch. It was therefore necessary to develop the missing functionalities in the native application but necessary for our project. These functions are:

- function relating to the filtering of networks on the basis of the SSID so as to consider only those that are part of the ACT protocol,
- function relating to filtering networks based on the distance from the access point. This feature is useful in the performance evaluation phase because it allows you to understand, based on the environment in which the ACT protocol is applied, which are the networks that must be considered to avoid false positives as much as possible,
- creation of a foreground service that allows the app to run in the background so as to perform network scans and all the functions subsequently developed while being in the background,
- function relating to the definition of the interval that indicates the time frame in which the user has detected that particular network,
- creation of the local database that contains, for each network, the time intervals

that indicate from when to when that network was detected queries relating to the search within the database and the insertion and updating of data,

- function relating to obtaining the position and converting to Geohash,
- function relating to sending the position in the form of geohash to the Control Service through a call to RESTful API,
- function related to the management of user permissions related to the use of the application in the background, obtaining the position, use of the WiFi receiver.

4.6 oneM2M TIM ICON© interfacing and components development

The goal of this task is to understand what TIM ICON©'s oneM2M platform is and how it works and subsequently develop the components of the ACT protocol by integrating them with it. For this reason, it was initially necessary to study the IoT oneM2M protocol first, then I moved on to the study of the TIM ICON© platform and the RESTful APIs that allow communication with it. Once I acquired the necessary knowledge, I moved on to the development of the ACT protocol components that are part of the TIM ICON© platform: Detection Service, Local Service, Control Service. Finally, a simulation of the National Health Authority was developed in order to test the system in its entirety. For the development of the entire system, a server was created with various APIs that allow the sending of JSON as described by the ACT protocol. Initially the API is executed which receives a JSON containing the information described by the protocol from a POST call, once this information is received a POST call is made on the TIM ICON© platform which stores the data received by the Detection Service. It was decided to use the TIM ICON© platform due to the fact that the entire system processes sensitive data which, if not properly processed, would allow us to know the health status and location of users. By using TIM ICON©, on the other hand, you have maximum security that the information uses channels that are impossible to intercept, thus avoiding the problem of illicit appropriation of informa-

tion. Thanks to TIM for the access credentials provided, thanks to which it is possible to use the services provided. Subsequently, thanks to the TIM ICON© Subscription / Notification mechanism, a second POST is launched to the Local Service which, as per protocol, adds other data to the JSON received. Once this is done, a POST to TIM ICON© is launched to load this JSON on the relevant TIM ICON© container. At this point the Subscription / Notification mechanism is used again to launch a POST to the Control Service. The Control Service, once received this POST request, forwards the information to the National Health Authority which decides what color to give to the receiver for which the test was performed. Once this is done, the new JSON is returned to the Control Service which extracts the necessary information to be loaded into its database and finally performs a POST request to TIM ICON© in order to load the new JSON on its container.

4.7 Web application development

The objective of this task is to develop the component that within the Asynchronous Contact Tracing protocol is called the ACT Display Application. This is a Web Application that shows users an interactive map by means of which the user can view red ACT Peripheral Services, i.e. indicating places where users are at risk of being infected. Through this Web app, the user can draw one or more polygons within the map to display the positive Peripheral Services by means of dots and, by clicking on the various dots, can see information about it. It is also possible to apply a time filter by defining a search time interval. In this way, once the filter is enabled and the area of interest has been drawn, markers relating to positive Peripheral Services during the selected time interval will appear inside.

Chapter 5

Implementation

In this chapter I will describe how they were developed: the mobile application, the interface between TIM ICON and the other components, the web application. As notational conventions it was decided to use `verbatim` to indicate file and class names and *italics* to indicate the name of components present in the Android operating system, the name of the JSON object fields, the name of some components of the web application. Regarding copyright, I decided to apply the Apache v2 license. The entire code is visible on the following GitHub page: `https://github.com/AlessioDD98/Asynchronous_Contact_Tracing`

5.1 Mobile application development

In this section we will see how the ACT Smart Mobile Application was developed, focusing more on how it was possible to extend the forked WiFi Analyzer© application and how the most important functions were developed.

5.1.1 Description of the functions already present

As for the extension of the application, it was first of all necessary to understand how the various classes interact with each other and what their task is. From the study of the application it has been understood that when the application is started the `MainActivity.kt` is launched which instantiates the `MainContext` class, the

application settings and checks that the user has given all the necessary permissions to use the application. The `MainContext`, once instantiated, takes care of checking if the smartphone WiFi is active, if it is not, it opens the settings screen relating to the networks so as to allow the user to enable it, finally it instances the `ScannerService` class. It is a class that deals with:

- manage the scanning of WiFi networks by instantiating the `Scanner` class. This class enables WiFi, checks if the user has given all the necessary permissions to the application and starts scanning for networks,

- manage the behavior of the application when the scan is successful through the instantiation of the `ScannerCallback` class,

- perform scans at regular intervals as long as the application remains open by instantiating the `PeriodicScan` class. This therefore implies the fact that if the user returns to the smartphone home, the application stops working because it is not enabled to work in the background,

- define a Broadcast Receiver through the class `ScanResultsReceiver` which is called by the operating system every time the scan is successful and which in turn calls the `ScannerCallback`. The `ScannerCallback` then calls the `Cache` class, specifically the Add function.

This function allows you to manage the information received as scan results. For this reason, it has as input parameters the list of WiFi networks obtained and the information associated with each network, these information are: SSID, BSSID, signal strength in Decibel, estimate in meters of the distance from the access point, information relating to the manufacturer of the access point.
This application also allows you to find the best WiFi channel available and create graphs showing how the signal strength varies over time and the quality of the networks. This part of the application will not be explained in detail because it is not relevant in relation to the extensions I have developed.

5.1.2 Extensions development

The first extension that was created in such a way as to provide the application with the functionality required by the Asynchronous Contact Tracing protocol was a Splash screen that is launched when the application is started. Through it, it is possible to choose whether to start the application in *normal* mode, that is without the extensions of the ACT protocol or in *ACT* mode, that is with the extensions of the ACT mode. This activity, once launched, instantiates the database inside the application and shows 2 buttons on the *"splash" screen*:

- the button with the wording "Normal" which will start the *normal* mode and which after pressing it will add the value "normal" to the mode field of the Shared Preferences.

- the button with wording "ACT" that will start the *ACT* mode and that after pressing it will add the ACT value to the mode field of the *Shared Preferences*.

In both cases, once one of the two buttons is pressed, the `Main Activity` will be launched.
Regarding the `Main Activity`, some small changes have been made in order to avoid the automatic start of the `Periodic Scan` class. This was done because a *Foreground Service* was developed that allows periodic scans to be carried out in the background. This *Foreground Service* is called `ACTScanner`. Once the `Main Activity` has been launched in *ACT mode*, the execution of the `ACTScanner` is then called up which initially creates the notification that will remain fixed and which will notify the user of the execution of the process in the background, subsequently a *Looper* is instantiated, that is a process that runs in the background indefinitely. The Update function is called within the *Looper*. This method enables WiFi if disabled, checks if the user has given the necessary permissions to scan networks, if yes, starts the scan and then calls the `ScannerCallback` which will manage the results obtained from the scan. An interval between one scan and the next is also set at the user's choice in the settings menu so that the *Looper*, after calling the *Update* function, goes into sleep mode for the number of seconds chosen by the user. As previously said, the

management of the scan results is entrusted to the `ScannerCallback` which, when called, calls the *Add* function of the `Cache` class. Thanks to this function, you can then view the scan results and all information relating to the networks found. This function also allows you to filter the networks detected by name, this is necessary because the ACT protocol accepts only networks for which the SSID starts with "ACT-" and manages the insertion and updating of data in the local database. The database created is called *NetworkDatabase* and contains a table called *Network* . Its purpose is to store the detected networks together with the timestamp of the first detection of that particular network (TimeFound) and the timestamp of the first detection that no longer finds that particular network (TimeLost). For this reason the table is made up of 4 fields: SSID, BSSID, TimeFound, TimeLost. The BSSID and the TimeFound have been defined as primary keys, this is because in the future it is possible to find again a network already detected but it will be made unique with respect to the same network found at a different TimeFound. The database was developed using the *Android Room framework*, the following queries were defined:

- insertion of a new detected network,
- update of an already detected network to set the TimeLost,
- search for networks with TimeLost not yet defined,
- filtered search for networks with TimeLost not yet defined based on BSSID.

The last two queries have been defined because the main task of the database is to memorize when the network was found for the first time and when it was no longer detected, only when we have both timestamps will it be possible to add another row to the table in reference to the same network but with different TimeFound. To define the TimeLost, a special algorithm has been developed in the file `Cache.kt`, visible in the Listing 5.1. An empty list array called *listNetwork* is defined which will contain all the networks found during the scans, once the results of a scan have been obtained, for each network, it is checked whether it is already present in *listNetwork*, in case it does not is present, it is added and it is checked if the same network is

present in the database but with an undefined TimeLost, if it is not present, the network is added to the database by setting the current timestamp as TimeFound. In other words, in this way we add to the database only networks or not yet present in it or networks present but with the TimeLost different from NULL. Subsequently, for all the networks present in *listNetwork* it is verified that they are also present in the list of networks just detected, if a certain network is not present it means that it has not been detected and therefore we must update its entry in the database in such a way as to define the TimeLost. An update of that network is then made within the database defining the current timestamp as TimeLost. Once this is done, that network is deleted from *listNetwork* so that if it is found again in the future, it will then be added to *listNetwork* and consequently to the database. Alternatively, it is possible to define the TimeLost by closing the application, for this reason a query is called in the *OnDestroy* function of the *ForegroundService* that returns all the networks with TimeLost equal to NULL and updates them by setting the current timestamp as TimeLost. In order not to overload the Main Thread and avoid blocking the UI, each query is inserted inside its *AsyncTask*.

```
var networksAlreadySeen= getAllNetworks(MainContext.INSTANCE.mainActivity.
    application).execute().get()
 for(r in scanResults){
   if(!listNetwork.contains(r.BSSID)){
     listNetwork.add(r.BSSID)
     if(networksAlreadySeen.filter {
       s->s.BSSID==r.BSSID && s.TimeLost==null
     }.isEmpty()){
       var n= Network(r.SSID,r.BSSID,Date(),null)
       InsertNetwork(n,MainContext.INSTANCE.mainActivity.application).
    execute()
     }
   }
 }
 for(l in listNetwork){
   if(scanResults.filter {
     scanResult -> scanResult.BSSID==l
   }.isEmpty()) {
```

```
17      var networkFiltered = networksAlreadySeen.filter {
18        s ->s.BSSID == l && s.TimeLost == null
19      }
20    if(networkFiltered.isNotEmpty()){
21      var n= Network(networkFiltered.get(0).SSID,l,networkFiltered.get(0).
    TimeFound,Date())
22      UpdateNetwork(n,MainContext.INSTANCE.mainActivity.application).
    execute()
23      del=true
24      remListNetwork.add(l)
25    }
26  }
27 }
28  if(del){
29    for(i in remListNetwork){
30      listNetwork.remove(i)
31    }
32    del=false
33    remListNetwork=ArrayList()
34  }
```

Listing 5.1: Algorithm for adding and updating networks inside the database

Later a second *Foreground Service* called `ACTPosScanner` was developed. The latter, once called, instantiates a *Looper* that executes a method that allows you to obtain the user's position and convert it into Geohash. To do this it is first of all necessary to check that the user has provided permissions to access the location and has activated the GPS, after which the *Fused Location Provider Client* is instantiated, which is the system service that manages the location, and is requested the position. Once obtained thanks to the *ch.hsr.geohash*[1] library, the position obtained in the form of latitude and longitude is converted into Geohash defining the length of the generated string. Once this is done, a REST request is made to the API contained in the Control Service which, once the Geohash has been received, provides the application with all the BSSIDs of the Peripheral Services in which the virus was detected with the relative positive inter-

[1] `https://github.com/kungfoo/geohash-java/blob/master/src/main/java/ch/hsr/geohash/GeoHash.java`

vals contained within the area generated by the Geohash. Once this information has been obtained, it is checked which Peripheral Service the user has come into contact with by matching the Control Service response and the information contained in the local application database and, if there are any matches, checks are made on the time intervals. The requests were implemented using the Volley framework[2]. Through the latter, a POST request is created containing a JSON with the Location field to which the Geohash is associated inside. Once the response from the Control Service has been obtained, it is necessary to properly extract the information inside and check the various time interval cases as shown in the Listing 5.2. This is due to the fact that the standard says that the response received will be a JSON document containing an array of infected Peripheral Services with associated time slots of positivity. For this reason the first thing to do once the answer was obtained was to convert it into a JSON array, in this way it was possible to create a loop that extracts the information from all the elements of the array. This loop, at each iteration, obtains a Peripheral Service ID, which corresponds to the BSSID, then queries the application database to see if the user has detected that Peripheral Service, in the positive case the time interval is extracted from the JSON of positivity of that Peripheral Service and is compared with the time interval contained in the application considering various cases:

- the first case is that in which the time interval of the user is contained within the time interval of the response, this means that the user was at risk of infection for the entire time he/she was inside that particular area,

- the second case is the case in which the user arrived in that area when it was not yet infected but while he/she was there it became infects. This means that the TimeFound is less than the start time of the infection but the TimeLost is greater,

- finally, the last case is the one in which the TimeLost has not yet been defined because the user is still within that area. In this case it only occurs if the TimeFound is greater than the time of onset of the infection.

