Design and implementation of a distributed architecture for embedded devices to provide real-time location and similar services

---

Author:
Guido Fioravantti Rassat

Supervisors:
Joerg Widmer
Albert Banchs

Tribunal

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día 10 de Julio de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Abstract

Ever since Global Positioning System technology appeared, there has been an active discussion about developing a similar service for indoor environments. Applications for indoor RILS (Real-time Indoor Location System) range from Business Intelligence to security appliances and their use is sure to have a significant impact on society. Though several alternatives have been explored throughout the years, no solution has been able to strike a balance between ease of use, expandability and low cost that would allow one of them to become mainstream.

Taking on this challenge, this document explores the state of the art of RILS, classifies them, identifies their advantages and disadvantages and proposes an alternative solution: FIBER. Furthermore, a real world deployment of said solution and further discussion about the results obtained is also included.

Agradecimientos

Quiero dar las gracias a todas las personas que me han ayudado a realizar este trabajo, ya sea con su consejo, cariño, experiencia o apoyo. También quiero hacer una mención especial a mis compañeros y compañeras de carrera, sin los que esta aventura hubiera sido menos emocionante culminar y cuyo compañerismo y amistad agradeceré siempre.

19 de Junio de 2015
Madrid, España.

Contents

List of Figures

1 Motivation and objectives

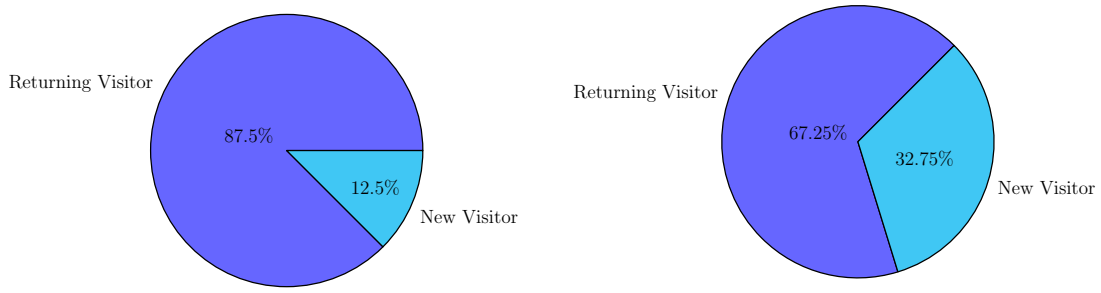
Business intelligence, marketing, employee tracking, security and safety are just some of many applications that can be built on top of Real-time Indoor Location Systems (or RILS). RILS can contribute to these areas, making them more intelligent, less intrusive and more accurate overall. In this section we will describe real scenarios where this technology is being used and the benefits it has yield thus far.

1.1 Business Intelligence and Marketing

The most fitting scenario for describing RILS and business intelligence combined is the so called *Smart Shopping Mall*. This type of mall includes a deployment of RILS and therefore knows more about smartphone users that shop in it. This information (a collection of user's MAC addresses and an estimate of their location inside the shopping mall), is compiled by business intelligence agents and used to aid the mall's commercial staff in a variety of ways.

In a similar fashion as website traffic analysis tools (such as Google Analytics [11]) have aided web owners [5], RILS provides mall owners with information about new vs returning visitors, visitor flow through the mall, visitor to client conversion, etc. This information, as happens in the web, can be critical to assess factors such as:

- Marketing campaign success rate (How many new visitors came to the mall?)
- Floor plant design (Why are users flowing more through certain corridors?)
- Mall space valuation (This area is more expensive because more people flow through it)



(a) Before Marketing campaign (b) After a successful Marketing campaign

Figure 1: A representation of how New Visitor vs. Returning Visitor information can provide useful insight on Marketing campaign success evaluation

Significant statistical inference can be made from such a data set. For this reason, it's rapidly becoming a powerful tool for Marketing applications. Many of these uses are still to be discovered, but others are Internet-born Marketing techniques that can be adopted directly. Such techniques enable:

- User profiling (Where does this user usually spend time and what is he/she interested in?)
- Shop profiling (What do customers of this shop do before/after shopping in it?)
- Trajectory based offers (This user did not buy in this shop and is leaving: ¿last opportunity offer?)
- Bounce rate, Average stay time, Number of stores visited, etc.

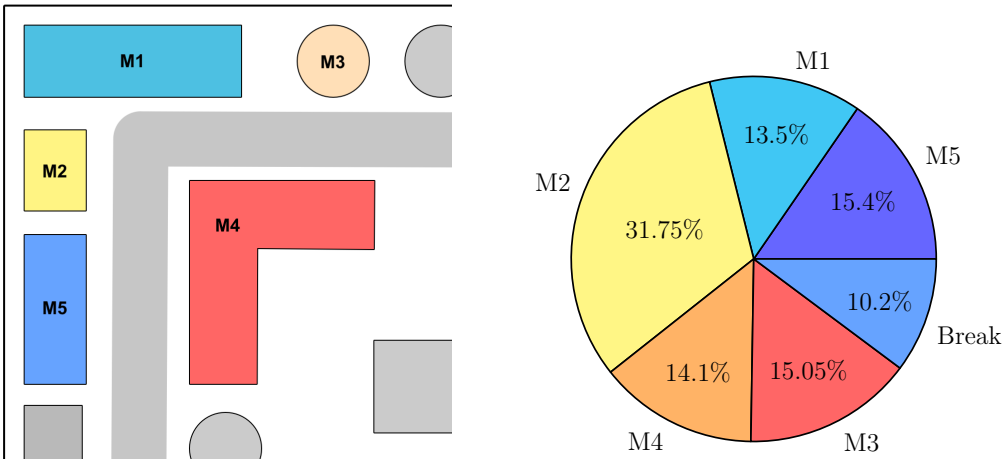
1.2 Employee Tracking

Companies from diverse industries have long tracked their employees. Motivations for doing so are diverse, ranging from measuring productivity to making sure employees are well rested.

Some of these companies are trying new techniques in hopes of providing seamless employee tracking (or ET). One of these new trends, for example, has been using quantifiers to achieve tracking. While quantifier can report on employee activity accurately, they're oblivious of where such activity takes place.

RILS can provide a solution for this scenario by pinpointing employees' locations and mapping them to working places. By building an employee tracking application on top of RILS, developers can keep historical data of worker's locations for later analysis as well as react to real-time events. As a complete solution, employee tracking on top of RILS could, for example, provide:

- Automated alerts on physical stress
- Hours worked vs. productivity analysis
- Machine usage analysis
- Accident reenacting (improving safety)



(a) A corner section of a factory floor plan representing machines labeled from M1 to M5 and a corridor in a right angle

(b) Time spent by a given employee on different machines

Figure 2: An example of a RILS + ET application tracking time spent by an employee on different machines

The RILS described in this report was developed for use in combination with employee tracking in a factory that produces metal products. The implementation details of the system will be described in further sections.

1.3 Security

Most modern security system is based on one way of tracking or another. Like employee tracking, where employees are tracked based on the mobile devices they use, infrastructure-based IEEE 802.11 [10] and Bluetooth [13] RILS (which this report focuses on and will be further explained) can track a user that is using a device for transmission without their active collaboration (unlike client-based RILS). Based on this capability, many security based applications can be envisioned.

For example, in some places like commercial airplanes, it is vital that electronic equipment is turned off. In this scenario, RILS can detect active interfaces and locate them so that offenders can be stopped without having to laboriously check every passenger's device.

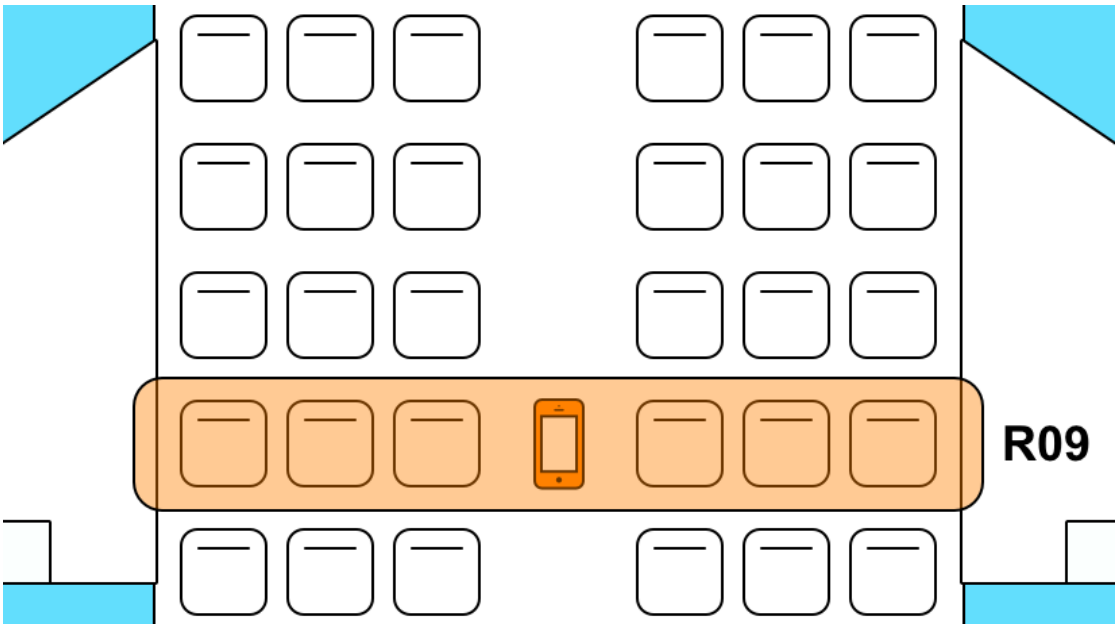


Figure 3: An example of an airplane application of RILS where an offender is detected in row 9

Other uses for this *non user-consented device detection* capability could be alerting about transmitting devices in conference rooms where such behavior is banned. A simply alert could be triggered when active interfaces are detected, warning meeting members to take appropriate action.

1.4 Safety

RILS can be combined with safety in a number of ways. One example is building evacuation. According to firefighters [6], assessing if there are people trapped inside a burning house or office is one of the most difficult tasks they face, for which they often resort to a risk/probability approach based on partial data. RILS could provide useful insight in these situations. If the system detects moving workers in a given area, such information may aid firefighters on planning rescue operations. Another thing to note is that for a conventional RILS + ET system to provide this service, backup batteries so that the equipment can run through emergency power cuts is all that it needed.

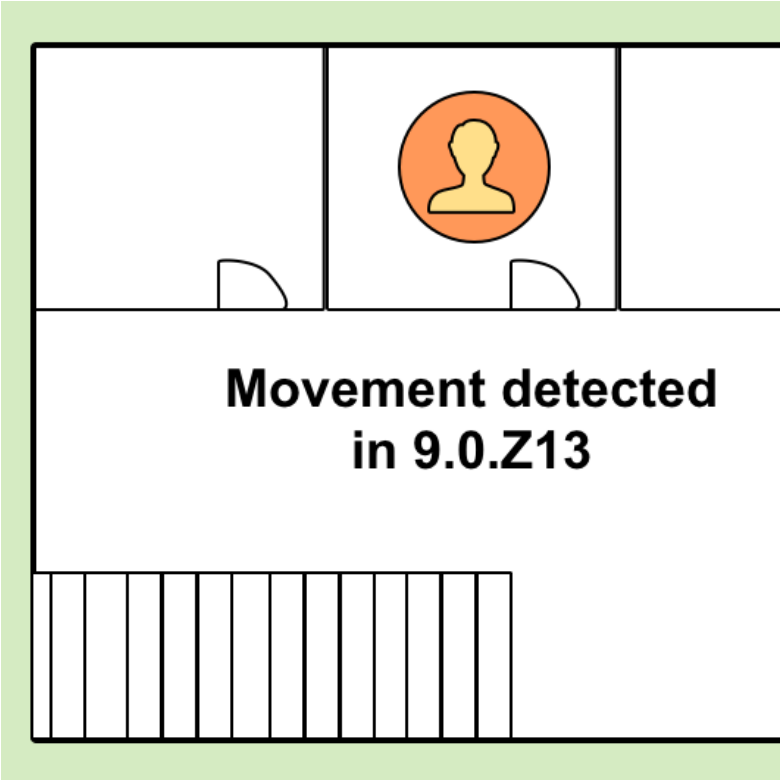


Figure 4: An example of what a RILS + Safety implementation may look like detecting movement during a fire

Other inspiring scenarios for RILS and safety include locating doctors in a hospital when needed, knowing the last position of a stranded worker or verifying devices are turned off in a 'explosion risk' environment.

2 Discussion of the problem to solve

Finding an adequate way of solving RILS is striking a balance between four aspects: convenience, expandability, cost and accuracy. In order to formally define these three aspects:

- **Convenience** refers to how easy it is to deploy the system with current wide-spread technologies. For example, a RILS aided by sensor present in most smartphones will be more convenient than a system which requires specific hardware.
- **Expandability** is measured by how straight forward it is for developers to build applications on top of the RILS in question. A RILS that is highly modular and provides well known interfaces for developer interaction will generally score higher in terms of expandability. For example, a FIBER provides a JSON API with location information is highly expandable since it uses a well known and simple interface to output relevant information on top of which a web based application is straight forward to build.
- **Cost** is simply defined as the amount of money that must be invested in order to get the RILS operative. If a RILS uses license spectrum or requires the user to bear a specific device, the cost will be high in comparison to a system that can run on off the shell hardware and existing user devices.
- **Accuracy** is related to the precision with which a client can be located in terms of meters. **However**, we are not looking for the best accuracy possible, just for a 'good enough' accuracy (3-4m). All the systems analyzed deliver this type of accuracy.

2.1 Classification of real-time location systems

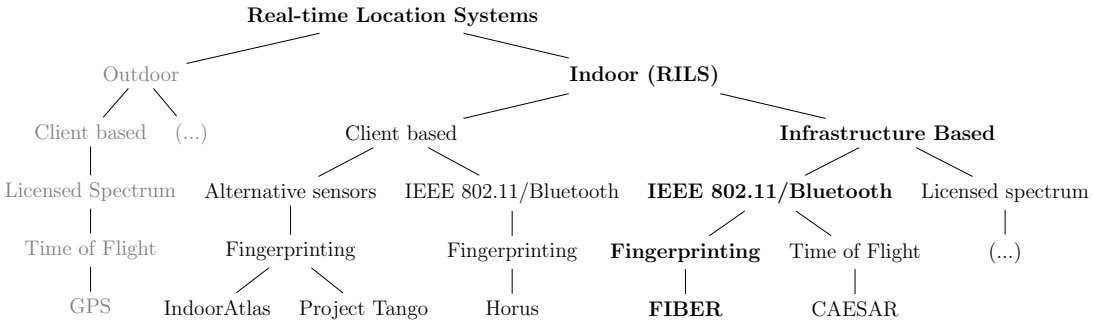


Figure 5: Proposed location systems classification

A complete classification scheme for location systems and RILS is proposed in Figure 37. **Our development, FIBER** (Fingerprinting Infrastructure-BasEd Rils), classifies as *IEEE 802.11 [10]/Bluetooth [13] fingerprint infrastructure based RILS*. Other state of the art RILS (IndoorAtlas [4], Project Tango [3], Horus [2] and CAESAR [1]) have been placed in it for reference.

2.2 State of the art

Current implementations of RILS are highly diverse in terms of technology used to achieve indoor location. We will now go over some of them, explaining how they work and why they pose limitations that made FIBER necessary.

2.2.1 Client side with alternative sensors

Systems like Google's Project Tango [3] and IndoorAtlas [4] focus on meter-accuracy of location. To achieve said accuracy, they rely on alternate sensors such as depth-aware cameras (such as [7]) and magnetic sensors [8].

Google's Project Tango [3] works on devices bearing special cameras and sensors providing depth perception, area learning and motion tracking. This information is then processed by the device in order to infer information about its position within a building. A developer API is also available, allowing developers to expand the core functionality of the system and C, JAVA, Android or Unity written applications to run on top of it.

Advantages	Disadvantages
High precision	Requires new sensors
API for developers	Client side
	Requires user involvement

Table 1: Table showing advantages and disadvantages of Google's Project Tango

While Project Tango provides modularity/expandability thanks to the provided developer API, requiring specific hardware not available to the public make cost and ease of use an issue. Moreover, being a client-side solution, many real-life scenarios (see Section 1) are not possible without user collaboration. Therefore, there is still room for FIBER to take on Project Tango's weak points.

IndoorAtlas [4] combines traditional IEEE 802.11 [10]/Bluetooth [13] fingerprinting techniques [9] with magnetic-field fingerprinting [8]. Relying on the earth's magnetic field is more convenient than 2.4GHz/5GHz spectrum, since it will not be affected by physical object's movement in the environment (i.e., furniture rearrangement, moving pieces in a factory, etc.).

Advantages	Disadvantages
Robust fingerprinting	Requires magnetic sensor
Less calibration	Client side
	Requires user involvement

Table 2: Table showing advantages and disadvantages of IndoorAtlas

IndoorAtlas [4] inherits all the disadvantages of client-side RILS when it comes to our target scenarios (see Section 1). Additionally, as in the case of Project Tango [3], using specific hardware makes cost and ease of use an issue.

2.2.2 Client side with IEEE 802.11/Bluetooth

The Horus [2] location system provides some of the mathematical framework for FIBER as well as some of the techniques for medium calibration and data analysis. FIBER and Horus systems are similar except for a crucial difference: Horus is client-based while FIBER is infrastructure based.

Client based systems require some degree of active user involvement. The user must run a location app on their device, generally helping said user to locate his/herself inside a building. Infrastructure based location systems do not depend on active user involvement and can operate without user-side consent. Moreover, infrastructure based location systems will only require a user to be sending information using IEEE 802.11 [10] or Bluetooth [13] for their device to be located.

Another important aspect about client-based vs. infrastructure-based systems that use fingerprinting is that, generally speaking, fingerprinting yields accuracy of 3-4 meters. In short, IEEE 802.11 [10]/Bluetooth [13] fingerprinting [9] provides *area-level accuracy*, which is good enough for infrastructure-based applications (see Section 1), but not quite good enough for most client-side applications (i.e., indoor navigation).

Advantages	Disadvantages
Uses existing sensors	Area level accuracy not enough
	Recalibration on environment change
	Requires user involvement

Table 3: Table showing advantages and disadvantages of the Horus system

Horus provides cost effectiveness and ease of use thanks to the use of common sensors as well as certain expandability but being client-side makes it incompatible with our targeted scenarios. The Horus system paper states [2] that porting the system to infrastructure-based is trivial, but there are some important issues in doing so that will be discussed further on (see Section 3).

2.2.3 Infrastructure-based with Time of Flight techniques

Location systems using Time of Flight (or ToF) attempt to compute the location of a given device based on very accurate measurement of time intervals between transmissions towards and from said device. Famously, GPS [12] uses this technique.

