Delft University of Technology
Master's Thesis in Embedded Systems

WiFi Indoor Localization Using Channel State Information

Patricia García Ferrín

TU Delft Delft University of Technology

WiFi Indoor Localization Using Channel State Information

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Patricia Garc a Ferr n
p.garciaferrin@student.tudelft.nl

22nd April 2019

Author
Patricia Garc a Ferr n (p.garciaferrin@student.tudelft.nl)
Title
WiFi Indoor Localization Using Channel State Information
MSc presentation
15th May 2019

Graduation Committee

prof. dr. K. G. Langendoen (chair)	Delft University of Technology
dr. P. Pawelczak	Delft University of Technology
dr.ir. G. J.M. Janssen	Delft University of Technology
dr. M. A. Zuniga Zamalloa	Delft University of Technology

Abstract

Indoor positioning has several variants as a result of multiple years of study on the topic. Using WiFi signals as the technology to compute the position of the target device is one of the most extended and researched techniques. WiFi Access Points are extensively deployed in all indoor environments where WiFi indoor localization has a potential application. Several of these indoor localization techniques use the Received Signal Strength (RSS) to compute the location of the device. However, the accuracy of the methods that use this parameter is usually low, in the meter range. Eight years ago, the new advances in wireless communications allowed to retrieve the phase of an incoming signal from a commercial chipset, and not only its strength. With the phase, we can use the physical and mathematical principles of signal transmission to compute the distance to a device. In the long term, it can contribute to having a more robust and long-lasting method to perform indoor localization if we can overcome the challenge of obtaining a delay-free phase measurement. For indoor distances, a small delay in the time estimation can be translated into a signi cant error in the ranging result.

In this work, we re-implement the ranging technique of the paper *Chronos: Decimeter-Level Localization with a Single WiFi Access Point*. This well-known research work explains how to obtain an accurate phase measurement from a WiFi chipset and the technique to compute the distance to the target. We provide a guide of the installation and implementation of the hardware and software required for a system like this, indicating the critical points to allow future researchers a quick set up of the equipment. We revisit the procedure of *Chronos* to obtain a delay-free phase measurement and show how to re-implement it from scratch. Finally, we perform a validation of the system in four di erent steps, from the use of ideal data to progressively introducing the real distortions of indoor wireless data. The evaluation of the performance of the system leads us to present the nding of a non-linear delay that can produce signi cant distance errors.

Preface

Science and technology progress over the years and face new challenging goals. Once the outdoor localization was mastered, a new field of technologies and techniques arose to overcome the challenges of indoor positioning.

After attending the course *Smartphone Sensing*, given by Marco Zuniga, I was interested in the innovations in indoor localization. In this course, we used WiFi signals and motion sensors in combination with machine learning to localize and track people inside a building; and I felt driven to find a way of eliminating the inconvenience of spending hours gathering data for the training phase. This way, the idea of going back to the origins of localization arose: the signal triangulation techniques. Technology just needed to allow the extraction of signal parameters like the phase from commercial chipsets, which was not possible until eight years ago.

This Thesis has given me the chance of working in the theory and application of a state-of-the-art technique. I hope I have contributed my little grain of sand for future researchers in this group that want to keep investigating this technique of multiple potential applications.

Firstly, I would like to thank Marco for his daring and encouragement when I chose this topic; and for showing me the indoor localization world. I would also like to thank Przemek for his enthusiasm, eagerness to learn new things and support in the challenging moments. Secondly, thanks to the VLC group for the progress meetings and their valuable suggestions. I would like to extend thanks to all people that made me feel at home these two years, friends and colleges that I have met in The Netherlands. To all my family and friends in Spain, thank you for you encouragement in the rough times. Finally, I would like to thank my parents and my sister. You are my pillars and who brought me this fantastic opportunity. Thank you for your infinite patience and your unconditional support; this is mine as much as yours.

Patricia Garcia Ferrin

Delft, The Netherlands
22nd April 2019

Contents

Chapter 1

Introduction

1.1 Motivation

Localization techniques have been continuously evolving for years. Its outdoors accuracy has signi cantly increased, and it is so widespread that the GPS has almost banished the use of maps. Millions of users have navigators that can guide them from door to door on a road trip or real-time smart-phone applications that can direct them while doing sightseeing in a city. However, people spend most of their time inside buildings: at home, at work, in the supermarket; places where the GPS signal can't reach. Applications that allow, for instance, to guide a blind person inside a building to the required location, to locate at any time the assets of a hospital or the monitoring of senior people at home to extend their autonomy; are now a closer reality.

The possibility of doing indoor localization has been under study since the 2000s for di erent radiofrequency technologies: RFID, WiFi, IR, and even ultrasound. Among all of them, we will put our focus on WiFi. The main reason is that the infrastructure is already present in the buildings where indoor localization has a potential application. However, it isn't a system designed for it; so its purpose is not giving an accurate signal measurement but reliable communications to the users connected to it.

For this reason, the most popular techniques until 2013 use metrics that do not provide exact physical information of the signal like the phase or the amplitude do. These are based on Machine Learning and use the Received Signal Strength Indicator (RSSI) parameter received from the closest WiFi Access Points (AP) as training data. The RSSI is just an indicator of the signal quality that accounts for the received power level after the possible signal loss. Therefore, it is not a parameter that we can use to extract accurate information about the signal. Machine Learning can deal with the uncertainty of the RSSI with long training processes and big datasets of this indicator in di erent locations. However, this procedure has to be repeated

every time the environment changed signi cantly.

Since six years ago, with the possibility of measuring the phase and amplitude of the incoming signal (also called Channel State Information), a new opportunity appeared to perform robust indoor WiFi localization: compute the position by triangulation using the phase of the received signal. This approach uses an accurate metric like the phase and mathematics to locate the device. Its main positive aspects are:

- We do not need long processes of data gathering anymore. The system can localize the device just knowing where in the building the APs are installed.
- The clock synchronization between the transmitter and the receiver of the signal is not necessary. One of the problems of transferring the GPS triangulation technique to WiFi was that GPS always requires to be in time synchronization with the receiver.
- The accuracy has improved so much that, in the last two years, some approaches have been able to narrow it down to the decimeter level.

On the other hand, we need to deal with the following weak points:

- The number of APs and the antennas receiving the signal is a signi- cant factor to obtain accurate results. The antennas will in uence the one-dimensional localization (also called ranging) results, while the number of APs can be essential to triangulate the position.
- It is necessary to have accurate phase information of the signal to compute the distance between devices. The data cannot be a ected by any delays because in such short distances the smallest delay can lead to a signi cant distance error. Removing the possible external delays is a challenging process as we will see in the course of this Thesis.

1.2 Problem Statement

In spite of all the advantages of this new method, only around ten research groups have been working on this technique. The pioneers built devices that needed big arrays of antennas to localize other devices. In the last four years, three mainstream papers have provided solutions that can apply this method to systems resembling actual o -the-shelf hardware.

A big part of this Thesis is centered on researching why such a promising technique like this one is not widespread, not even in systems for commercial use. In general terms, the complexity of the method and the fact that there are only two hardware options available to report the phase are the main bottlenecks of this approach.

We will choose one of the latest publications on this topic and build-up from scratch the method that they propose. This way, we aim to point out the problematic aspects of the process. We will also be able to compare the results we obtain, analyze the method and indicate possible improvement options for future work. We choose the system in *Chronos: Decimeter-Level Localization with a Single WiFi Access Point*[36] from the CSAIL research group at MIT as our objective. In Chapter 2 we will show the reasons for selecting this paper among the existing prior work on the topic.

Summarizing, we can formulate the research objective of this work as follows:

Analyze indoor localization techniques that rely on CSI metrics using the method of the paper *Chronos: Decimeter-Level Localization with a Single WiFi Access Point* as the basis, and identify the reasons why such a promising technique is not widespread.

1.3 Contributions

The key contributions that we provide in this work are:

1. A guide that explains how to perform WiFi indoor localization using the phase and, in particular, the method used in *Chronos*.
2. An analysis of the build-up of this system, where we can point out two possible bottlenecks:

   The installation and setup of the hardware and software to collect the phase information.

   The implementation of the code to process the data and compute the localization.
3. A thorough evaluation of the method that leads us to observe the presence of a non-linear delay that does not appear in *Chronos* and causes a mismatch between the obtained results and the expected output.

The objective of this analysis is to get the future researcher one step closer to the state-of-the-art in this subject. Give her a starting point in the topic, helping to solve future problems and avoid making similar mistakes.

1.4 Structure

In Chapter 2, we will compare the prior work on the topic with *Chronos*, and show the reasons why we chose the re-implementation of this method as the basis of our work. Chapter 3 will explain all the general theoretical concepts

of this approach, and how the paper *Chronos: Decimeter-Level Localization with a Single WiFi Access Point* [36] gets to localize, using only one Access Point. Chapter 4 will show the steps we took to recreate the method of *Chronos*. A big part of this chapter will be about the problems encountered during the process, and the solutions and alternatives found for them. In Chapter 5, we will evaluate the system and method performance, and we will analyze the results. Finally, in Chapter 6 we will present the conclusions and some possible ideas for future work.

Chapter 2

Prior Work

In this Chapter, we will cite some of the most relevant papers about WiFi indoor localization using CSI of the past few years. We will justify the choice of the work *Chronos: Decimeter-Level Localization with a Single WiFi Access Point* as the system to reimplement with a comparative chart of di erent approaches.

WiFi indoor localization is a research problem that has been under study for around two decades. There are some approaches that we can only apply if we know the CSI. Some others, do not rely on the precise measurement of the signal to obtain the position of the device that sent it. For this reason, there are multiple proposals that combine both techniques, especially in the past four years when the idea of using CSI for localizing started to grow. We can summarize the most widespread techniques in the following classification:

Techniques that do not require CSI

a. *Sensor Fusion:* Consists of using the data of several sensors like accelerometers, gyroscopes or barometers to track the movement of the device. The method known as dead reckoning can compute the position taking as a starting point a known location and measuring the direction and distance traveled from there. [12, 8]

b. *Machine Learning:* There are multiple techniques related to the use of Machine Learning, from the K-Nearest Neighbours (KNN), where it decides the location of a data point based on its closest neighbors'; to the Deep Neural Networks, where we train a complex network to learn one or more physical parameters of the chosen locations. Fingerprinting is another popular technique based on building radiofrequency maps from snapshots taken at training time. In the online phase, the comparison between the live picture and the previously generated maps outputs the solution. In multiple occasions, the Received

Signal Strength Indicator (RSSI) is the parameter used to build the radio map database, which is just a power level indicator and thus not completely reliable as a physical parameter of the signal. More recent techniques combine other signals like the magnetic field or directly use the CSI to build the fingerprints. [2, 1, 31, 5]

c. *Time Difference of Arrival (TDoA):* The TDoA uses the time difference of arrival at two different receivers of a beacon sent by the target. This method requires, either the synchronization between APs or between the APs and the device so that we can interpret the timestamp of the transmission correctly. *Trilateration* technique computes the final position using the distance, that in this case, we can get from the TDoA. Mathematics plays a significant role in this method to determine the final location of the target. Theorems like the triangle inequality can filter out distance results that are physically impossible, and we can use them in combination with other techniques apart from TDoA. [50, 49]

Techniques that require CSI

a. *Time of Flight (ToF):* The ToF is the time taken by the signal to travel between endpoints. Knowing the speed of transmission of the medium and the time taken, we can compute the distance. As we will see in the next Chapter, it is possible to use the phase of the incoming signal to calculate the ToF.

b. *Angle of Arrival (AoA):* The AoA indicates the direction from where the signal was sent, from the receiver's point of view. This parameter, when combined with the distance at which the emission happened, can be used to compute the bi-dimensional location of the transmitter. This technique is called *triangulation*.

Looking at Figure 2.1 we can see an incoming signal arriving at two different antennas separated at a distance d. The signal travels an extra path to get to the antenna further away, so there is a difference φ between the phase measured at both antennas. This phase difference, in turn, can be expressed in terms of the Angle of Arrival (θ) and the distance between antennas (d) as follows [20]:

$$\varphi = \frac{2\pi f}{c} d \sin\theta \mod 2\pi \tag{2.1}$$

There is always one reference antenna whose φ is zero.

Chronos doesn't use the AoA to compute the bi-dimensional position of the device. In Table 2.1 we can see how several papers make use of it.

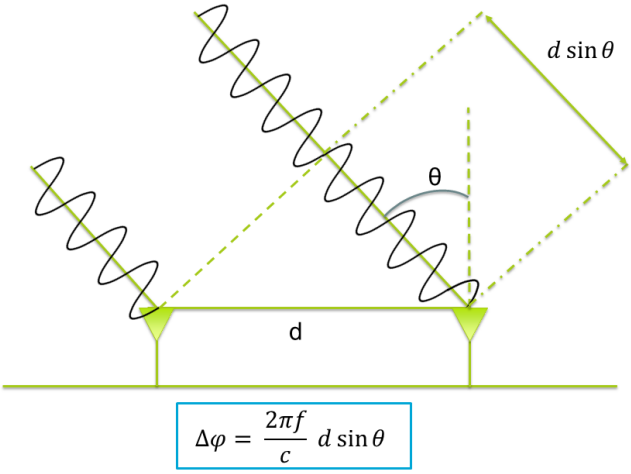


Figure 2.1: Graphical representation of the Angle of Arrival.

In Table 2.1, we summarize the most relevant features of 10 mainstream papers about WiFi indoor localization with CSI.

In this Thesis we chose to reimplement the system explained in *Chronos* [36]. It shows good accuracy, with a median error for both Line of Sight and Non-Line of Sight measurements below 1m. It does not require a training phase like the deep learning approaches. It is one of the few methods that needs only one AP to be able to localize, and it does not require any extra infrastructure. Finally, the system does not need a network synchronization to compute the ToF, which reduces the dependency of the network with the target and vice-versa to obtain results. For all these reasons, it is a suitable choice to analyze the feasibility of CSI for indoor localization based on WiFi.

Table 2.1: Prior Work on WiFi Indoor Localization Using CSI. The techniques marked with ( ) require a training phase.