[2] `https://google.github.io/volley/`

```
if(networkFiltered.isNotEmpty()){
 for(x in networkFiltered) {
   println("ELE: $x")
   val rf=rep.getJSONObject(i).getJSONObject("RED-Forecasts")
   var ts=rf.getString("Start")
   var te=rf.getString("End")
   ts=ts.replace("T"," ")
   te=te.replace("T"," ")
   println("TS: $ts\n TE: $te")
   var tf=x.TimeFound
   var td=x.TimeLost
   val formatter = SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
   val dataTS = formatter.parse("2022-06-09 14:35:00.000000")
   println("DATA START: $dataTS")
   val dataTE = formatter.parse("2022-09-02 14:35:00.000000")
   println("DATA END: $dataTE")
   //Time interval check
   if(tf>=dataTS && td!!<=dataTE) {
     notificationManager.notify(2, notification.build())
   }
   else if(tf<dataTS && td!! >=dataTS){
     notificationManager.notify(2, notification.build())
   }
   else if(tf>=dataTS && td==null){
     notificationManager.notify(2, notification.build())
   }
 }
}
```

Listing 5.2: Algorithm that check the time interval considering the different cases

Finally, the functionality of filtering networks based on the distance from the access point has been added using a function already present in the original application that provides the user with the distance for each network. This function calculates the distance between the user and the access point in meters based on the strength of the received signal and the frequency of the network using the formula ($distance$ is

calculated in meters):

$$distance = 10^{((27.55-(20*log10(frequency))+signalLevel)/20)}$$

This formula is derived from the formula that calculates the Free Space Path Loss (FSPL) [10]. Once the distance between the access point and the user is calculated for each network, the distance filter value selected by the user is checked and, if the distance of the network is less than the value of the filter value, the network is added to the vector of valid networks.

5.2 oneM2M TIM ICON© interfacing and components development

In this section we will see how the components of the Asynchronous Contact Tracing protocol: Detection Service, Local Service, Control Service protocol and the simulation of the National Health Authority have been developed. Each of the components should be a separate server thus creating a communication system between the various servers. Since it is not possible to reproduce this case, only one server has been created with various APIs that allow you to perform the tasks of each component. This server communicates with the oneM2M TIM ICON© platform through the APIs accessible thanks to the credentials provided by TIM. To allow the TIM ICON© platform to send replies to the server it was necessary to obtain a public address, to do this the Ngrok[3] tunneling mechanism was used. All components were developed using Python programming language and the Flask framework[4] was used to create the server.

5.2.1 Detection Service development

An API has been developed which receives POST calls from a simulated Peripheral Service containing a JSON and which in turn creates a JSON to be loaded on

[3] https://ngrok.com/

[4] https://flask.palletsprojects.com/en/2.2.x/

TIM ICON©. This JSON exactly matches the JSON defined by the ACT protocol. As shown in the Listing 5.3, this API receives the JSON inside the message variable and inserts it inside a JSON called payload which contains the structure required by TIM ICON© to add entries to the containers. Finally, through the requests library[5], a POST request is made to the identifying URL of the Detection Service container within TIM ICON©. A subscription has been defined inside the TIM ICON© container relating to the Detection Service that allows you to call the API relating to the communication between Detection Service and Local Service every time a container instance is added. Finally, with regard to the Detection Service, an API has been developed that manages the JSON response defined in the ACT protocol through a POST call, which allows communication between Local Service and Detection Service.

```
def DetectionNodeData(message):
    payload={
      "m2m:cin": {
      "cnf": "application/json:0",
      "con": message
      }
    }
    request= requests.post(url= "https://icon-lab.tim.it/onem2m/act_test/
   ACT_DetectionService", headers=headers, auth= HTTPBasicAuth('act_test',
    '@Test_99'),
      data=  json.dumps(payload))
    return request.json(), request.status_code
```

Listing 5.3: API for communication between Detection Service and TIM ICON

5.2.2 Local Service development

With regard to the Local Service, an API has been developed that receives a POST call containing a JSON and integrates the information received with the information generated by the Local Service itself. In fact, its purpose, in addition to receiving the test results from the Detection Service, is to register information about area's disinfection and subsequently build a new JSON to be communicated to the Control Service

[5] https://pypi.org/project/requests/

containing all the information relating to the test carried out by the Detection Service and the disinfestation carried out by the cleaning crew. For this reason, after creating the new JSON, the latter is sent via a POST request to the container relating to the Local Service within TIM ICON© in order to create a new container instance containing that JSON. As you can see in the Listing 5.4, this API receives a JSON from the Detection Service and adds the fields: *Location* in Geohash type, *Disinfection* in boolean type, *Disinfection_Time* in timestamp type, *Peripheral_Service_Id*, *Test_Result*, *Test_Time* which are extracted from the received JSON. It then inserts all the information into a JSON and performs a POST request to the URL of the Local Service container within TIM ICON©. A Subscription has also been defined inside the container in TIM ICON© which, every time a new container instance is inserted, sends a POST request containing the contents of the container instance just created to the address associated with the Control Service. Finally, the API was developed that allows sending the JSON defined by the ACT protocol for communication between Local and Detection Service.

```
def LocalToControlService(message):
    x = message['m2m:sgn']['nev']['rep']['m2m:cin']['con']
    Disinfection= True
    Disinfection_Time= datetime.now().isoformat()
    geohash_code = pgh.encode(30, -6.9)
    dict= {'Location': geohash_code,'Disinfection': Disinfection, '
   Disinfection_Time': Disinfection_Time,
           "Peripheral_Service_Id":x["Peripheral_Service_Id"],"Test_Result
   ": x["Test_Result"],"Test_Time": x["Test_Time"], 'Location_Service_Info
   ': 'https://NationalHealth.com/forecast'}
    payload= {'m2m:cin': {'cnf': 'application/json:0', 'con': dict}}
    new_payload= json.dumps(payload)
    write = requests.post(url= "https://icon-lab.tim.it/onem2m/act_test/
   ACT_LocalService", headers=headers, auth= HTTPBasicAuth('act_test', '
   @Test_99'), data=  new_payload)
    return write.json(), write.status_code
```

Listing 5.4: API for communication between Local Service and TIM ICON

5.2.3 Control Service development

The purpose of the Control Service is to receive information from the Local Service relating to its coverage area and forward it to the National Health Authority (NHA) so that it can decide whether to quarantine the area and then notify users who have visited that area informing them about the possible risk of infection. Subsequently, the Control Service will receive in response from the National Health Authority a JSON containing the information previously sent in addition to the Forecast Color and infection risk time interval defined by the NHA and, if the forecast color is red, it will store this information in its database and inside its TIM ICON© container. The developed database is non-relational and was developed using MongoDB [21]. In this way, the Control Service can communicate to the Smart Mobile Application and the Display Application the information relating to the areas considered at risk.
Communication between Control Service and NHA does not take place via TIM ICON© because NHA is not a oneM2M component. For this reason, the API shown in the Listing 5.5 has been developed that allows to send, via a POST call, a JSON to the NHA. In this API the Control Service receives a JSON and through the requests library makes a POST call to the URL of the NHA that does not belong to TIM ICON©, for this reason it is hosted on a separate server which can be accessed via the URL provided by Ngrok tunneling.

```
1 def ControlServiceToPHA(message):
2     x = message['m2m:sgn']['nev']['rep']['m2m:cin']['con']
3     header = {'content_Type': 'application/json'}
4     x=json.dumps(x)
5     write = requests.post(url="http://84a6-46-193-67-245.ngrok.io/PHA",
    headers=header, data=x)
6     return write.status_code
```

Listing 5.5: API for communication between Control Service and NHA

The NHA will respond by sending another JSON containing the additional information relating to the forecast. The trigger to start this API is the notification received from the Local Service. The Control Service in fact receives the notification from the Local Service containing the JSON defined by the ACT protocol and forwards it to

the NHA through a POST call. Subsequently, the Control Service performs the API shown in the Listing 5.6 that check if the *Forecast_color* JSON field's is red, if this is true it stores the following fields in its database: *Peripheral-Service-Id, Location, Timestamp relating to the beginning of the infection risk, Timestamp relating to the end of the infection risk, Local-service-info, Control-service-info*. It then sends a POST request using the requests library to TIM ICON© using the URL of the Control Service container to insert a new container instance inside its container containing the entire JSON received from the NHA.

```
def ControlserviceData(message):
        message["Message"] = 'Message from National Health Service'
    if(message["Forecast_Color"]=="Red"):
            entry={"Peripheral_Service_Id":message["Peripheral_Service_Id"
    ],"Location":message["Location"],"Start":message["Frame"]["Start"],
                   "End":message["Frame"]["End"],"Local_service_info":
    message["Location_Service_Info"],"Control_service-info":"https://
    NationalHealth.com/forecast"}
            collection.insert_one(entry)
            for x in
        payload= {'m2m:cin': {'cnf': 'application/json:0', 'con': message
    }}
        new_payload= json.dumps(payload)
        write = requests.post(url= "https://icon-lab.tim.it/onem2m/
    act_test/ACT_ControlService", headers=headers, auth= HTTPBasicAuth('
    act_test', '@Test_99'), data=  new_payload)
        return write.json(),write.status_code
```

Listing 5.6: API for communication between Control Service and TIM ICON

Regarding the communication between the Smart Application and the Control Service, the user sends the Geohash corresponding to his position within a JSON through the application as shown in the Listing 5.7. Once the Control Service has obtained the Geohash, it searches its database for all the Peripheral Services classified as "red" and returns them to the user. To do this, a query is performed within the database, filtering all the rows based on the user's Geohash, if it finds results it builds a JSON with the fields described by the ACT protocol, otherwise it builds a JSON with an empty field. Finally it sends the resulting JSON to the user as a response to the POST call.

```
def ControlToSMAPP(location):
  message="The area is infected, please get ASAP a PCR test or go to
    isolation"
  result={}
  first=True
  find=False
  i=0
  for x in collection.find({"Location": location}):
      find=True
      if first:
          result= {"Replies":[{"Peripheral_Service_Id":str(x["
    Peripheral_Service_Id"]),"RED_Forecasts":{
            "Start":str(x["Start"]),"End":str(x["End"])},"
    Local_service_info":x["Local_service_info"],"Control_service_info":x["
    Control_service_info"]}],"Message":message}
          first=False
      else:
          result["Replies"].append({"Peripheral_Service_Id":str(x["
    Peripheral_Service_Id"])})
          result["Replies"][i]["RED_Forecasts"]={"Start":str(x["Start"]),"
    End":str(x["End"])}
          result["Replies"][i]["Local_service_info"]=x["Local_service_info
    "]
          result["Replies"][i]["Control_service_info"]=x["
    Control_service_info"]
          result["Message"]=message
      i+=1
  if not find:
      result = {"Replies":[{}]}
  return result
```

Listing 5.7: API for communication between Control Service and Smart Mobile Application

5.2.4 National Health Authority development

The National Health Authority (NHA) is a component not managed by the oneM2M standard but autonomous and independent because it is managed by the government of the country in which the ACT protocol is used. They will define the risk parameters relating to the probability of infection or not on the basis of the advice and studies made by their specialists. For this reason what has been developed is a simulation of the NHA in order to carry out testing and experiments of the system. In this simulation the NHA runs the API shown in the Listing 5.8.

```
def VirusDetection(message,last_dt):
    last_dt=datetime.fromisoformat(last_dt)
    last_dt += timedelta(seconds=1)
    tr=message["Test_Result"]
    if tr > 5:
        message["Forecast_Color"]="Red"
        message["Frame"]={"Start":datetime.strftime(last_dt,"%Y-%m-%dT%H:%
   M:%S.%f"),"End":message["Disinfection_Time"]}
    else:
        message["Forecast_Color"] = "Green"
    last_dt = message["Disinfection_Time"]
    return message, last_dt
```

Listing 5.8: API for communication between NHA and Control Service

In this API the NHA, received the JSON, checks the value of the *Test_Result* and, if it exceeds a certain threshold, adds two new fields inside the received JSON. It adds the *Forecast_color* field assigning it the red value and the *Frame* assigning it as *Start time* the timestamp of the End field of the last detection carried out and as *End time* the timestamp contained in the *Disinfection_time* field of the received JSON. Otherwise, assign the value green to the *Forecast_color* field. Once this is done, the NHA sends a POST request containing the JSON previously described to the Control Service.

5.3 Web application development

The Django framework [7] was used to create the web application. Thanks to this framework it is possible to create web applications by separating the management of the static template from the dynamic code in JavaScript, giving the possibility to write functions using Python programming language. Furthermore, various frontend and backend technologies were used to develop the web application:

- Leaflet [19]: it was used for dynamic map management, thanks to it it is possible to create layers containing map, markers, polygons. I decided to use Leaflet over other mapping libraries because its clear documentation allows faster learning.
- Leaflet geoman [16]: it was used to implement the map drawing functionality, which is the functionality that allows polygons to be drawn on the map.
- Leaflet marker cluster [17]: it was used to implement the marker clustering functionality.
- Turf [35]: it was used to implement the features analysis functions inside the map, specifically thanks to this library it was possible to check if a point is contained within the polygon or if two polygons intersect.
- Leaflet messagebox [18]: it was used to show messages within the map.
- JQuery [13]: it was used for the dynamic management of screen elements by inserting logic into the input fields.
- Openstreetmap[29]: it was used as a free provider to view the map.

The main files of this web app are:

- `homepage.html`: this page contains the layout and style of the site page and a form-type script that manages the communication between the user and the server and the interactions between the elements of the map.
- `homepage.js`: it contains the features implemented with Leaflet, Openstreetmap, Leaflet-geoman, Jquery.

Inside the `homepage.html` file we initially have the header that contains the inclusion of the aforementioned libraries, then the definition of the elements displayed on the screen which are: a *div* that contains the *checkbox* that manages the time filter and two dividers, which they appear only when the user checks the checkbox, which contain two input fields of type date time picker. Next, we have another *div* that contains: the div for the map, the button to start drawing mode and the button to reset the map. Subsequently the div that manages the *messagebox*, the inclusion of the script contained in `homepage.js` and the module script. Finally we have the definition of the style that is applied to the elements, having the buttons inside the map but in the foreground it was necessary to add the property `position:absolute` and `pointer-events:auto` to the style, otherwise the button could not have been clickable. The module script contains: the import of the library that allows to decode the Geohash and the definition of the graphic settings related to the markers. Next, in the Listing 5.9, we find the logic of the reset button which is managed by an event listener that listens to the click event, in this way when the button is clicked it executes a cycle for all the layers of the map removing them all except the Openstreetmap one which would be what show the map. It also clears the array that contains all the polygons present in the map and resets the *clustermarkers*.

```
//RESET BUTTON
Reset.addEventListener('click', function(e){
  map.eachLayer(function (layer) {
  if (layer._url != "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"){
     map.removeLayer(layer);
  }
  console.log(layer._url);
  });
  tfig=[];
  markers = markers.clearLayers();
});
```

Listing 5.9: Algorithm to delete all the layers in the map, reset polygon's array and marker cluster

Next, in the Listing 5.10, we have the definition of the cluster marker which defines the colors that the clusters must have based on how many elements they contain.