According to CAESAR [1], the reason ToF has not been more present in IEEE 802.11 [10] based RILS is that, since they are used mainly for communication, ranging techniques suffer from *"noise in the measurement due to clock drift, interaction with system tasks, and protocol overhead"*. Caesar will also tend to loose accuracy when not in direct line of sight of the device due to reflections.

Advantages	Disadvantages
Uses existing sensors	Requires strict time sync with device
Meter accuracy	Reflection sensitive

Table 4: Table showing advantages and disadvantages of the CAESAR system

All things considered, CAESAR poses as a solid option for our target scenarios (see Section 1) since it's infrastructure-based. Anyhow, the need for strict timing synchronization with the device and reflection problems may be an issue for certain scenarios (i.e, a metal factory).

2.2.4 Licensed spectrum solutions

Although some RILS solutions take advantage of higher frequencies with better accuracy than IEEE 802.11 [10] (2.4GHz/5GHz) or Bluetooth [13] (2.4GHz), they require specific hardware on the client-side and need a license for operation within their spectrum range. Since cost effectiveness is one of our goals, we do not consider these systems.

2.2.5 Infrastructure-based Fingerprinting

Our development (FIBER) is an infrastructure based fingerprinting RILS that solves Horus' [2] client-side inconvenience. While not as accurate as other non-fingerprinting systems like CAESAR [1], ease of use, cost effectiveness and expandability of FIBER, along with some robustness against reflections, makes it the better match. Table 32 shows FIBER against other systems according to the criteria we defined at the beginning of this section:

	Convenience	Expandability	Cost	Accuracy
Project Tango [3]	low	high	high	✓
Indoor Atlas [4]	low	medium	high	✓
CAESAR [1]	medium	medium	medium	✓
Horus [2]	medium	high	low	✓
FIBER	high	high	low	✓

Table 5: Comparison table of SOA RILS according to the evaluation criteria.

Requiring only widespread sensor on current smartphones, convenience is a selling point for FIBER. Using free spectrum technologies for fingerprinting [9] such as IEEE 802.11 [10] and Bluetooth [13] help with cost-effectiveness and enables the use off-the-shelf hardware for sniffers (also alleviating regulatory frameworks). Finally, building software in exportable modules and providing a JSON API interface for developers to easily build applications on top of FIBER contributes to the system's expandability. If FIBER can provide area-level accuracy (see results and evaluation in Section 4), it's the best solution for our needs.

3 Technical implementation

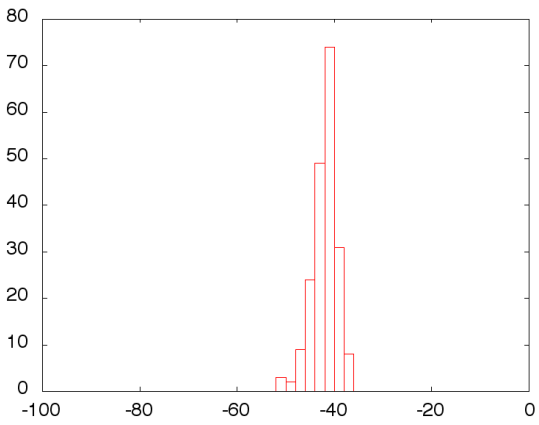
In this section we will see how the elements composing the FIBER system were implemented and made to work together. The development of the system was focused around three fundamental pillars, deemed necessary to succeed. These pillars are (you might want to refer to terminology at Section 3.1):

1. Obtaining good RSSI histograms during fingerprinting [9]
2. Having a good calibration technique
3. Adequately placed sniffers

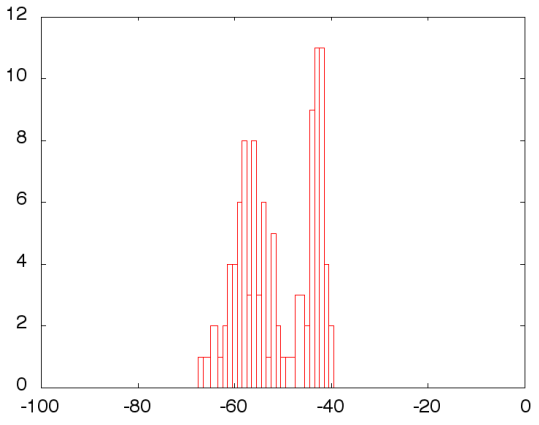
3.1 Terminology

When discussing the system's details, the following terms will be used:

- **RSSI** stands for Received Signal Strength Information. It provides information about the received signal strength in dBm for a given transmission. Values are usually negative, and range from -90dBm (bad signal reception) to 0dBm (perfect signal reception).
- **RSSI histogram (or RH)** is the result of compiling **RSSI** values for a certain amount of time.
- **Single-spiked histogram (or SSH)** refers to a **relative histogram** where **RSSI** values are distributed towards a single value (see Figure 38 [a]).
- **Multiple-spiked histograms (or MSH)** are **RSSI histograms** where **RSSI** values are centered around two (or more) values instead of one (see Figure 38 [b]).



(a) An example of a single-spiked histogram (SSH) centered around -41dBm



(b) An example of a multiple-spiked histogram (MSH) centered around -41dBm and -58dBm

Figure 6: Comparison of single-spiked and double-spiked RSSI histograms

- Each **radio map point** (or RMP) is made up of a set of **sniffer reports** taken during the **calibration**. A **radio map point** is associated a to the GPS coordinates [12] corresponding to the physical location where the calibration of said point took place.
- A **client** (or **user**) is a person carrying an IEEE 802.11 [10] capable device that will be tracked by the system and whose location the system tries to infer.
- A **sniffer** is a device bearing at least one NIC (network interface card) used by the **sniffing software** running in it to sniff **RSSI** values from **client** transmissions.
- **Sniffer report** refers to the **RH** provided by a **sniffer**.
- **Offline phase (or calibration)** consists in gathering **sniffer reports** from all available sniffers when a **client** is at a known position. This technique is known as **fingerprinting** [9]. The collected **Sniffer reports** make a **radio map point** that will be associated to the GPS coordinates [12] of the location at which the calibration takes place.
- A **radio map** (or **RM**) is a collection of several **radio map points**.
- **Online phase** consists in gathering **RSSI** values from **clients** and comparing them to the **radio map** recorded during the **offline phase** in order to determine which is the most probable location of said **client**.

3.2 Architecture

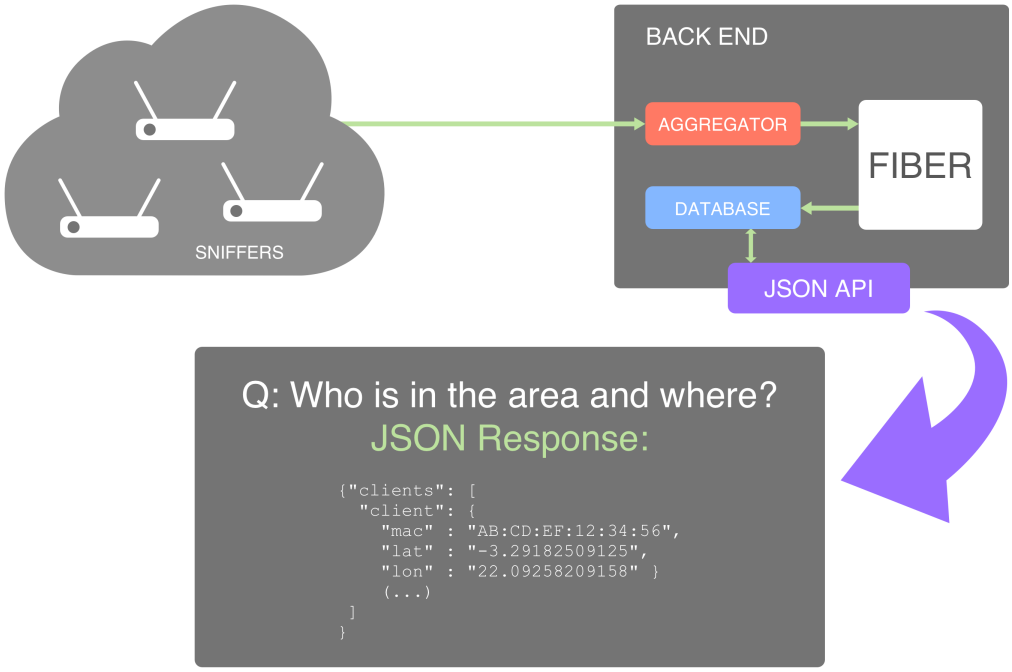


Figure 7: FIBER architecture showing Sniffers + Backend + JSON API interface

The FIBER architecture can be divided into two functional groups: sniffers and back end. Sniffers are devices that will act as the sensing part of the system and will forward measurements (RSSI values from clients' communications) to the back end. The back end, will process this measurements and use them for calibration or to infer location information about the clients detected. This information is accessible to the exterior via a JSON [27] API, that will send JSON [27] formatted responses to queries about clients' locations (see Section 3.4.7).

3.3 Sniffers

Sniffers are the sensing part of FIBER. They exist to provide sniffer reports for the offline and online phases. They are based on small ARM chips and require one NIC for sniffing purposes, another NIC for communication with the back end and power. We will now review the sniffers' requirements, hardware and software.

3.3.1 Requirements

For adequate sniffing, sniffers must fulfill a number of requirements:

1. **Adequate placement** is crucial for proper sniffing. The altitude at which the sniffer's antenna is located drastically affects the range at which it can detect clients, cause unwanted shifts in sniffers reports and increase the need for re-calibration. Sniffers should be placed at an altitude and orientation

such that the amount of radio map points in direct line of sight of sniffing antenna should be maximized.

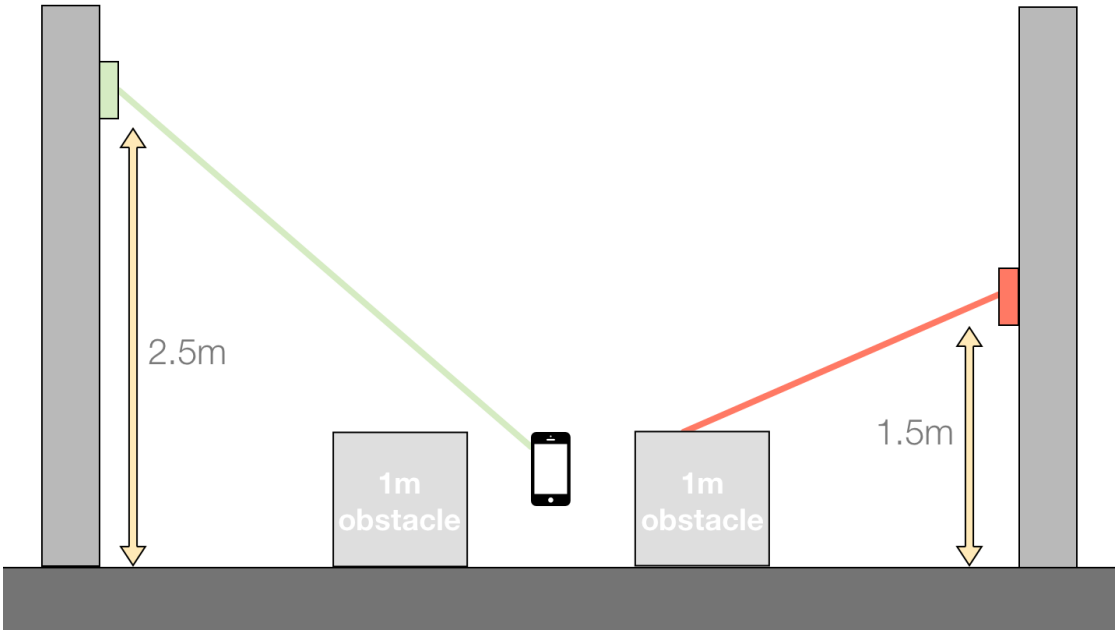
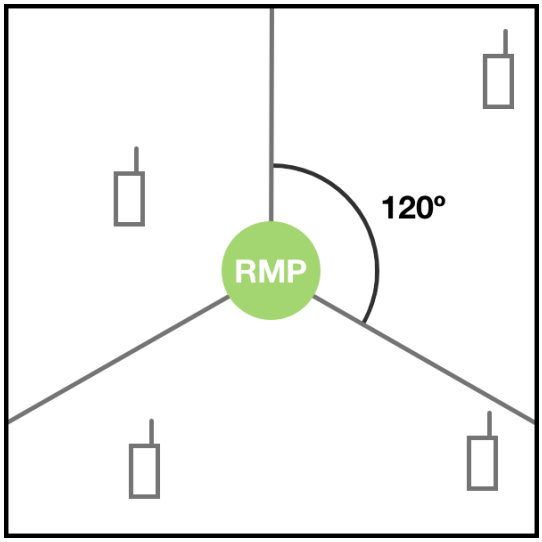
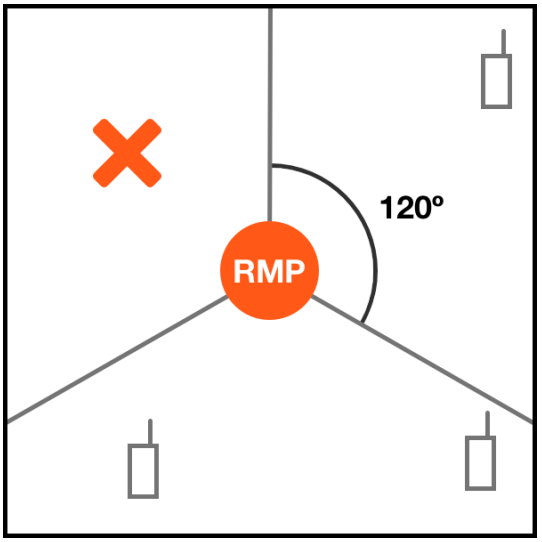


Figure 8: An example of how altitude can affect line of sight contact with clients with 1 meter height obstacles

2. **Direct line of sight** from any radio map point to at least one sniffer per 120º sector (see Figure 9).



(a) An example of well placed sniffers with respect to a radio map point (RMP): At least one sniffer per 120º sector



(b) An example of badly placed sniffers with respect to a radio map point (RMP): One 180º sector has no sniffers in it

Figure 9: Comparison of well and badly placed sniffers with respect to a radio map point (RMP)

3. **Automated setup** is required to make batch deployment at site seamless. Sniffers' configuration must be easily parametrizable as to connect to network infrastructure automatically and connect to the back end.

4. **Self reparation** is also required in case of software failure (hardware failure cannot be solved without human intervention). Sniffers should be able to self-diagnose issues and attempt fixes to avoid human intervention. Since sniffers will typically be located at non-easily accessible places, technician work should be reduced as much as possible.

5. **Remote access** is the ability to remotely control devices to change configurations or debug issues. Access through SSH [35] (or similar) is a requirement for sniffers, specially since they will be located in remote places where physical access is limited.

3.3.2 Hardware

The sniffer form factor should be as compact as possible as to enable its placement in remote areas with minimum inconvenience. In order to achieve a compact form factor, ARM architecture [14] based systems are preferred. Three platforms are considered: Raspberry Pi 2 model B [15], Hardkernel's Odroid C1 [16] and Solid Run's HummingBoard i1 [17]. We will now discuss them in detail.

Originally intended to run Android OS [32], the Odroid C1 also supports ARM Debian Linux [20]. It improves USB capabilities over the Raspberry Pi 2, as it doesn't suffer from the *USB/Ethernet performance anomaly issue*, where the Ethernet [33] bus is shared with the USB bus and can reduce network performance under load. It also has a dedicated charging port, separate from the micro-USB port, which allows both of them to be used simultaneously.

Advantages	Disadvantages
No USB/Ethernet performance issues	Not as good Debian as Raspbian
Dedicated charging port	

Table 6: Advantages and disadvantages of Hardkernel's Odroid C1

The Odroid C1's front and back component schemes can be found in Figure 10 and 11 respectively. A full specification list can be found in Figure 12.