Name	Year	No. APs	Extra Infrastructure	Median Error	Technique
[47] ArrayTrack	2013	6	Yes Antenna array	0.31m	CSI + AoA with multiple antennas (8 ant.)
[30] CUPID	2013	5	No	2m	CSI + AoA + triangulation
[40] FILA	2013	1 - 6	No	LoS: $<$0.5m NLoS: $<$1.2m	CSI + KNN( ) + trilateration
[22] SAIL	2014	1	No	2.5m	CSI + ToF + dead reckoning
[20] SpotFi	2015	5	No	0.6m	CSI + AoA + triangulation
[48] ToneTrack	2015	4	Yes Rice WARP platform	0.9m	CSI + ToF + triangle inequality
[36] Chronos	2016	1	No	LoS: 0.65m NLoS: 0.98m	CSI + ToF + geometrical constrained quadratic optimization
[38] DeepFi	2017	1	No	LoS: 0.8m NLoS: 1.5m	CSI ngerprinting with deep learning( )
[7] ConFi	2017	1	No	1.4m	CSI + Convolutional Neural Netw.( )
[37] CiFi	2017	1	No	LoS: 2.2m NLoS: 1.8m	CSI + AoA as input for Deep Convolutional Neural Netw.( )

Chapter 3

Theoretical Foundation

In this chapter, we will show how to compute the location of a device with the phase of a WiFi signal and the system that *Chronos* implements to do it. In Section 3.1, we will explain the physical foundations of how the phase allows computing the distance at which the transmitter sent the signal. In Section 3.2, we will introduce the Channel State Information and its importance to know how the communication channel a ects the phase. Finally, in Section 3.3 we will describe in detail the process *Chronos* that follows to compute the Time of Flight with high accuracy.

3.1 Using the Phase for Localization

The *phase* and the *amplitude* are the main parameters that de ne a signal. The amplitude provides information about the attenuation that a signal experiences during its course over the transmission medium. From the phase, we can extract the time that has passed for the whole journey. Either for a direct path transmission between the transmitter and the receiver or after any re ections.

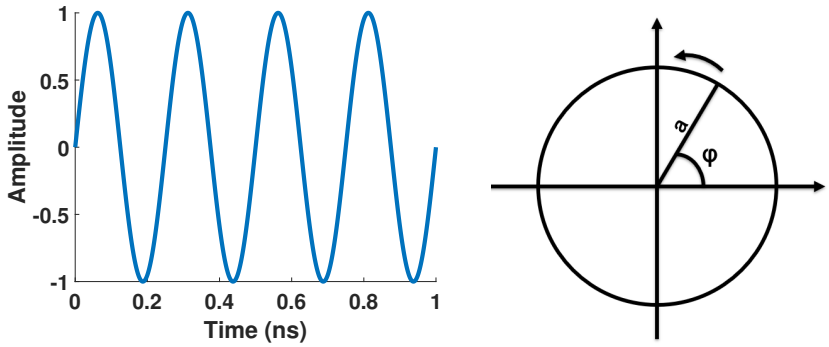
There are two common ways of representing a signal, as a sinusoidal function, and as a complex number in the polar representation, like in Figure 3.1. While the time-dependent representation is probably easier to picture, the angular representation gives us a better chance of observing the progression of the phase.

The mathematical expression of the sinusoidal function is the following:

$$y = A\sin(2\pi ft + \quad) \tag{3.1}$$

being A the amplitude, f the frequency, t the time and the initial phase o set.

We can express a signal in the polar representation and its translation to



(a) Sinusoidal Representation (b) Polar Representation

Figure 3.1: Two ways of representing a signal.

the complex number expression as:

$$h = ae^{\,j2\ ft} \tag{3.2}$$

$$z = a\,(\cos(2\pi ft) + j\sin(2\pi ft)) \tag{3.3}$$

where h is the channel response comprising the phase and amplitude of the signal and a is the amplitude. Likewise, the phase can also be expressed as:

$$\varphi = \backslash h = \quad 2\pi ft \quad \text{mod } 2\pi \tag{3.4}$$

The Time of Flight or ToF is the time taken by the signal to travel from the transmitter to the receiver, that we will call τ. In real setups, this parameter gets distorted by the hardware imperfections in the signal processing process. Therefore, the objective is to obtain the \clean" ToF that corresponds to the true time taken for the journey, without the hardware delays. We will talk about them in Section 3.3.

Another issue to face is how to compute the ToF deterministically if after every period the phase is \reset" to 0 and a new turn starts. Looking back at the phase expression in Equation (3.4) the mod 2π shows that, it is impossible to tell how many periods of the signal have passed before arriving at the receiver. We can see in the picture in Figure 3.2 that, we obtain in nite possibilities for the ToF (τ) with the same phase measurement (φ). Which in turn, translates to in nite distance solutions. In Section 3.3 we will show the technique that *Chronos* uses to unequivocally obtain this parameter.

3.2 Channel State Information in WiFi chipsets

Now that we know the possibility that the phase o ers to measure the distance between two devices; we need to see how to obtain it from an ordinary

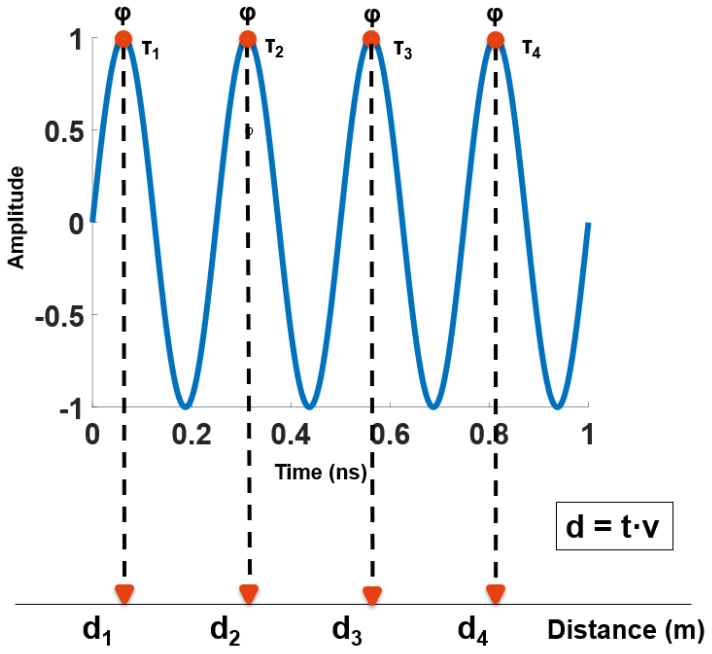


Figure 3.2: Phase uncertainty for distance calculation.

commercial WiFi chipset and the format in which this data is reported to be able to interpret it.

To begin with, the Channel State Information (CSI) is the matrix that will give us the phase information and the amplitude of the incoming signal at each receiving antenna.

In WiFi transmissions, the transmitters send the data spread in di erent frequencies, not only the carrier frequency or center frequency. This data is multiplexed using Frequency Division Multiplexing so that the amount of information that can be sent simultaneously increases and the communication can happen at a higher rate. Each of those frequencies is called a *subcarrier* of the signal. The modulation scheme indicates the selection of the subcarriers where to send the data. Speci cally, in WiFi, the subcarriers are chosen orthogonal to one another, following the Orthogonal Frequency-Division Multiplexing (OFDM) method, to prevent the interference between carriers [28].

The key concept here is not the technique used to transmit the signal, but the fact that, for a single emission there is data sent in multiple frequencies. We can take pro t of that, and obtain more information measuring the phase in all of them.

Consequently, the CSI matrix should comprehend the data collected by each antenna in the receiver side, sent by each transmitter's antenna in all the signal subcarrier frequencies. This matrix is always formed by complex

numbers that represent the amplitude and phase of the incoming signals as in Equation (3.3), but the arrangement of the dimensions usually depends on the software that reports it. Figure 3.3 shows an illustration of the process. The channel response $h_{i;k}$ reports the phase and amplitude of the signal that the RX antenna k received referenced to the pilot tone that the TX antenna i sent. Therefore, in the matrix in the picture, each row is for a transmitting antenna, and each column represents the receiving antenna. Finally, we will have a matrix like this one per subcarrier of the transmitted signal.

However, not every WiFi chipset has the capability of computing the phase. On top of that, only a few of those can report it to the user. The hardware calculates the CSI at the physical layer, and only specialized software designed for those particular chipsets can pass it up to the userspace.

There are two essential requirements that a chipset must ful ll to be valid to compute and report these values to the user:

1. It must have the possibility of operation with the IEEE802.11n WiFi protocol.

2. There must be a dedicated software for that device, that can access the physical layer and extract the phase information up to the userspace.

Operation in IEEE802.11n WiFi protocol

The IEEE 802.11n WiFi standard has an operating mode that can measure the transmission medium. This mode was added to the protocol to enhance the range and quality of the communications. If the characteristics of the channel at the current time are known, the transmission of each transmitter antenna can be intentionally desynchronized to avoid creating, to a certain extent, destructive interference at the receiver. The purpose of this channel sounding process is to perform what is called beamforming. Beamforming, as the name indicates, aims to form a directional beam between the receiver and transmitter. This way, the chance of avoiding the obstacles that block the direct transmission between them increases.

The procedure is the following: the transmitter sends a series of reference pilot tones known to the receiver. Each of the receiving antennas can compute the attenuation and phase rotation the signal su ered during the over-the-air journey, based on the di erence between the expected and the actual incoming signal. This computation is what we call the CSI matrix. From here, the system builds the so-called steering matrix to change the phase of the outgoing signal and reduce the destructive interference at the receiver.

CSI extraction tool

Furthermore, we encounter the barrier of the data accessibility. In general terms, it is never accessible to the user. The Channel State Information

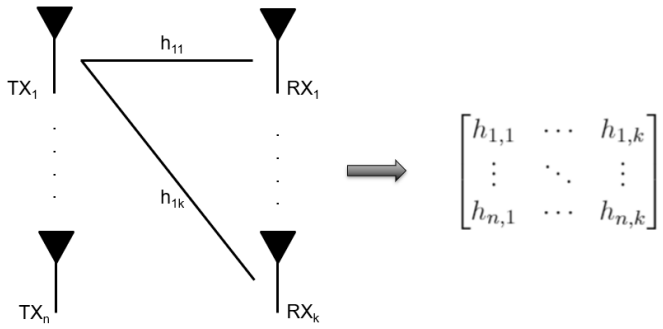


Figure 3.3: Channel State Information matrix generation.

is used by the physical layer to adjust the transmission at the lowest level; however, the higher layers do not need this information at all. For this reason, it remains hidden to the user with no inner knowledge of how the hardware is programmed to compute it. This problem creates the need of finding a \software bridge" that can pass the information from the lowest layer, up to any layer in which the user has access granted.

Currently, this custom made software exists for two different commercial hardware options. In both cases, it is a solution given by developers that know the firmware and hardware architecture of the chipset and not by the chipset manufacturers.

The first one was developed by Daniel Halperin et al. [18] in 2011 for the Intel Wireless Link 5300 (IWL5300) chipsets. As this was the first software to appear, most of the oldest papers about WiFi indoor localization with CSI use it for their implementations. The second option was developed in 2015 by Yaxiong Xie et al. [46] for Atheros chipsets. In theory, it works for NIC cards in laptops or chipsets installed in APs. Although this feature provides a versatility that some recent papers have leveraged; in our case the lack of testing of the tool in the new Qualcomm Atheros chipsets prevented us from using them in this work. For this reason, we use the IWL5300 and Halperin's Linux tool.

Besides these two options, even though not explored in this project, there is a possibility of using research equipment like Software Defined Radio to measure the phase, simulating how a realistic WiFi system works. However, it is more convenient to work with off-the-shelf WiFi chipsets. It will give us a real approach to the delays that appear when we work with commercial hardware, oriented to functionality rather than precision.

3.3 Chronos Approach

In this section, we will explain the technique of the paper *Chronos: Decimeter-Level Localization with a Single WiFi Access Point*. The main achievement

of *Chronos* is to compute the Time of Flight down to nanosecond accuracy. Once the accurate ToF is known, the distance between devices is obtained multiplying it by the speed of light. However, the possibility of finding a precise solution for the ToF depends on how the following three important issues are solved:

1. **An isolated phase measurement is not enough information to compute the ToF unequivocally.** Among all the periodic solutions that a measured phase can show, *Chronos* can select the one corresponding to the real distance. The phase measured from only one signal cannot output a deterministic ToF. However, the combination of the solutions from multiple transmissions at different frequencies can pinpoint the actual ToF, and therefore the distance between sender and receiver. In Figure 3.4a we illustrate this first step of the functioning of *Chronos*. The transmitter sends a beacon to the receiver, located at a distance d, at three different frequencies. The receiver measures a phase that can correspond to any of the positions marked with the vertical dashed black lines, repeated period after period. If we only use the frequency of the top, we could obtain four different locations marked with the red dot including the true d. When we add the second signal in the middle, we reduce from four to two possible outputs that coincide in both blue and green frequencies. Finally, we obtain the deterministic solution when only one solution for the yellow frequency coincides with the previous two.

   Chronos obtains the deterministic ToF by measuring the signal in different WiFi channels and matching the coinciding solution in all of them.

2. **The hardware introduces delays to the phase measurement that distort the distance results.** These delays show a mismatch between the real phase of the signal at the distance where it was measured and the value reported by the hardware. *Choronos* identifies three of them: Packet Detection Delay, Carrier Frequency Offset and PLL Offset. They distort the ideal or \delay-free" data that we expect to obtain, like the one in Figure 3.4a. We could see up to tenths of meters error in the final result if we do not remove these delays. In Section 3.3.2 we will explain the methods that *Chronos* presents to get rid of them, and in Chapter 5 we will see their effect on the data and the results.

   Chronos detects three hardware delays that distort the data we measure. If left uncompensated, they can cause errors of tenths of meters in the final location estimation of the device.

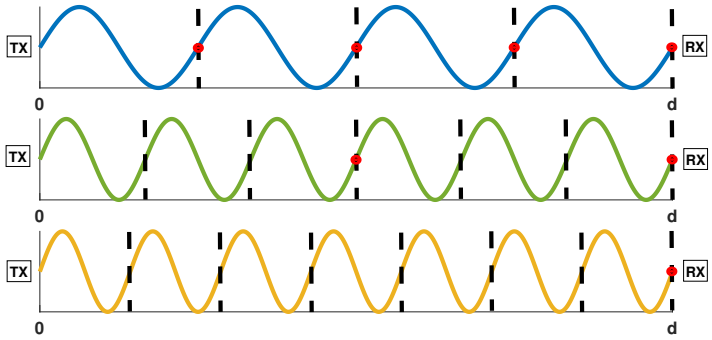
3. **It is necessary to know how to distinguish the direct path among all the reflections that arrive at the receiver to compute the real ToF.** The data in Figure 3.4a is ideal, not only because it is \delay-free", but also because it represents a unique and direct path transmission between the sender and receiver. In practice, the incoming signal is formed by the di erent re ections of the same signal transmission in the obstacles of the environment. Only with the phase of the direct path, we can compute the real ToF. As we will see in Section 3.3.3, *Chronos* uses the Inverse Non-Uniform Discrete Fourier Transform (INDFT) to disentangle the di erent paths. Figure 3.4b shows the output of this algorithm, where each peak represents one path. The X-axis shows the ToF of those trajectories and the Y-axis their magnitude. We can deduce that following the direct path the signal gets faster to the receiver, even though its intensity might not be the highest. The direct path transmission will still be the fastest even if it is attenuated after passing through an obstacle in its way. Therefore, in a pro le like the one in Figure 3.4b, the rst peak indicates a ToF for the direct path of 1 ns, and two re ections arriving at 3 ns and 7 ns.