Specifically, in our case it was decided to create a green cluster when the number of elements is less than 50, orange when it is less than 200, red above 200.

```
//DEFINE CLUSTER COLOR ACCORDING TO MARKER'S NUMBER
 var markers = L.markerClusterGroup({
 iconCreateFunction: function (cluster) {
     var childCount = cluster.getChildCount();
    var c = ' marker-cluster-';
    if (childCount < 50) {
      c += 'small';
    } else if (childCount < 200) {
      c += 'medium';
    } else {
      c += 'large';
    }
    return new L.DivIcon({ html: '<div><span>' + childCount + '</span></
   div>', className: 'marker-cluster' + c,
     iconSize: new L.Point(40, 40) });
 }
 });
```

Listing 5.10: Definition of cluster group's behaviour

Then, in the Listing 5.11 we have the definition of the behavior after the creation of a polygon event inside the map. According to the requirements of this web app it is not possible to have two polygons that intersect or contain each other, for this reason it was necessary to develop an algorithm to manage this functionality.

```
//CHECK IF THERE IS POLYGON CONTAINS OR INTERSECTIONS
tfig.push(tpolygon);
if(tfig.length>1){
 for(var i=0;i<tfig.length;i++){
   for(var j=0;j<tfig.length;j++){
     if(i!=j){
       if(turf.booleanContains(tfig[i],tfig[j]) || turf.booleanIntersects(
   tfig[i],tfig[j])){
         map.removeLayer(layer);
         tfig.pop(tpolygon);
         box.show( 'You cannot draw polygon that intersect or contains
```

```
    another' );
11          return;
12        }
13      }
14    }
15  }
16 }
```

Listing 5.11: Algorithm that checks and prevents the intersection between two polygons

Within this code snippet we see how initially each polygon created is inserted into the array of polygons, then we check if we have more than one polygon and through two nested loops we check, using Turf [35] library, if there are intersections or containments between all the pairs of polygons, if so, you delete the drawn polygon, remove it from the polygon array and show a messagebox with an error message.
Subsequently, before the asynchronous call to the server to get the data, a check is made in case the time filter is enabled but no time interval has been selected and the user draws a polygon. In this case, the user will receive an error message warning him that if he/she does not indicate the dates he/she cannot draw the polygon.
If the polygon creation is successful, an asynchronous GET call is made to a Control Service API via AJAX. This call returns a JSON containing all the Peripheral Services contained within the database internal to the Control Service. For this reason, checks are made to show the user only the Peripheral Services of interest to him. A specific function called `addPoint` shown in the Listing 5.12 has been developed for adding markers to the map. This function takes as input the JSON obtained from the Control Service, the polygon that the user has drawn and the cluster marker object. Once recalled, it first converts the position contained within each instance of the JSON in the form of Geohash and converts it into latitude and longitude, after which it initializes a point object of the Turf library containing the aforementioned coordinates in such a way as to be able to verify if that point is contained in the polygon. If the answer is positive, the popup is created which will then be shown when the marker is clicked, the marker is defined, the contents of the popup are inserted and it is added to the marker cluster.

```
function addPoint(data, tpolygon, markers){
 var pos=Geohash.decode(data.Location);
 //console.log("Peripheral ID: "+data.Peripheral_Service_Id+" pos: "+pos.
    lon +" "+pos.lat);
 var tpoint=turf.point([pos.lon, pos.lat]);
 if(turf.booleanPointInPolygon(tpoint, tpolygon)){
 const popupContent =
 '<h4 class = "text-primary">Peripheral Service</h4>' +
 '<div class="container"><table class="table table-striped">' +
 "<tbody><tr><td> ID </td><td>" +
 data.Peripheral_Service_Id +
 "</td></tr>" +
 "<tr><td>Risk of infection time start </td><td>" +
 new Date(data.Start) +
 "</td></tr>" +
 "<tr><td> Risk of infection time end</td><td>" +
 new Date(data.End) +
 "</td></tr>";
 var marker = new L.circleMarker([pos.lat, pos.lon],geojsonMarkerOptions);
 marker.bindPopup(popupContent);
 //MARKER CLUSTERING
 markers.addLayer(marker);
 }
}
```

Listing 5.12: addPoint function

Finally, the cases in which the time filter is active was managed. In this case the user must therefore view only the markers that respect that filter, for this reason controls have been added to verify that the times contained in the point fall within the defined time range.

Chapter 6

Results

In this chapter the results produced by the developed software will be shown. There will follow images representing what is shown in output to the user of different types: as regards the ACT Smart Mobile Application and the ACT Display Application, the screens present in the application and in the web application will be shown respectively. As for the ACT Detection Service, ACT Local Service and ACT Control Service will be shown sample images of the instances created within TIM ICON©.

6.1 Mobile application development

When the application is started the user will be in front of the Splash Screen shown in the Figure 6.1. Here you will have two buttons in front of you: one to launch the app in the *normal mode* that is the original mode without the developed extensions, the other to launch the app in the *ACT mode* that is the mode that uses all the extensions developed and previously described.

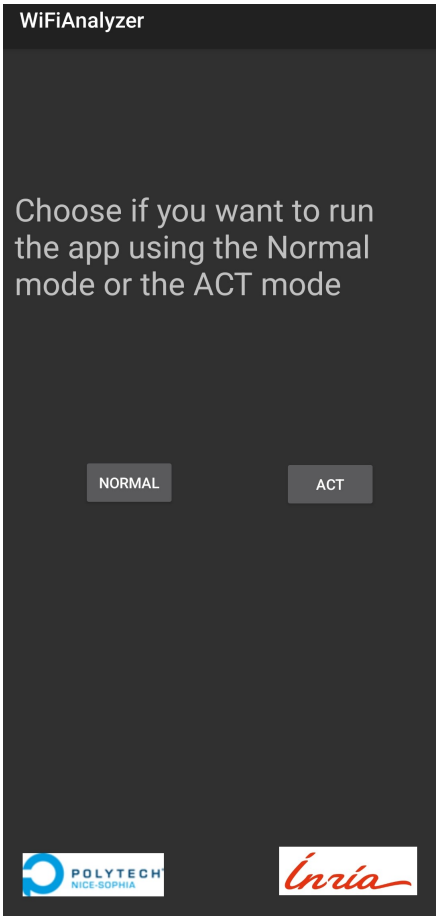

Figure 6.1: ACT Smart Mobile Application Splash screen

After pressing one of the two modes, the user will find himself in front of the screen represented in the Figure 6.2. This screen is the same for both modes and shows the user all WiFi networks found.

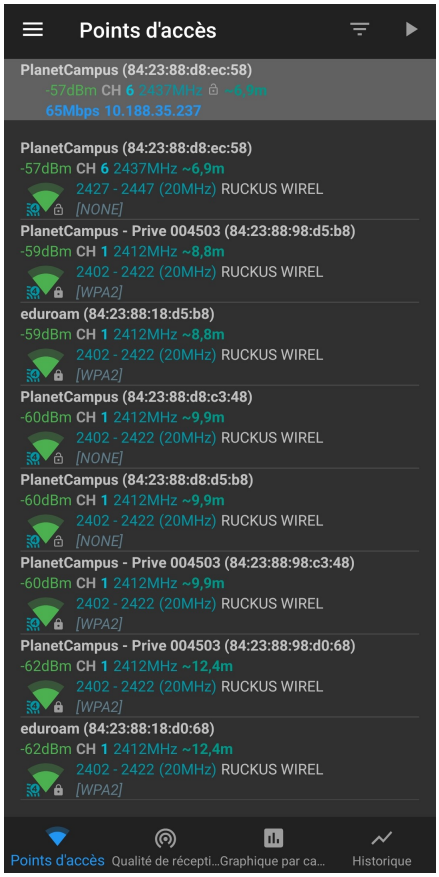

Figure 6.2: ACT Smart Mobile Application Main screen

A difference between the two modes is that if the ACT mode is performed the scanning of the networks is performed in the background as you can see from the notification shown in the Figure 6.3.

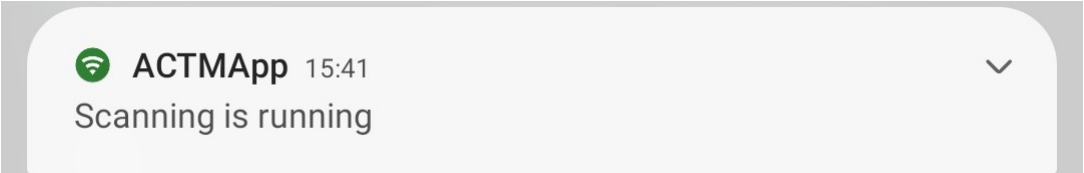

Figure 6.3: ACT Smart Mobile Application Scanning running notification

From this screen you can reach the Settings screen. As shown in the Figure 6.4, 2 new settings have been added to the settings screen: one relating to the scan of the

position to be sent in the form of Geohash to obtain the access points positive for COVID-19 and the other relating to the filter of the distance between user and access points.

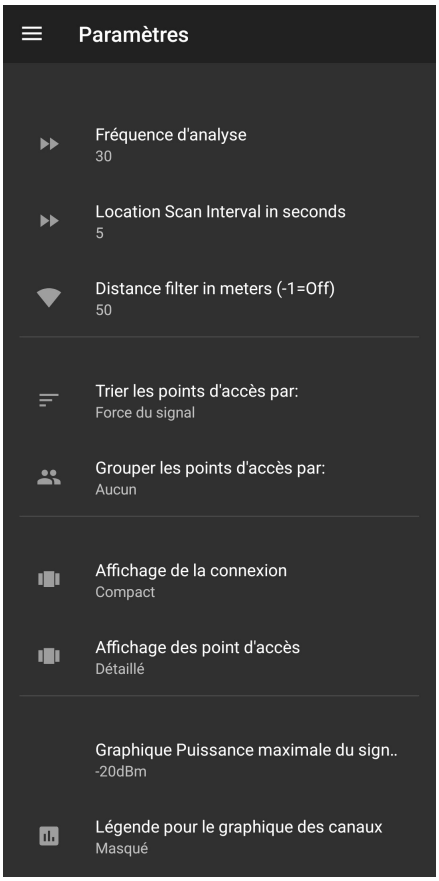

Figure 6.4: ACT Smart Mobile Application Settings screen

Finally, in the event that a matching occurs between the access points received in the application and the access points present in its internal database, the user will receive a notification as shown in the Figure 6.5.

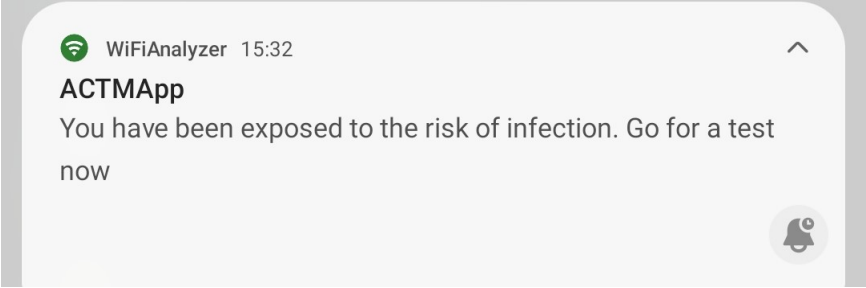

Figure 6.5: ACT Smart Mobile Application COVID-19 exposure notification

6.2 oneM2M TIM ICON© interfacing and components development

As for the communication between the ACT Local Service, Detection Service and Control Service and TIM ICON© components, from the Figures 6.6, 6.7, 6.8 you can see examples of the container instances that are automatically created every time a communication is made between the various components. Each JSON contained within each container instance reflects the format described by the ACT protocol.


```
{
    "Disinfection": true,
    "Disinfection_Time": "2022-08-01T16:47:54.75190
    "Location": "ev74dvrddm74",
    "Location_Service_Info": "https://NationalHealth
    "Peripheral_Service_Id": "C8:60:00:4C:27:A5",
    "Test_Result": 8,
    "Test_Time": "2022-08-01T16:47:53.222335"
}
```


Figure 6.6: Local Service TIM ICON© container instance

```
{
    "Peripheral_Service_Id": "C8:60:00:4C:27:A5",
    "Status": "Active",
    "Test_Result": 8,
    "Test_Time": "2022-08-01T16:47:53.222335"
}
```


Figure 6.7: Detection Service TIM ICON© container instance


```
{
    "Disinfection": true,
    "Disinfection_Time": "2022-08-01T16:47:54.751908"
    "Forecast_Color": "Red",
    "Frame": {
        "End": "2022-08-01T16:47:54.751908",
        "Start": "2022-08-01T16:47:44.681817"
    },
    "Location": "ev74dvrddm74",
    "Location_Service_Info": "https://NationalHealth.
    "Message": "Message from National Health Service"
    "Peripheral_Service_Id": "C8:60:00:4C:27:A5",
    "Test_Result": 8,
    "Test_Time": "2022-08-01T16:47:53.222335"
}
```


Figure 6.8: Control Service TIM ICON© container instance

6.3 Web application development

As shown in the Figure 6.9, the ACT Display Application homepage contains the filters section, better shown in the Figure 6.10, where, if selected, it is possible to apply the filter by time interval and the map containing the *Draw polygon* button to draw a polygon and *Reset* button to delete the polygons present.