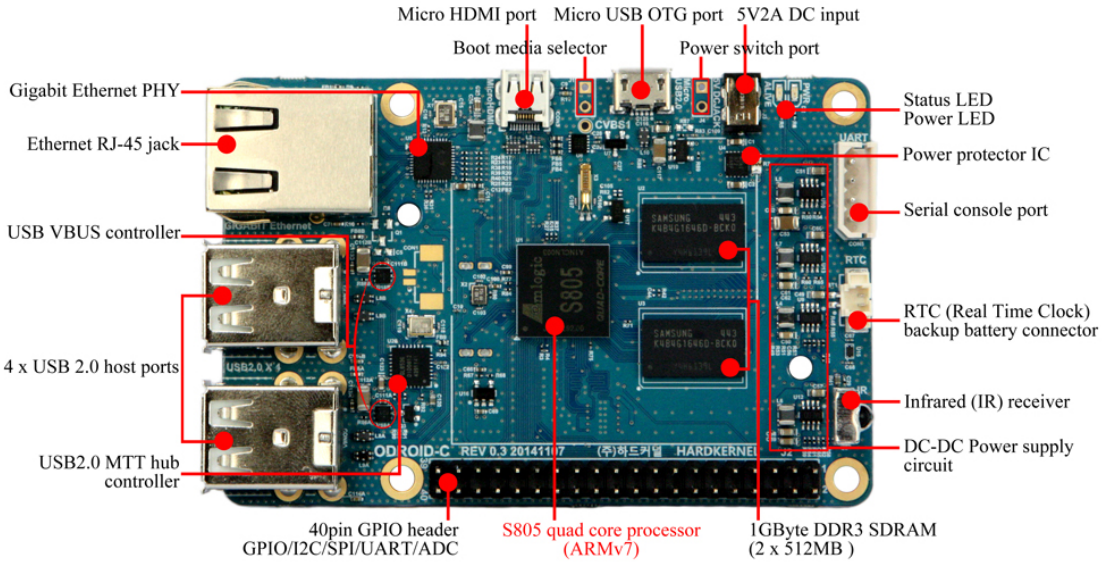


Figure 10: The Odroid C1's front component scheme

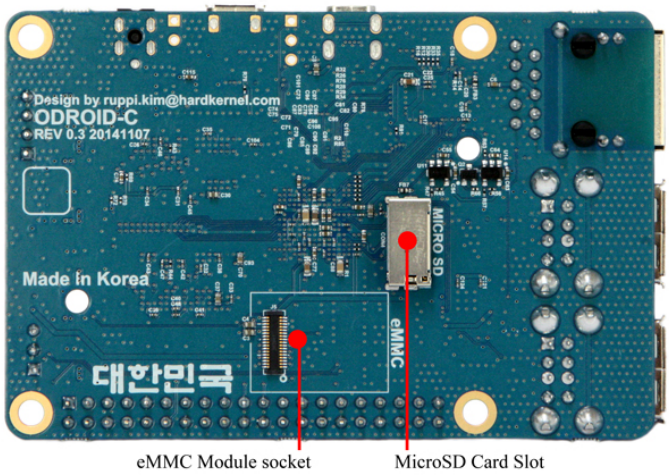


Figure 11: The Odroid C1's back component scheme

Processor	**Amlogic S805** : Quad Core Cortex™-A5 processor with Dual Core Mali™-450 GPU
RAM	**Samsung K4B4G1646D** : 1GByte DDR3 32bit RAM (512MByte x 2pcs)
eMMC module socket	8GB/64GB : **Toshiba eMMC** 16GB/32GB : **Sandisk iNAND Extreme** The eMMC storage access time is 2-3 times faster than the SD card. You can purchase 4 size options: 8GB, 16GB, 32GB and 64GB. Using an eMMC module will increase speed and responsiveness, similar to the way in which upgrading to a Solid State Drive (SSD) in a typical PC also improves performance over a mechanical hard drive (HDD).
Micro Secure Digital (MicroSD) Card slot	There are two different methods of storage for the operating system. One is by using a MicroSD Card and another is using an eMMC module, which is normally used for external storage for smartphones and digital cameras. The ODROID-C1 can utilize the newer UHS-1 SD model, which is about 2 times faster than a normal class 10 card. Note that there are some cards which needs additional booting delay time around 30 seconds. According to our test, most Sandisk Micro-SD cards don't cause the booting delay. We will make a compatibility list soon.
5V2A DC input	This is for 5V power input, with an inner diameter of 0.8mm, and an outer diameter of 2.5mm. The ODROID-C1 consumes less than 0.5A in most cases, but it can climb to 2A if many passive USB peripherals are attached directly to the main board.
USB host ports	There are four USB 2.0 host ports. You can plug a keyboard, mouse, WiFi adapter, storage or many other devices into these ports. You can also charge your smartphone with it! If you need more than 4 ports, you can use a powered external USB hub to reduce the power load on the main device.
Micro HDMI port	To minimize the size of the board, we used the Type-D micro-HDMI connector.
Ethernet RJ-45 jack	The standard RJ45 Ethernet port for LAN connection supports 10/100/1000Mbps speed. Green — Flashes when there is 100Mbps connectivity Yellow — Flashes when there is 1000Mbps connectivity
Status / Power LEDs	The ODROID-C1 has four indicator LEDs that provide visual feedback. Red — Power — Hooked up to 5V power Blue — Alive — Solid light : u-boot is running Flashing : Kernel is running (heart beat)
Infrared (IR) receiver	This is a remote control receiver module that can accept standard 37.9Khz carrier frequency based wireless data in NEC format.
Micro USB OTG port	You can use the standard micro-USB connector with Linux Gadget drivers on your host PC, which means that the resources in the ODROID-C1 can be shared with typical PCs. You can also add a micro-USB to HOST connector if you need an additional USB host port. Note that this port cannot be used for power input.
General Purpose Input and Output (GPIO) ports	These 40pin GPIO port can be used as GPIO/I2C/SPI/UART/ADC for electronics and robotics. The 40 GPIO pins on an ODROID-C1 are a great way to interface with physical devices like buttons and LEDs using a lightweight Linux controller. If you're a C/C++ or Python developer, there's a useful library called WiringPi that handles interfacing with the pins. We've already ported the WiringPi v2 library to ODROID-C1. Please note that pins #37, #38 and #40 are not compatible with Raspberry Pi B+ 40pin header. Those pins are dedicated for Analog input function. Note that all the GPIO ports are 3.3Volt. But the ADC inputs are limited to 1.8Volt.
Serial console port	Connecting to a PC gives access to the Linux console. You can see the log of the boot, or to log in to the C1 to change the video or network settings. Note that this serial UART uses a 3.3 volt interface. We recommend the USB-UART module kit from Hardkernel.
RTC (Real Time Clock) backup battery connector	If you want to add a RTC functions for logging or keeping time when offline, just connect a Lithium coin backup battery (CR2032 or equivalent). All of the RTC circuits are included on the ODROID-C1 by default. Molex 53398-0271 1.25mm pitch Header, Surface Mount, Vertical type (Mate with Molex 51021-0200)
Gigabit Ethernet PHY	**Realtek RTL8211F** is a highly integrated Ethernet transceiver that complies with 10Base-T, 100Base-TX, and 1000Base-T IEEE 802.3 standards.
USB MTT hub controller	**GENESYS LOGIC GL852G** is used to implement the 4-port Hub function which fully complies with Universal Serial Bus Specification Revision 2.0.
USB VBUS controller	**NCP380** Protection IC for USB power supply from OnSemi.
Boot media selector	If this port is opened, the first boot media is always eMMC. If this port is closed, the first boot media is always SD-card.
Power switch port	You can add a slide switch or rocker switch on this port if you want to implement a hardware on/off switch. If this port is closed, the power is off. If this port is opened, the power is on.
Power supply circuit	Discrete DC-DC converters and LDOs are used for CPU/DRAM/IO power supply.
Power protector IC	**NCP372** Over-voltage, Over-current, Reverse-voltage protection IC from OnSemi.

Figure 12: The Odroid C1's Specification

Solid Run's Hummingboard i1 is also considered as a candidate. Having a higher computational power that the Odroid C1 and the Raspberry Pi 2, it can handle heavier tasks than its contenders. The extra power, however, causes a bigger heat dissipation and a higher power consumption. This board is the only board of the three considered that needs a heatsink. Another issue is its lower popularity, that translates into a less community-debugged version of ARM Debian Linux [20]. It also has less USB 2.0 ports than both the Raspberry Pi 2 and the Odroid C1.

Advantages	Disadvantages
Higher CPU computational power	Needs heatsink
	Reduced community support
	Vaguely optimized Debian
	Only two USB ports

Table 7: Advantages and disadvantages of Solid Run's Hummingboard i1

The Hummingboard i1's front component scheme can be seen in Figure 13. A technical specification list of the board can be found in Table 8.

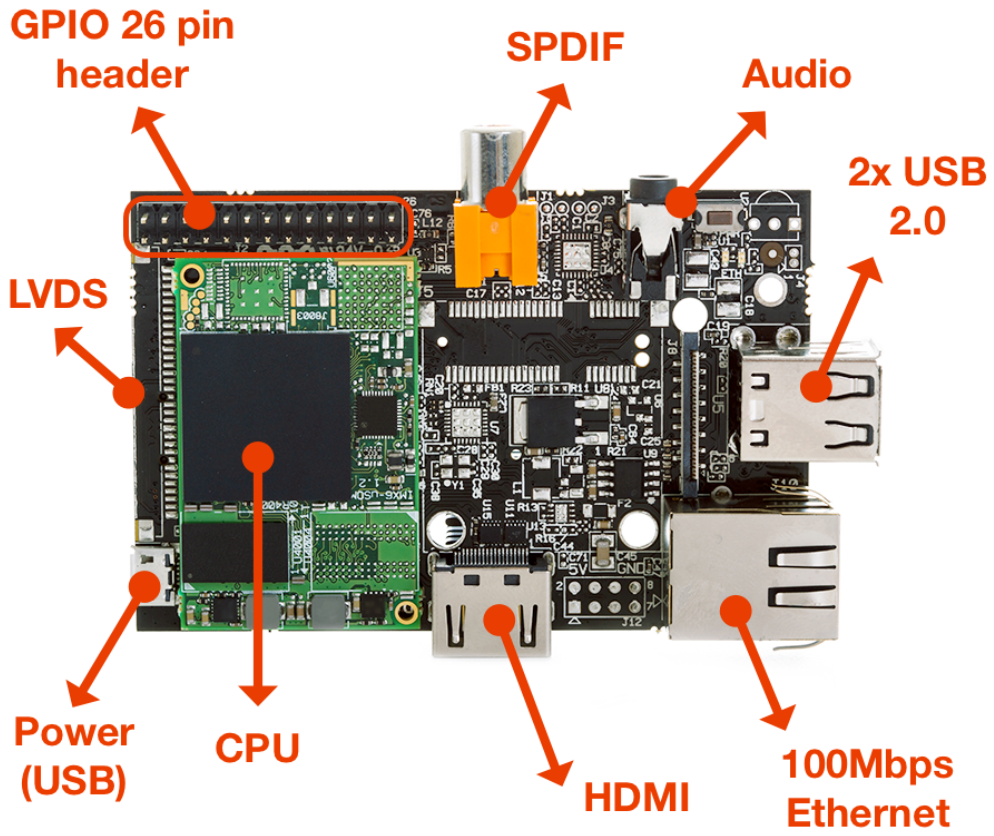


Figure 13: The Solid Run Hummingboard i1's front component scheme

Technical Specifications	
CPU	i.MX6 Dual Lite 2 cores
RAM	1GB 64 bit, 1GB @ 800Mbps
GPU	GC880 OpenGL ES1.1,2.0 Multi-format video decoder and encoder
Video IO	HDMI 1080p with CEC 1.4, 3D support MIPI CSI 2.0 Camera 2 Lane CSI-2
IO	Ethernet 10/100/1000 2x Powered USB 2.0 GPIO header UART, 8 GPIO, SPI with 2 CS, i1C
Audio IO	Coax SPDIF audio out PWM Mono output
Storage	UHS-1 Micro SD interface

Table 8: The Solid Run HummingBoard i1's specifications

Being the most popular of the three, the Raspberry Pi 2 model B has a big support community in its favor. The curated Debian Linux [20] it uses, called Raspbian OS [19], is also a big advantage, as it offers the biggest compatibility with well known Linux programs. A specification list can be seen in Table 10 and a component scheme is available in Figure 14.

Advantages	Disadvantages
Raspbian OS	USB/Ethernet performance anomaly
Big support community	
High compatibility with Linux software	

Table 9: Advantages and disadvantages of Solid Run's Hummingboard i1

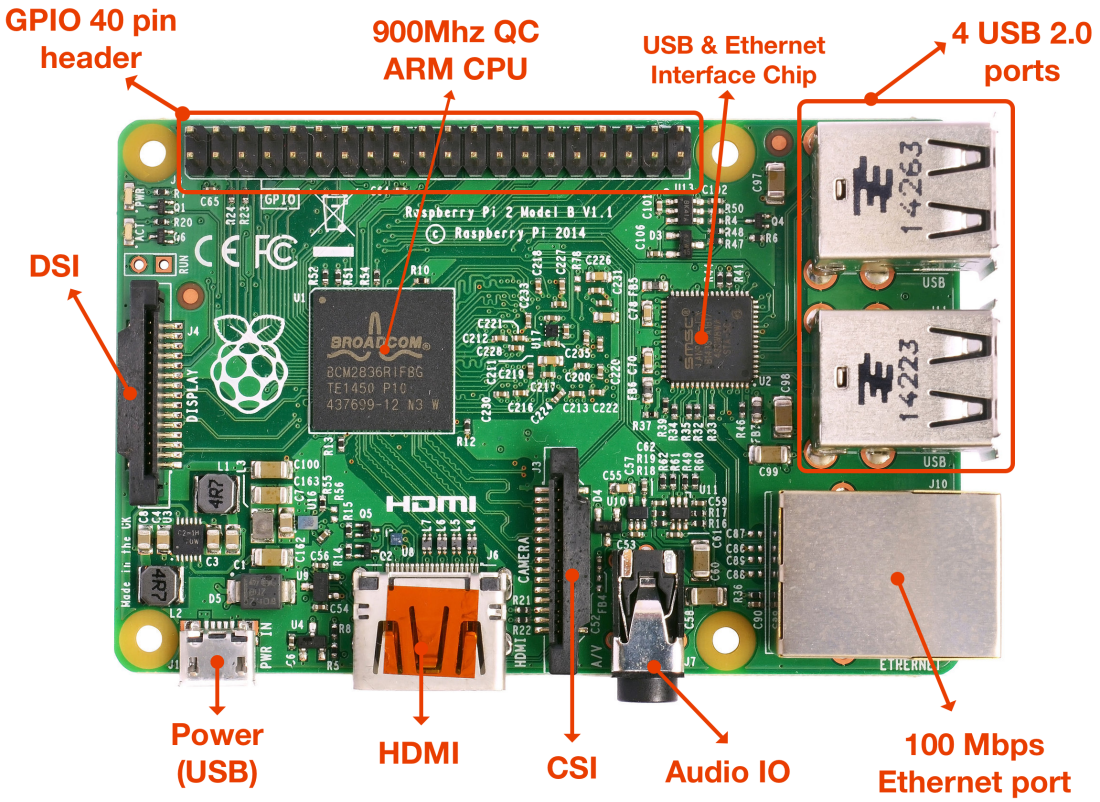


Figure 14: The Raspberry Pi 2 Model B's front component scheme

Technical specifications	
Processor	Broadcom BCM2836 ARMv7 Quad Core Processor powered Single Board Computer running at 900MHz
RAM	1GB RAM
IO	40pin extended GPIO 4 x USB 2 ports 10/100 Ethernet Port 4 pole Stereo output and Composite video port Full sized HDMI CSI camera port DSI display port
Storage	Micro SD port
Power	Micro USB power source

Table 10: The Raspberry Pi 2's technical specifications

Using a stripped down version of the Raspbian OS on the Odroid C1's and Raspberry Pi's, we were able to use them indistinctly, which helps with stock, customs and shipping delays either one may suffer. The customized Raspbian OS [19] is also mounted as a read-only image, as to increase the system's robustness in case of power failure or other abrupt power-offs.

Each sniffer is equipped with two IEEE 802.11 [10] capable usb dongles. One of them will provide a sniffing interface for Kismet [18]. The other will connect to the network infrastructure (see Section 3.5.1) in order to provide connectivity with the back end (see Section 3.4).

The dongle that provides the sniffing interface had to be carefully selected. Not all drivers will report accurate RSSI values and not all dongles are guaranteed to be perfectly compatible with the operating system's kernel. Therefore, the following criteria was employed to select a good match:

- RSSI value report consistency
- Raspbian GNU/Linux 7 [19] compatibility
- Driver compatibility with used Linux kernel

The best match turned out to be the TP-Link TL-WN722N (see Figure 15). The TL-WN722N wireless and hardware features are listed in Table 13 and 11 respectively. Operation details such as certification and operating temperatures can be found in Table 12

Figure 15: A photograph of the TL-WN722N network interface

HARDWARE FEATURES	
Interface	USB 2.0
Button	WPS Button
Dimensions (W x D x H)	3.7 x 1.0 x 0.4 in. (93.5 x 26 x 11mm)
Antenna Type	Detachable Omni Directional (RP-SMA)
Antenna Gain	4dBi

Table 11: TL-WN722N Hardware features

OPERATION DETAILS	
Certification	CE FCC RoHS
Package Contents	Wireless Adapter 4dBi detachable Omni directional antenna Resource CD
Environment	Operating Temperature: 0°C - 40°C Storage Temperature: -40°C - 70°C Operating Humidity: 10% - 90% non-condensing Storage Humidity: 5% - 90% non-condensing

Table 12: TL-WN722N Operation details

WIRELESS FEATURES	
Wireless Standards	IEEE 802.11n IEEE 802.11g IEEE 802.11b
Frequency	2.400-2.4835GHz
Signal Rate	11n: Up to 150Mbps(dynamic) 11g: Up to 54Mbps(dynamic) 11b: Up to 11Mbps(dynamic)
Reception Sensitivity	130M: -68dBm@10% PER 108M: -68dBm@10% PER 54M: -68dBm@10% PER 11M: -85dBm@8% PER 6M: -88dBm@10% PER 1M: -90dBm@8% PER
Transmit Power	<20dBm
Wireless Modes	Ad-Hoc Infrastructure mode
Wireless Security	Support 64/128 bit WEP WPA-PSK/WPA2-PSK
Modulation Technology	DBPSK DQPSK CCK OFDM 16-QAM 64-QAM
Advanced Functions	WMM PSP X-LINK(For Windows XP) Roaming

Table 13: TL-WN722N Wireless features

As for the dongle used for communication, there are no particular requirements. The cheapest is chosen and no homogeneity is required among devices.

3.3.3 Software

The most important program in the sniffers is Kismet [18]. For the purpose of the project, it's important to know that Kismet can run in server mode, in which it will open a local socket (`127.0.0.1:2501`). Any program can connect to this local socket and ask for certain information about what Kismet is sniffing through the configured interface. This information can be RSSI values, client MAC addresses, etc. They are referred to as 'sources' in Kismet's terminology.

The developed software inside the sniffers is focused on wrapping the Kismet [18] program running in server mode. The name of the wrapper program is `kismet_wrapper` and it was written in C [21] as to increase performance as much as possible. The task of `kismet_wrapper` is to:

1. Interact with the Kismet in order to enable sources and have Kismet report MAC address, RSSI and timestamp values from sniffed clients.

2. Collect information sniffed by Kismet from the local socket provided (`127.0.0.1:2501`)

3. Translate Kismet information into the following format:

```
<sniffer mac> <timestamp> <rssi> <client mac>
                          ...
ab:cd:ef:12:34:56 1234567890 -37 f6:e5:d:4:c3:b2:a1
```

4. Handle Kismet failures by monitoring its local socket's state. If failed, the program shall exit with a given code that will signal the bootscripts to take appropriate action (see Table 14).