Chronos uses the INDFT to disentangle the di erent paths of the transmission. The rst peak of the output pro le indicates the direct path and therefore is the one to use for computing the ToF.

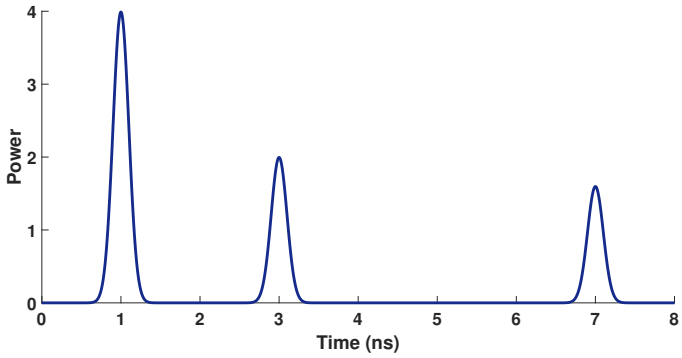
3.3.1 Deterministic ToF Computation

First of all, we know that with a single phase measurement, only sampling the signal at one frequency, it is mathematically impossible to get a unique solution for the ToF. Furthermore, the timer resolution is proportional to $\frac{1}{B}$, and consequently, the smallest distance that we can tell apart is proportional to $\frac{c}{B}$; where B is the bandwidth and c the speed of the light. Therefore, with the bandwidth of one channel, of 20 MHz and 40 MHz for the 2.4 GHz and 5 GHz band respectively; in the worst-case scenario we can get up to 15 m and 7.5 m error. For indoor setups, this spatial resolution is too broad to be useful. What *Chronos* does to x this limitation, is to increase the bandwidth by measuring the signal in the whole band of available WiFi channels.

When we measure the signal at di erent frequencies, the amount of data available to compute the ToF increases. In Equation (3.2) we can see the link between the phase and the frequency. For the same ToF (τ), transmissions at di erent frequencies ($f_1, ..., f_n$) result in di erent phase measurements ($\backslash h_1, ..., \backslash h_n$). *Chronos* combines the data from all the channels and selects, among all the possible solutions the one $\backslash h$ that coincides in all of them.



(a) Deterministic ToF computation



(b) Direct path identi cation

Figure 3.4: **Conceptual scheme of Chronos to obtain the ToF.** (a) The combination of multiple frequency signals allows us to obtain a single solution for the ToF. (b) After the removing the delays introduced by the hardware, the INDFT algorithm presented in *Chronos* allows to identify the direct path and its ToF. In the picture, the output of the INDFT for a transmission with three signi cant paths.

Combining the bands of 2.4 GHz and 5 GHz, we end up with data from a total of 37 channels, which adds up to 560 MHz and reduces the worst-case error to 1.79 ns and 53.55 cm. There are 13 channels in the 2.4 GHz band, whose center frequency spacing is uniform at a 20 MHz distance and 24 in the 5 GHz band, but in this case, organized in non-contiguous frequency chunks [29]. There are groups of channels with central frequencies 20 MHz apart, and then empty spaces in the frequency spectrum separating these groups. This arrangement of the channels is bene cial for converging to a deterministic result with fewer data. If the chosen frequency distribution is uneven, the probability of having just one $\backslash h$ that coincides for all the channels increases because we do not favor the periodic nature of the phase. If the spacing between frequencies is equal, we need a larger frequency sampling

to obtain the same deterministic result.

Chronos implements a Request-Acknowledgement protocol to do the synchronous channel swap for the data gathering process. It is impossible to measure all the channels simultaneously, so we already know that this method will introduce an unavoidable error. This error would be a factor to take into account for the channel coherence time, that is the time for which the channel conditions are assumed to remain unchanged. However, in our case, this factor is not as relevant, because we want to test the concept for static objects and not for tracking moving objects. So, we can assume that we are meeting the requirement of the coherence time.

3.3.2 Hardware Delays

The principal goal of WiFi chipsets is to perform a reliable communication between devices, and therefore, the precision in time measurement is not an indispensable requirement. The packet reception chain introduces a series of delays in each stage of the signal processing process before the Channel State Information recording step. The CSI recording ends the signal processing and leads to the packet processing stage.

The delay compensation is crucial for the accurate ToF computation. *Chronos* is the first system that gets to compute the ToF down to nanosecond accuracy. The system can achieve this precision because apart from augmenting the resolution with the bandwidth increase, it can identify and remove effectively the delays introduced at the signal processing stage.

In this section, we will explain the delays identified in *Chronos*: the Packet Detection Delay (PDD), Carrier Frequency Offset (CFO) and Phase Locked Loop (PLL) Offset, and their effects on the time measurement and the way of counteracting them.

Packet Detection Delay

The Packet Detection Delay, as the name itself indicates, is the delay introduced while detecting the presence of a packet. The uncertainty in reaching the energy threshold indicative of an incoming packet introduces a delay. This delay will translate into an extra phase rotation. Its quantity is somewhat arbitrary, as it depends on variable factors like the TX power or the transmission medium to reach that threshold.

Chronos identifies the introduction of this delay in the *downconversion* process of the signal processing stage. The received signal is sampled, and the subcarriers are downconverted to the carrier frequency before reaching the Packet Detector block, that will acknowledge the presence of a packet if these samples exceed the energy threshold. To illustrate better this process, we can take a look at the example in Figure 3.5. The figure in the middle represents the center frequency, while the one below and the one above

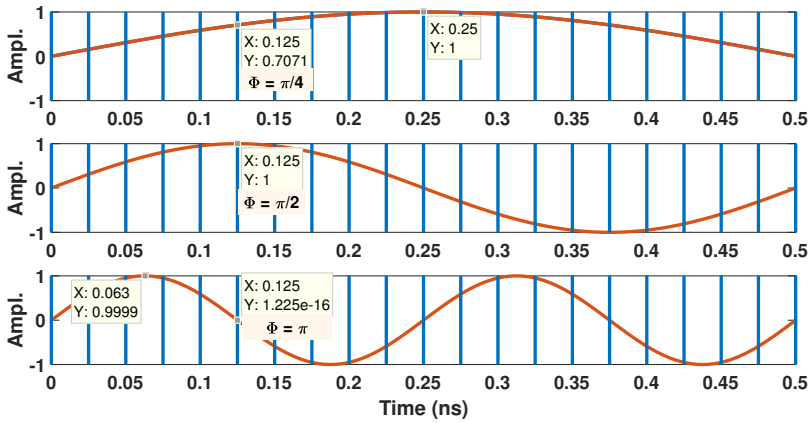


Figure 3.5: **Downsampling example for two subcarriers and the carrier with sampling frequency of 20 times the central frequency.** These frequencies do not correspond to any of those used in OFDM. On top, the subcarrier representation with frequency below the carrier frequency. In the middle, the carrier or central frequency representation. At the bottom, the subcarrier representation with frequency above the carrier frequency.

represent a faster and a slower subcarrier respectively. The downconversion consists on moving these frequencies from passband to baseband. In other words, to transport them to the vicinity of the carrier frequency. To do this, the sampler downsamples all the subcarriers using the center frequency sampling rate. The vertical lines the Figure 3.5 represent this sampling [1].

We can see that with this rate one signal is oversampled and the other one is undersampled. We get extra information from the slowest signal while we lose some samples from the fastest ones. The center frequency, on the contrary, su ers no variations. For instance, let's assume that the rst maximum of each signal indicates the energy threshold and the time passed to reach that point for the central frequency (0.125 ns) is the Packet Detection Delay.

The receiver chain can compensate the delay for the carrier frequency (subcarrier 0); this is, it will apply $2\pi f_{i;0}\delta_i$ to all subcarriers, where δ_i is the PDD. In Figure 3.5 this corresponds to $\frac{}{2}$ rad. With this compensation, the zero subcarrier will always measure the real channel. However, if we deduct $\frac{}{2}$ rad from the faster subcarrier, we will not be able to remove the delay entirely. On the contrary, for the slower subcarrier, we will overcompensate with $\frac{}{2}$ rad, and it will also not indicate the true phase of the incoming signal. Knowing that the extra phase due to the PDD for every subcarrier is $2\pi f_{i;k}\delta_i$, in Equation (3.5) we see the mathematical expression for the

[1] The rate doesn't adjust to real OFDM so that the e ect of the delay could be more visible in the example

resulting added phase ($_{i;k}$) because of the PDD.

$$_{i;k} = \quad 2\pi(f_{i;k} \quad f_{i;0})\delta_i \tag{3.5}$$

The solution that *Chronos* introduces is to use the data that is not a ected by the PDD to make all the computations. It will not get rid of the delay; instead, it will use only the zero subcarrier from all the available subcarriers of each channel. With the data of just the carrier frequency, it is still possible to apply the method depicted in Figure 3.4a, where the combination of all possible solutions outputs the real ToF. However, isolating the zero subcarrier it is not that straightforward, because in WiFi there is no transmission in this frequency. The solution adopted is to interpolate the zero subcarrier using the rest of the subcarrier's data. We will see in Section 4.2.2 how to do it.

Chronos uses only the data of subcarrier zero or carrier frequency ($\backslash h_{i;0}$) of each channel i for all the computations, because it is not a ected by the Packet Detection Delay.

PLL Phase Offset

The PLL (Phase Locked Loop) is the closed-loop system in charge of generating the signal for the carrier frequency. It compares the phase of the reference signal generated by an oscillator, with the output carrier phase that the system is producing and adjusts the Voltage Controlled Oscillator (VCO) accordingly. The VCO is the nal component in charge of generating the carrier [3].

In a protocol that continuously changes the channel of operation, it is impossible to control the state in which the PLL starts. This hardware will at some point synchronize the phase of the reference and output signal, but it will not necessarily happen in the signal's zero crossing corresponding to 0 rad. On the other end, the receiver has to generate its carrier frequency to demodulate the signal. The PLL O set appears due to the imbalance of the carrier's initial phase between the transmitter and the receiver.

We can take a look at Figure 3.6 to see the e ect of this o set. The transmitter's in orange and receiver's signal in blue have a phase mismatch of $\frac{}{2}$, one has an initial phase $^{tx}_{0} = 0$ and the other $^{rx}_{0} = \frac{}{2}$. The green vertical line represents the real ToF. When the orange TX signal arrives at the receiver, this one interprets the phase with the carrier that it has generated; this is, the blue signal. Therefore, we can see a mismatch of $_{rx} = \frac{}{2}$. Similarly, the transmitter would su er an o set of $_{tx} = \frac{}{2}$ if it had to interpret the incoming phase over the orange signal. Equations 3.6 and 3.7 show the mathematical expression of this e ect.

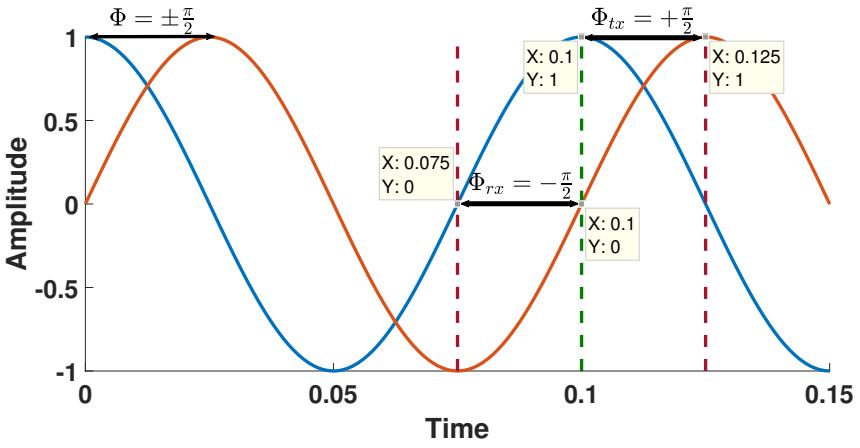


Figure 3.6: PLL O set illustration. TX signal in orange and RX signal in blue with a mismatch of $\frac{}{2}$ is translated to an o set of $\frac{}{2}$ when measuring the phase at each end. The green line marks the real ToF, while the red shows the output a ected by the delay.

$$ {}^{rx}_{i;0}(t) = e^{j(\ {}^{tx}_{i;0} \quad {}^{rx}_{i;0})} \tag{3.6} $$

$$ {}^{tx}_{i;0}(t) = e^{j(\ {}^{rx}_{i;0} \quad {}^{tx}_{i;0})} \tag{3.7} $$

This delay will produce a shift in the data. All the subcarriers of the signal will experience the same mismatch, but there will be variations between channels, as the generation of each carrier frequency will produce its initial phase o set.

The nature and e ect of the Carrier Frequency O set is very similar to this one. We will use the same procedure to eliminate both; explained in the following section.

Carrier Frequency Offset

The Carrier Frequency O set (CFO) is caused by the mismatch between the expected carrier frequency and the real one used to codify and send the signal. This di erence entails an imbalance that adds an extra rotation to the phase. If left uncompensated, it can also create interference between carriers and complicate the decoding and recovery of information [6]. Even though the margins in the fabrication of components are tight, there is always a chance of having a little deviation in the frequency generation.

In Figure 3.7 we can see an illustration of the CFO [2]. Again, in orange we have the carrier frequency signal for the transmitter and in blue for the

[2] For both CFO and PLL O set the phase and frequency mismatch has been exagerated to appreciate the e ect.