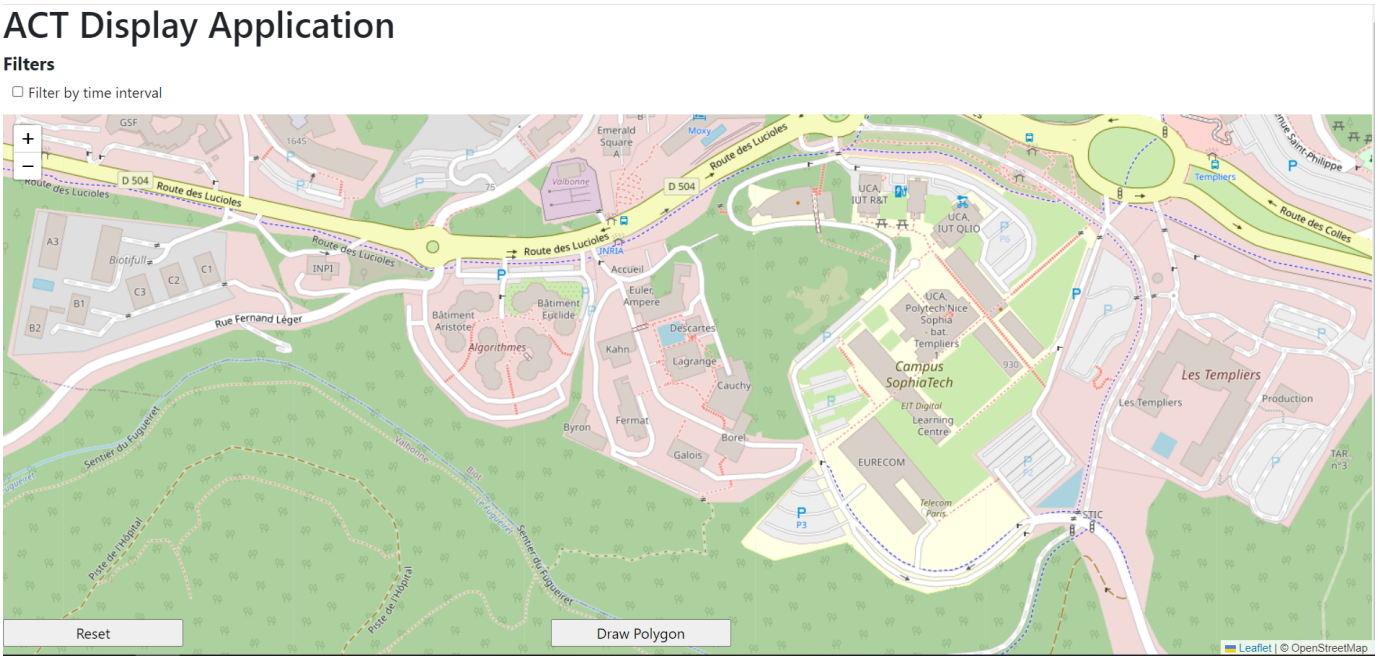

Figure 6.9: ACT Display Application homepage

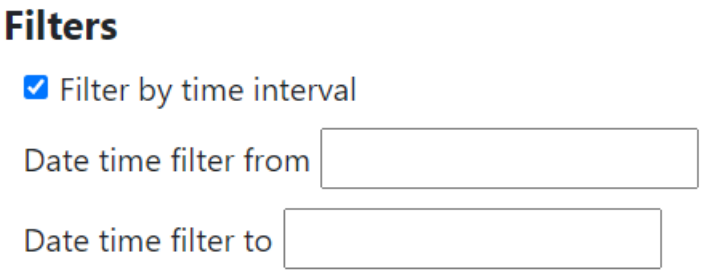

Figure 6.10: Filters section

After pressing the Draw polygon button you can draw any polygon. After finishing the drawing, the clusters will be shown that contain the markers related to the WiFi positive access points inside the polygon as shown in the Figure 6.11.

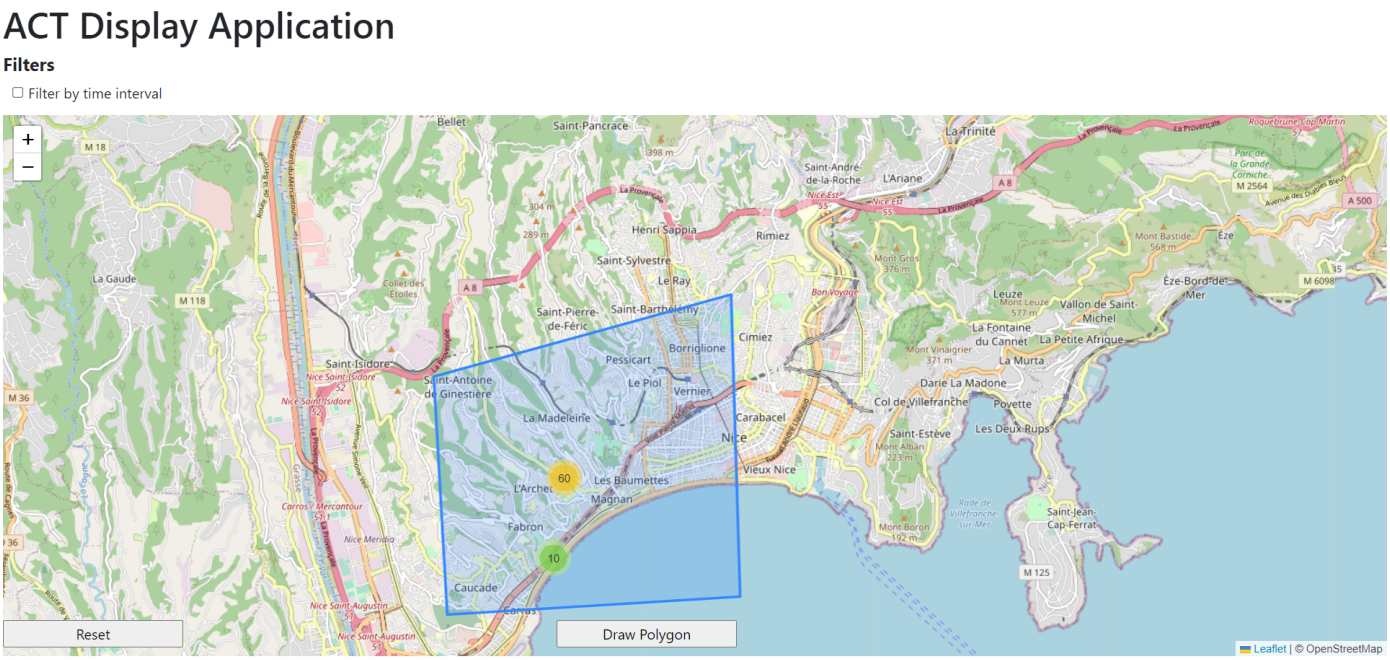

Figure 6.11: Homepage showing cluster

Pressing on a cluster, the latter will be opened and, as shown in the Figure 6.12, all the markers inside it will be shown.

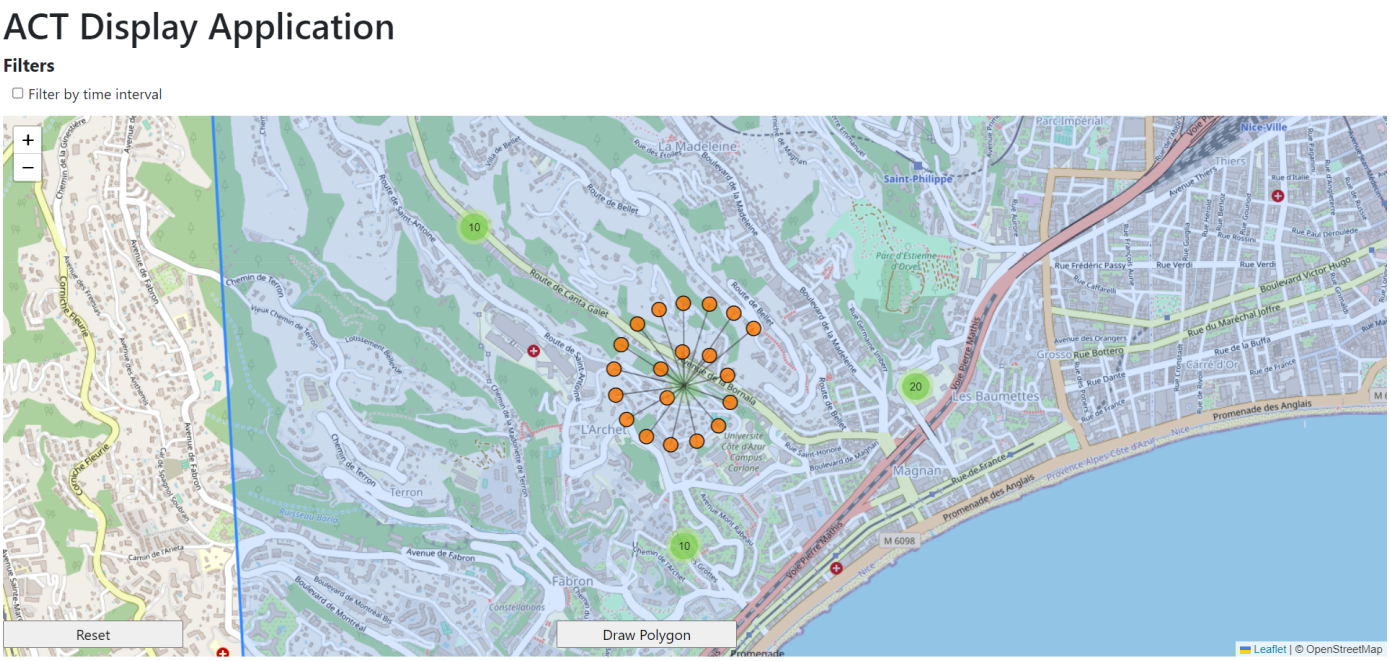

Figure 6.12: Homepage showing markers inside cluster

Finally, as shown in the Figure 6.13, pressing on a marker will open a popup that shows information about the BSSID of the access point and the time interval in which there was the risk of being infected.

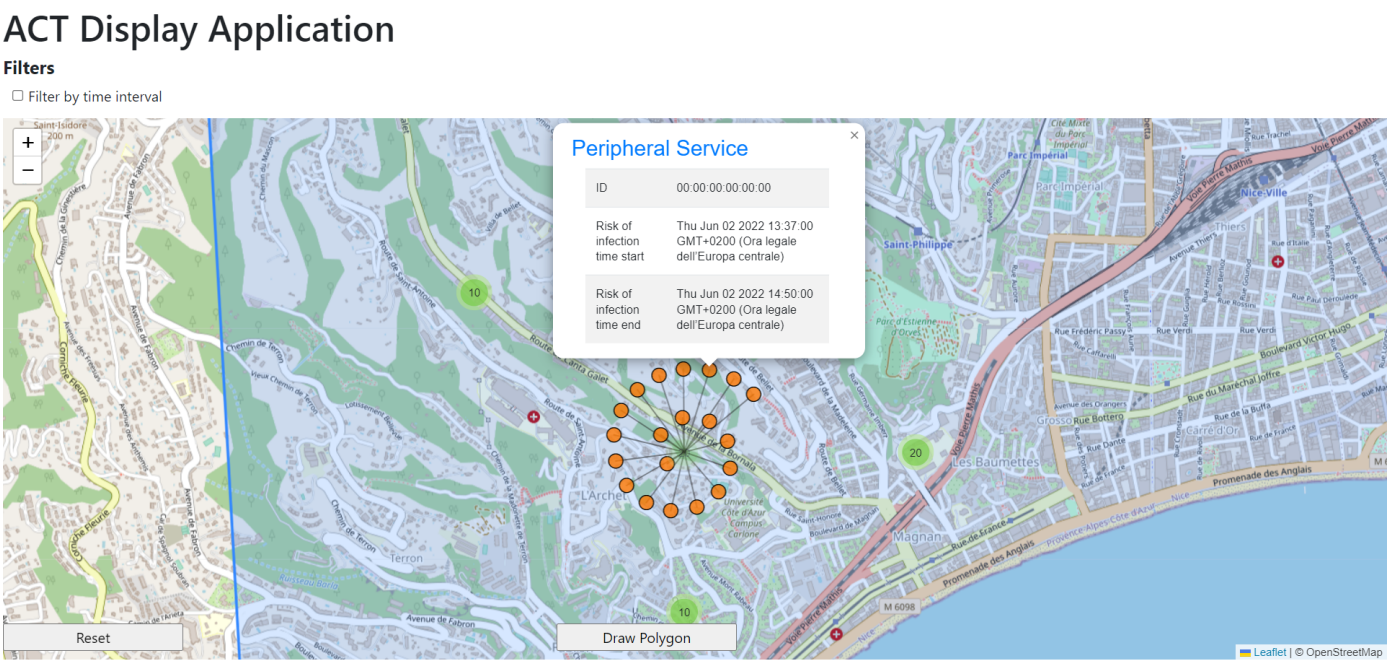

Figure 6.13: Homepage showing marker's popup

Chapter 7

Testing and Experiments

In this chapter we will see how the testing of the entire system was carried out, also seeing the script used to simulate new cases of COVID-19 with the relative explanation. It will also show how the experiments to evaluate the performance of the ACT Smart Mobile Application were carried out and what recommendations were derived.