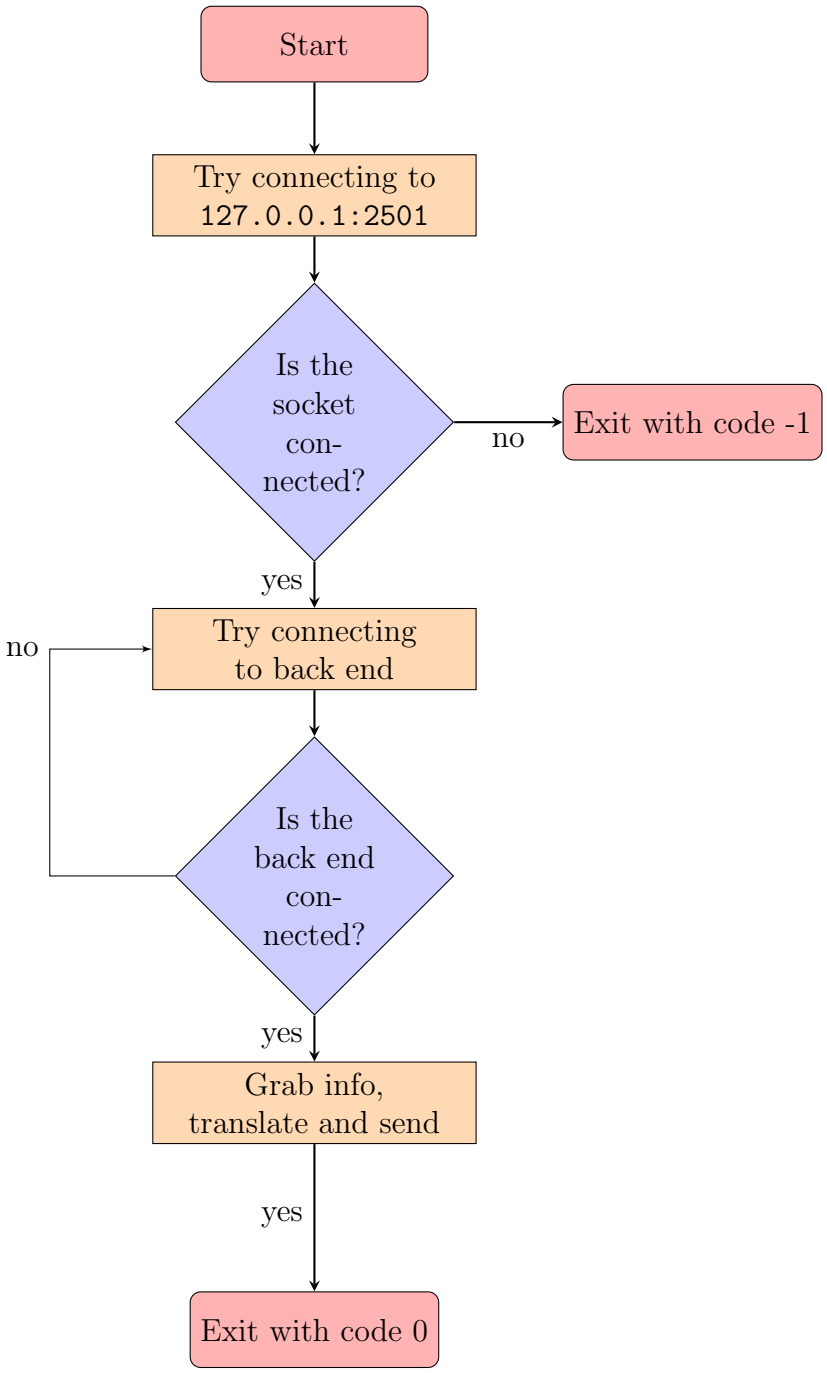
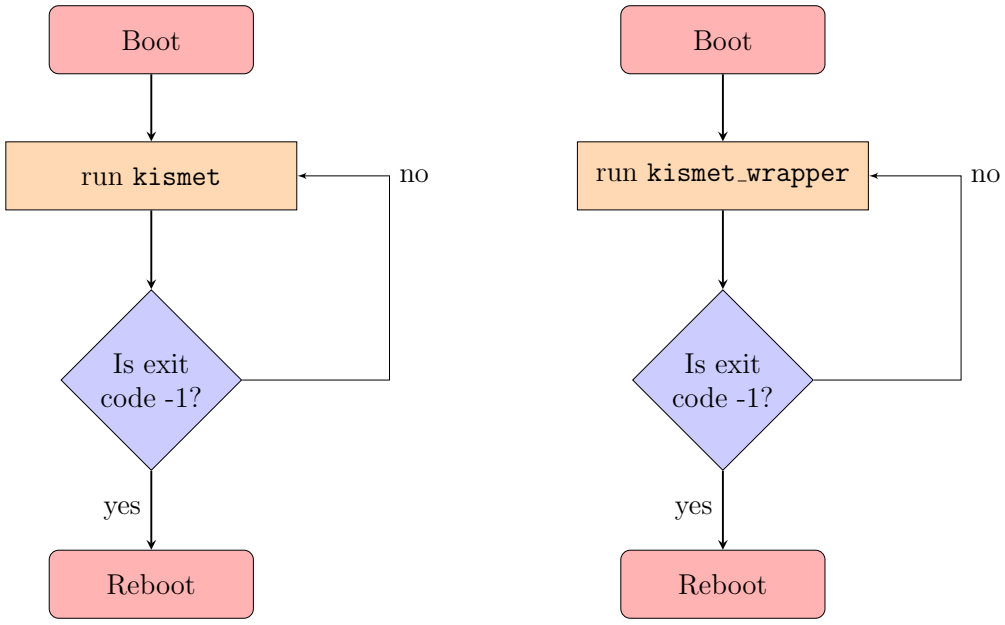


Figure 16: `kismet_wrapper` flow diagram

Kismet and `kismet_wrapper` are launched by two bash scripts that will relaunch them if they terminate. This bash scripts, called `kismet_bootscript` and `kismet_wrapper_bootscript` respectively, are scheduled to launch at boot time using `cron`. Watchdogs are used to reboot the system in case of system crash.

In order to facilitate configuration, a file named `location_service.conf` is placed inside the home folder, containing the address of the backend and other information used by the sniffer software. Once this info is changed, a simple

reboot is guaranteed to restart all software in an ordered way thanks to `cron` and the bootscripts.



(a) `kismet_bootscript.sh` flow diagram (b) `kismet_wrapper_bootscript.sh` flow diagram

Figure 17: Flow diagrams for bootscripts

Sniffers are also programmed to take actions autonomously in case of specific failures. Table 14 shows different failure points that are taken into account and the response that the sniffer takes to overcome them.

Error	**Response**
Kismet failure	Kismet reboot. Third time failing causes system reboot.
General communications failure	Communication interface reset. Third time: system reboot.
Back end communication failure	System reboot.
System block (watchdog)	System reboot.
Memory availability issue	Kismet (known memory leak) reset. Third time: system reboot.

Table 14: Table describing course of action in case of Sniffer failures

3.4 Back end

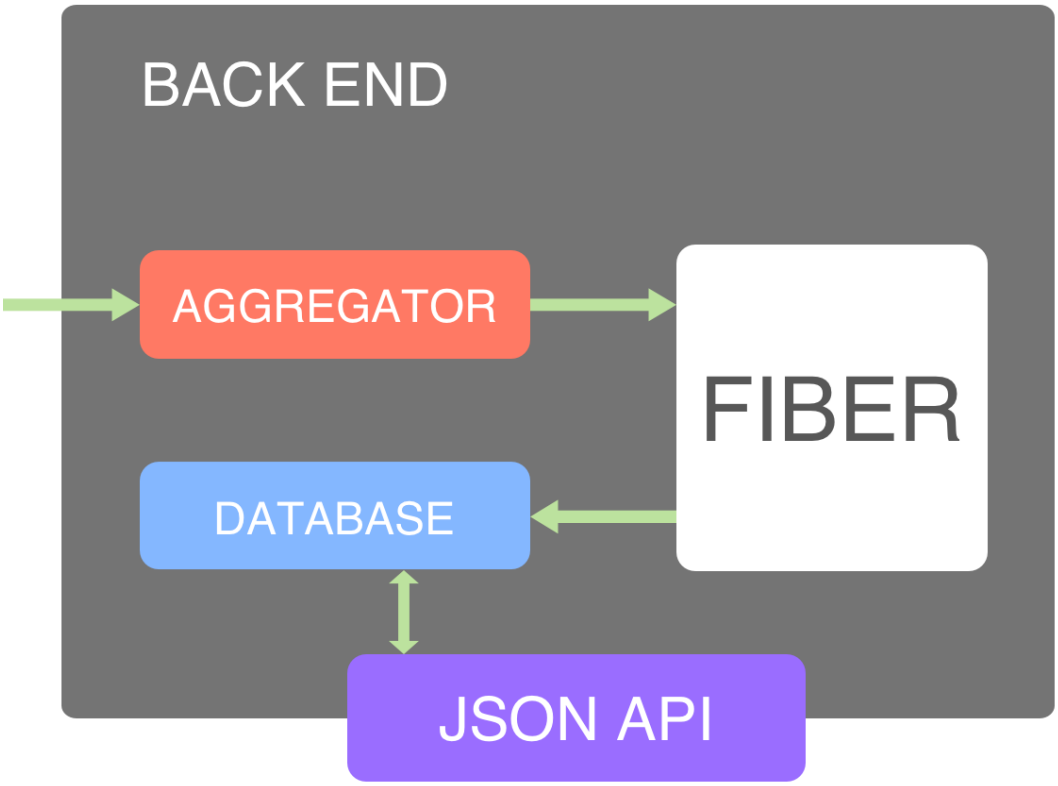


Figure 18: A closeup of the back end architecture

3.4.1 Requirements

The back end of the system is in charge of aggregating all the information provided by the sniffers, use it for calibration or to determine a user's location, returning that information in a JSON [27] formatted response via a web service.

3.4.2 Hardware

The back end integrates four separate software modules (location software, aggregator, database and API) that run in the same machine. This machine will be called 'back end server', and its recommended minimum requirements can be found in Table 15.

Back end server hardware
Server-grade power supply
Server-grade hard drive (1TB+)
Intel Core 2 duo (2.0GHz) +
Ethernet capable NIC

Table 15: A list containing requirements for the back end server

Basically, the hardware used for the server can be generic. Any box containing a sever-grade power supply (with always-on performance focus), server-grade hard drive (with continuous IO performance focus), standard Ethernet connectivity and dual core 2.0GHz or more processing power is more than enough.

3.4.3 Software Requirements

Operating System wise, a Unix [23] or Linux [24] based solution is required, as some of the software developed for FIBER relies on Unix Sockets [22] for communication among processes. Porting the system to a DOS based OS like Windows is possible but not advised. A list of software packets that must be present in the server can be found in Table 16

Software	**Version**
MySQL Server	14.0 (or higher)
Java JRE	1.8.0 (or higher)
Apache Tomcat Server	8.0 (or higher)

Table 16: Software requirements for the back end

3.4.4 Aggregator

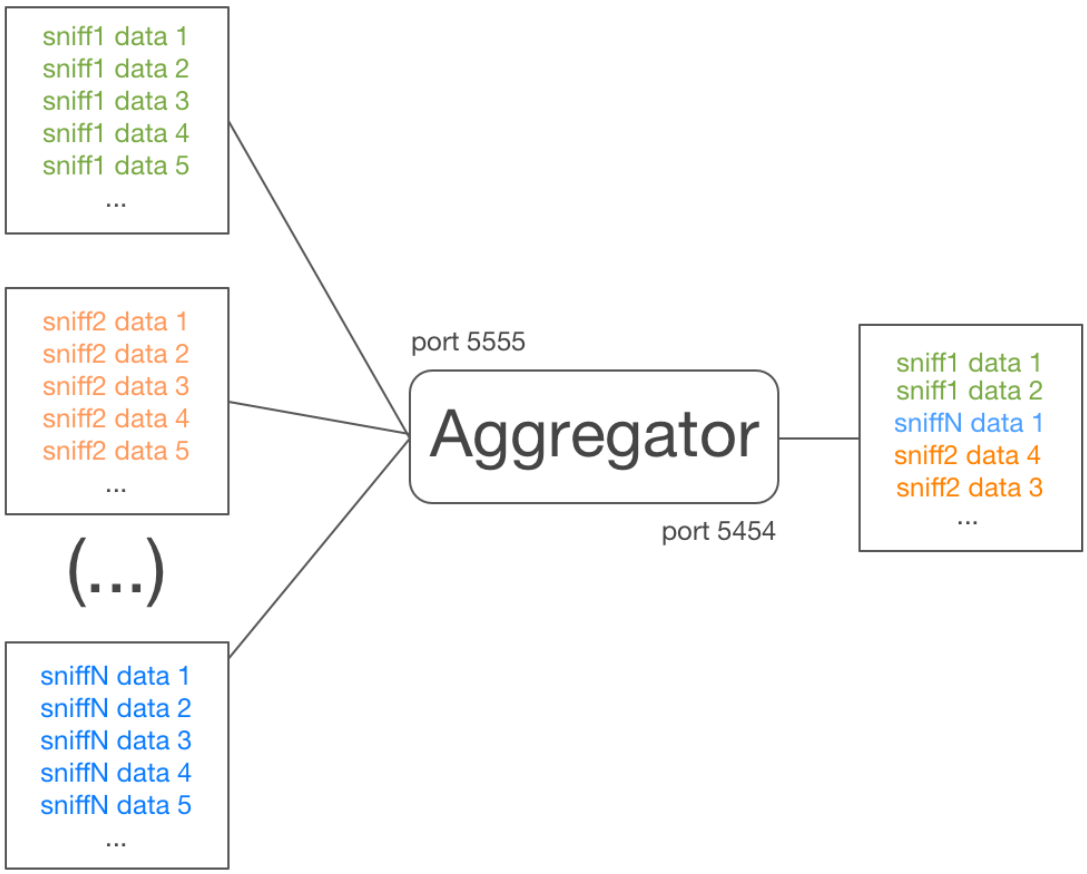


Figure 19: An illustration of how the aggregator module works

The aggregator is in charge of aggregating al data flows sent by sniffers into a single socket. Typically, sniffers will send flows to the aggregator on port 5555 and the aggregator will combine them into a single stream onto port 5454 of the same machine or a different one (more about this feature later).

The aggregator is written in C and is aimed is to provide the fastest and most efficient aggregation (one of the reasons it was not directly included in the Java written FIBER module). To achieve fast aggregation of up to hundreds of sniffers at a time, a memory-sacrifice strategy is used. A memory pool with reserved memory for each sniffer is managed by a thread that waits for newline on a flow before sending it off. The result is a line-per-measurement-per-sniffer joint output flow. The flow diagram for the aggregator program can be seen in Figure 20.

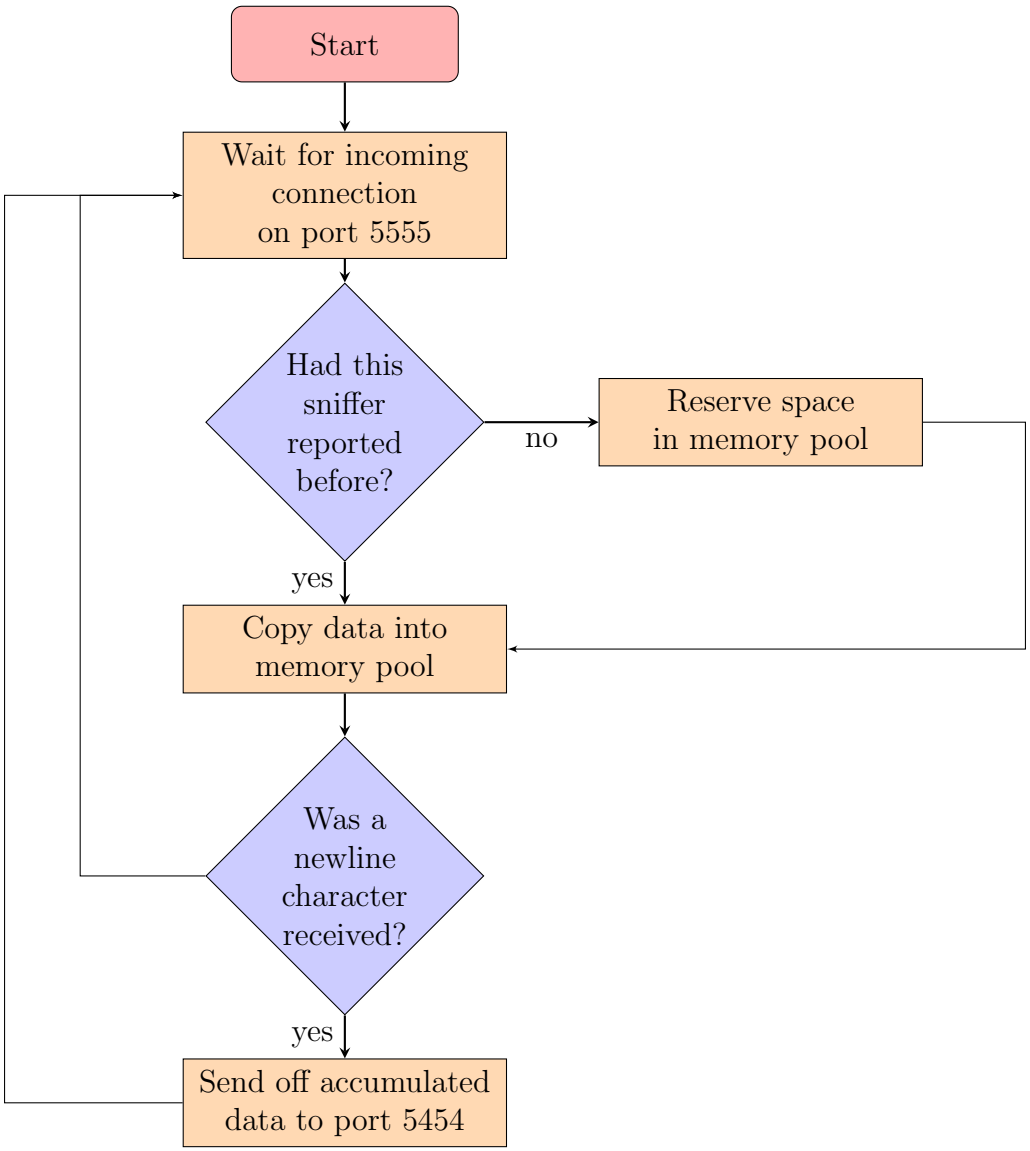


Figure 20: Aggregator flow diagram

Since the aggregator module is independent from the rest of the system, it can be exported onto another machine different from back end server. During the deployment, the range on the IEEE 802.11 [10] network might not be enough to cover the extension of the site. In that situation, one could run an instance of the aggregator on a sniffer, that will act as a hop in order to forward the data flows from out-of-range sniffers. This mesh network is described in Figure 21.

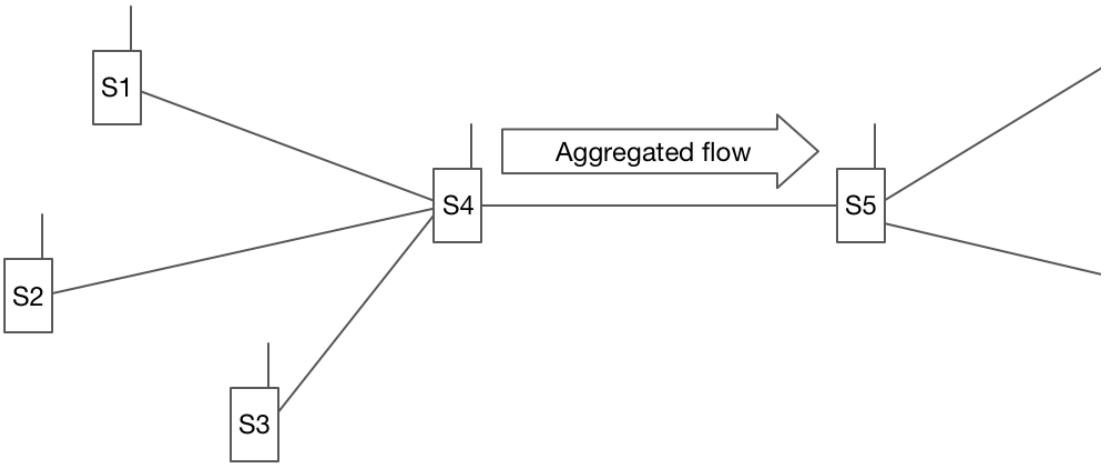


Figure 21: An example of how the aggregator module can be exported to a given sniffer (S4 in this example) in order to create a mesh network

3.4.5 Location software

The location software is the core of the system. It grabs the information from the aggregator as a unified stream and provides the following functionality:

- **Record map points** during calibration
- **Create radio maps** with calibrated points. This map is saved to the hard drive and used during localization.
- **Load radio maps** from the hard drive.
- **Computing the location of clients** based on live reports from all sniffers.
- **Insert new clients into the database** as they appear. Even if a location is not yet computed for the client, a registry is added so that information for 'counting' (see Section 1.1) is available immediately after the client is sniffed.
- **Update the database's** client records with new information on location. The task of removing 'old' clients relies on the database itself (see Section 3.4.6).