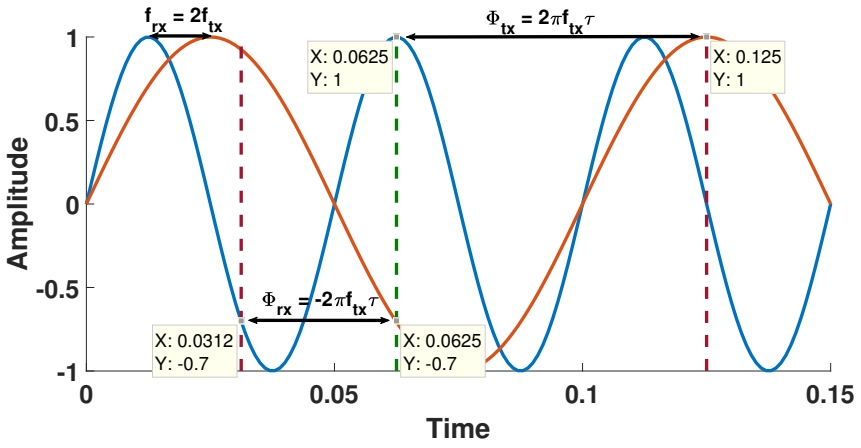


Figure 3.7: CFO illustration. TX signal in orange and RX signal in blue with a frequency mismatch of $f_{rx} = 2f_{tx}$ is translated to an o set of $= 2\pi f_{tx}t$ when measuring the phase at each end. The green line marks the real ToF, while the red shows the output a ected by the delay.

receiver. Now, the RX frequency is two times the TX frequency $f_{rx} = 2f_{tx}$. We assume that the initial phase is the same for both to visualize it better. As before, the green line marks the real ToF. We can transform the frequency mismatch into phase with the well-known $= 2\pi ft$. The receiver will interpret the phase value of the orange signal as the one marked in red in its own blue signal, which is deviated $_{rx} = 2\pi f_{tx}t$. The transmitter experiences the opposite $_{rx} = 2\pi f_{tx}t$ deviation from the real value marked with the green line and it is shifted to the red mark. Mathematically, we can express it with in Equations (3.8) and (3.9).

$$CFO^{rx}_{i;0}(t) = e^{j(f^{tx}_{i;0} \quad f^{rx}_{i;0})t} \tag{3.8}$$

$$CFO^{tx}_{i;0}(t) = e^{j(f^{rx}_{i;0} \quad f^{tx}_{i;0})t} \tag{3.9}$$

The CFO will cause the same e ect as the PLL O set in the data, and we can take advantage of how similar they are to remove them as a whole. We can notice that in all the Equations (3.7), (3.6), (3.9) and (3.8), the only di erence between transmitter and receiver is the sign. We can take pro t of the channel hopping protocol's need to communicate transmitter and receiver to collect data from both sides, and if we put all of them together we can express the extra added phase as:

$$CSI^{rx} = h_{i;0}e^{j(f^{tx}_{i;0} \quad f^{rx}_{i;0})t+(\quad ^{tx}_{i;0} \quad ^{rx}_{i;0})} \tag{3.10}$$

$$CSI^{tx} = h_{i;0}e^{j(f^{rx}_{i;0} \quad f^{tx}_{i;0})t+(\quad ^{rx}_{i;0} \quad ^{tx}_{i;0})} \tag{3.11}$$

Multiplying the data from both sides, we can remove the o set and obtain

the phase free from all delays:

$$CSI = h_{i;0}^2 e^{j(f_{i;0}^{tx} \quad f_{i;0}^{rx})t + (\quad_{i;0}^{tx} \quad \quad_{i;0}^{rx}) + j(f_{i;0}^{rx} \quad f_{i;0}^{tx})t + (\quad_{i;0}^{rx} \quad \quad_{i;0}^{tx})}$$

As this is an exponential multiplication, terms of di erent sign cancel each other. We end up obtaining the squared Channel State Information, and no added phase rotation at all:

$$CSI = h_{i;0}^2 \tag{3.12}$$

Chronos multiplies the CSI from the transmitter and the receiver together to get rid of the CFO and PLL O set.

In the next section, we will see how this new expression for the CSI a ects the computation of the nal result.

3.3.3 Direct Path Determination

All the process explained until this point assumes that there is just one path from the transmitter to the receiver. However, we know that in an environment full of obstacles and relatively small like the inside of a building; that will never be the case. The signal that arrives at the receiver is formed by a group of re ections of the same transmission, that has traversed di erent distances to reach the endpoint. This phenomenon is called multipath.

Because of this, it is essential to nd a way of disentangling all the trajectories and discriminate the direct path, if it exists. Once we know the ToF of the path with no re ections, the distance computation is straightforward, just multiplying it by the speed of light. In this kind of applications there is usually a direct path; however, it is also typical that this one has no Line of Sight (LoS). Which means that the signal arrives attenuated by going through the obstacles that encounters in its way.

We will work under the premise that the direct path, with or without LoS, is the one that needs the least time to get to the receiver. The intensity of that fastest signal is a factor that will only tell us the presence or not of a LoS. In other words, the direct path will be the shortest, and therefore the fastest, even if it has to go through an obstacle to get to the destination.

Therefore, given a multipath environment, the received signal must be the sum of all the incoming re ections and the direct path:

$$h_{i;k} = \sum_{l=1}^{p} a_l e^{\quad 2 \quad j f_{i;k} \quad l} \tag{3.13}$$

which is the sum of all the signals that have travelled through the p di erent paths, with amplitude a_l, in channel (i), subcarrier (k) and Time of Flight τ_l.

Taking a closer look, we can see that this equation has the form of a Discrete Fourier Transform (in Equation (3.14)), so it would come up to our minds that computing the Inverse Discrete Fourier Transform (IDFT) it would be possible to disentangle all the di erent transmission paths.

$$X_k = \sum_{n=1}^{N} x_n e^{\frac{2}{N} j kn} \tag{3.14}$$

However, it is not as straightforward as inverting this equation, because the sampling of the signal in the frequency domain is not uniform. Therefore, we need an algorithm to invert the Non-Uniform Discrete Fourier Transform (NDFT). This equation has a problem: there is no direct and unique result for it.

The aim of this equation is to compute the parameters x_n from the known X_k; that is, to obtain the time response from the response in frequency. The time response vector (x_n) is called the Power Delay Pro le (PDP), and it is the output of the INDFT. The PDP will show us the instants in time at which the signal coming from di erent paths has arrived. We can see an example in Figure 3.4b. These three peaks indicate that the signal has arrived at the receiver from three di erent paths. The rst one, and therefore the direct one took 1 ns. The second and third travelled for 3 ns and 7 ns and were attenuated in their way. We can see that this PDP corresponds to a LoS transmission because the direct path is also the most powerful trajectory. In the case of a NLoS transmission, it could happen that, depending on the attenuation caused by the obstacle blocking the direct path, any other trajectory arrives with an intensity higher than the direct path transmission.

The direct path, for both LoS and NLoS conditions, is the shortest and fastest transmission. Therefore, the rst peak of the PDP corresponds to the direct path, as it arrives the rst to the receiver, regardless of its intensity.

Finally, we need to include the e ect of inputting the data as in Equation (3.12). Now the phase information is squared, so we need to work with the square of the Equation (3.13). As explained in [36][p.171], applying the binomial expansion we can see the new output of the time vector.

Following the same example given in *Chronos*, let's assume a signal that arrives at the receiver along two paths, then $p = 2$:

$$h^2_{i;0} = \left(\sum_{l=1}^{p=2} a_l e^{\ 2\ j f_{i;0}\ _l} \right)^2 = (a_1 e^{\ 2\ j f_{i;0}\ _1} + a_2 e^{\ 2\ j f_{i;0}\ _2})^2$$

So, applying the binomial expression:

$$\begin{aligned} h^2_{i;0} &= (a_1 e^{\ 2\ j f_{i;0}\ _1})^2 + 2(a_1 e^{\ 2\ j f_{i;0}\ _1})(a_2 e^{\ 2\ j f_{i;0}\ _2}) + (a_2 e^{\ 2\ j f_{i;0}\ _2})^2 \\ &= a_1^2 e^{\ 2\ j f_{i;0} 2\ \ _1} + 2\ \ a_1 a_2 e^{\ 2\ j f_{i;0} (\ _1 + \ _2)} + a_2^2 e^{\ 2\ j f_{i;0} 2\ \ _2} \end{aligned} \tag{3.15}$$

The ToF for both paths will show up as double their real times. Apart from this, a new term appears in the equation, giving a ToF that doesn't correspond to any existing trajectory. If we assume that τ_1 is the time for the direct path, the following applies:

$$\begin{aligned} \tau_1 < \tau_2 &\Rightarrow 2\ \tau_1 < 2\ \tau_2 \\ &\Rightarrow 2\ \tau_1 < \tau_1 + \tau_2 \end{aligned}$$

Therefore, the shortest path time will continue being the smallest result of the time response vector, but it will show up as double its real value. We can obtain the actual time response of the channel dividing this number by two.

Using CSI^2 as the input data for the INDFT will cause that the first peak appears at twice the distance, but this one will still be the first. If we halve this value, we can obtain the true ToF of the signal.

In Section 4.2.3, we will explain and analyze the algorithm of approximation used by *Chronos* to make the INDFT converge to a solution.

Chapter 4

Implementation

In this chapter we will describe the implementation of that explained in Section 3.3. We will indicate the key points and the problems found during the process, as well as the provided solutions. This chapter has two parts: *hardware setup and installation* and *software implementation*. In the hardware section, we will describe the characteristics and critical points for the installation of the chosen hardware. In the software section, we will go through the steps to install the software for collecting the CSI information. We will also explain the *Chronos* software re-implementation that we have developed, emphasizing the parts that may cause any issues in future projects.

4.1 Hardware Selection

Recall that to select a valid chipset it has to comply with the following two conditions: the operation in 802.11n WiFi protocol and the existence of specialized software that can access the physical layer from the userspace to recover the CSI. Nowadays there are only two known software tools for this kind of WiFi chipsets. The oldest one, developed by Daniel Halperin et al. in 2011 [18], works exclusively for the Intel Wireless Link 5300 (IWL5300) NIC card. In 2015 Yaxiong Xie et al. [46] developed a second tool designed to work for any device with an Atheros manufacturer chipset, installed either in an OpenWRT OS Access Point or NIC card of a laptop.

With the aim of using and testing a system as close to reality as possible, the first approach of this project was to re-implement *Chronos* using some commercial access points from TP-Link. According to the specifications [27], it integrates an Atheros AR9380 chipset (currently Qualcomm Atheros). Unfortunately, this approach didn't work, and we decided to change and use the same card as *Chronos* is using, the IWL5300. In the following sections, we will explain the process to install the Intel card and the software to recover the data it gathers. In Appendix A, we describe the Atheros tool,

the di erent attempts we performed to make it work and the reasons why we think it is not working for the selected hardware.

4.1.1 IWL 5300 Card and Linux CSI Tool

In this Section, we will pinpoint all the problems found during the installation of the hardware and building up of the setup. Then we will talk about the installation steps of the Linux tool that was developed to work along with it. We will give a user guide for its utilization and an example of the data representation obtained with it.

The Linux software tool for the IWL5300 card computes the CSI at the receiver side when the packets that arrive are sent in 802.11n rates and pass it to higher layers of the operating system, where the user can access it.

To do so, there are two main places where to apply software modi cations: the operating system and the wireless card rmware. The change in the rmware is transparent to us as we can't see the code, but we are aware of its purpose. The new rmware enables the debug mode in the card. This mode was designed by the chipset developers to check the functioning of the CSI computation, so the chipset will compute it for any packet that arrives with an 802.11n rate. Thanks to the transfer of the code from their side, we are able now to see those values outside the physical layer.

On the other hand, there are some changes to the wireless drivers in the operating system, so that the data can arrive at the userspace. These changes will allow logging multiple aspects as well as the data, and to pipe it upstream to the userspace.

IWL5300 Card Installation

The Hardware installation consists of two main components: the IWL530 wireless card and the laptop where to plug it. There is a third important part, the antennas, about which we will talk later on.

The main problem with the card, mainly due to its availability in the market, is the size to choose. There is a full-size card and a half-size one. The small one was designed to t in the newest and more compact laptops, where the big one doesn't t. The second important feature of any of the cards is that they use a PCIe connector [1]. These two factors will restrict the computer we can choose to install the card. Both size cards should support this tool, but we have only tested it with the full-size one.

The full-size IWL5300 card will not t in the newest computers, where the components are more compact to reduce the size of the laptop.

[1] The complete datasheet of the IWL5300 card can be found in https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ultimate-n-wifi-link-5300-brief.pdf

For this project, we had access to full-size cards, and we reused a couple of old IBM Lenovo R60 laptops, kept by the TU Delft's ICT department after no longer in use. Their most interesting specifications are:

Intel Centrino Duo processor.

Intel PRO/Wireless 3954ABG as the native wireless card.

PCIe connector.

On the laptop side, the critical issue is what is called the hardware whitelisting. Some laptops include a list of accepted hardware components. If we substitute one of the native components by a new piece of hardware that is not on the list, the computer will not boot. For the Lenovo computers, and in particular for this model, we find this problem.

We found three different approaches to overcome the whitelist problem:

1. Modify the BIOS changing a bit in the nonvolatile BIOS memory. In a Linux Operating System, we can access this memory reading the `/dev/nvram` file.
2. Install a modified BIOS that removes the hardware whitelist restriction.
3. Change the card ID so that it matches with one that is in the approved hardware list.

The first and third option are explained in detail in [32]. For the first option, they provide a small program that changes one bit in the BIOS memory space to disable the whitelist check. Unfortunately, this method is not extensible to every Lenovo Thinkpad model, and it didn't work out for the R60. The third option seems simple, but it is difficult to find the original table of accepted cards, and the procedure is quite risky.

The final solution was to find or build ourselves a modification of the BIOS that bypasses the check. After an intensive search, a modified BIOS was found in [13]. The steps for the installation of the new BIOS are the following:

1. Update the Lenovo BIOS version to v2.23 [21]. The tool to install it only works with Windows OS, so a prior step to this is to have installed any version of Windows [2].
2. Then, use the same tool to install the modified v2.23 BIOS with the procedure of step one.

[2] I tried to use a portable version of Windows XP for this laptop, but it didn't work out because some drivers for the battery were not working fine. The battery needs to be charged and plugged for the BIOS installation to work.

3. Finally, the laptop should be able to boot with the new card, and the Windows OS is not needed anymore.

The hardware whitelisting is a common feature in some laptops. If the new piece of hardware is not on the list of accepted devices, the computer will not boot. We need to modify the BIOS of the laptop if we want to make it accept the substitution of the card.

Channel State Information Tool Installation

Now that the laptop can boot, we can install the CSI tool for Linux [18]. The only requirement here is to choose a Linux based Operating System with a supported kernel version between 3.2 and 4.2. For this project, we chose an old Ubuntu distribution, specifically 14.04 with v.3.13 kernels. Be aware of the support that the Ubuntu community provides to the kernel versions [35]. Most of the old versions of the OS will have updated v.4.4 kernels, so the right 14.04 distribution can be found in [34].

The CSI tool needs to merge its modifications with the whole kernel structure of the operating system. For the tool to work, it is necessary a Linux based operating system with a supported kernel version between 3.2 and 4.2.