7.1 Testing

To verify the correct functioning of the entire system, then verify that the communication between all the components takes place correctly and, in the case of a positive result, the user is informed, a script shown in Listing 7.1 has been developed that simulates the detection of COVID-19 in some areas and adds then the BSSIDs of the Peripheral Services associated with these zones in the Control Service database labeling them as "red" defining a random infectiousness interval. To do this, an arbitrary number of BSSIDs and infectiousness intervals are generated, after which a cycle is defined for which an infectiousness interval is associated with each BSSID, the JSON containing Peripheral service ID, Location, start and end of infectiousness interval is created and is inserted into the Control Service database. Then a user is simulated who sends exactly the same position contained in the Location previously defined and entered in the database in order to check if the system, given a Location, is able

to provide the user with all the occurrences that indicate Peripheral Service ”red”, i.e. infected from the virus. Once this is done, it occurs if the mobile application, obtained the response from the Control Service, is able to make an internal matching between the information obtained and those present in its internal database by comparing the Peripheral Service ID and the time intervals.

```
def addEntry():
    BSSID = []
    n=10
    example_mac = "00:00:00:00:00:00"
    startDate = datetime(2022, 5, 6, 13, 00)
    endDate = datetime(2022, 6, 6, 14, 00)
    startTimes=[]
    endTimes=[]
    for x in random_date(startDate, n):
        r=x.strftime("%Y-%m-%dT%H:%M:%S.%f")
        print("TEMPO:" +r)
        r=datetime.fromisoformat(r)
        startTimes.append(datetime.strftime(r, "%Y-%m-%dT%H:%M:%S.%f"))
    for x in random_date(endDate, n):
        endTimes.append(datetime.strftime(x, "%Y-%m-%dT%H:%M:%S.%f"))
    for i in range(n):
        generated_bssid = RandMac(example_mac)
        BSSID.append(example_mac)
    for i in range(n):
        entry = {"Peripheral_Service_Id": str(BSSID[i]), "Location": "
    spv0sd", "Start": str(startTimes[i]),
                 "End": str(endTimes[i]), "Local_service_info": "https://
    NationalHealth.com/forecast",
                 "Control_service_info": "https://NationalHealth.com/
    forecast"}
        collection.insert_one(entry)
```

Listing 7.1: COVID-19 detection simulation

7.2 Fieldwork Experiments

The goal of this task is to carry out performance evaluation by carrying out fieldwork experiments in order to evaluate the detection system of the networks of the Mobile application. In this way it is possible to understand how to set the parameters related to the scanning interval of the networks and the filter of the detection distance of the networks in such a way as to limit as much as possible the number of false positives detected. To do this, it was first of all necessary to map the WiFi networks of the building used for the experiments in order to know which WiFi networks identify the user's location and which belong to places not visited by the user. Subsequently it was necessary to modify the Mobile application in such a way that at the end of each experiment it memorized the database created on a file. In this way it was subsequently possible to analyze the data contained in the file to detect the networks that were not present in the path made by the user but which were still detected by the application and then create charts that made comparisons on the number of false positives found based on different parameters such as: smartphone brand, network scan interval, distance filter. To carry out the experiments it was first of all necessary to decide the building to be used and to map all the WiFi networks present inside. The choice of the building fell on the INRIA Lagrange building as it is a building with floors all organized in the same way, making it easier to map the building and run the experiments. The ACT Smart Mobile Application was used to perform the mapping because being able to view the SSID, BSSID, distance from the network referring to the detected networks, it was possible to go near the various repeaters and identify the network associated with that particular repeater through the app. Since each repeater provides multiple networks such as: INRIA, INRIA-internal, INRIA-guest, eduroam I decided to map the *eduroam* networks. After having mapped all three floors of the Lagrange building, I created maps shown in Figures 7.1, 7.2, 7.3 of the various floors that show where the repeaters and the associated BSSID are located, in this way, during the analysis of the collected data it is possible to understand where are the detected networks that are not part of the plan the user is in.

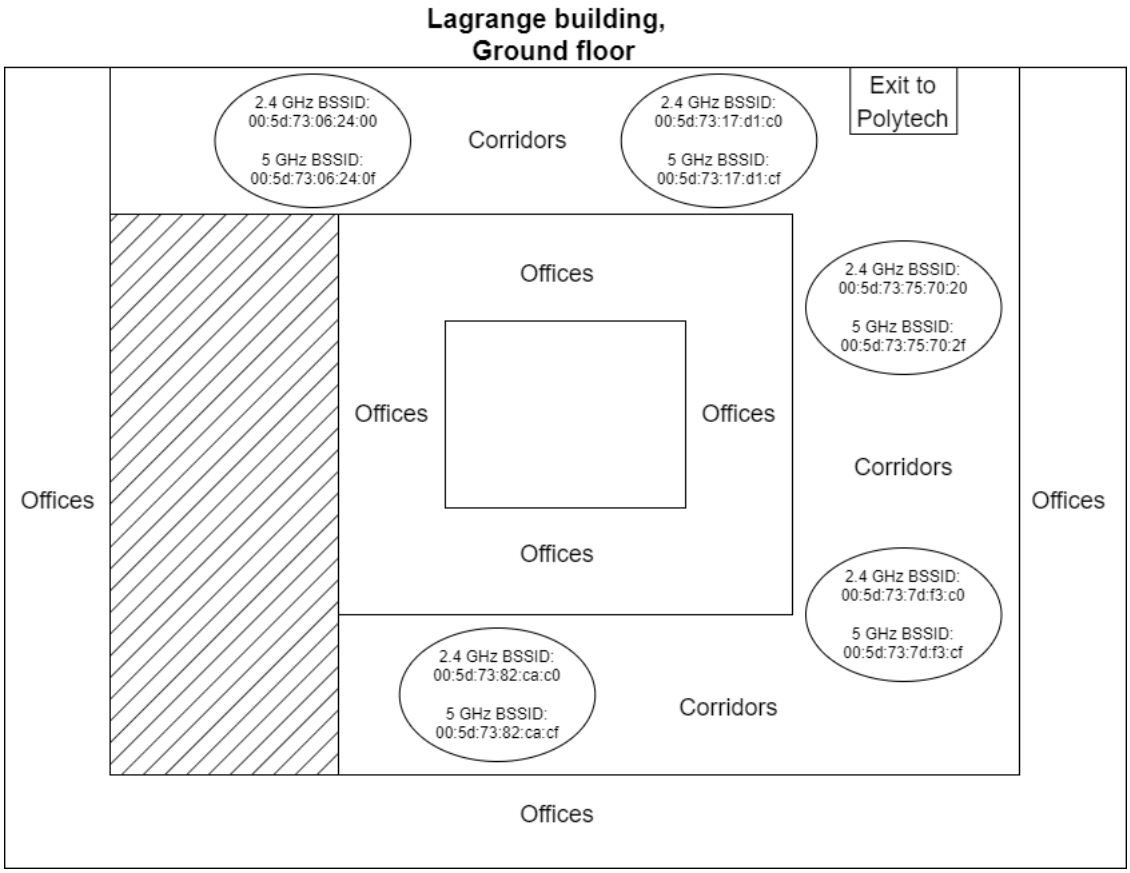

Figure 7.1: Lagrange building, ground floor

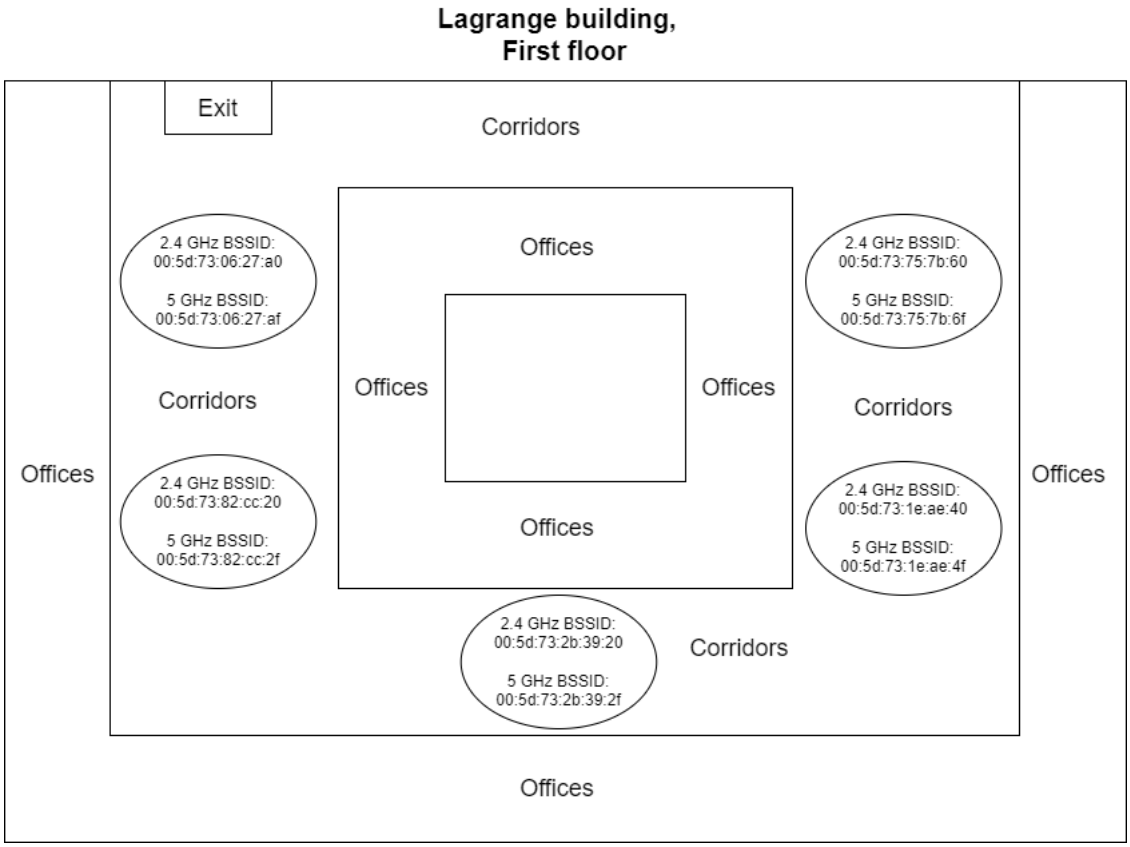

Figure 7.2: Lagrange building, first floor

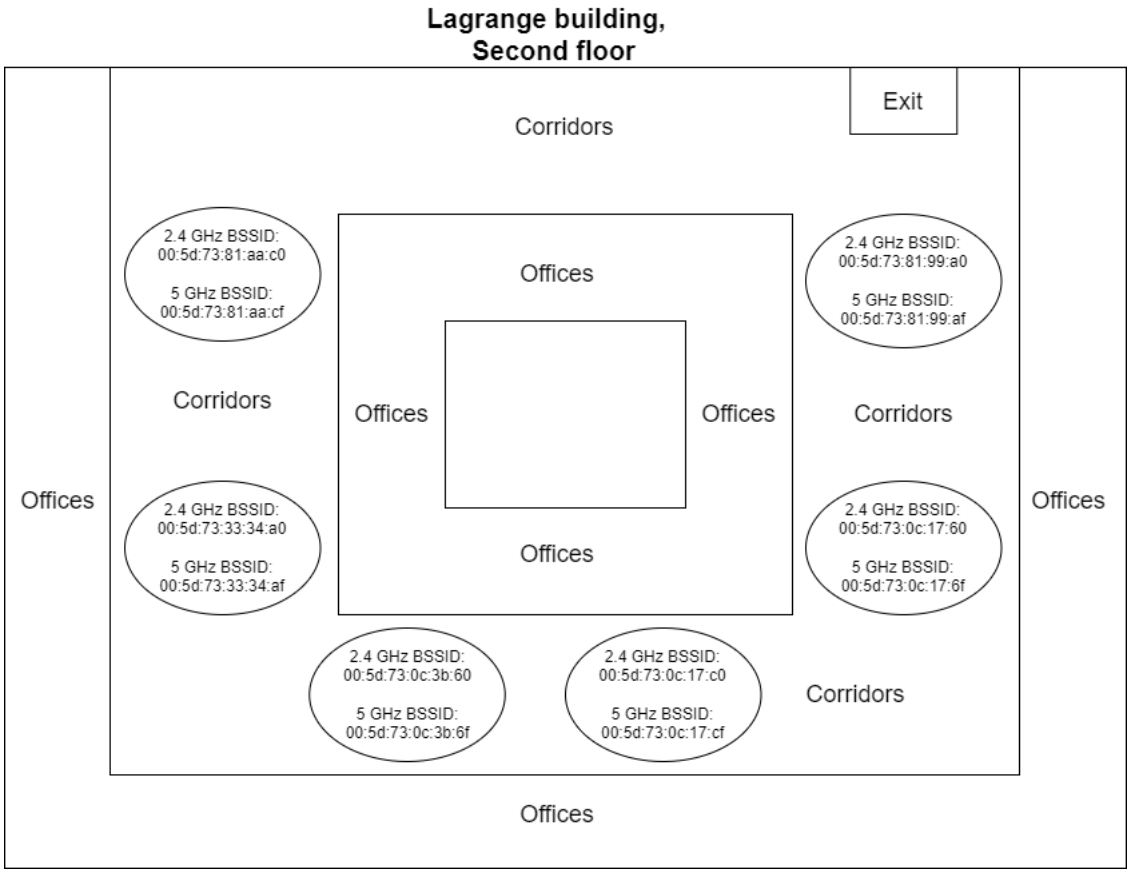

Figure 7.3: Lagrange building, second floor

After creating the maps I moved on field experiments phase. In this phase first I modified the app by adding a button that allows you to create a file with extension `.csv` containing the data of the database inside the application, finally inside the `SplashActivity` I added the line of code to delete the database every time the application is launched. The `.csv` file will then contain a list of SSID, BSSID, TimeFound, TimeLost and information relating to the Scan interval and Distance filter set by the user. To export the database, a request was therefore added to obtain permissions to write files in memory and a function was developed that is activated when the "Database backup" button is pressed. The function shown in the Listing 7.2 first of all accesses the *SharedPreferences* to obtain the information relating to the Scan interval and the Distance filter set by the user, subsequently through a query to the database

it obtains all the data contained within it and, thanks to the Kotlin-CSV library[1], the file `.csv` is created. Once created, a cycle is performed that writes all the information present in the database into the file. If the TimeLost has not yet been set, the timestamp of the moment when the user backs up the database will be set.

```
fun exportCSV() {
    var mc=MainContext.INSTANCE
    var settings: Settings =Settings(Repository(mc.context))
    val sharedPreference =  getSharedPreferences("sharedPrefs", Context.
   MODE_PRIVATE)
    var editor = sharedPreference.edit()
    var n= sharedPreference.getInt("Version",1)
    val exportDir = File(filesDir, "/CSV")// path where file's saved
    if (!exportDir.exists()) {
        exportDir.mkdirs()
    }
    val listNetworks =getAllNetworks(MainContext.INSTANCE.mainActivity.
   application).execute().get()
    csvWriter().open(File(exportDir, "test"+n+".csv"), append = false) {
        // Header
        writeRow(listOf("Scan interval: "+settings.scanSpeed()+" s", "
   Distance filter: "+settings.distance()+" m"))
        writeRow(listOf("[SSID]", "[BSSID]", "[TimeFound]", "[TimeLost]"))
        for (e in listNetworks) {
            if (e.TimeLost == null) {
                writeRow(listOf(e.SSID, e.BSSID, e.TimeFound, Date()))
            } else {
                writeRow(listOf(e.SSID, e.BSSID, e.TimeFound, e.TimeLost))
            }
        }
    }
    editor.putInt("Version",n+1)
    editor.commit()
}
```

Listing 7.2: Database backup in .CSV file function

[1] https://github.com/doyaaaaaken/kotlin-csv

After having properly modified the application it was then possible to carry out the experiments. What has been done is:

1. start the application so that the database was reinitialized,
2. walk for 5 minutes throughout the floor,
3. export the database in the form of `.csv` thanks to the appropriate button,
4. change the Scan interval and Distance filter parameters,
5. repeat the experiment with the new parameters.