This software implements a programatic version of the methodology for calibration and localization that can be found in Section 3.6. The programs in this module can be though of as a suite, containing all programs listed in Table 17.

Program name	description
Mapper	Uses measurements from the calibration device to compose a radio map point. It then adds said point to an existing map or creates a new one depending on the parameters specified.
MapMerger	Takes two maps which's names are specified in the arguments and creates a new one as result of merging them together.
PointDeleter	Takes the coordinates of a given point and the name of the map in which it's contained and deletes it from said map. This facilitates the task of repeating the calibration of a given point without having to dump the entire map.
MapInfo	Displays information of the map which name is passed as a parameter. This information includes point's coordinates and the time and date they where captured at.
MapEvaluator	Once the ground truth on the client's location is fed as a parameter, it can evaluated the percentage of time the estimation of location matches said ground truth. It then scores the point, allowing FIBER deployers to evaluate a radio map point performance and ponder the need to recalibrate it.
Fiber	The location software itself. Takes a single parameter with the name of the radio map to locate users on.

Table 17: A table containing all programs in the location module suite and their descriptions

3.4.6 Database

The database used to store client's locations is a MySQL [29] database. In it, only one table named 'Clients' is defined. This table contains all the information relative to clients located by the system. The schema followed for this table is seen in 22. The table engine is `INNODB`.

```
+----------+---------------+------+-----+-------------------+-----------------------------+
| Field    | Type          | Null | Key | Default           | Extra                       |
+----------+---------------+------+-----+-------------------+-----------------------------+
| id       | int(11)       | NO   | PRI | NULL              | auto_increment              |
| mac      | varchar(255)  | NO   | UNI | NULL              |                             |
| floor    | int(11)       | YES  |     | NULL              |                             |
| lat      | decimal(10,7) | YES  |     | NULL              |                             |
| lon      | decimal(10,7) | YES  |     | NULL              |                             |
| lastseen | timestamp     | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+----------+---------------+------+-----+-------------------+-----------------------------+
```

Figure 22: The 'Clients' table database schema

- `id` is an integer value assigned to each client that is unique and is set automatically by the database.
- `mac` stores a string containing the client's MAC address.
- `floor` indicates the floor on which the client was located.
- `lat` stores the client's location in terms of latitude.
- `lon` stores the client's location in terms of longitude.
- `lastseen` is a timestamp that is automatically updated when the client record is updated (i.e, when the client's location is recomputed).

The database is also configured to launch an event scheduled every 10 seconds that deletes clients whose `lastseen` timestamp is older than a given amount of time. This is intended so the system 'forgets' clients that have presumably left the building.

3.4.7 JSON API software

In order to provide an API that returns JSON-formatted [27] data (see Figure 23) regarding located clients, the back end uses an Apache Tomcat [28] server, for which a web app has been developed using Java Servlets [26].


```
{
"clients": [
    {
      "floor": 0,
      "lat": -3.12847858,
      "lon": 22.09835672,
      "mac": "AB:CD:EF:12:34:56"
    },
    {
      "floor": 2,
      "lat": -3.42378558,
      "lon": 22.12452879,
      "mac": "A1:B2:C3:D4:E5:F6"
    },
    {
      "floor": 2,
      "lat": -3.54698451,
      "lon": 22.87564502,
      "mac": "AB:12:CD:34:EF:56"
    }
  ]
}
```


Figure 23: An example of a JSON-formatted response from the API

The API is developed following standard MVC [25] practices. There is a single view, which plots the locations of clients on a map. This is supposed to be an example, since the idea is for developers to develop much more complex front ends that can take advantage of FIBER through the API. The controller responds to both GET and POST petitions. The model for a client location is direct representation of the database's client (see Section 3.4.6).

A full interface to the database is provided through the API, so queries will be translated into MySQL [29] equivalents and the response from the database will be formatted into JSON [27].

This JSON-to-MySQL and vices-versa feature makes this solution very flexible, as it can accommodate any developer requirement. For example, if the developer only wants location information related to clients located on the second floor, a simple GET petition like `/fiber?floor=2` (or its POST equivalent) could return something like Figure 24.


```
{
"clients": [
    {
      "floor": 2,
      "lat": -3.42378558,
      "lon": 22.12452879,
      "mac": "A1:B2:C3:D4:E5:F6"
    },
    {
      "floor": 2,
      "lat": -3.54698451,
      "lon": 22.87564502,
      "mac": "AB:12:CD:34:EF:56"
    }
  ]
}
```


Figure 24: An example of a JSON-formatted response filtered by floor

There are many other uses of this direct database interfacing. In another example, instead of filtering by floor, a developer might be interested in geographical area, for which he/she would simply forge the following request:

```
fiber?lat=mt-3.64817264&lat=lt-3.89294822&lon=mt22.38274652&lon=lt22.87264512
```

(where `lt` stands for 'less than' and `mt` stands for 'more than'). The result will be location information regarding clients located on the rectangular area defined between latitudes -3.64817264 and -3.89294822 and longitudes 22.38274652 and 22.87264512. An illustrated example can be seen in Figure 25.

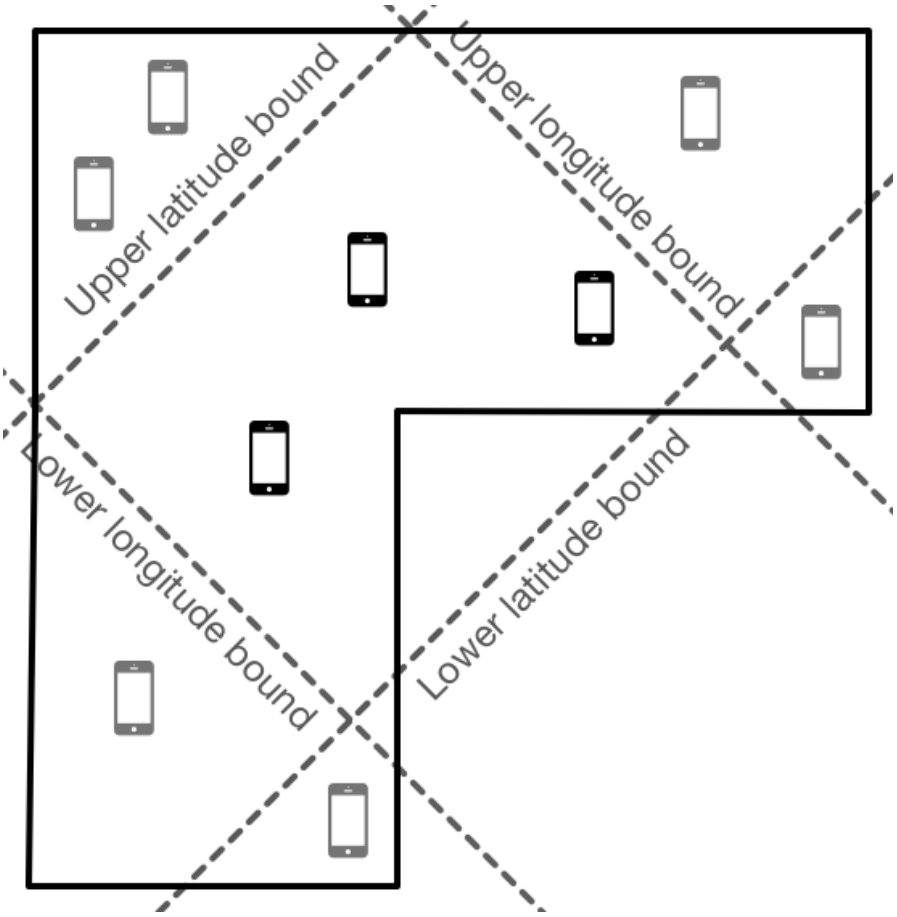


Figure 25: An example of latitude and longitude bound filtering

3.5 Infrastructure

In order for the system to be properly operated, power and network infrastructures are required. In this section we will go over the specific requirements that both need to fulfill.

3.5.1 Network

Two separate networks need to be deployed to ensure adequate operation of the FIBER system. One network, called client network, will be the one the clients are connected to (which the system won't require control of) and the other one, called sniffer network, will provide connectivity between the sniffers and the back end.

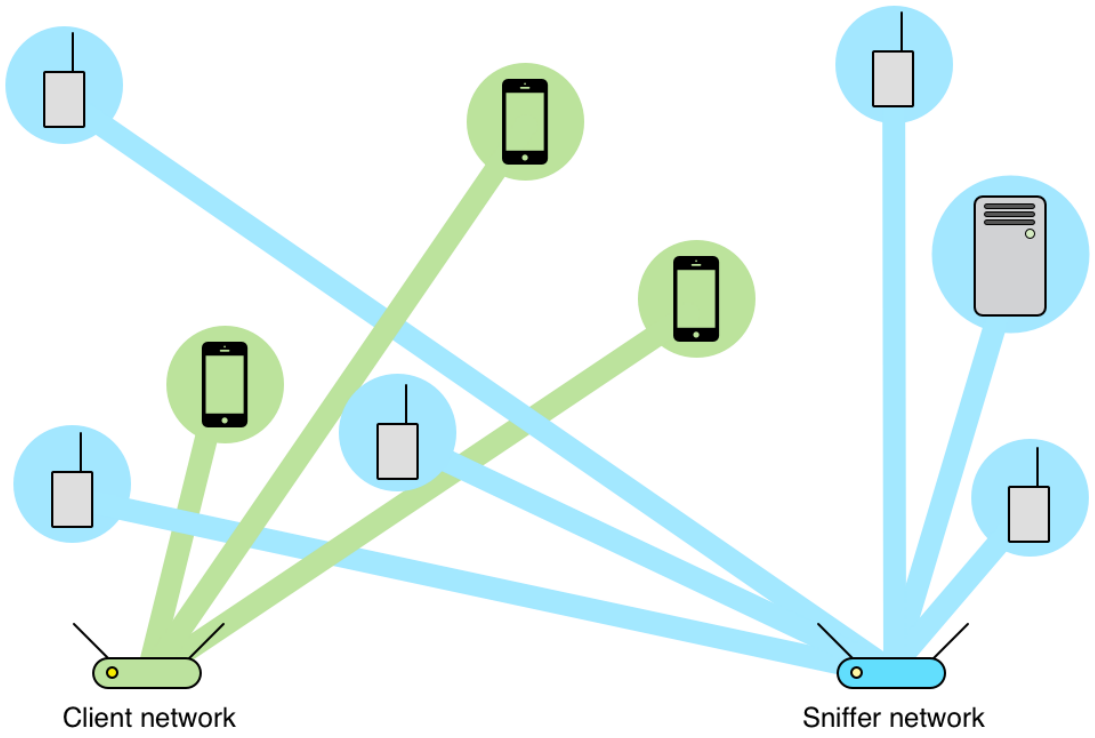


Figure 26: An illustration of how client and sniffer networks are separated

As will be mentioned in the methodology for calibration (see Section 3.6.1), there is a problem with co-channel interference that can cause multiple-spiked histograms (refer to terminology in Section 3.1) to appear in sniffer reports, causing an overall loss in accuracy. One way to avoid this interference is to have the sniffer and client networks operate in far apart IEEE 802.11 [10] channels. For example, one would run in channel 1 whilst the other one in channel 11. If the client network isn't in control of the system, a spectrum analysis should reveal an 'unpopulated' channel on which the sniffer network should be set to operate.

Another issue that could cause multiple-spiked histograms is power adaptation on the client's device or switching physical rates. To avoid this, IEEE 802.11g [10] (for fixed physical rate) is forced if the client network is under the system controls. If the client network is not being controlled by the system, alternative solutions can be found, which will be discussed in Section 5.

3.5.2 Power

All sniffers, access points and back end must be fed with electrical current to operate. This is a problem for sniffer placement, since areas with power availability are not necessarily the ones for better sniffer placement.

To overcome this problem, batteries are used for sniffers during the initial deployment. Once the calibration supports that the sniffer positions are optimum, proper power infrastructure should be deployed (i.e, cabling) to those locations. Information on the batteries used can be examined in Figure 27 and Table 18.

Figure 27: The RavPower Deluxe Series RP-PB22 battery

Specifications	
Model	RP-PB22
Capacity	13000mAh
Output	5V / 2.4A 5V / 2.1A
Input DC	5V/1.5A
Weight	0.31 Kg
Size	12.7 x 8.13 x 2.18 cm

Table 18: The RavPower Deluxe Series RP-PB22 specifications

3.6 Methodology

This sections discuses the procedures followed by our system to achieve client location. The way FIBER computes client locations can be explained in two stages: offline (or calibration) phase and online (or localization) phase.

3.6.1 Offline phase (or calibration)

The calibration is where a known device (called 'calibration device', i.e., a smartphone) is placed in a known physical location. Then, the device generates traffic (typically by pinging the access point) in order to be sniffed by a cloud of sniffers previously deployed at the site. In the back end, a histogram per sniffer (called sniffer report) containing RSSI values reported over calibration time is stored. All sniffer reports associated to that particular physical location are stored as a radio map point (or RMP) as well as the physical location's GPS coordinates (which should be known at the time of the calibration). This point will be one of several points that comprise the radio map. This process is repeated for all physical locations that will be contained in the radio map and constitutes what is known as fingerprinting [9].

The resulting radio map contains a set of radio map point formed by sniffer reports. These sniffer reports, should be comprised of single peaked histograms and in some cases, due to reflections in the medium, multiple-spiked histograms.

There is an issue, however, where co-channel interference in IEEE 802.11 [10] networks can cause the appearance of multiple-spike histograms in absence of reflections. Although not necessarily breaking the methodology, this low-RSSI interference reduces the amount of useful data in the histogram, reducing its 'resolution'. This can negatively impact the statistical inference that will be made in the online phase, causing an overall loss of accuracy.

To avoid co-channel interference, far apart IEEE 802.11 channels are used for communication between the sniffer cloud and the back end (sniffer network) and the calibration device (client network).

3.6.2 Online phase (or localization)

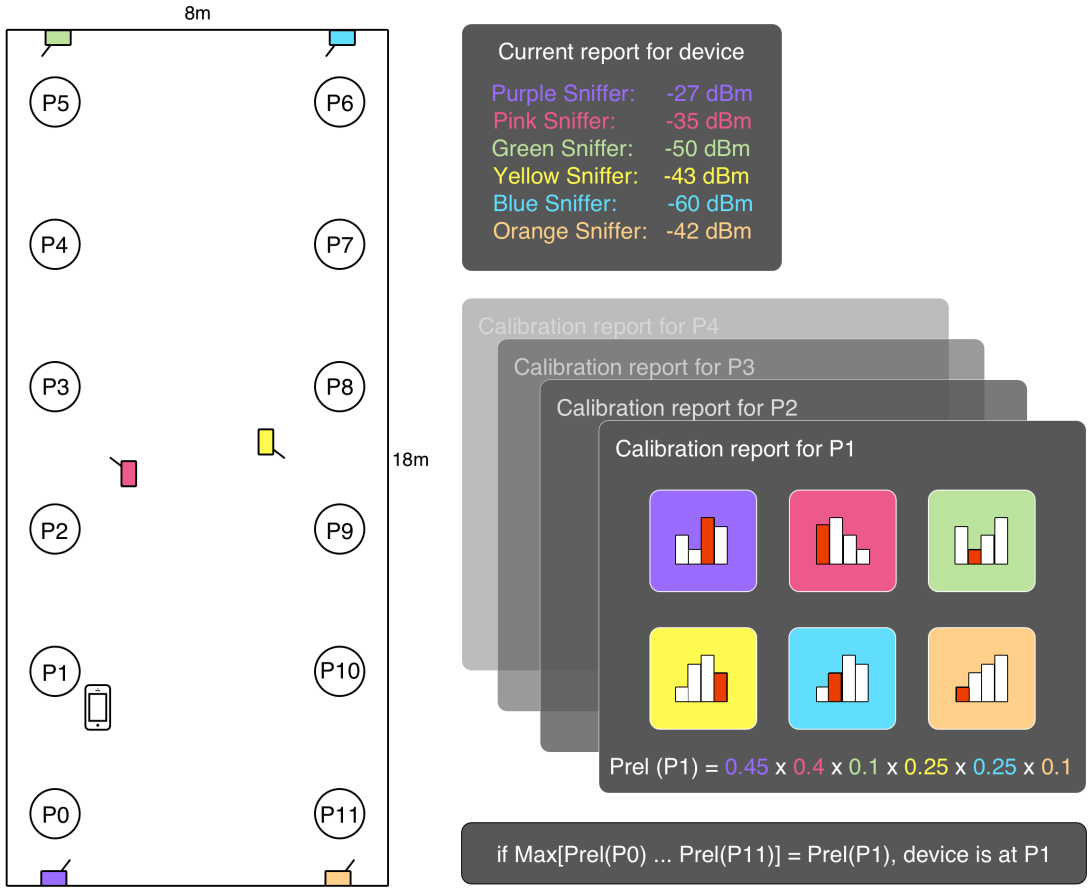


Figure 28: An explanation of how FIBER localization works

Let there be a client at a pre-calibrated site bearing a device that is transmitting through an IEEE 802.11 [10] interface. The sniffer cloud will start reporting on every packet the smartphone sends, including the device's MAC address and RSSI value of the transmission (see Section 3.3.3). The system will record a radio map point corresponding to the location of the user's device in the same way as during calibration but with two exceptions:

1. The physical location of the point is not known beforehand (that's what we are trying to determine)

2. Instead of requiring a number of measurements per sniffer report (as to guarantee sufficient histogram resolution), a parametrizable number of different sniffers must report at least one measurement

Once these two conditions are satisfied, the system has a new radio map point representing the location of the user that can be compared to the points in the radio map. Let the new point corresponding to the user's location be called *user point* (ϕ_u), the set of sniffer reports contained in it S_u and the latitude-longitude pair location values L_u. If there are N sniffer reports in S_u, then:

$$L_u = (lat, lon)$$
$$S_u = \{s_0, ..., s_N\}$$
$$\phi_u = [S_u, L_u]$$

The system must now determine what points in the radio map are more related to the user point. To do so, it determines what radio map points contain all sniffer reports in S_u. While the methodology could still work comparing the user point against all points in the radio map, filtering points that don't contain all sniffer reports in the user point (hence useless at this time) saves time and processing.