Once the Operating system is set up, we can start with the installation of the CSI tool following the instructions in [17]. As indicated, this process has five steps to be performed from the command line, and can be summarized as follows:

1. **Prerequisites**
   Install the build tools, the Linux development headers according to the kernel version, and the Git client. After this, my recommendation is to follow the first tip and disable the Network Manager to avoid conflicts in the management of the wireless connections. All the configurations can be done with the `iw` utility from the command line.

2. **Build and Install the Modified Wireless Driver**
   Obtain the CSI Tool Linux source tree, with the appropriate changes to the wireless driver that match the kernel version. **Note:** The direct cloning process of the CSI Tool repository took a really long time for this laptop ($>$ 24h). The processing capability of the laptop has an important role in this step. To save some time, cloning the repository in one computer with the same kernel version tag, and copying it in the wanted one was proven to work.

   It is indispensable to merge these driver modifications with the Linux source tree for the distribution-provided kernel version, as explained

in the first tip. This process should resolve without conflicts if we are using a supported kernel version by the tool (like v.3.13). If the merge outputs conflicts, we should check we are indeed working with the correct kernel version[3]. The tool won't work if we ignore this mismatch in the merging.
Finally, we have to build and install the modified driver as explained in the instructions. For this laptop, we will get the message `can't read private key` after loading the new driver, but it will not affect the functioning of the tool.

3. **Install the Modified Firmware**
   Substitute the original IWL5300 card firmware by the modified firmware.

4. **Build the Userspace Logging Tool**
   Consists on building the tool that logs at the userspace the CSI that comes from the driver.

5. **Enable Logging and Test**
   Finally, we have to unload the driver and reload it enabling the logging capabilities. To enable the different logging possibilities there is a parameter called `connector_log` loaded at the same time as the driver.

How To Use the CSI Tool for Linux

`connector_log` indicates the amount and type of information to log. In the code of the tool, there are several masks defined associated with different kinds of information which we contain in Table 4.1.

Table 4.1: **Different masks to set the value of `connector_log`.**

Mask	Value
IWL_CONN_BFEE_NOTIF_MSK	0x1
IWL_CONN_RX_PHY_MSK	0x2
IWL_CONN_RX_MPDU_MSK	0x4
IWL_CONN_RX_MSK	0x8
IWL_CONN_NOISE_MSK	0x16
IWL_CONN_TX_RESP_MSK	0x32
IWL_CONN_TX_BLOCK_AGG_MSK	0x64
IWL_CONN_STATUS_MSK	0x128

Either doing a bitwise AND or adding their hexadecimal values we combine all the information to record. So, if we want to get the computed

[3] The command `uname` outputs the kernel version.

CSI as well as log the payload of the incoming packet, we need to add IWL_CONN_BFEE_NOTIF_MSK and IWL_CONN_RX_MPDU_MSK and set connector_log to 0x5. For this project, this is the information that we need. We can get the CSI and use the payload of the packet to contain information to use as protocol control that we will later explain.

The connector_log parameter loaded with the driver indicates the type of information we want to log. It is formed combining the masks of the di erent types. For example, connector_log = 0x5 if we want to obtain the CSI and get the payload of the incoming packet.

In principle, this would be enough to set up the tool if we want to compute the CSI for packets coming from an AP. We need to establish an unencrypted connection with an AP and run the following command to start the CSI logging utility:

```
sudo linux 80211n csitool supplementary/netlink/log_to_file csi.
    dat
```

Then, we can compute the CSI from the packets that the AP pings back in 802.11n rates. However, the developers of the tool already pointed out that the most reliable mode is the monitor mode. We could test that indeed, the AP-Client mode did not work. As both ends need to work as AP and Client (transmitter-receiver) at certain moments, the monitor mode suits the requirements of the project. From now onwards, we will explain and refer to the monitor mode for the communication mode between devices.

The standard monitor mode de nes one end as the packet injector in the network and the other end as receiver in monitor mode. So, in the tool source code there is a script to run for the device acting as the sender, and another one acting as the receiver (setup_inject.sh and setup_monitor_csi.sh). As for our setup, we need to have both devices injecting and receiving we built a script that combines both in Listing 4.1.

Listing 4.1: Script to set the injector mode and monitor mode in the same machine

```
1  cd linux 80211n csitool supplementary/injection/
2
3  modprobe -r iwlwifi mac80211 cfg80211
4  sleep 1
5  modprobe iwlwifi connector_log=0x5 debug=0x40000
6  sleep 1
7
8  iwconfig wlan0 mode monitor 2>/dev/null 1>/dev/null
9  while [ $? -ne 0 ]
10 do
11  iwconfig wlan0 mode monitor 2>/dev/null 1>/dev/null
12 done
13 sleep 1
14 ifconfig wlan0 up
```

```
15 sleep 1
16 iw wlan0 set channel $1 $2
17
18 echo 0x1C113 | sudo tee `sudo find /sys -name monitor_tx_rate`
19 cd
```

Note: The `sleep 1` are additions to the original injection and monitor mode scripts. They give some time for the con guration to take place, and we reduce the chance of obtaining a \device busy" error when running the script.

The line 18 in Listing 4.1 is used to set the transmission rate. This step is crucial, as only packets sent with High Throughput (802.11n) rates can be used to compute the CSI. The bits conforming this number adjust di erent parameters of the transmission, but in our case, the important one is the number of streams or transmitting antennas. For further details about this number, the code can be accessed in [15].

> We must always set the transmission data rate to High Throughput (802.11n). If the con gured bitrate is not an 802.11n rate, the receiver will not be able to compute the CSI from that packet.

The value of the rate is formed putting together the following parameters:

- The antenna or antennas that are transmitting the signal. A, B, C, AB or ABC. These correspond to the antennas 1, 2 and 3 of the chipset. Each of these possibilities has a corresponding mask associated.
- A bit indicating if this is a High Throughput (HT) rate or not.
- The Modulating and Coding Scheme (MCS), which is a reference number of the combination of the number of spatial streams, modulation type, and coding scheme [39, 25].

In Table 4.2 we can see all these values together forming the rates that we have used for this project.

Table 4.2: **Values to set the transmission rate.**

No. Antennas	Tx Ant.	Ant. mask	HT (yes)	MCS	Full Value
1	A	0x04000	0x00100	0x00003	0x04103
1	B	0x08000	0x00100	0x00003	0x04103
1	C	0x10000	0x00100	0x00003	0x04103
2	AB	0x0C000	0x00100	0x0000B	0x0C10B
3	ABC	0x1C000	0x00100	0x00013	0x1C113

Finally, we need to run the script of Listing 4.1 using as rst parameter the WiFi channel where to set the device, and as second parameter HT20

con guring it as a High Throughput 20MHz width channel. For example, if we want to con gure the system in channel 64 (center frequency 5320MHz):

```
sudo ./script.sh 64 HT20
```

Phase Reported by the Tool

The CSI tool includes a series of functions for the data processing in MATLAB, besides the already mentioned software to log the CSI. In this part, we will explain the signi cant steps to get the phase from the data reported by the tool [16].

First of all, there is a function that opens the `.dat` le containing the logged raw data and parses it searching for the CSI. After this step, we will be able to visualize a structure with several parameters for each packet, such as the RSSI values of each antenna or the rate at which it was sent. The second essential function that they provide is called `get_scaled_csi`. As explained in [16]: \this function will compute the CSI in absolute units, rather than Intel's internal reference level". It will use the values of the structure, like the RSSI and noise, to obtain the normalized CSI.

The nal format of the data that we will use is a three-dimensional matrix of complex numbers representing the phase and amplitude of the incoming signal. The rst dimension is associated with the stream from which that signal has been measured, this is, which transmitter's antenna has cast it. The second dimension corresponds to the antenna that has received the signal and reported the phase information. Finally, the third dimension is related to the subcarrier of the signal carrying the data. This card only reports 30 subcarriers, which for the case of 20 MHz bandwidth channels are 312.5 kHz apart from each other at both sides from the center frequency. So, if for instance, we are working with three transmitting and three receiving antennas, the format of the data will have the shape of a 3x3x30 dimension matrix of complex numbers. For a general view of this, we have Figure 4.1.

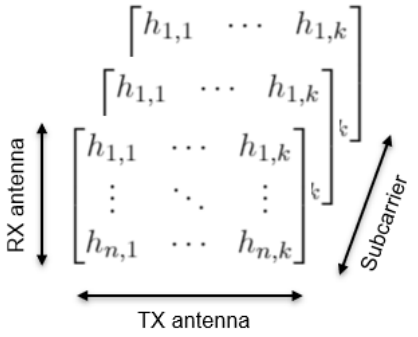


Figure 4.1: Channel State Information matrix format.

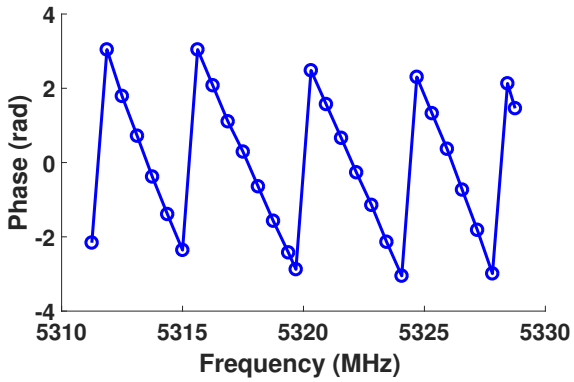


Figure 4.2: Phase reported by the CSI tool for one of the antennas of the laptop acting as receiver (values contained between π and $-\pi$). Each dot corresponds to a different subcarrier.

Finally, in Figure 4.2 we can see an example of the phase reported by the tool using real wireless data. We gathered this data with one of the two internal antennas of the laptop to prove the correct installation of the tool. Using the function `angle()` in MATLAB, we can obtain the phase from the values of the matrix contained between π and $-\pi$. In Chapter 5, we will talk about the final hardware setup with the external antennas, and analyze in depth the data that these obtain.

4.2 Software Implementation

In this Section, we will explain the implementation of the software in this work. In the first place, we will show the design of the software that is running in both laptops to synchronously change the channel and at the same time collect CSI data on both sides. *Chronos* gives a general view of its functioning; however, the implementation belongs to the contributions of this project. We will walk through it and explain its main features. Then, we will comment on the MATLAB code developed for the data processing and the delay removal. Finally, we will focus on the computation of the Inverse Non-Uniform Discrete Fourier Transform algorithm presented by *Chronos*, and we will pinpoint the problems found here and the given solutions.

4.2.1 Channel Hopping Protocol and Data Gathering

As mentioned in Chapter 3, to collect CSI data in every channel we need to implement a protocol that performs the synchronous channel change. This mechanism needs to allow the sending of channel sounding packets and recording of CSI at both sides. Therefore, we can use this need for

synchronizing the channel change to measure the CSI, reusing the channel jump request to compute the CSI.

Apart from recording the data, we need a way to recover it to perform the post-processing and compute the distance between devices. For this, we have developed a server program where both devices send the data that they collect.

Channel Hopping Protocol

In this system, we have two participating devices that we can call Access Point and Client. The AP is the device that starts the channel changing process; this is, the one that sends the requests to the other end asking for a change to the next channel in the band. The Client has a more passive role because it waits for the AP's petition to change. The AP keeps track of the current channel and tells the Client to which one has to switch.

For this implementation, we have two di erent programs, one to run in each device and started by the user. Following the path of *Chronos*, we have implemented a Request-Acknowledgement (REQ-ACK) protocol. We will leverage the protocol design to compute the CSI from both sides. The REQ and ACK are 802.11n packets and include in its payload all the information that we need, for example, the next channel in the sweep. We can compute the CSI from them and get the necessary information to control the process.

Once the CSI data is ready, a server collects it from both devices. The data processing to compute the distances between devices is done in MATLAB running in another machine. The top-level architecture of the system with its inputs and outputs is shown in Figure 4.3.

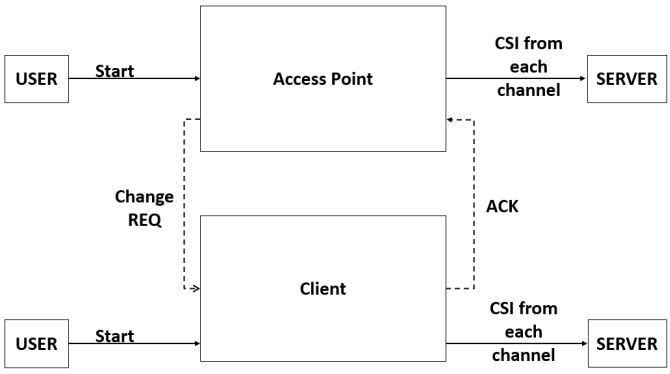
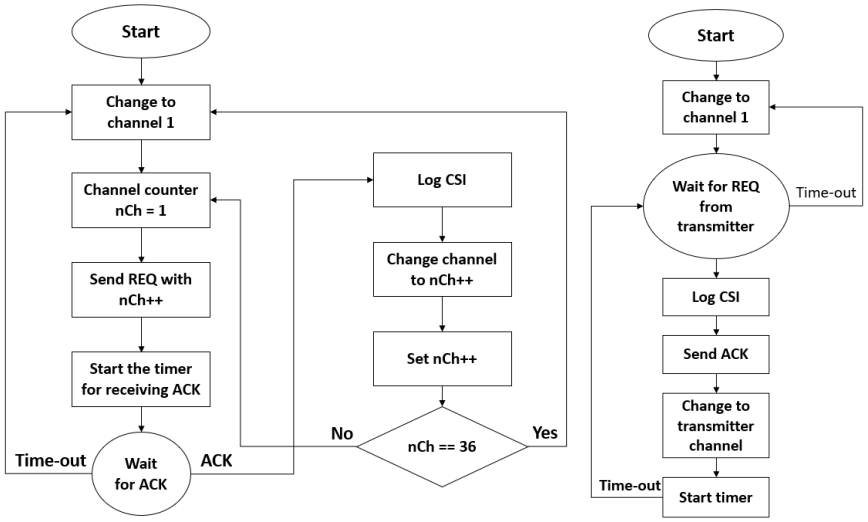


Figure 4.3: General view of the functioning of the system.

To explain what happens when the packets sent don't arrive at the destination, we can take a look at Figure 4.4a and Figure 4.4b. In these diagrams we have a ow chart of the program running in the AP and the Client. The AP starts a timer every time it sends a channel change REQ to wait for the

ACK from the Client. If that ACK is received, the AP logs the CSI and sends the next REQ. In the case that the ACK is never received, after the timer expires, the channel is reset to the rst one on the list. On the side of the Client, the timer is set continuously to wait for REQs. If the REQ is received, the device changes to the next channel, logs the CSI and waits for the next one. On the contrary, if a time-out occurs, the same as the AP, the device is reset to the rst channel of the list.