I made various experiments by changing the parameters relating to the Scan interval and to the network detection distance filter. Two smartphones were used: a Samsung Galaxy A52 and a UMIDIGI F1 Play[2], for each of them various experiments were made by changing the parameters mentioned above in order to define the number of false positives, i.e. networks detected by the smartphone but for which the user did not pass by, based on the parameters set. All the experiments were made on the first floor that is the middle floor of the building, in this way it is possible to check whether during the experiments both networks coming from the floor below and networks coming from the floor above are detected. Once I got the data from the experiments I moved on to analyzing the data in order to find the false positives. To do this I wrote the script shown in the Listing 7.3 which reads the `.csv` file containing all the networks detected and creates a text file containing only the *eduroam* networks. Then a counter is defined which counts all the networks present in the text file and a counter which counts the false positives. The text file mentioned above is then read and, for each network, a comparison is made on its BSSID and an array containing the BSSIDs of the networks present on the first floor of the Lagrange building and if a matching is not found, that network is a false positive, in this case the false positive counter is incremented and the data relating to that network is printed.

```
content=open("./Test Umidigi/test34.csv","r").readlines()
file=open("./Test Umidigi/test34_mod.txt","w")
```

[2]Umidigi is a Chinese smartphone brand of medium-low range

```
for line in content:
    print(line)
    if("eduroam" in line):
        file.write(line)
file.close()
file=open("./Test Umidigi/test34_mod.txt","r").readlines()
ap=["00:5d:73:06:27:a0","00:5d:73:06:27:af","00:5d:73:75:7b:60","00:5d
   :73:75:7b:6f","00:5d:73:1e:ae:40","00:5d:73:1e:ae:4f","00:5d:73:2b
   :39:20","00:5d:73:2b:39:2f","00:5d:73:82:cc:20","00:5d:73:82:cc:2f"]
find=False
c=0
total=0
for line in file:
    for ele in ap:
        if(ele in line):
            find=True
    if find==False:
        print(line)
        c+=1
    find=False
    total+=1
print("\n\nErrors: "+str(c))
print("\n\nTotal networks number: "+str(total))
```

Listing 7.3: Data analysis algorithm

The results are visible in the following tables.

Samsung Galaxy A52 experiment results with Scan interval = 10 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	3	23	13%
10 meters	16	46	35%
20 meters	39	73	53%
30 meters	54	90	60%
50 meters	70	105	67%

UMIDIGI F1 Play experiment results with Scan interval = 10 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	1	9	11%
10 meters	2	14	14%
20 meters	5	13	38%
30 meters	14	24	58%
50 meters	27	47	57%

Samsung Galaxy A52 experiment results with Scan interval = 20 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	2	18	11%
10 meters	7	30	23%
20 meters	7	16	44%
30 meters	44	78	56%
50 meters	57	89	64%

UMIDIGI F1 Play experiment results with Scan interval = 20 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	0	6	0%
10 meters	1	10	10%
20 meters	5	22	23%
30 meters	11	20	55%
50 meters	28	48	58%

Samsung Galaxy A52 experiment results with Scan interval = 30 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	1	15	7%
10 meters	3	21	14%
20 meters	25	54	46%
30 meters	26	53	49%
50 meters	39	62	63%

UMIDIGI F1 Play experiment results with Scan interval = 30 seconds

Distance filter	**False positive**	**Total networks**	**FP\TN in %**
5 meters	0	5	0%
10 meters	2	8	25%
20 meters	5	20	25%
30 meters	10	35	29%
50 meters	22	43	51%

Subsequently, I created graphs for both smartphones by first relating the number of false positives based on the Distance filter with the Scan interval set and then relating the false positive\total number of networks ratio based on the Distance filter with the Scan interval set. Below we will see the representation and explanation of these graphs.

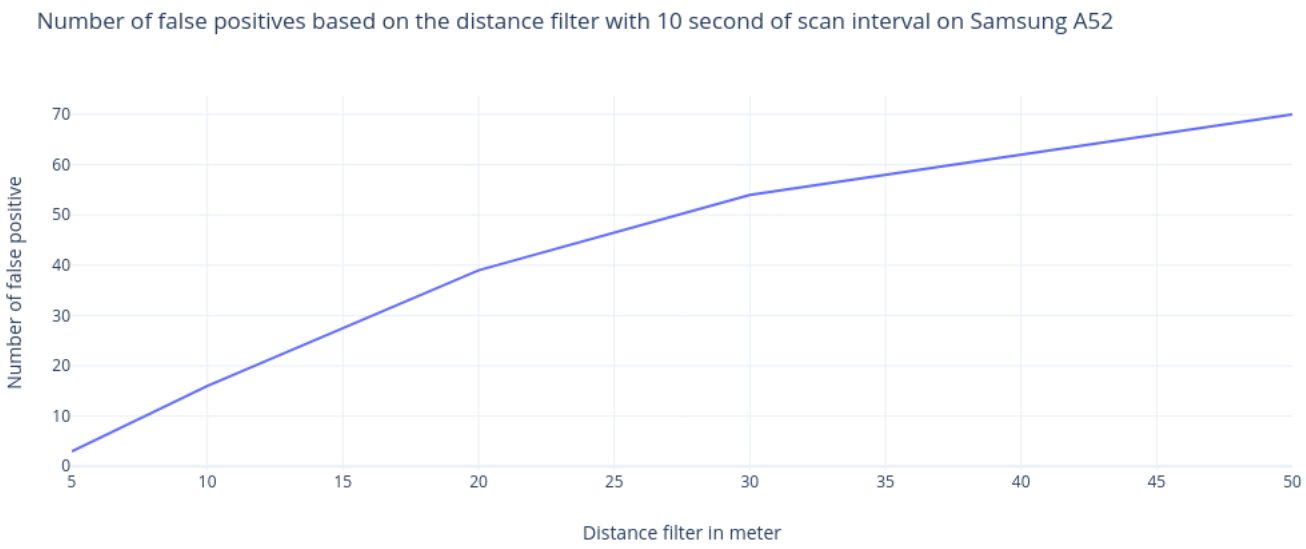

Figure 7.4: Number of false positives based on the distance filter with 10 seconds of scan interval on Samsung A52

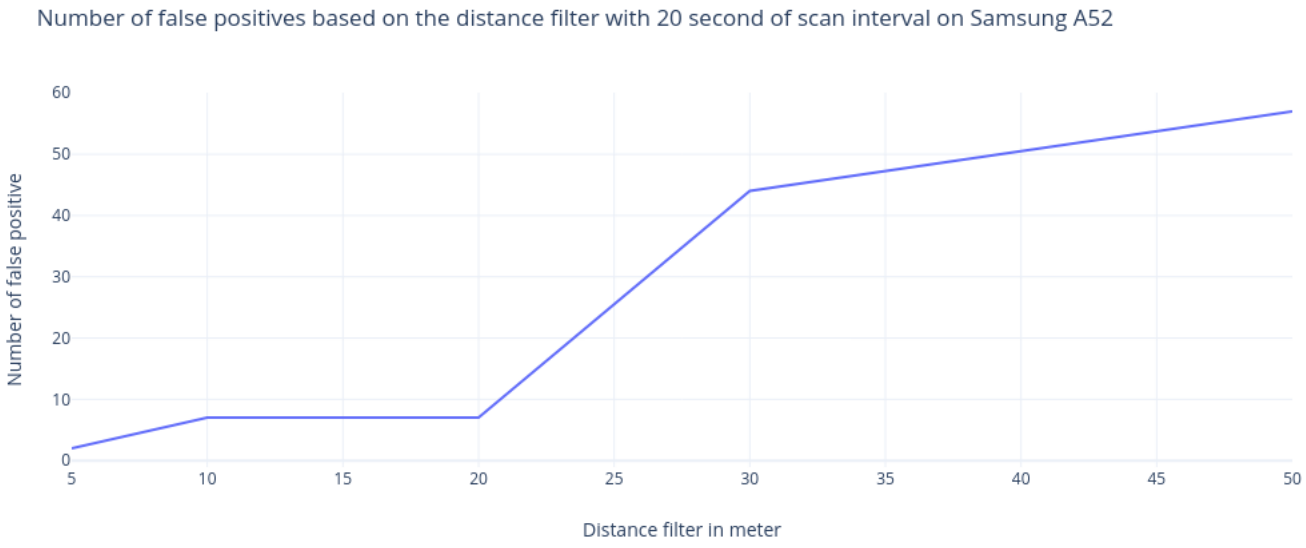

Figure 7.5: Number of false positives based on the distance filter with 20 seconds of scan interval on Samsung A52

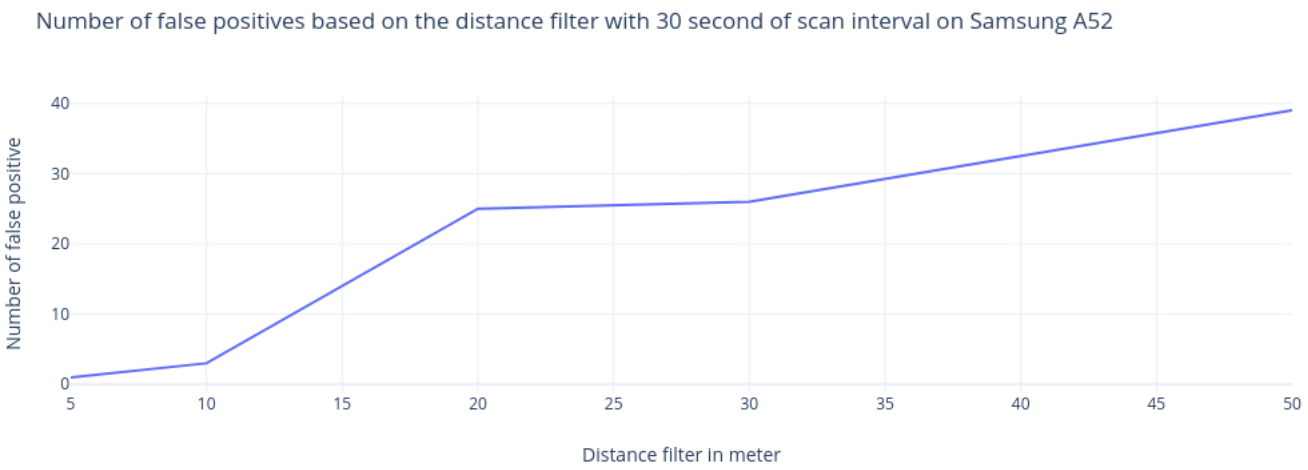

Figure 7.6: Number of false positives based on the distance filter with 30 seconds of scan interval on Samsung A52

From the three graphs shown in the Figures 7.4, 7.5, 7.6 we can see that in the Samsung Galaxy A52 as the distance filter value increases, regardless of the scan interval, the number of false positives increases. We can see that with a lower scan interval we have a higher maximum number of false positives, this is probably due to the fact that by performing the scans more frequently the device finds more networks in total and therefore is more likely to find false positives. Below we will see the 3 graphs related to the Umidigi F1 Play.