Now, we have the user point (ϕ_u) and a subset of candidate points (Φ_c). The candidate point that is most similar to the user point is probably the point the user is nearest to. The system must compare all sniffer reports contained in ϕ_u with those contained in all candidate points in Φ_c.

In order to do the comparison, a naive joint probability approach is used. Let the histogram in each sniffer report be normalized. Then, given an RSSI value, one could simply obtain the probability of that value happening in that particular histogram by looking at its frequency. In the following example, the probability with which the value -45dBm occurs in the histogram is 0.25 (or 25%).

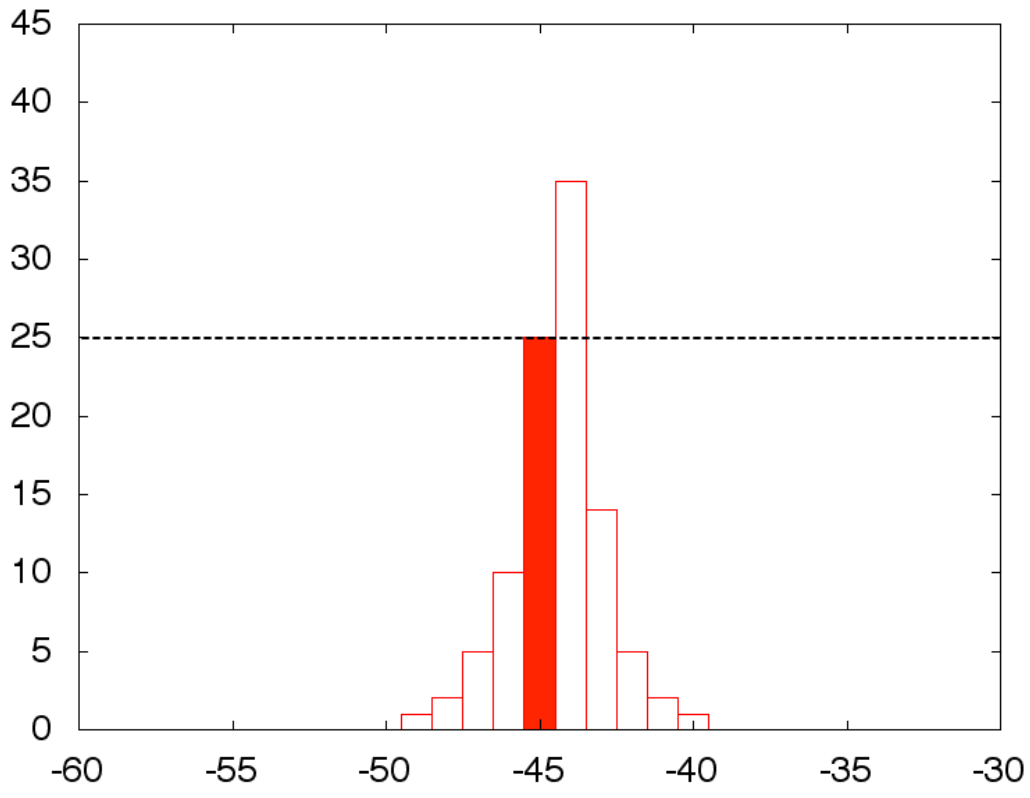


Figure 29: An example of normalized histogram, showing a 0.25 probability of a -45dBm measurement belonging to it

It follows that for the set of sniffer reports S_u in the user point ϕ_u, the probability that they would come from a given candidate point ϕ_c is simply the

joint probability that every sniffer report from S_u 'came from' its homologue S_c. We define this probability as *relation probability* P_{rel}. The location L_c of the candidate point with higher P_{rel} is the most probable user location L_u. So, the relation probability that the user is at candidate point 'c' (P_{rel}) is :

$$P_{rel} = p(s_{c0}|s_{u0}) \cap ... \cap p(s_{cj}|s_{uj}) \ \forall s_{cj} \in S_c$$
or
$$P_{rel} = \prod_{\phi_{cj} \in \phi_c} p(S_{cj} \mid S_{uj})$$

If we call the candidate point with the largest relation probability ϕ_{pref}, we can finally state:

$$\phi_{pref} = \phi_{ck} \text{ where } k = \arg\max_k \left(P_{rel0}, ..., P_{relk}\right)$$

Client location is:
$$L_u = L_{pref}$$

4 Results and evaluation

Once the system prototype is developed, it must be tested. The objective of these tests is to determine if the FIBER can deliver on the 3-4 meter accuracy promised. Having a good point-to-sniffer ratio is also crucial, since if too many sniffers need to be deployed to achieve target accuracy, cost will be excessive. We consider $\frac{1sniffer}{2points}$ to be the minimum ratio allowable.

The testbed is a 18 x 8 meter room, where 12 radio map points have been calibrated and 6 sniffers have been deployed. A single device will have to be located at the site as being in one of the 12 calibrated points. In order to more precisely measure the system's accuracy, the floor is divided into 1 m^2 tiles. The device will stay on each of these tiles long enough to be located 10 times (less than 10 seconds). The percentage of accurate localizations is represented as the legend in Figure 30 shows.

- **Accurate localization** means that the client device was said to be at the nearest radio map point. Nearest means on the 4 adjacent blocks or on the 2 blocks on top of which the point lays.
- **Failed localization** means that the client device was said to be elsewhere.

During the tests, the device is carried in a volunteers pocket. The device will be pinging the client network's access point constantly during the entire experiment. The client network is set to channel 11 and fixed to IEEE 802.11g [10] physical rate (to avoid power adaptation causing rssi shifts, see Section 3.1). The sniffer network is set to channel 1 with automatic physical rate. It should also be noted that the calibration was done with four angles (90º apart) per point and 30 samples per angle (30 sample resolution per histogram).

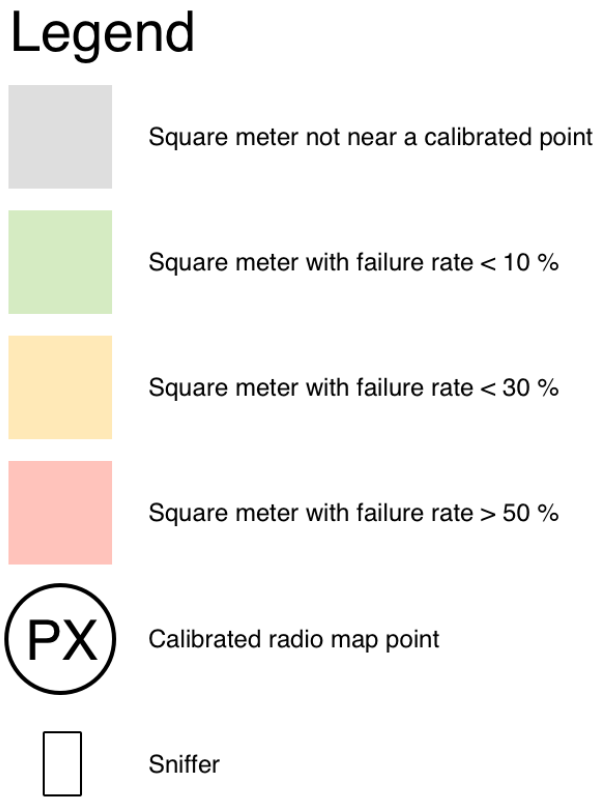


Figure 30: Legend for Figures 31 and 32

4.1 First set of results

The resulting heat map can be seen in Figure 31. Data inspection reveals 48 highly reliable tiles (less than 10% failure rate), 44 moderately reliable tiles (less than 30% failure rate) and 16 non reliable tiles (more than 50% failure rate). Out of 108 tiles, **44.45% were reliable**, **40.74% were moderately reliable** and **14.81% were not reliable**.

Highly reliable	44.45%
Moderately reliable	40.74%
Non reliable	14.81%

4.1.1 Window method

Moderately reliable tiles can be more useful by taking a time window of N location attempts. For example, storing 10 locations and reporting the mode location (the most occurring one) as the client's location, will make the system's guess right most of time.

Using this technique, we can re-evaluate the system performance. In this case, we take into account that green and yellow tiles will yield good results and red tiles will not. If each point is assigned 6 tiles that should be mapped to it, we can talk about *useful area* around a point. We now seek points with 100% useful area

and call them *good points*. **Note** that the rooms' corners will always yield bad results and therefore will not be considered.

Using this analysis (and ignoring corners), we see that 7 points are *good points* and 5 are *bad points*. The result is a **58.33%** of accurate location, which is not impressive.

To improve this ratio, we use FIBER's software tools (see Section 3.4) to re-calibrate *bad point*. The results are shown in the following section.

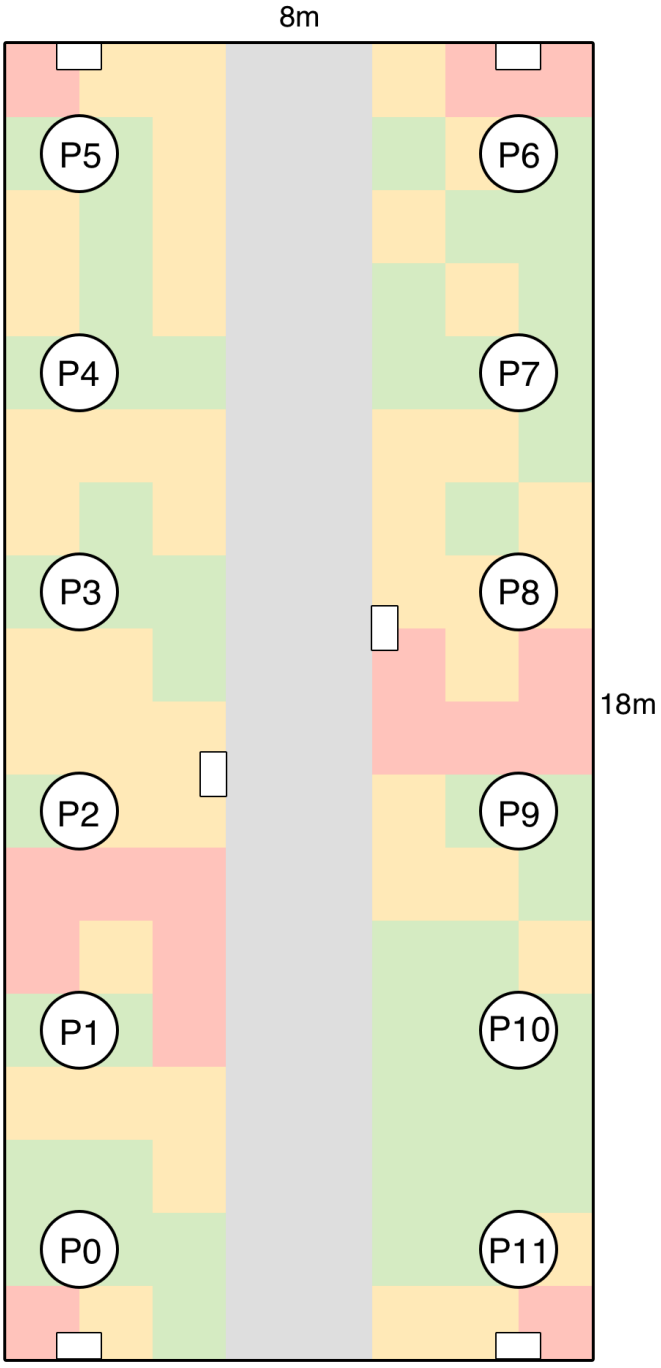


Figure 31: Results on overall system accuracy (legend in Figure 30)

4.2 Results after recalibration

After the recalibration, we see that direct analysis yields better results than before, with 57 highly reliable tiles (green), 46 moderately reliable tiles (yellow) and only 5 non reliable tiles (red). So, this time, out of 108 tiles, **52.78% were**

reliable, **42.59% were moderately reliable** and **4.63% were not reliable**. The resulting heat map can be seen in Figure 32.

Highly reliable	52.78%
Moderately reliable	42.59%
Non reliable	4.63%

The most noticeable change is that **non reliable tiles (red) have decreased from almost 15% to less than 5%**. By applying the window method with these results we obtain **11 good points** and only one bad point (P6). **The overall accuracy of the system has increased to 91.67%**, making the system accurate enough for our needs.

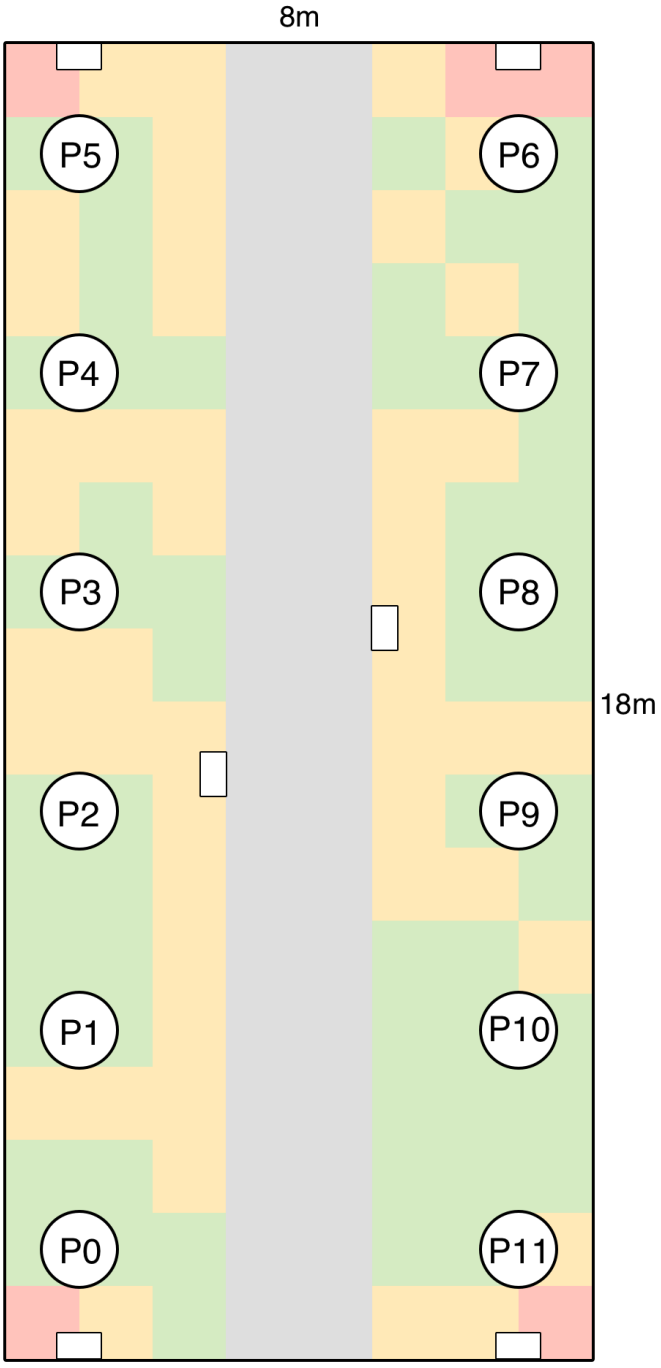


Figure 32: Results on overall system accuracy after recalibration (legend in Figure 30)

5 Future work

There are some improvements that can be made to the FIBER system in the future. These improvements focus on increasing the system's robustness and speed, while maintaining or improving the system's overall accuracy.

5.1 Improving speed

As discussed in Section 4, a windowing method is used to filter bad location estimations based on the fact that they happen less often than good estimations. This method consists in waiting for a parametrizable amount of estimations on the client's location to be made and then taking the mode estimation as the valid one (since it happened more often). While this technique improves the system's overall accuracy, it also implies a time delay, which slows the system down and makes moving clients harder to track. Two approaches can be followed to overcome this problem:

1. Bluetooth [13] integration
2. Viterbi-like trajectory reconstruction for moving clients

5.1.1 Bluetooth integration

Let there be a window of locations formed by several estimations. Some of the estimations will be wrong, and some will be right. Let the wrong estimations be majority and, hence, the mode estimation be wrong. If sniffers are equipped with Bluetooth [13] sensors that can detect the device and their range is smaller (a few meters) than a typical IEEE 802.11 [10] wireless range, some sniffers' reports may be discarded as being to far away from the client to be relevant and not be considered for the estimation.

If this range-based discrimination is applied to the previous window of locations, it may happen that wrong estimations are discarded, thus the good estimations considered only. This technique can help when reflections not accounted for during calibration are altering RSSI measurements from the sniffers.

Using this technique might help reduce the window for locations, reducing the time needed for the estimation of a clients locations, increasing system speed.

5.2 Viterbi-like trajectory reconstruction

In a very similar way as the Viterbi algorithm [34] is used to reconstruct symbol sequences in channels with memory, a succession of possible locations for a moving client can be used to determine the most plausible trajectory taken by said client. For every step, a constellation of possible points could be plotted. If one particular set of estimations suggests a client is moving at vehicle speeds, that potential trajectory can be discarded. With enough steps, a single possible trajectory may remain, which would probably the one taken by the client.

Another benefit of this method is that its complexity only grows linearly with the number past locations (steps) take into account. If the number of past locations

is smaller than the window used in the current implementation, it will also improve speed.

5.3 Improving robustness

One weak point of the current system is that the use of fingerprinting [9] implies a calibration that is subject to be rendered useless on significant medium change or device heterogeneity. To overcome the medium issues, a set of recommendations on sniffer placement and calibration techniques have been discussed in Section 3.3.1. The problem with tracking different devices, however, has to be solved in another manner.

The problem with trying to locate different devices is that different device interfaces may have different RSSI mean power shifts, that make the calibration made with one device useless for another. To approach this problem, relative histograms are proposed.

5.3.1 Relative histograms

Instead of storing one histogram per sniffer that sees the calibration device during calibration, relative histograms will store the difference in RSSI values between the sniffers. Say, for example, that sniffers A, B, C and D see the calibration device. Then the following histograms will be stored (note that some of these histograms are redundant due to transitivity, so the implementation would be more efficient):

A - B
A - C
A - D
B - A
B - C
B - D
C - A
C - B
C - D
D - A
D - B
D - C

Once a map is constructed this way, when a client is trying to be located, its reported online values would be relativized in the same way. The advantage is that relative histograms do not take into account mean values, hence different devices with different mean RSSI values should be able to be located without major problems.

6 Budget and work planning

6.1 Budget

This section is devoted to present an estimated cost for the FIBER system development in terms of labor force, hardware, software and indirect costs.

6.1.1 Labor force cost

Five workers were involved in the development of FIBER. Three professors who provided guidance in a weekly meeting, one telecommunications engineer with expertise in hardware and operating systems and a telecommunications student intern (the author of this report) with some background in system development. All salaries are shown before taxes.