(a) Access Point ow chart. (b) Client ow chart.

Figure 4.4: **Program architecture flow charts.** (a)AP side ow chart. (b)Client side ow chart.

Getting into the actual implementation of the code, we use three threads to achieve the functionalities explained up to this point. We need a thread that can send the packets using the procedure of the tool so that the other end can compute the CSI, but that can also embed the necessary information of the protocol in the payload. Another thread should be in charge of receiving those packets and logging the CSI data. Finally, we need a third thread that can keep track of the time for the packet reception time-outs. Therefore, we can call these threads Packet Transmission Thread (main thread), Packet Reception Thread, and Timer Thread. They communicate with each other through the generation of events. The events, in turn, are capable of changing the state of the state machine running in the main thread.

Summarizing, the purpose and functionality of the threads are the following:

Packet Transmission Thread

– This thread is the main execution thread. It is a state machine

with two essential states: \Waiting for ACK" and \Changing Channel".

- The state will change if it receives a *Time-out event* or a *Packet Received event* from one of the other threads.
- The *Packet Received event* will unchain an ACK transmission and a channel switch in the side of the Client. In the case of the AP, the reception of the ACK will cause a channel change, the transmission of a REQ for the next channel and the restart of the timer.
- The *Time-out event* will cause, in both cases, a reset to the rst channel of the list.

Packet Reception Thread

- This thread is in charge of receiving the packets, logging the CSI and generating the corresponding event (*Packet Received event*) to the Packet Transmission Thread.
- For the Client, as the number of the channel is in the payload of the packet, this thread has to parse it and pass it to the main thread to perform the channel change.

Timer Thread

- This thread gets executed periodically to implement the time-outs of the AP and Client to wait for the reception of an ACK or REQ.
- When the timer expires, it sends the corresponding event (*Time-out event*) to the main thread.

After testing the implementation, we concluded that the system was working with a low rate of loss packets (around 5%) for the band of 5 GHz following this protocol and logging the CSI correctly. However, for the 2.4 GHz band, the packet loss rate was unbearable (around 90%), and the system had to start over in the rst channel too often. The reasons for this behavior could be diverse, like the wireless tra c of the environment or the position of the antennas. We performed various tests modifying these two factors. We tested the system in an isolated environment with no wireless transmissions that could disturb our process, and we also tried di erent positions and angles for the antennas, but the improvement in the loss rate was insu - cient[4]. We nally decided to use only the 5 GHz band (24 channels add up to 645MHz bandwidth) for the experiments, and adjust all the computations to this situation. We will see the new granularity in Section 4.2.3.

[4] This issue with the band of 2.4 GHz has been reported by other users of the tool: https://github.com/dhalperi/linux-80211n-csitool-supplementary/issues/287

Server Implementation

In the setup for this system, an external computer plays the part of the server, while the two Lenovo laptops interact as AP and Client, and send the CSI information that they collect to it. We should note that the modi ed IWL5300 rmware does not allow the connection to encrypted APs [16]. Also, as both sides need to be working in injector mode, we cannot be connected to an AP. For these two reasons, the three devices mentioned before have a wired connection through a hub. The IP addresses of the devices are xed and hardcoded in the server source code.

The laptops send every packet that they log to the server, indicating in which channel it was received. Then, in the processing stage in MATLAB, we perform a preliminary step to detect and use only the scans that are complete.

4.2.2 Delay Compensation Code

Once we have the data, we need to process it to remove the delays that we explained in Section 3.3.2. In this Section, we will talk about how to manipulate the data, perform the data interpolation to obtain the zero subcarrier value and remove the nal delays with the data multiplication all using MATLAB.

As seen in Figure 4.1, we have a matrix of dimensions 3x3x30, and we want to get the data from every receiving antenna corresponding to each transmitter stream. Therefore, for 3 RX and 3 TX antennas, we end up with nine vector shape structures of size 1x1x30. We use this vector to interpolate the data for the zero subcarrier, based on the other 30 subcarriers reported.

The interpolation function we use, as in *Chronos*, is the Cubic *spline* interpolation [23] provided by MATLAB. We enquire for the zero subcarrier using the code in Listing 4.2 and use the output for the rest of the process, only that subcarrier. The other two arguments of the function are the subcarrier indexes and their corresponding data.

Listing 4.2: Cubic spline interpolation to obtain the zero subcarrier data.

```
1 SubCarrInd = [ 28, 26, 24, 22, 20, 18, 16, 14, 12, 10,
      8, 6, 4, 2, 1, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21,
     23, 25, 27, 28];
2 query = (0);
3 h0 = spline(SubCarrInd, data, query)
```

Finally, we remove the last delays, the CFO and PLL O set, multiplying the CSI measured from both sides. We have to be careful and multiply the corresponding zero subcarrier of a channel with its respective one at the other device from the same RX-TX antenna combination. After this process, the values will be ready to feed the INDFT algorithm.

4.2.3 Inverse Non-Uniform Discrete Fourier Transform Algorithm

If we recall the previous chapter, we already pointed out the problems of this algorithm. Due to its irregular frequency sampling, it is not possible to obtain a direct and unique solution to the inversion of the Fourier Transform. The equation of the Discrete Fourier Transform in (3.14) and we can reformulate it as a matrix form expression like in Equation (4.1). This system of equations is under-determined as some of the responses in frequency are missing and therefore it has multiple possible solutions.

Among all these possible solutions, *Chronos* will choose the one that gets closer to satisfying one of the characteristics of indoor transmissions: the sparsity; in other words, to have a small number of dominant paths. According to *Chronos*, mathematically this characteristic is translated into finding the minimum of the norm of the INDFT's output vector [5]. We want to obtain the response in time ($x(t_i)$) from the equation in (4.1) adding this last constraint of minimizing the norm.

$$\begin{bmatrix} X(f_1) \\ \vdots \\ X(f_n) \end{bmatrix} = \begin{bmatrix} F_{1;1} & & F_{1;k} \\ \vdots & \ddots & \vdots \\ F_{n;1} & & F_{n;k} \end{bmatrix} \begin{bmatrix} x(t_1) \\ \vdots \\ x(t_k) \end{bmatrix} \tag{4.1}$$

As said before this expression is just Equation (3.14) in the matrix representation. Therefore, $X(f_i)$ is the response in frequency from the measured in every channel of the non-contiguous frequency chunks of the WiFi band; and $F_{i;k}$ is the Fourier matrix term that for the i^{th} channel and k^{th} path of Equation (4.2).

$$F_{i;k} = e^{\ j2\ f_{i;0}\ k} \tag{4.2}$$

The approach used by *Chronos* to converge to a solution in an optimization problem with these characteristics, is the so-called Proximal Gradient method. Specifically, within these methods, it employs a Proximal Gradient Descent style algorithm.

The gradient [24] can be seen as the derivative of a multi-variable function, so it represents the slope of the tangent of a function. It shows the direction of the highest increase of the function (potentially a maximum) and the magnitude of this increment.

The Proximal Gradient Descent method [19] follows the opposite direction of what the gradient is pointing searching for the minimum. This iterative algorithm will advance one step further from the maximum in each iteration. If we consider $f(\vec{x})$ the multi-variable function to optimize and r representing its gradient; x_t and x_{t+1} the current and next iteration of the algorithm

[5] The full mathematical expression can be seen in (8), (9) and (10) of Chronos paper [36].

and λ the size of the step we want to take following the gradient, we can see in Equation (4.3) the mathematical expression of this method.

$$\vec{x}_{t+1} = \vec{x}_t \quad \lambda r(f(\vec{x}_t)) \tag{4.3}$$

We will soon see how the size of the step plays a crucial role in the convergence of this iterative algorithm. When λ is small, we run the risk of needing a large number of iterations to get to a solution. However, if the step is too big, we might skip the minimum we are searching for and diverge completely.

We can see in Listing 4.3 the piece of code extracted from the paper to perform the mentioned algorithm. Here $\vec{p}_t$ and $\vec{p}_{t+1}$ are the current and next iteration values of the algorithm, where $\vec{p}$ is the objective output. F is the Fourier matrix and λ the iteration step. Next we will explain what the sparsity parameter α and the convergence parameter ϵ are.

Listing 4.3: INDFT algorithm based on Proximal Gradient Descent method [36].

```
1  while converged = false do
2      p_{t+1} = SPARSIFY(p_t    F*(Fp_t    h),    )
3      if ||p_{t+1}  p_t||_2 <  then
4          converged = true
5          p_t = p_{t+1}
6      else
7          t = t + 1
8      end if
9  end while
10
11 function SPARSIFY(p,d)
12     for i = 1, 2, ...length(p) do
13         if |p_i| < d then
14             p_i = 0
15         else
16             p_i = p_i (|p_i| - d)/|p_i|
17         end if
18     end for
19 end function
```

We want to emphasize certain parts of the code that are important to understand the implementation of the algorithm:

> **line 2**: This line of code performs the main functionality of the algorithm. Here we are iterating over the Proximal Gradient Descent equation as in Equation (4.3). The expression should have the form of Equation (4.4); however, they set it as one of the arguments of the function called *sparsify* (de ned in line 11). This function enforces sparse solutions. It maintains the paths of power over a certain threshold and sets the rest to zero. This way, the next iteration value

will have less dominant paths, and those that remain will be computed with a higher resolution.

$$\vec{p}_{t+1} = \vec{p}_t \quad \lambda F \ (F\vec{p}_t \quad \hbar) \tag{4.4}$$

l.3: In this line, we will decide if the iteration is over or not. The criterion to decide it resides in the di erence between the current and next iteration. If this di erence is below a certain threshold, we will consider that the algorithm has converged. The value of the threshold ϵ is up to the user. In our case, a value of 10 ^{6} gave us a good execution time vs sharp solution ratio.

l.11: The *sparsify* function gets as arguments the output vector for the next iteration of the Proximal Gradient Descent algorithm, and what we could call sparsity threshold. The sparsity parameter α and the iteration step λ form this threshold. The output of the function, as already mentioned, will be a more sparse next iteration vector.

Problems Found

While implementing and testing this piece of code, we spotted a typo in one of the parameters of the paper that prevented the algorithm from converging. Another sensitive point was how to choose the resolution of the time vector, this is, the τ_k in the Fourier matrix expression of Equation (4.2).

The step (λ) size

The step size will determine the progression of each iteration alongside the direction determined by the gradient. As mentioned before, if the step is too big we run the risk of passing over the minimum. While if the step is too small, the time and computation resources to reach that minimum can be excessive. On top of this, the initial value of the output vector (iteration 0) is set to a random value, so the step needs to be carefully chosen to converge in a reasonable time.

Getting back to *Chronos*, the step size is $\frac{1}{jjFjj_2}$, where F is the Fourier Matrix of Expression (4.2). However, when testing the implementation of the whole algorithm, we saw that the solution was always diverging to in nite. After an extensive search, we determined that the step size was too big and it was preventing the algorithm from converging.

Looking deeper into the gradient method, we see that it is inside a group of algorithms used to solve the optimization problem called Base Pursuit Denoising (formulation in expression (4.5)). This problem has the form of Expression (10) in *Chronos* [36]. There are several alternative possibilities to obtain a solution, like the in-crowd algorithm that can solve large and sparse problems [14]. However, we focus on the

gradient method for the L-1 norm minimization and its step size. The expression in (4.5) can be reformulated and appear under the name of Lasso formulation [6].

$$\min_x \frac{1}{2}ky \quad Axk_2^2 + \lambda kxk_1 d \tag{4.5}$$

Looking at information about the in uence of the step size in the convergence of the gradient method ([19], [33], [11]), we concluded that the step needs to be smaller for the algorithm to converge. Based on the step de nition that E. Birgin et al. [4] give in their study (Expression (2)) and corroborated by Deepak Vasisht[7], the main author of the paper *Chronos*, we deduced that the step size should change to $\frac{1}{jjF\ Fjj_2}$ or the equivalent $\left(\frac{1}{jjFjj_2}\right)^2$.

With the step size stated in *Chronos*, $\frac{1}{jjFjj_2}$ the algorithm doesn't converge to a solution. Instead, we have to use the smaller value $\left(\frac{1}{jjFjj_2}\right)^2$.

The time vector (τ_k)

To compute the values of the Fourier matrix of Equation (4.2), we need the time vector τ_k; where k represents the number of possible paths of the multipath pro le. We can choose the length and granularity of this vector. For each of these time inputs, the INDFT algorithm will output a value that determines if a re ection of the signal has arrived at that time and its intensity.

If the resolution of this vector is low, we lose the chance of distinguishing several paths that arrived in a short time lapse. There is always a limit to the smallest distance that we can tell apart and having values below that threshold will not report any better results. However, in *Chronos* it is not indicated how to obtain this lower threshold. We determine our time vector according to the frequency that the 5 GHz band channels span, even if not in a continuous spectrum, which is 645 MHz. With this bandwidth, we can have a worst-case time resolution and space resolution of 1.55 ns and 46.5 cm. However, as we still need to divide the result by two to compensate for the squared input data, it increases to 0.78 ns and 23.25 cm. Finally, the time vector that we will provide as input to the INDFT will range from 0 ns to 50 ns

[6] https://en.wikipedia.org/wiki/Lasso_(statistics)#cite_note-Tibshirani_1996-2

[7] We communicated the typo to the author who corroborated the change of the parameter as we proposed

with values every 1.55 ns. For our tests in Chapter 5, we will see that we need to distinguish values every 30 cm and this time vector can provide us that resolution.

Chapter 5

Evaluation

In this Chapter, we will validate the implementation of Chapter 4 and evaluate the results of the method. To do so, we have divided the Chapter into three parts. In Section 5.1, we will show the evaluation setup of the system. We present the four different datasets that we will use to validate the implementation. In Section 5.2, we present the results of the evaluation with the different types of data. Finally, in Section 5.3 we will focus on the findings of Section 5.2. In particular, we will analyze the presence of an unexpected delay.

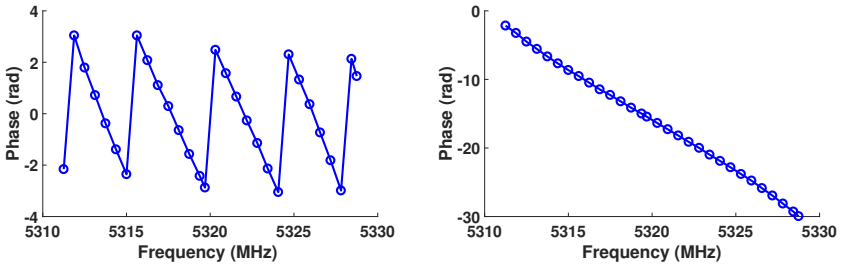
5.1 Evaluation Setup

In this section, we describe the setup for the evaluation. We use four types of data: Ideal, Ideal with Delays, Cable and Wireless data. We explain the purpose and the expected output of each of them.