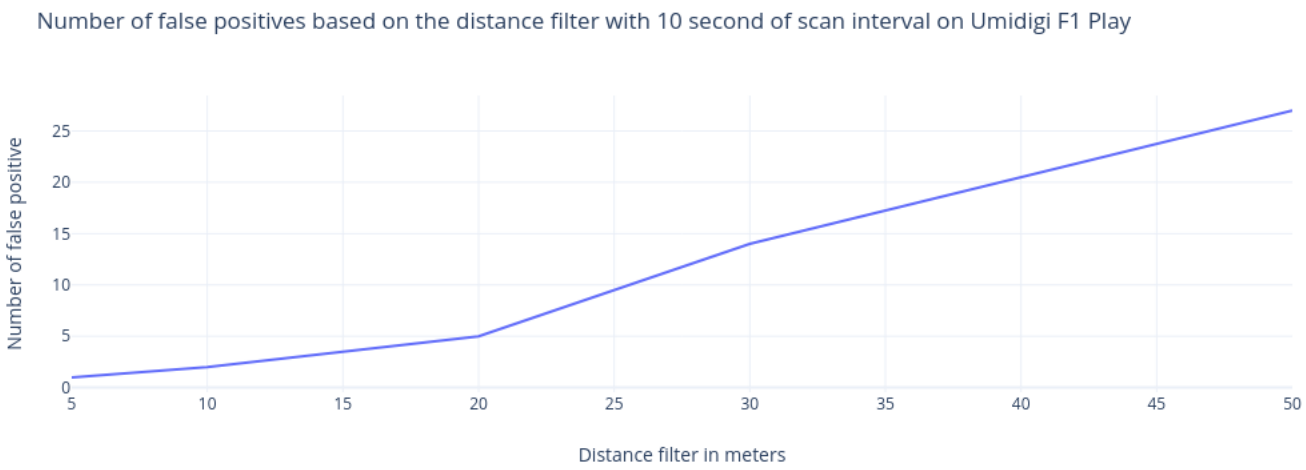

Figure 7.7: Number of false positives based on the distance filter with 10 seconds of scan interval on Umidigi F1 Play

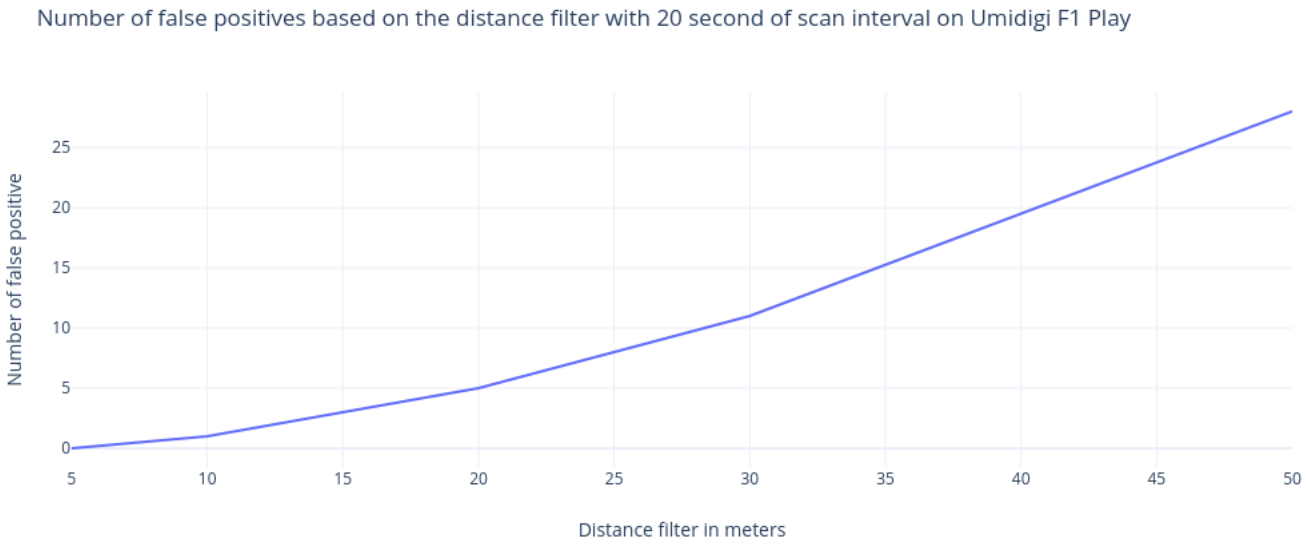

Figure 7.8: Number of false positives based on the distance filter with 20 seconds of scan interval on Umidigi F1 Play

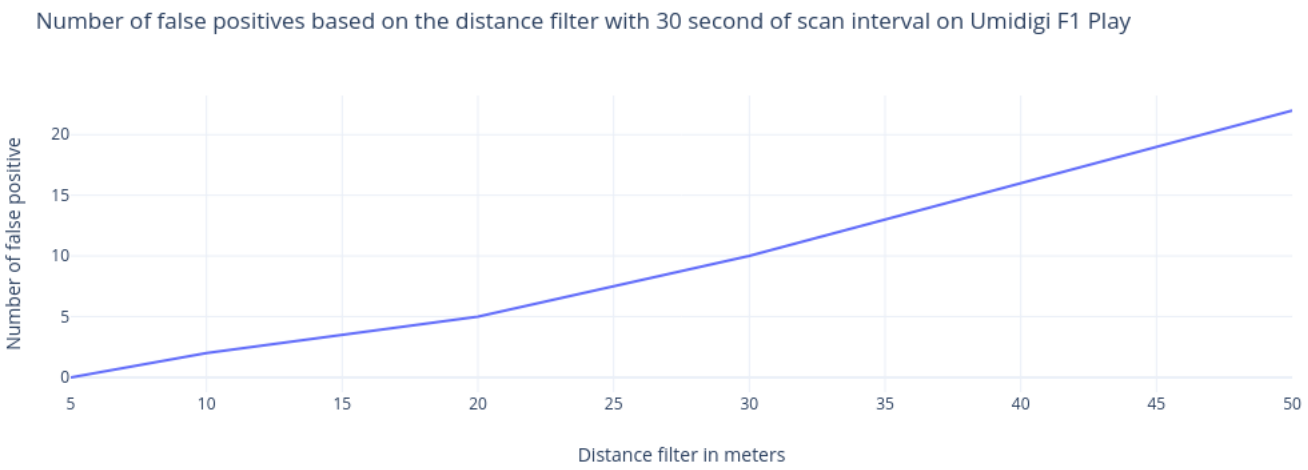

Figure 7.9: Number of false positives based on the distance filter with 30 seconds of scan interval on Umidigi F1 Play

From the three graphs shown in the Figures 7.7, 7.8, 7.9 we see an equally similar situation in the Umidigi F1 Play unlike the fact that the differences between the 3 maxima based on the Scan interval are smaller. We can also note that more false positives are detected in the Samsung in general than in the Umidigi, this is probably due to the fact that the Samsung in general is able to detect more networks. The graphs about the relationship between the false positive\total number of networks ratio based on the Distance filter with the Scan interval set are shown below.

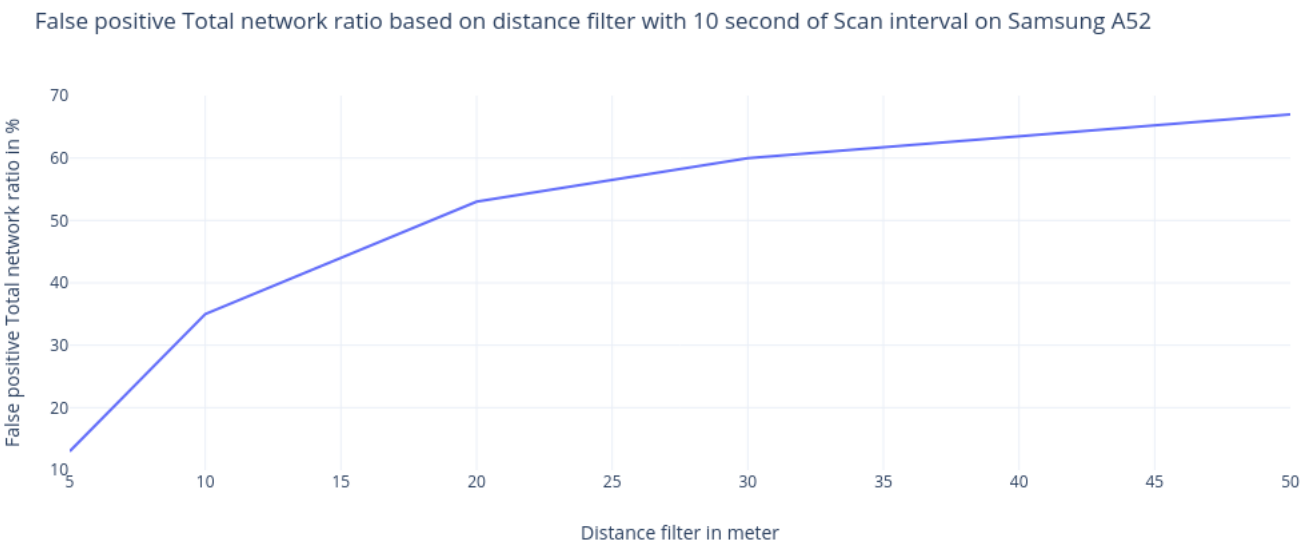

Figure 7.10: False positive\Total network ratio based on distance filter with 10 seconds of Scan interval on Samsung A52

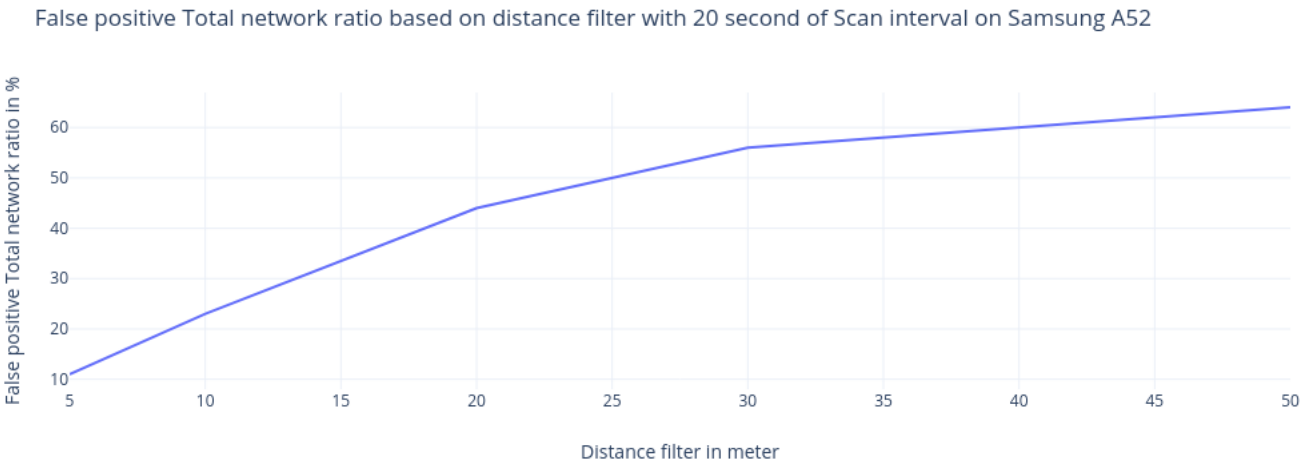

Figure 7.11: False positive\Total network ratio based on distance filter with 20 seconds of Scan interval on Samsung A52

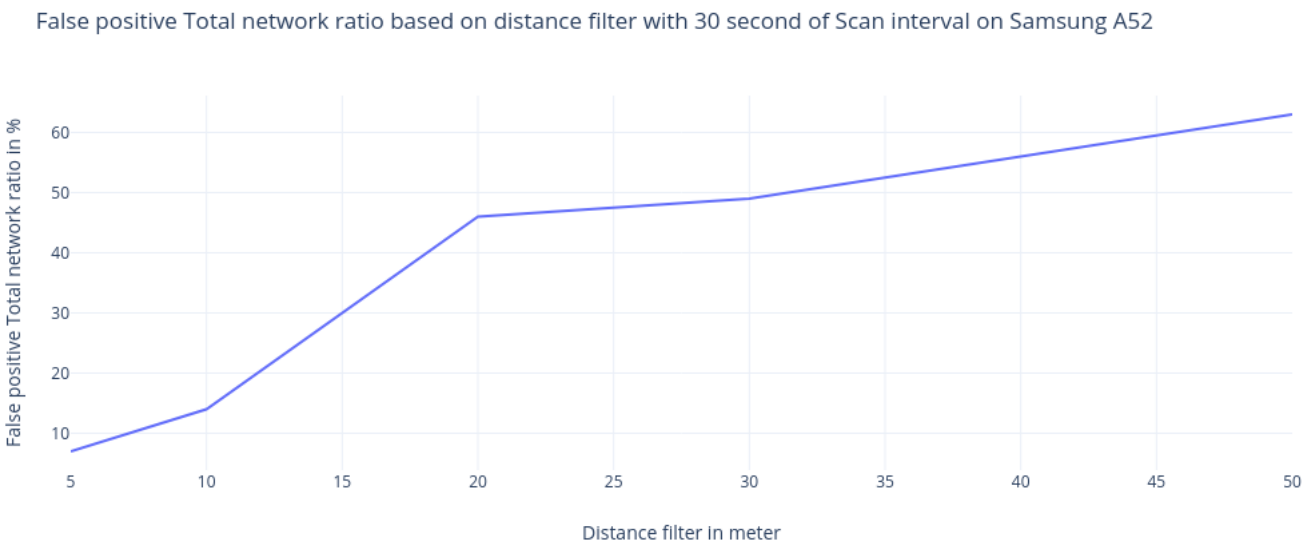

Figure 7.12: False positive\Total network ratio based on distance filter with 30 seconds of Scan interval on Samsung A52

From the three graphs shown above in the Figures 7.10, 7.11, 7.12 we can see how as the meters increase the percentage ratio of false positives\total networks increases, reaching peaks over 60%, this means that more than half of the networks detected during the experiments are false positives thus denoting major performance problems. We can also note that as the scan interval increases, the peak is lower, this is probably a direct consequence of what we have described in the previous graphs. That is, given that as the scan interval increases, the total number of networks detected decreases, there is less chance of finding false positives. Below we will see the 3 graphs related to the Umidigi F1 Play.