Qualification	Contribution	Amount	Salary [€/h]	Total [€]
Professor	2 h/week guidance	10	18.5	3700
Professor	2 h/week guidance	10	18.5	3700
Professor	2 h/week guidance	10	18.5	3700
Telecom. Engineer	6 h/day	47	12.5	3525
Intern	5 h/day	47	6	1410
TOTAL				**16035 €**

Table 19: Labor force costs table

6.1.2 Hardware costs

A considerable amount of hardware is needed for the system to work. Since the prototype of this project is destined to a client, no amortization is taken into account, as it's a one-time cost for the team that will not be reused. Detailed information on hardware costs is shown in the following tables, organized by sniffer hardware, back end hardware and infrastructure hardware.

Sniffer hardware costs			
Expenditure	**Amount**	**Cost per unit [€]**	**Total [€]**
Raspberry Pi 2 Model B	17	41.2	700.4
SolidRun Hummingboard i1	3	61.63	184.9
HardKernel Odroid c1	6	30.82	184.92
SunDisk MicroSD cards	26	5	130
5V power source	26	5.28	137.28
Enclosures	26	7.04	183.04
TpLink TL-WN722N	26	9	234
LogiLink	26	8	208
SUBTOTAL			**1962.54 €**

Table 20: Sniffer hardware costs table

Back end costs			
Expenditure	**Amount**	**Cost per unit [€]**	**Total [€]**
Dell Server	1	645.3	645.3
SUBTOTAL			**645.3 €**

Table 21: Back end hardware costs table

Infrastructure hardware costs			
Expenditure	**Amount**	**Cost per unit [€]**	**Total [€]**
Batteries	26	23.6	613.6
Ethernet cables	5	6	30
Power cable extenders	3	8.3	24.9
SUBTOTAL			**668.5 €**

Table 22: Infrastructure hardware costs table

Some prices were converted from dollars to euros. Shipping costs and customs fees were not included. VAT was included. A total hardware cost table is presented in Table 23.

Total cost table	
Expenditure	**Total [€]**
Sniffer hardware	1962.54
Back end hardware	645.3
Infrastructure hardware	668.5
TOTAL	**3276.34 €**

Table 23: Total hardware cost stable

6.1.3 Software costs

Only free software under MIT license [30] or BSD 2-clause [31] license were used in the development of this project. No costs are observed in this area.

6.1.4 Total cost

If we add the costs for labor force, hardware and software, we have an estimate of the FIBER's development cost. The total cost is 20137.14 €. Infrastructure cost still have to be discussed, as they depend on the total cost (see Section 6.1.5). For a more detailed view, see Table 24.

Total project cost	
Expenditure	**Total [€]**
Labor force	16035
Hardware	3276.34
Software	0
TOTAL	**19311.34 €**

Table 24: Total project costs table

6.1.5 Indirect costs

Indirect costs are considered to be all expenses that are related with the development and deployment of the system. As this project was developed in a research centre, a 25% of the total cost is given to the centre to pay for office area, computers, materials, etc. Other indirect costs are related to the trip made to a Spanish city for the deployment.

Indirect costs			
Expenditure	**Amount**	**Cost per unit [€]**	**Total [€]**
Working infrastructure	-	-	4827.84
Train tickets	2	82.3	164.6
Hotel room night	2x3	75.2	451.2
Daily trip expense	2x3	70	210
TOTAL			**5653.64 €**

Table 25: Indirect costs table

6.2 Planning

In this section we will discuss what activities took place in the development of FIBER and their duration. We will list them in Section 6.2.1, analyze how they interact with each other using a PERT diagram in Section 6.2.2 and finally show them on a timeline using a Gantt diagram in Section 6.2.3.

There are some general notes to take into account in this section. The system is developed as an evolving prototype and only the first few iterations are developed (stays as a prototype) in this report. In time planning, a day counts as 5 hours, since that is the time per day devoted to the development of the system.

6.2.1 Activities table

The activities described in Table 26 sum 300 hours. It's important to note that some activities where done in parallel to others, so the PERT (see Figure 33) and Gantt (see Figure 34) will reveal lower 'minimum' times. In reality however, the project spanned roughly 250 hours. The rest of the time was spent assembling this report. A list of these activities and their required predecessors follows:

Activity	Label	Requirements	Duration
Background research on RILS	A	-	10
Develop a list of hardware candidates for sniffers	B	A	10
Obtain sniffer hardware candidate test units	C	B	75
Test sniffer candidate hardware	D	C	15
Obtain definitive sniffer hardware	E	D	75
Develop sniffer software	F	C	25
Test SOA alternatives	G	D	30
Design FIBER system	H	G, F	10
Implement FIBER prototype	I	H, E	30
Test system	J	I	10
Improve system	K	J	10

Table 26: A table showing the activities that took place during the development of FIBER as well as an assigned label for further reference, predecessor activities and their duration in hours

These activities can be categorized into four groups: Previous work, Hardware decisions, Design and development and Evaluation and tweaking. The following tables show a further description of each activity within its group (Tables 27, 28, 29 and 30 respectively).

Activity	Description
A	**Background research on RILS** consists in a finding out what is the state of the art in real-time indoor location systems. Said work can be found in Section 2.2.

Table 27: Activities belonging to the 'Previous work' category

Activity	Description
B	**Developing a list of hardware candidates for sniffers** is necessary for the project as sniffers are an essential part of the system that can't be emulated on any other hardware (not like the back end that runs on a laptop during development). Information about the hardware candidates considered can be found in Section 3.3.2.
C	**Obtaining sniffer hardware candidate test units** consists in finding adequate prices and most convenient shipping. Customs and shipping delays are considerable and therefore are taken into account.
D	**Testing sniffer candidate hardware** is done once it arrives. This activity consists in finding out which hardware is best according to the needs of the project.
E	**Obtaining definitive sniffer hardware** is done once the tests have revealed the best hardware for the project's needs. The same process as for C is followed.

Table 28: Activities belonging to the 'Hardware decisions' category

Activity	Description
F	**Developing sniffer software** comprises developing all the software programs the sniffer needs to operate as such. A list of said programs and details about their implementation can be found in Section 3.3.3.
G	**Testing SOA alternatives** meant gathering some hands-on knowledge of the state of the art systems (see Section 2.2) in order to learn from them. This process is essential to better design FIBER.
H	**Designing FIBER system** consists in creating a list of requirements for the system and determine how they will be met. A detailed description of FIBER's architecture can be found in Section 3.2.
I	**Implementing FIBER prototype** is done once the design has been completed. A technical discussion of the FIBER implementation can be found in Section 3.

Table 29: Activities belonging to the 'Design and development' category

Activity	Description
J	**Testing the system** means evaluating how many of the requirements set in activity H are met. A discussion on the results evaluation of FIBER can be found in Section 4.
K	**Improving the system** is refactoring the code and refining the original procedures based on the test's experience.

Table 30: Activities belonging to the 'Evaluation and tweaking' category

6.2.2 PERT diagram

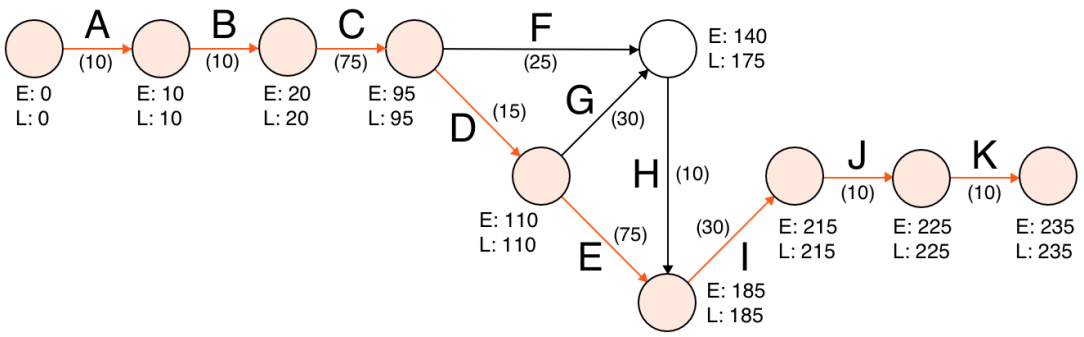


Figure 33: A PERT diagram showing all activities from Table 26 (critical path in orange)

The critical path analysis of the PERT diagram (shown in orange) reveals that the minimum duration of the project is 235 hours which, in terms of a 5 hour

per day contract, means 47 days. If weekends are considered, the duration of the project is 9.5 weeks. In reality, some slack was used to further research each stage and improve over software implementations. Slack times can be inspected in Table 6.2.2.

Activity	**Free** $(E_j - E_i - d_{ij})$	**independent** $(E_j - L_i - d_{ij})$	**Total** $(L_j - E_i - d_{ij})$
A	-	-	-
B	-	-	-
C	-	-	-
D	-	-	-
E	-	-	-
F	20	20	55
G	-	-	35
H	35	-	35
I	-	-	-
J	-	-	-
K	-	-	-

Table 31: A table showing free, independent, and total slack times for all activities

Slack times reveal that activities F, G, and H which are related to developing sniffer software, testing SOA alternatives and designing FIBER can have 55, 35 and 35 hours of delay respectively. This is optimum, since development and design activities are prone to consume more time than estimated and, in reality, did so.

6.2.3 Gantt diagram

In this section, a Gantt diagram of the project is shown in Figure 34. A work day is considered to last 5 hours. The project starts on Monday 16th of March and ends Tuesday 19th of May for a total duration of 47 days or 235 hours (as the PERT diagram in Figure 33 shows).

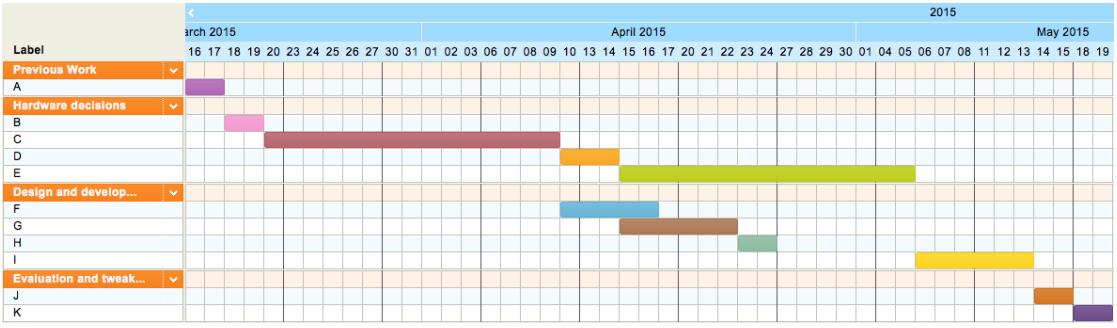


Figure 34: The project's Gantt diagram (one day is 5 hours of work)

7 Conclusions

With FIBER we set out to fill a gap in RILS for a 'good-enough' accuracy system that would be convenient, expandable and at a low cost. In roughly 300 hours, we met our objective and presented a system that could deliver on all promises made, localizing clients indoors with a 3-4 meter accuracy. On top of this, the system is attractive to developers thanks to the easy JSON interface it offers.

The future of FIBER is planned out, with improvements to be made to the methodology and software. Therefore, the health of the project is good and its future is assured provided it attracts clients and developers, which shouldn't be difficult thanks to its versatility and low cost.

Appendices

A Summary

What you're about to read is a section-by-section summary of the whole document. In each of these summarized sections, you will find a reference to the corresponding full document's section where you may continue reading if interested. While this summary is good enough to transmit a *rough idea* of the project, bear in mind that some important aspects of the development are not featured in it. Please be encouraged to read the full document.

A.1 Motivation and objectives

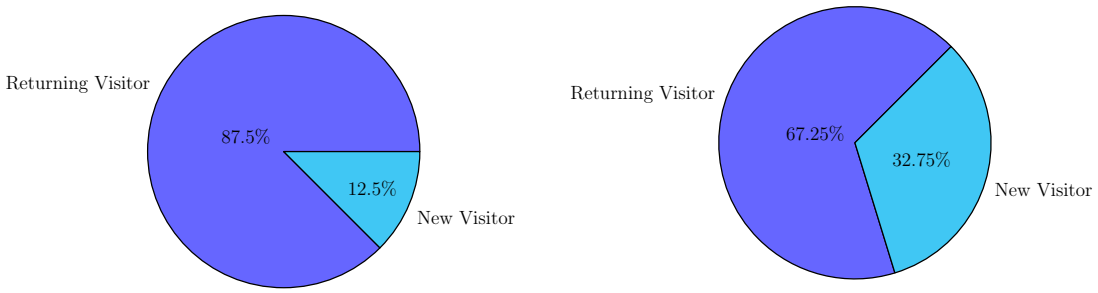
Business intelligence, marketing, employee tracking, security and safety are just some of many applications that can be built on top of Real-time Indoor Location Systems (or RILS). RILS can contribute to these areas, making them more intelligent, less intrusive and more accurate overall. In this section we will describe real scenarios where this technology is being used and the benefits it has yield thus far.

A.1.1 Business Intelligence and Marketing

The most fitting scenario for describing RILS and business intelligence combined is the so called *Smart Shopping Mall*. This type of mall includes a deployment of RILS and therefore knows more about smartphone users that shop in it. This information (a collection of user's MAC addresses and an estimate of their location inside the shopping mall), is compiled by business intelligence agents and used to aid the mall's commercial staff in a variety of ways.

In a similar fashion as website traffic analysis tools (such as Google Analytics [11]) have aided web owners [5], RILS provides mall owners with information about new vs returning visitors, visitor flow through the mall, visitor to client conversion, etc. This information, as happens in the web, can be critical to assess factors such as:

- Marketing campaign success rate (How many new visitors came to the mall?)
- Floor plant design (Why are users flowing more through certain corridors?)
- Mall space valuation (This area is more expensive because more people flow through it)



(a) Before Marketing campaign (b) After a successful Marketing campaign

Figure 35: A representation of how New Visitor vs. Returning Visitor information can provide useful insight on Marketing campaign success evaluation

[Read more in Section 1.1]

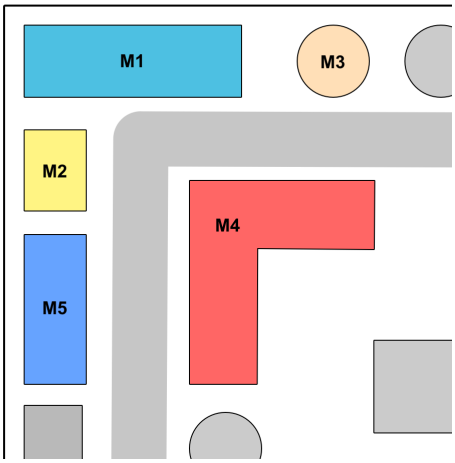
A.1.2 Employee Tracking

Companies from diverse industries have long tracked their employees. Motivations for doing so are diverse, ranging from measuring productivity to making sure employees are well rested.

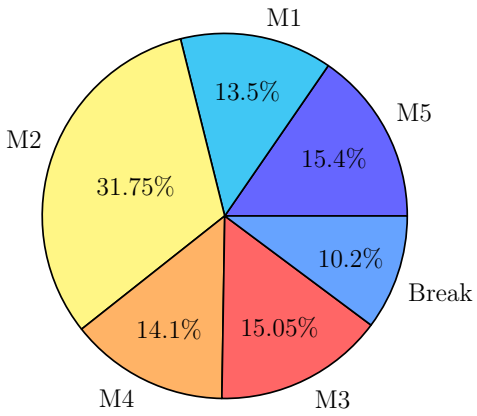
(...)

RILS can provide a solution for this scenario by pinpointing employees' locations and mapping them to working places. By building an employee tracking application on top of RILS, developers can keep historical data of worker's locations for later analysis as well as react to real-time events. As a complete solution, employee tracking on top of RILS could, for example, provide:

- Automated alerts on physical stress
- Hours worked vs. productivity analysis
- Machine usage analysis
- Accident reenacting (improving safety)



(a) A corner section of a factory floor plan representing machines labeled from M1 to M5 and a corridor in a right angle



(b) Time spent by a given employee on different machines

Figure 36: An example of a RILS + ET application tracking time spent by an employee on different machines

[Read more about RILS applications in Sections 1.2, 1.3 and 1.4]

A.2 Discussion of the problem to solve

Finding an adequate way of solving RILS is striking a balance between four aspects: convenience, expandability, cost and accuracy. In order to formally define these three aspects:

- **Convenience** refers to how easy it is to deploy the system with current wide-spread technologies.
- **Expandability** is measured by how straight forward it is for developers to build applications on top of the RILS in question.
- **Cost** is simply defined as the amount of money that must be invested in order to get the RILS operative.
- **Accuracy** is related to the precision with which a client can be located in terms of meters. **However**, we are not looking for the best accuracy possible, just for a 'good enough' accuracy (3-4m). All the systems analyzed deliver this type of accuracy.

[Read more in Section 2]

A.2.1 Classification of real-time location systems

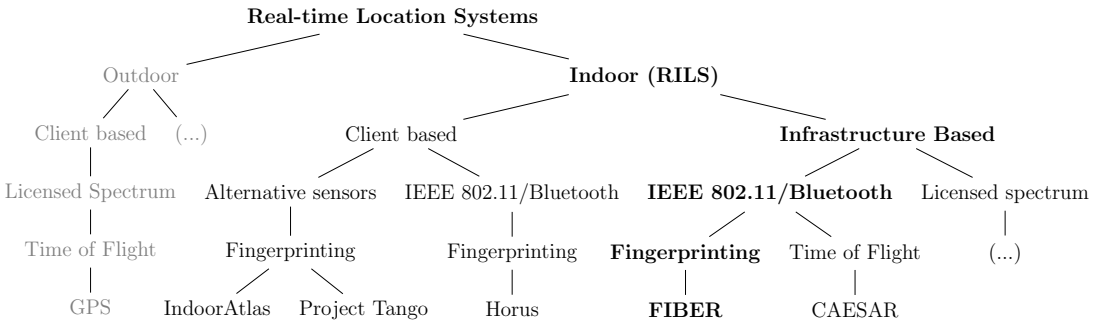


Figure 37: Proposed location systems classification

A complete classification scheme for location systems and RILS is proposed in Figure 37. **Our development, FIBER** (Fingerprinting Infrastructure-BasEd Rils), classifies as *IEEE 802.11 [10]/Bluetooth [13] fingerprint infrastructure based RILS*. Other state of the art RILS (IndoorAtlas [4], Project Tango [3], Horus [2] and CAESAR [1]) have been placed in it for reference.

A.2.2 State of the art

Current implementations of RILS are highly diverse in terms of technology used to achieve indoor location. We will now go over some of them, explaining how they work and why they pose limitations that made FIBER necessary.

Client side with alternative sensors

Systems like Google's Project Tango [3] and IndoorAtlas [4] focus on meter-accuracy of location. To achieve said accuracy, they rely on alternate sensors such as depth-aware cameras (such as [7]) and magnetic sensors [8].