5.1.1 Ideal Data

The ideal data of this section is mathematically generated data. It shows the phase that we would obtain if a signal could arrive at the receiver without being affected by the multipath effect. The hardware delays are also not included here. The aim is to validate the correct functioning of the INDFT algorithm. If it is working correctly, we should obtain a single path in the Power Delay Profile for the distance at which we have generated the phase information.

To understand the process of data generation we need to recall some concepts. First of all, we have to remember the difference between the phase rotation due to the distance traveled through the air (the ToF), and the phase rotation introduced by the hardware. For now, we want to generate data with just the first type of phase rotation because we are working with no delays. In the next subsection, we will show how to introduce the delays.



(a) Wrapped phase (b) Unwrapped phase

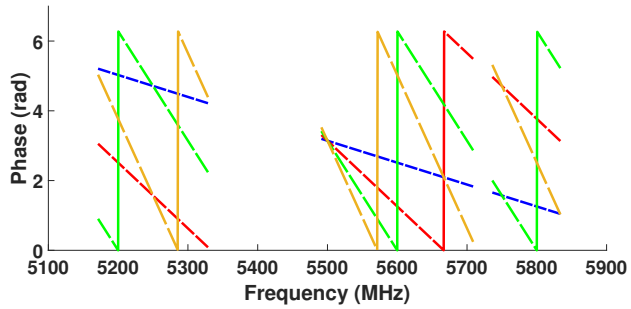
Figure 5.1: (a) Example of wrapped phase between π and π. (b) Example of unwrapped phase of a signal recorded at the Receiver. The channel frequency (center or carrier frequency) is 5.32 GHz and the dots represent each subcarrier.

Second, we need to go back to the representation of the data. In Figure 4.2 (also in Figure 5.1a) we could see an example of what is called *wrapped* phase. However, there is another possibility that appears more often called *unwrapped* phase. In this case, the data is plotted in a continuous way avoiding the 2π jumps in the border cases. As we can see in Figure 5.1b, the *unwrapped* phase continues after reaching 2π, and ts a linear regression (following the phase expression of Equation (3.4)).

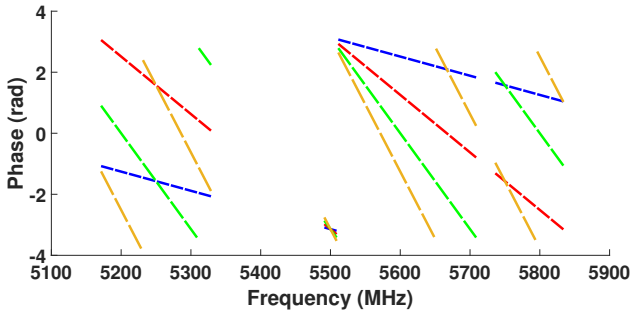
Then, to generate the data we use the formula in Equation (3.4). As we cannot distinguish how many periods of the signal have passed when it gets to the receiver, we have to apply the mod (2π) operation. For this test, we have created phase data in each of the 5 GHz band available frequencies for the distances of 30 cm, 90 cm, 150 cm and 210 cm. In Figure 5.2a we see the data in its wrapped representation, while in Figure 5.2b the unwrapped phase of each channel. The lines in the gure represent the data for each channel, which is, in turn, formed by the data of its subcarriers like the markers in Figure 5.1. The \jumps" in the data of Figure 5.2b appear because we unwrap every channel individually and there is no previous reference of the period of the signal. If we add the corresponding multiple of 2π to that data, then it will show up linear without discontinuities. As the phase of the incoming signal is dictated by $2\pi ft$, the more distance the signal travels, the higher its ToF will be. Therefore, the slope of the data in Figure 5.2b is steeper for longer distances.

Finally, another aspect that we have to highlight is that the data that we build remains in between 0 and 2π, while the real one, due to the characteristics of the software tool, is reported between π and π. Therefore, this data presents a shift upwards compared to the wireless one. However, the output of the INDFT algorithm will not be a ected.

The evaluation results for the Ideal Data are shown in Section 5.2.1.



(a) Wrapped phase



(b) Unwrapped phase

Figure 5.2: **Ideal phase data for different distances:** (a) Ideal wrapped phase for distances of 30 cm, 90 cm, 150 cm and 210 cm. (b) Ideal unwrapped phase for distances of 30 cm, 90 cm, 150 cm and 210 cm.

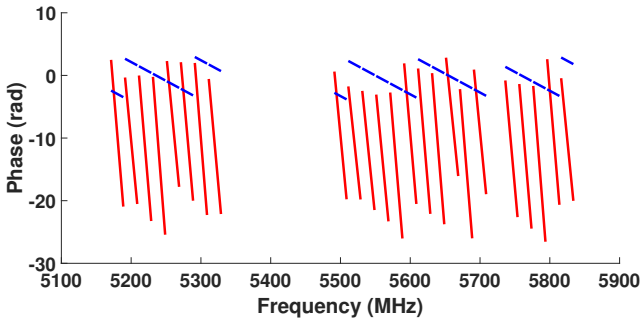
5.1.2 Ideal Data With Delays

In this section, we add the hardware delays to the ideal data. The purpose is to have phase information that is a ected by the delays explained in *Chronos* due to the hardware imperfections in the signal processing. This data does not include the e ects of multipath. It simulates the nature of a transmission through a cable that joins both ends, an experiment that we will carry out in Section 5.1.3.

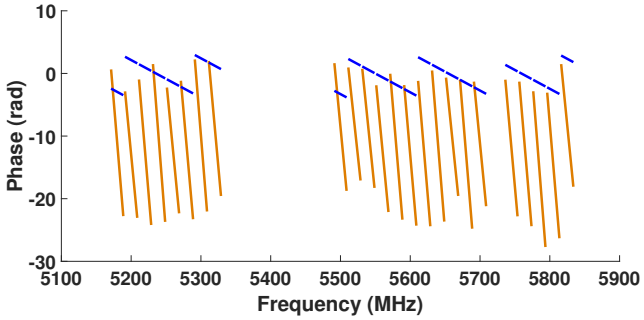
To generate this data we have to go back to Section 3.3.2 where we explained the Packet Detection Delay, the Carrier Frequency O set and the PLL O set. To generate data with PDD, we use the expression in Equation (5.1), that is a combination of Equation (3.4) for the ideal phase of a transmission at a certain distance, and Equation (3.5) that applies the delay to every subcarrier but zero.

$$\backslash h_{i;k} = (\quad 2\pi f_{i;k}\tau \quad 2\pi(f_{i;k} \quad f_{i;0})\delta_i) \quad \text{mod } 2\pi \tag{5.1}$$

To obtain τ, this is, the ToF, we divide the distance by the speed of transmission of the medium. For wireless transmissions, that speed is the speed



(a) Ideal Data with PDD Delay



(b) Ideal Data with PDD, CFO and PLL O set

Figure 5.3: **Ideal Data with Delays:** (a) Ideal data for the cable transmission (in blue) and the same data a ected by Power Detection Delay (in red). (b) Ideal data for the cable transmission (in blue) and the same data a ected by Power Detection Delay, CFO and PLL O set (in orange).

of light (c). For the case of the wire, it is $\frac{2c}{3}$. δ_i is the delay, that according to *Chronos* paper is large and unpredictable, with a median value of 177 ns and a standard deviation of 24 ns. Based on this, we generate the delay using a Gaussian distribution with these parameters. In Figure 5.3a we can see the data with PDD for a distance of 191 cm and the ideal data for comparison. This distance relates to the cable length of 191 cm that we will use in the next section. The other cable lengths are 341 cm and 491 cm, for which we will also generate data and compare the PDP results in Section 5.2.

We see in Figure 5.3a the expected e ect on the data: the subcarriers at the extremes su er a bigger shift and separate more from the zero subcarrier than the ones closer to it. The visible e ect is that the data is now more spred out: the length of the blue line (data with no delay) and the red line is no longer the same and the reason is that the subcarriers do not experience the same delay. We know that the zero subcarrier of the ideal and delayed data should coincide because, as seen in Chapter 3, it does not experience the delay. The displacement that we see in the plot is due to how MATLAB

represents the unwrapped phase with the function `unwrap()`. The phase is the same, only with a number of extra periods (2π). To see them coinciding in the plot, we would need to apply $\mod 2\pi$ the same number of times.

For the complete emulation of the data, we have to add the CFO and PLL Offset. We recall with expressions (3.10) and (3.11) that these delays apply to every subcarrier equally. Therefore, we can add them after the PDD using $2\pi ft$, where t is the delay. There is not a reference value for these offsets in the paper, so; we apply a random quantity enough to see its expected effect on the data. In Figure 5.3b we can see the output for one of the transmission ends. With CFO we saw a change in the length of the line that represents the phase information for the whole channel because not all subcarriers experience the effect of the delay in the same amount. In this case, for CFO and PLL Offset, the lines suffer a shift as a whole because the delay equally affects all subcarriers including zero. We saw in Chapter 3 that the receiver and transmitter experience the same delay but with opposite signs. Therefore, for the other transmission end, the data would look like Figure 5.3b but shifted in the opposite direction.

The evaluation results for the Ideal Data With Delays are shown in Section 5.2.2.

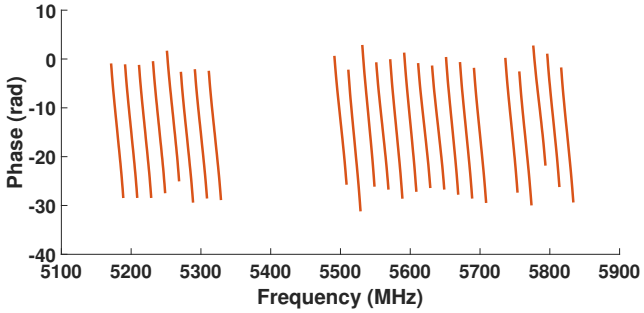
5.1.3 Cable Data

In this section, we already work with real data collected joining the antennas of both AP and Client. The purpose is to evaluate the results of using data that is not affected by the multipath effect but includes the real hardware delays. In the previous section, we generated ideal delays according to the formulas in *Chronos*. Therefore, the delay compensation techniques will always work correctly for the ideal data, and we will obtain the expected PDP. For the case of the cable data, the hardware delays are real. We will be able to isolate the effect of the hardware imperfections in the results, and check if *Chronos* is capable of getting rid of them and obtain a delay-free PDP.

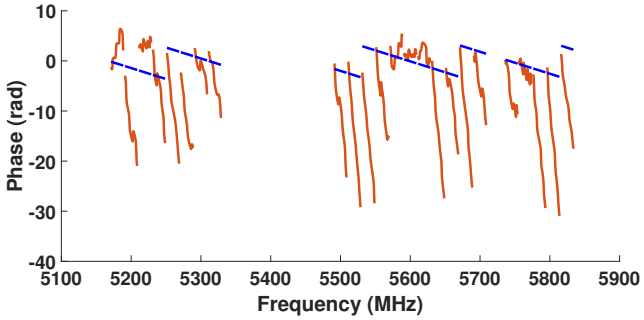
The setup consists of a cable, a 20dBm attenuator and a couple connectors that join everything together with a total length of 191 cm. In addition to this, we could obtain two extension cables that allowed us to collect data at 341 cm and 491 cm distance too.

In Figure 5.4a we plot the data collected at one of the ends of the 191 cm cable. This data is analogous to the one generated with PDD, CFO and PLL Offset, therefore we can compare it with Figure 5.3b. The data in both figures has a similar shape. We see that the data maintains its linearity in every scan as it is not affected by any other factor besides the hardware delays.

The evaluation results for the Cable Data are shown in Section 5.2.3.



(a) 191 cm Cable Data



(b) Wireless Data at 209 cm distance in orange and reference ideal data for 209 cm in blue

Figure 5.4: **Cable data and Wireless data:** (a) 24 channel scan in the 5 GHz band measured at one of the laptops for a cable length of 191 cm. (b) Wireless data for 24 channel scan in the 5 GHz band measured at one of the laptops at a distance of 209 cm.

5.1.4 Wireless Data

Finally, the real wireless data includes all the effects that may distort the signal in an indoor environment. With this data, we will evaluate the complete implementation in Section 5.2 and observe how the multipath affects the result in comparison with the output of the previous section data.

The setup to collect the data of this section consists of the already mentioned laptops acting as AP and Client with three external antennas separated at 11 cm, which is around two times the wavelength of the 5 GHz transmissions. The three antennas are aligned and placed in parallel to the other device antennas. For the experimental setup, we measure the distance between the antennas that are placed face to face. The length of the cable joining the connector of the NIC card with the antenna is 168 cm, which will introduce a delay of 8.4 ns at each side. We will take this delay into account when computing the PDP output of the wireless data. Finally, all the data that we will use for the next Section 5.2 is obtained placing the laptops in

Line of Sight.

For the tests, we collect data at several distances separated by 30 cm, specifically at 29 cm, 59 cm, 89 cm, 119 cm, 149 cm, 179 cm and 209 cm. In Figure 5.4b we plot the data for a scan at 209 cm. We can observe the effect of the multipath if we compare it with Figure 5.4a. The data loses the linearity in some of the scans because of the multiple reflections arriving at the same time to the receiver. This effect should not affect the results, as the INDFT algorithm already accounts for the multipath with the appearance of multiple peaks in the Power Delay Profile. If the hardware delays are properly corrected, the first peak of the PDP should still indicate the direct path, and we should only notice the appearance of smaller peaks around the main one.

The evaluation results for the Wireless Data are shown in Section 5.2.4.

5.2 Evaluation Results

In this section, we will show the Power Delay Profile for the data of Section 5.1. We will analyze these results and see how they compare to the ones presented by *Chronos*.

5.2.1 Ideal Data Results

For this data, with no delays or multipath effect, we expect to obtain a single peak at the exact time that links to the distance for which we generated the data. For the distances of 30 cm, 90 cm, 150 cm and 210 cm, we expect to see a single output at 1 ns, 3 ns, 5 ns, and 7 ns. We will express the PDP results in distance units instead of time because it is easier to visualize. The conversion to distance units is as straightforward as multiplying the time by c and adjusting the units to cm. In Figure 5.5 we can see the resulting PDP. Notice that we use a spatial resolution of 30 cm instead of the previously mentioned 23.3 cm. The purpose of this change is to have outputs that coincide with the input values of the data and it is easier to validate the performance of the INDFT algorithm. For the PDPs using wireless data, we use the 23.3 cm spatial resolution. In the case of the cable, as the speed of transmission of the medium is $\frac{2}{3}$ slower, the spatial resolution increases $\frac{2}{3}$ up to 15.5 cm ($\frac{2c=3}{B}$).