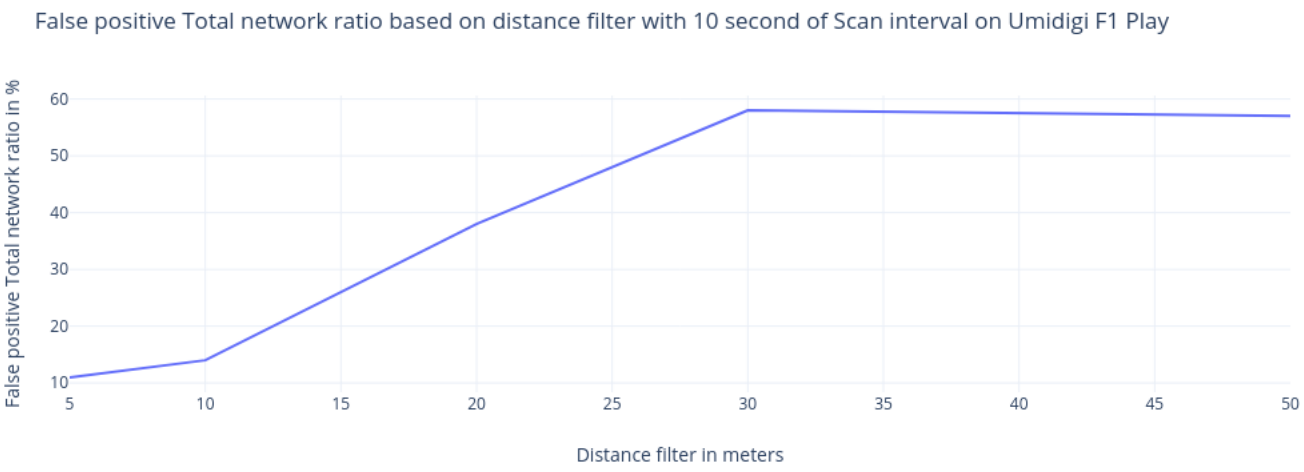

Figure 7.13: False positive\Total network ratio based on distance filter with 10 seconds of Scan interval on Umidigi F1 Play

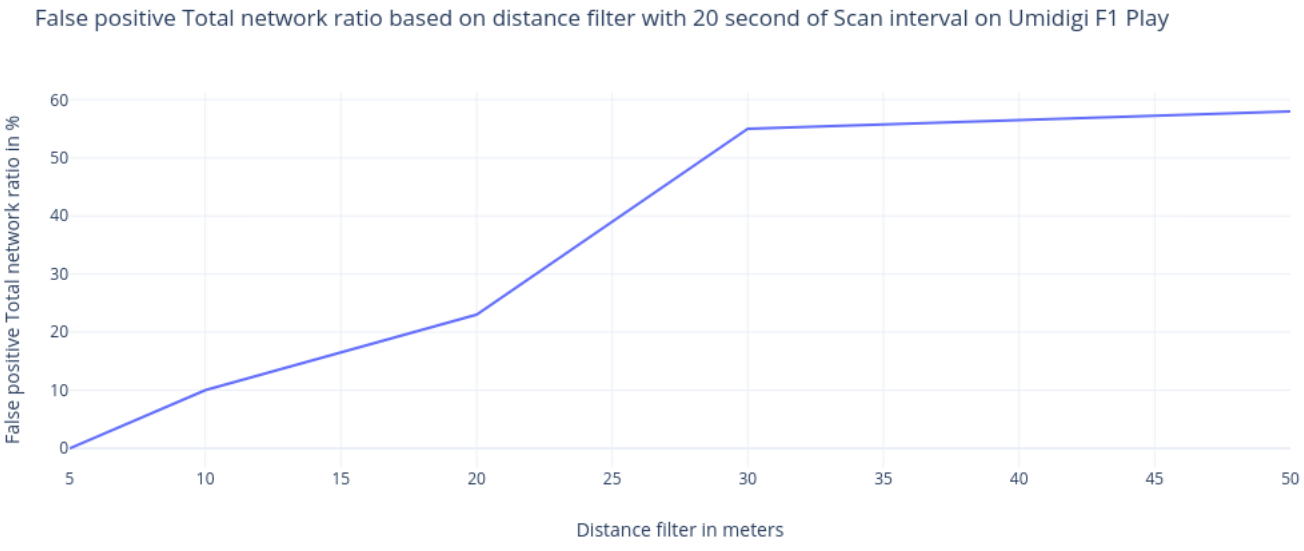

Figure 7.14: False positive\Total network ratio based on distance filter with 20 seconds of Scan interval on Umidigi F1 Play

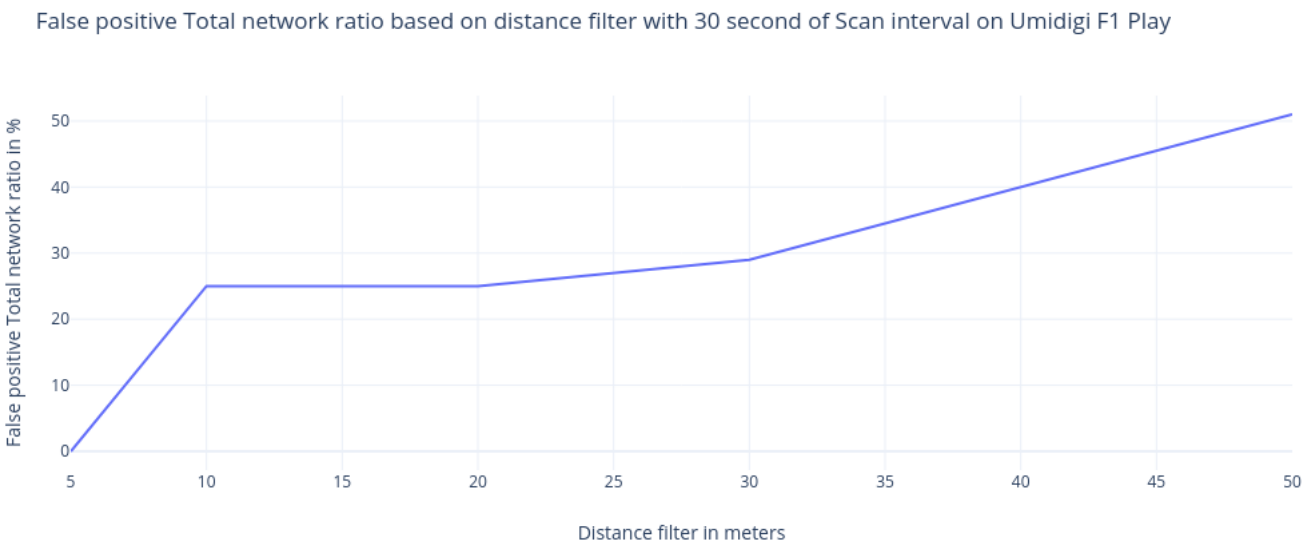

Figure 7.15: False positive\Total network ratio based on distance filter with 30 seconds of Scan interval on Umidigi F1 Play

From the three graphs shown above in the Figures 7.13, 7.14, 7.15 we see a situation similar to Samsung but with the value of the reduced peaks. This indicates that although Umidigi generally detects fewer networks, it is more accurate.

7.3 Recommendations

In conclusion, it is possible to deduce several recommendations from the experiments carried out:

- it is better to put a low Distance filter value so that smartphones detect fewer false positives. This is especially recommended for higher quality smartphones because it can be deduced that the higher the quality of the smartphone, the higher the number of total networks it will detect, therefore the higher the number of false positives detected,
- from a performance point of view related to the number of false positives it is better to put a high Scan interval value because the higher its value, the lower

the number of false positives. From a real-time performance point of view, however, it is better to set a low Scan interval value as in this way the places visited by users will be detected with more precision.

Chapter 8

Conclusions

In this thesis I implemented the Asynchronous Contact Tracing protocol ETSI Technical Specification 103757 [43], making improvements both from a practical and a theoretical point of view, demonstrating the feasibility of the protocol. I also carried out experiments to evaluate the performance of the protocol in order to be able to derive suggestions for improving it in the future. The results are promising as it is possible to apply this protocol very easily by exploiting existing WiFi networks, greatly reducing deployment costs. This protocol can also be used for future pandemics and will allow a better management of the tracing of infections avoiding an excessive number of infections and deaths.

8.1 Future works

In the future it is possible to apply artificial intelligence algorithms to improve the accuracy of the application so as to significantly reduce the number of false positives. Doing so would increase the usefulness of using the signal strength as this parameter would help the artificial intelligence algorithm to classify the user's location. For example, as mentioned above, it is possible to apply the k-neighbors algorithm in the first instance to see how performance improves. Another possible future development is related to the use of Bluetooth technology instead of WiFi as in this way it could be possible to increase the accuracy of the system at the expense of higher

deployment costs. Finally, it would be useful to carry out studies regarding the positioning of the ACT Peripheral Services within the building in order to find the best way to minimize the number of ACT Peripheral Services used and ensure that there are no areas of the building ”blind” that are unable to detect the user’s position.

8.2 Acknowledgements

I would like to thank first of all my family for having been by my side in this long journey, my friends and the University of Bologna, in particular Professor Davide Sangiorgi for allowing me to participate in this Erasmus+ program which allowed me to live a year abroad, study and work there. I would like to thank everyone I met on Erasmus, guys from all over the world with whom I have formed strong bonds that I am sure will last forever. Thank you for being by my side in difficult times and thank you for all the good experiences we shared. I would like to thank Inria in general, especially the Kairos team for giving me the opportunity to carry out this research internship in the company. I would like to thank Luigi Liquori (Inria) for accepting my application to work on this project, for the support, the lessons regarding the world of computer science, the world of work and life in general. I would like to thank Robert De Simone for the perfect integration into the team, Professor Arnaud Legout (Inria) for the advice on network technologies, Enrico Scarrone (TIM), for the explanations on oneM2M and TIM ICON© and for giving me the opportunity to collaborate with TIM on this project.

Bibliography

[1] 3gpp wikipedia. https://it.wikipedia.org/wiki/Third_Generation_Partnership_Project.

[2] Acme project. https://github.com/ankraft/ACME-oneM2M-CSE.

[3] Atis os-iot project. http://os-iot.org/.

[4] Backward vs forward contact tracing wikipedia. https://en.wikipedia.org/wiki/Contact_tracing#Backward_and_forward_tracing.

[5] Captcha wikipedia. https://it.wikipedia.org/wiki/CAPTCHA.

[6] Cuckoo filter wikipedia. https://en.wikipedia.org/wiki/Cuckoo_filter.

[7] Django framework. https://www.djangoproject.com/.

[8] Dogecoin wikipedia. https://it.wikipedia.org/wiki/Dogecoin.

[9] Eclipse om2m project. http://www.eclipse.org/om2m/.

[10] Free-space path loss wikipedia. https://en.wikipedia.org/wiki/Free-space_path_loss/.

[11] Geohash website. http://geohash.org/.

[12] Iotdm project. https://wiki.opendaylight.org/view/IoTDM:Main/.

[13] Jquery library. https://jquery.com/.

[14] k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[15] Kotlin programming language wikipedia. https://en.wikipedia.org/wiki/Kotlin_(programming_language).

[16] Leaflet-geoman website. https://geoman.io/leaflet-geoman.

[17] Leaflet markercluster. https://github.com/Leaflet/Leaflet.markercluster.

[18] Leaflet messagebox. https://github.com/tinuzz/leaflet-messagebox.

[19] Leaflet website. https://leafletjs.com/.

[20] Litecoin wikipedia. https://it.wikipedia.org/wiki/Litecoin.

[21] Mongodb website. https://www.mongodb.com/.

[22] Oasis si project. https://github.com/iotoasis/SI.

[23] Oauth2 wikipedia. https://en.wikipedia.org/wiki/OAuth.

[24] Ocean project. http://www.iotocean.org/main/.

[25] Onem2m iot standard. https://www.onem2m.org/.

[26] Onem2m wiki. https://wiki.onem2m.org/index.php?title=OneM2M_overview.

[27] Opendaylight project. https://www.opendaylight.org/.

[28] Openmtc project. http://www.openmtc.org/.

[29] Openstreetmap website. https://www.openstreetmap.org/.

[30] Pepp-pt github. https://github.com/pepp-pt/pepp-pt-documentation.

[31] Pepp-pt wikipedia. https://en.wikipedia.org/wiki/Pan-European_Privacy-Preserving_Proximity_Tracing.

[32] Privacy-preserving contact tracing. https://covid19.apple.com/contacttracing.

[33] Robert github. https://github.com/ROBERT-proximity-tracing/documents.

[34] Tcn wikipedia. https://en.wikipedia.org/wiki/TCN_Protocol.

[35] Turf library. https://turfjs.org/docs/.

[36] Wifi analyzer (open-source). https://play.google.com/store/apps/details?id=com.vrem.wifianalyzer&hl=it&gl=US.

[37] Jason Bay, Joel Kek, Alvin Tan, Chai Sheng Hau, Lai Yongquan, Janice Tan, and Tang Anh Quy. Bluetrace: A privacy-preserving protocol for community-driven contact tracing across borders. *Government Technology Agency-Singapore, Tech. Rep*, 18, 2020.

[38] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. DESIRE: A Third Way for a European Exposure Notification System Leveraging the best of centralized and decentralized systems. working paper or preprint, May 2020.

[39] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. ROBERT: ROBust and privacy-presERving proximity Tracing. working paper or preprint, May 2020.

[40] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, et al. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing. *arXiv preprint arXiv:2004.03544*, 2020.

[41] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

[42] Abdul Qadir Khan. *Asynchronous Contact Tracing, Fighting Pandemics with Internet of Things*. PhD thesis, INRIA Sophia Antipolis-I3S, 2021.

[43] Luigi Liquori, Enrico Scarrone, Suno Wood, Lanting Cees, Francisco daSilva, Markus Maass, Flynn Bob, Thomas Kessler, Holoyad Taras, and Massimo Vanetti. ETSI Technical Specification TS 103757. SmartM2M; Asynchronous Contact Tracing System, December 2021.

[44] Rainer Mautz. Indoor positioning technologies. 2012.

[45] Colin Percival and Simon Josefsson. The scrypt Password-Based Key Derivation Function. RFC 7914, August 2016.

[46] Beomju Shin, Jung Ho Lee, Taikjin Lee, and Hyung Seok Kim. Enhanced weighted k-nearest neighbor algorithm for indoor wi-fi positioning systems. In *2012 8th international conference on computing technology and information management (NCM and ICNIT)*, volume 2, pages 574–577. IEEE, 2012.

[47] Thomas Tegou, Ilias Kalamaras, Konstantinos Votis, and Dimitrios Tzovaras. A low-cost room-level indoor localization system with easy setup for medical applications. In *2018 11th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 1–7. IEEE, 2018.

[48] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth Paterson, Srdjan Čapkun, David Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel Smart, Aysajan Abidin, Seda Gürses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. Decentralized privacy-preserving proximity tracing, 2020.