Google's Project Tango [3] works on devices bearing special cameras and sensors providing depth perception, area learning and motion tracking. This information is then processed by the device in order to infer information about its position within a building. A developer API is also available, allowing developers to expand the core functionality of the system and C, JAVA, Android or Unity written applications to run on top of it.

(...)

IndoorAtlas [4] combines traditional IEEE 802.11 [10]/Bluetooth [13] fingerprinting techniques [9] with magnetic-field fingerprinting [8]. Relying on the earth's magnetic field is more convenient than 2.4GHz/5GHz spectrum, since it will not be affected by physical object's movement in the environment (i.e., furniture rearrangement, moving pieces in a factory, etc.).

[Read more in Section 2.2.1]

Client side with IEEE 802.11 [10]/Bluetooth [13]

The Horus [2] location system provides some of the mathematical framework for FIBER as well as some of the techniques for medium calibration and data analysis. FIBER and Horus systems are similar except for a crucial difference: Horus is client-based while FIBER is infrastructure based.

Client based systems require some degree of active user involvement. The user must run a location app on their device, generally helping said user to locate his/herself inside a building. Infrastructure based location systems do not depend on active user involvement and can operate without user-side consent. Moreover, infrastructure based location systems will only require a user to be sending information using IEEE 802.11 [10] or Bluetooth [13] for their device to be located.

[Read more in Section 2.2.2]

Infrastructure-based with Time of Flight techniques

Location systems using Time of Flight (or ToF) attempt to compute the location of a given device based on very accurate measurement of time intervals between transmissions towards and from said device. Famously, GPS [12] uses this technique.

[Read more in Section 2.2.3]

Licensed spectrum solutions

Although some RILS solutions take advantage of higher frequencies with better accuracy than IEEE 802.11 [10] (2.4GHz/5GHz) or Bluetooth [13] (2.4GHz), they require specific hardware on the client-side and need a license for operation within their spectrum range. Since cost effectiveness is one of our goals, we do not consider these systems.

Infrastructure-based Fingerprinting

Our development (FIBER) is an infrastructure based fingerprinting RILS that solves Horus' [2] client-side inconvenience. While not as accurate as other non-fingerprinting systems like CAESAR [1], ease of use, cost effectiveness and expandability of FIBER, along with some robustness against reflections, makes it the better match. Table 32 shows FIBER against other systems according to the criteria we defined at the beginning of this section:

	Convenience	Expandability	Cost	Accuracy
Project Tango [3]	low	high	high	✓
Indoor Atlas [4]	low	medium	high	✓
CAESAR [1]	medium	medium	medium	✓
Horus [2]	medium	high	low	✓
FIBER	high	high	low	✓

Table 32: Comparison table of SOA RILS according to the evaluation criteria.

Requiring only widespread sensor on current smartphones, convenience is a selling point for FIBER. Using free spectrum technologies for fingerprinting [9] such as IEEE 802.11 [10] and Bluetooth [13] help with cost-effectiveness and enables the use off-the-shelf hardware for sniffers (also alleviating regulatory frameworks). Finally, building software in exportable modules and providing a JSON API interface for developers to easily build applications on top of FIBER contributes to the system's expandability. If FIBER can provide area-level accuracy (see results and evaluation in Section 4), it's the best solution for our needs.

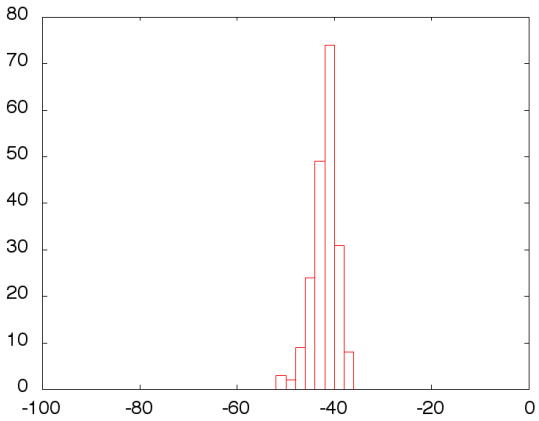
A.3 Technical implementation

NOTE: Please keep in mind that this is a very simplified version of the technical implementation of FIBER. A complete and detailed explanation of the system's architecture, sniffers, back end, infrastructure, hardware and software is made available in Section 3.

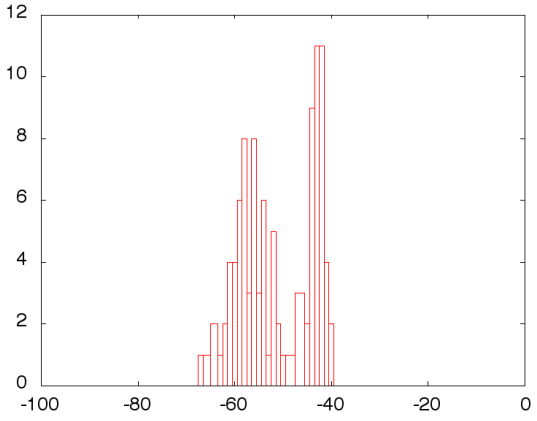
A.3.1 Terminology

When discussing the system's details, the following terms will be used:

- **RSSI** stands for Received Signal Strength Information. It provides information about the received signal strength in dBm for a given transmission. Values are usually negative, and range from -90dBm (bad signal reception) to 0dBm (perfect signal reception).
- **RSSI histogram (or RH)** is the result of compiling **RSSI** values for a certain amount of time.
- **Single-spiked histogram (or SSH)** refers to a **relative histogram** where **RSSI** values are distributed towards a single value (see Figure 38 [a]).
- **Multiple-spiked histograms (or MSH)** are **RSSI histograms** where **RSSI** values are centered around two (or more) values instead of one (see Figure 38 [b]).



(a) An example of a single-spiked histogram (SSH) centered around -41dBm



(b) An example of a multiple-spiked histogram (MSH) centered around -41dBm and -58dBm

Figure 38: Comparison of single-spiked and double-spiked RSSI histograms

- Each **radio map point** (or RMP) is made up of a set of **sniffer reports** taken during the **calibration**. A **radio map point** is associated a to the GPS coordinates [12] corresponding to the physical location where the calibration of said point took place.
- A **client** (or **user**) is a person carrying an IEEE 802.11 [10] capable device that will be tracked by the system and whose location the system tries to infer.
- A **sniffer** is a device bearing at least one NIC (network interface card) used by the **sniffing software** running in it to sniff **RSSI** values from **client** transmissions.
- **Sniffer report** refers to the **RH** provided by a **sniffer**.
- **Offline phase (or calibration)** consists in gathering **sniffer reports** from all available sniffers when a **client** is at a known position. This technique is known as **fingerprinting** [9]. The collected **Sniffer reports** make a **radio map point** that will be associated to the GPS coordinates [12] of the location at which the calibration takes place.
- A **radio map** (or **RM**) is a collection of several **radio map points**.
- **Online phase** consists in gathering **RSSI** values from **clients** and comparing them to the **radio map** recorded during the **offline phase** in order to determine which is the most probable location of said **client**.

A.3.2 Architecture

The FIBER architecture is divided into two parts: a cloud of sensors (sniffers) and a server (back end). Each sniffer continuously captures IEEE 802.11 [10] packet headers sent by mobile devices and decodes their MAC address and RSSI values. This information is then sent to the back end. If the system is instead being used

for calibration, the back end turns this information into radio map points that will become part of a radio map. If the system is being used for localization, the same information is now used to try infer where a given mobile device is located at. The methodology the back end applies for either use is described in the following section. The back end also makes location information about tracked devices available through a JSON [27] API, for which a web service is built (read more in Section 3.4).

A.3.3 Methodology

This sections discuses the procedures followed by our system to achieve client location. The way FIBER computes client locations can be explained in two stages: offline (or calibration) phase and online (or localization) phase.

Offline phase (or calibration)

The calibration is where a known device (called 'calibration device', i.e., a smartphone) is placed in a known physical location. Then, the device generates traffic (typically by pinging the access point) in order to be sniffed by a cloud of sniffers previously deployed at the site. In the back end, a histogram per sniffer (called sniffer report) containing RSSI values reported over calibration time is stored. All sniffer reports associated to that particular physical location are stored as a radio map point (or RMP) as well as the physical location's GPS coordinates (which should be known at the time of the calibration). This point will be one of several points that comprise the radio map. This process is repeated for all physical locations that will be contained in the radio map and constitutes what is known as fingerprinting [9].

The resulting radio map contains a set of radio map point formed by sniffer reports. These sniffer reports, should be comprised of single peaked histograms and in some cases, due to reflections in the medium, multiple-spiked histograms.

There is an issue, however, where co-channel interference in IEEE 802.11 [10] networks can cause the appearance of multiple-spike histograms in absence of reflections. Although not necessarily breaking the methodology, this low-RSSI interference reduces the amount of useful data in the histogram, reducing its 'resolution'. This can negatively impact the statistical inference that will be made in the online phase, causing an overall loss of accuracy.

To avoid co-channel interference, far apart IEEE 802.11 channels are used for communication between the sniffer cloud and the back end (sniffer network) and the calibration device (client network).

Online phase (or localization)

Figure 39: An explanation of how FIBER localization works

Let there be a client at a pre-calibrated site bearing a device that is transmitting through an IEEE 802.11 [10] interface. The sniffer cloud will start reporting on every packet the smartphone sends, including the device's MAC address and RSSI value of the transmission (see Section 3.3.3). The system will record a radio map point corresponding to the location of the user's device in the same way as during calibration but with two exceptions:

1. The physical location of the point is not known beforehand (that's what we are trying to determine)

2. Instead of requiring a number of measurements per sniffer report (as to guarantee sufficient histogram resolution), a parametrizable number of different sniffers must report at least one measurement

Once these two conditions are satisfied, the system has a new radio map point representing the location of the user that can be compared to the points in the radio map. Let the new point corresponding to the user's location be called *user point* (ϕ_u), the set of sniffer reports contained in it S_u and the latitude-longitude pair location values L_u. If there are N sniffer reports in S_u, then:

$$L_u = (lat, lon)$$
$$S_u = \{s_0, ..., s_N\}$$
$$\phi_u = [S_u, L_u]$$

The system must now determine what points in the radio map are more related to the user point. To do so, it determines what radio map points contain all sniffer reports in S_u. While the methodology could still work comparing the user point against all points in the radio map, filtering points that don't contain all sniffer reports in the user point (hence useless at this time) saves time and processing.

Now, we have the user point (ϕ_u) and a subset of candidate points (Φ_c). The candidate point that is most similar to the user point is probably the point the user is nearest to. The system must compare all sniffer reports contained in ϕ_u with those contained in all candidate points in Φ_c.

In order to do the comparison, a naive joint probability approach is used. Let the histogram in each sniffer report be normalized. Then, given an RSSI value, one could simply obtain the probability of that value happening in that particular histogram by looking at its frequency. In the following example, the probability with which the value -45dBm occurs in the histogram is 0.25 (or 25%).

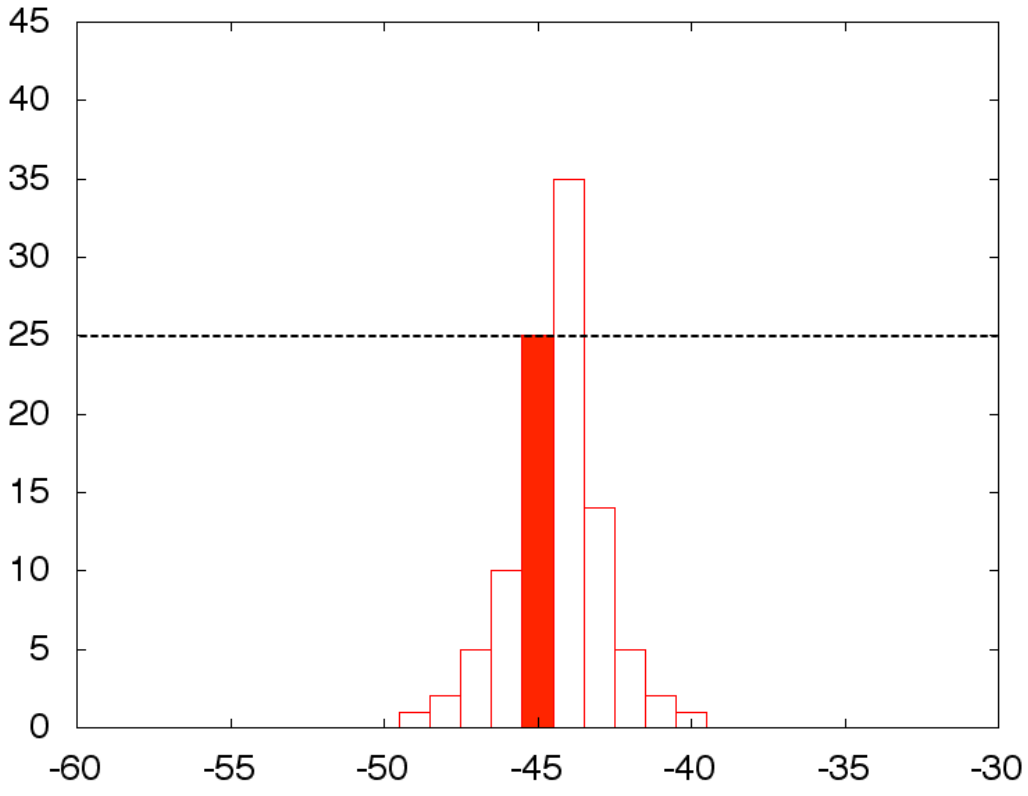


Figure 40: An example of normalized histogram, showing a 0.25 probability of a -45dBm measurement belonging to it

It follows that for the set of sniffer reports S_u in the user point ϕ_u, the probability that they would come from a given candidate point ϕ_c is simply the

joint probability that every sniffer report from S_u 'came from' its homologue S_c. We define this probability as *relation probability* P_{rel}. The location L_c of the candidate point with higher P_{rel} is the most probable user location L_u. So, the relation probability that the user is at candidate point 'c' (P_{rel}) is :

$$P_{rel} = p(s_{c0}|s_{u0}) \cap ... \cap p(s_{cj}|s_{uj}) \ \forall s_{cj} \in S_c$$
or
$$P_{rel} = \prod_{\phi_{cj} \in \phi_c} p(S_{cj} \mid S_{uj})$$

If we call the candidate point with the largest relation probability ϕ_{pref}, we can finally state:

$$\phi_{pref} = \phi_{ck} \text{ where } k = \arg\max_k \left(P_{rel0}, ..., P_{relk}\right)$$

Client location is:
$$L_u = L_{pref}$$

A.4 Results and Evaluation

After an initial calibration and further recalibration of some problematic points, **users are located correctly in 91.67% of the site's surface**. The system accurate enough for our needs. For more information on how this accuracy was measured and what it means, please refer to Section 4.

A.5 Budget and planning

The duration of this project was 300 hours which were distributed throughout 47 days. The cost of the project was 19311.34€, plus 5653.64€ infrastructure costs. For more detail on the project budget or planning, please feel encouraged to read through Section 6.1 and 6.2, respectively.

A.6 Future work

There are some improvements that can be made to the FIBER system in the future. These improvements focus on increasing the system's robustness and speed, while maintaining or improving the system's overall accuracy. For more information on future plans to achieve this, please refer to Section 5.

A.7 Conclusions

The FIBER system achieved all goals set. Improvements must be made but the system is a promising prototype at the moment. For more on conclusions, review Section 7.

References

[1] Domenico Giustiniano, Stefan Mangold, *CAESAR: Carrier Sense-Based Ranging in Off-The-Shelf 802.11 Wireless LAN*

[2] Moustafa Youssef and Ashok Agrawala, *The Horus WLAN Location Determination System*, `www.cs.umd.edu/~moustafa/papers/horus_usenix.pdf`

[3] Google Inc., *Project Tango*, `www.google.com/atap/project-tango/`

[4] IndoorAtlas Ltd., *Ambient magnetic field-based indoor location technology, Bringing the compass to the next level*, `web.indooratlas.com/web/WhitePaper.pdf`, July 2012

[5] Brian Clifton, *Advanced Web Metrics with Google Analytics*

[6] Mark van der Feyst, Eric Wissner, James Petruzzi, *Residential Fire Rescue*

[7] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, Andrew Fitzgibbon, *KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera*

[8] Jaewoo Chung, Matt Donahoe, Chris Schmandt, Ig-Jae Kim, Pedram Razavai, Micaela Wiseman, *Indoor location sensing using geo-magnetism*

[9] Kaemarungsi, K., Krishnamurthy, P. *Modeling of indoor positioning systems based on location fingerprinting* 2004

[10] Brian P. Crow, Indra Widjaja, Jeong Geun Kim, Prescott T. Sakai, *IEEE 802.11 Wireless Local Area Networks* `www.di.unisa.it/~vitsca/RC-0910I/pdf00001.pdf`

[11] Google Analytics, `www.google.com/analytics/`

[12] J. Parthasarathy, *Positioning and Navigation System using GPS*, `www.isprs.org/proceedings/XXXVI/part6/208_XXXVI-part6.pdf`

[13] Chatschik Bisdikian, *IBM Research Report*, 2001

[14] ARM Architecture `infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.research/index.html`

[15] Raspberry Pi 2, Raspberry Pi Foundation, `www.raspberrypi.org/help/what-is-a-raspberry-pi/`

[16] Odroid C1, Hardkernel co., Ltd., `www.hardkernel.com/main/products/prdt_info.php`

[17] Hummingboard i1, SolidRun Ltd., `www.solid-run.com/products/hummingboard/`

[18] Kismet, Kismet Wireless, `www.kismetwireless.net`

[19] Raspbian Linux, Raspberry Pi Foundation & Debian Project, www.raspbian.org

[20] Debian Linux, Debian Project, www.debian.org

[21] Brian Kernighan and Dennis Ritchie, *The C Programming Language*, 1978

[22] Unix Sockets, beej.us/guide/bgipc/output/html/multipage/unixsock.html

[23] The UNIX System, www.unix.org

[24] The Linux Operating System, www.linux.org

[25] Steve Burbeck, *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*, st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[26] Java Servelets, www.oracle.com/technetwork/java/index-jsp-135475.html

[27] JSON, json.org

[28] Apache Tomcat, Apache, tomcat.apache.org

[29] MySQL, www.mysql.com

[30] MIT License, opensource.org/licenses/MIT

[31] BSD 2-clause License opensource.org/licenses/BSD-2-Clause

[32] Android OS, www.android.com

[33] Robert M. Metcalfe and David R. Boggs, *Ethernet: Distributed Packet Switching for Local Computer Networks*, ethernethistory.typepad.com/papers/EthernetPaper.pdf

[34] G. David Forney, Jr., *The Viterbi Algorithm*, citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.412.6372&rep=rep1&type=pdf

[35] OpenSSH, www.openssh.com