With these results, we validate that the implementation of the INDFT algorithm is correct. The ideal data does not require the removal of any hardware delays, and therefore we can isolate the performance of the approximation algorithm for the INDFT explained in Section 4.2.3.

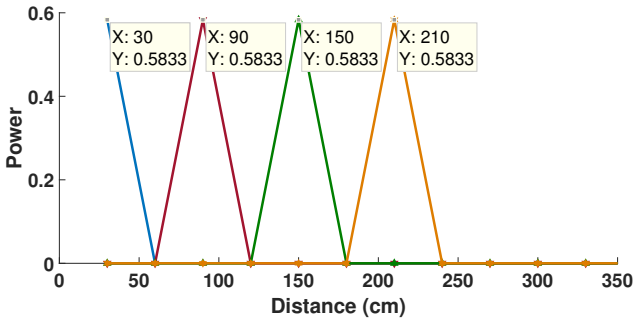


Figure 5.5: Power Delay Pro le for distances of 30 cm, 90 cm, 150 cm and 210 cm.

5.2.2 Ideal Data With Delays Results

In this section, we obtain the PDP results that act as a baseline for the following Section 5.2.3. We generated the data for 191 cm, 341 cm and 491 cm cable lengths with their corresponding delays, the PDD, CFO, and PLL O - set in Section 5.1.2. In Figure 5.6b, 5.6d and 5.6f we can see the expected or ideal output for the real cable data at those distances. In Figure 5.6a, 5.6c and 5.6e we have plotted the PDPs before removing the delays, at one of the transmission ends for its comparison. We can appreciate how the peaks are displaced due to the random delay. Once we multiply the data from both ends removing the delays we obtain the results of Figure 5.6b, 5.6d and 5.6f.

To generate the ideal delays, we have used the Equation (5.1) and Equation (3.10) and (3.11) from Chapter 3. In essence, we have generated the delays on the base of knowing how to eliminate them. In this step we will always obtain a PDP without delays related to hardware. Therefore, we cannot attribute to hardware delays any appreciable e ect or deviation from the expectations in the graphs.

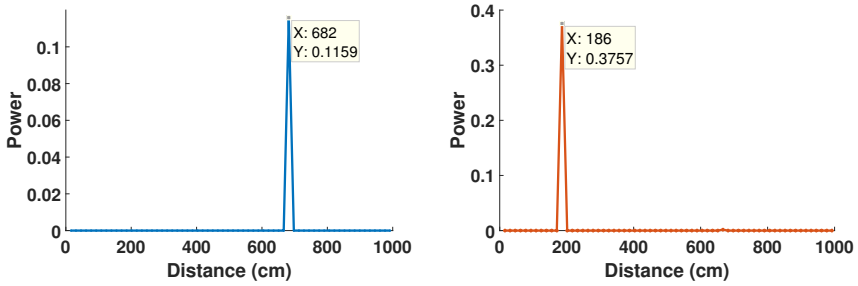
Analyzing the results, we see that we obtain the rst peak at the closest possible solution for 191 cm, 341 cm and 491 cm with this spatial resolution, which is 186 cm, 341 cm and 496 cm respectively.

The only unexpected e ect that we can appreciate in Figure 5.6f is the presence of an extra peak at 992 cm in a multipath-free environment. We will be able to see this e ect at a bigger scale in the following section. This section's data is mathematically generated, so we can already intuit that this e ect is not linked to any physical phenomenon.

5.2.3 Cable Data Results

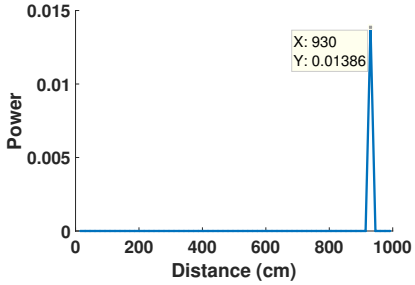
In this section, we show the PDPs for the cable measurements. In Figure 5.7a for 191 cm, in Figure 5.7b for 341 cm and in Figure 5.7c for 491 cm.

Almost all the rst peaks coincide around a single position. We will take

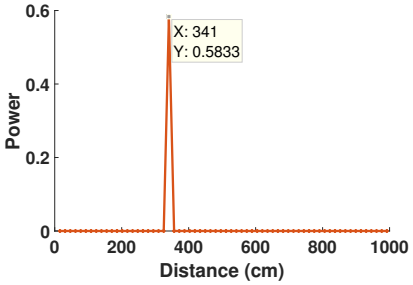


(a) Ideal 191 cm cable PDP w/ delays

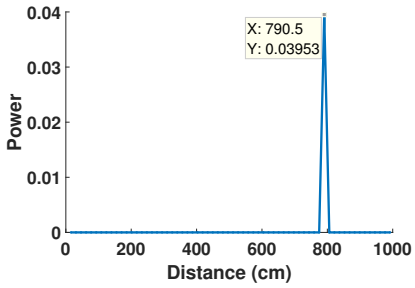
(b) Ideal 191 cm cable PDP



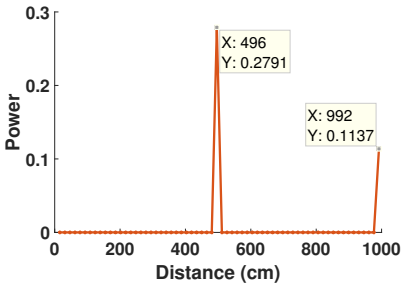
(c) Ideal 341 cm cable PDP w/ delays



(d) Ideal 341 cm cable PDP



(e) Ideal 491 cm cable PDP w/ delays

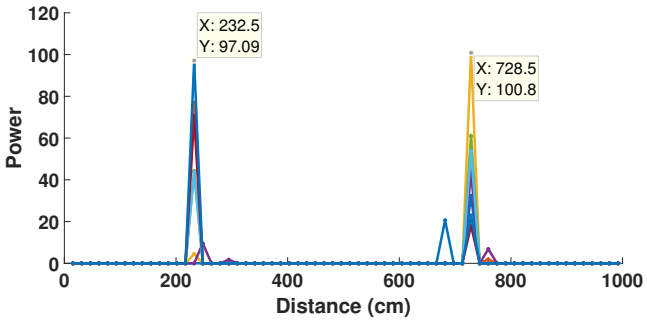


(f) Ideal 491 cm cable PDP

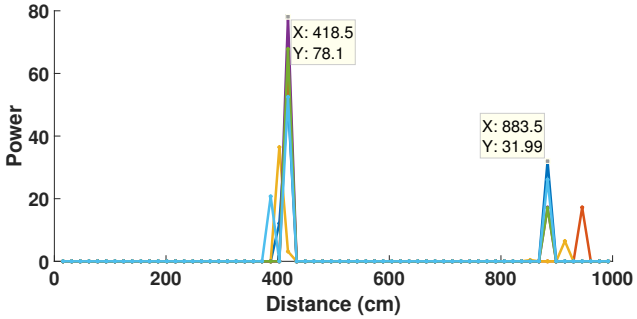
Figure 5.6: **Ideal cable transmission PDP.** Results of using ideally generated data with mathematically generated PDD, CFO and PLL O set delays.(a-b) Ideal 191 cm cable PDP w/ delays and after delay removal. (c-d) Ideal 341 cm cable PDP w/ delays and after delay removal. (e-f) Ideal 491 cm cable PDP w/ delays and after delay removal.

this distance as the average solution. For 191 cm the rst peak is in 232.5 cm, for 341 cm in 418.5 cm and for 491 cm in 573.5 cm. All these results have a signi cant delay from the expected results of Figure 5.6. In Table 5.1, we summarize the results and the delays in time, as well as the error in distance. In blue we compute the time compensation we need to apply to obtain the real distances, while in orange, we compute the delay to the expected PDP output.

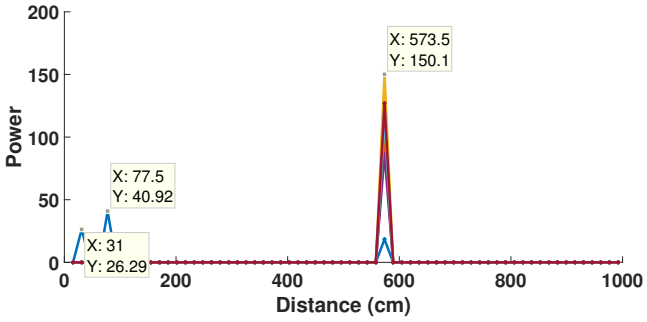
Analyzing these results we can observe that:



(a) Real 191 cm cable PDP



(b) Real 341 cm cable PDP



(c) Real 491 cm cable PDP

Figure 5.7: **Real cable transmission PDP.**Results of using real cable data.(a) Real 191 cm cable data PDP (8 scans). (b) Expected 341 cm cable PDP (7 scans). (c) Expected 491 cm cable PDP (7 scans).

There is an uncompensated hardware delay affecting the results. We can isolate this delay and attribute it to the hardware processing stage because we are working in a multipath-free environment.

The delay is not following a clear linear regression. We obtain the linear regressions in Equation (5.2) and Equation (5.3) for the time

Table 5.1: **Summary of the cable results.** Expected result vs obtained result (in gray). In blue the time error of the result to the real distance. In orange to the expected result.

Real Distance	(cm)	**191**	**341**	**491**
Expected Output	(cm)	186	341	496
PDP Output	(cm)	232.5	418.5	573.5
Error to Real	(cm)	41.5	77.5	82.5
Time Error to Real	(ns)	2.075	3.875	4.125
Error to Expected	(cm)	46.5	77.5	77.5
Time Error to Expected	(ns)	2.325	3.875	3.875

error to the real distance and the expected distance respectively.

$$y = 0.0068333x + 1.02817 \qquad R^2 = 0.83994 \qquad (5.2)$$

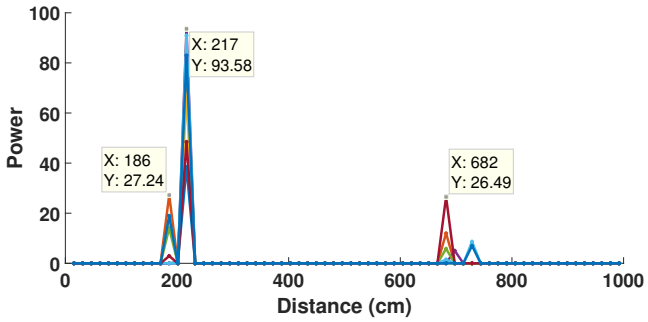
$$y = 0.0051667x + 1.5965 \qquad R^2 = 0.75 \qquad (5.3)$$

The R^2 error under 0.9 indicates that the data does not adjust accurately to a linear t, so we cannot use the Equation (5.2) nor Equation (5.3) to apply an \automatic" compensation of the delay.

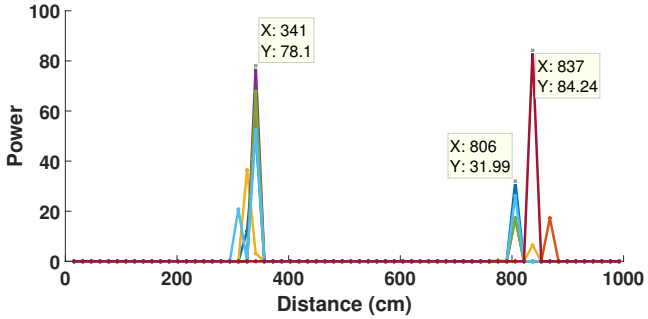
In Figure 5.8a, 5.8b and 5.8c we can see the result of compensating the delay to the real distance (in blue in Table 5.1). We can appreciate that, while for 341 cm we obtain the expected output, for 191 cm we cannot completely remove the delay and for 491 cm we are overcompensating it. In the PDP for 191 cm even if the rst peak is at the expected 186 cm, the highest intensity peak is at 217 cm, so we would need to increase the 2.075 ns compensation. For 491 cm, on the contrary, we get 480.5 cm instead of the expected 496 cm, meaning that the 4.125 ns compensation is excessive.

The reason for this is the spatial resolution. The output indicates the closest possible result given that we can only resolve a 15.5 cm distance. There might be an extra +/- distance that we cannot take into account to apply the compensation because we cannot see it in the PDP.

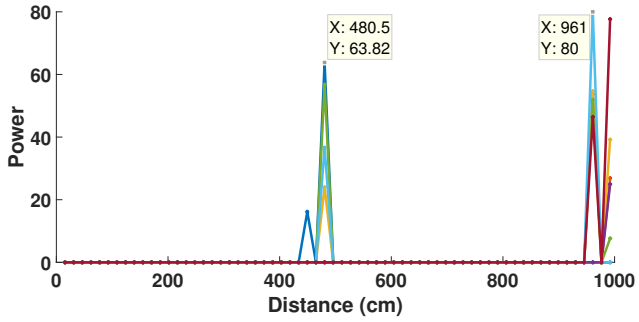
Summarizing, we see that the compensation works, and we obtain almost the expected error that the ideal data indicates. However, we have \manually" applied this compensation assuming that we already know the true distance. We have seen that these delays are not linear, so we cannot use a linear regression to compute the amount of the delay to compensate for any other distance. In the following sections, we will use the same manual compensation technique to try to nd a pattern for the delays in the wireless data.



(a) 191 cm cable PDP with 2.075 ns delay compensation.



(b) 341 cm cable PDP with 3.875 ns delay compensation.



(c) 491 cm cable PDP with 4.125 ns delay compensation.

Figure 5.8: **Cable data PDP with delay compensation to the real distance:** (a) 191 cm cable PDP with 2.075 ns delay compensation. (b) 341 cm cable PDP with 3.875 ns delay compensation. (c) 491 cm cable PDP with 4.125 ns delay compensation.

There is a predominant second peak in some PDPs at more than double the distance from the first one. We know that the multipath is not causing this e ect in the data. We also observed in Figure 5.6f that the same happened for the ideally generated data; so we discard the hardware as the cause too. Therefore, the only possibility left is that the data post-processing algorithm is provoking

the appearance of this peak.

We can see it with the following sequence of ideally generated data. We will use ideal cable data without delays to isolate the INDFT performance. In the first place, we generate the data to coincide with one of the outputs of the time vector. In this case, for the cable, we have a solution every 15.5 cm. Then, we will compute the PDP for a distance that is in between two possible outputs, this is, that has the maximum error. Finally, we will select a distance that is close to the next value that the INDFT can output, but not the exact solution. We can compare these three cases in Figure 5.9a, Figure 5.9b and Figure 5.9c for data generated at 15.5 cm, 23.25 cm and 27 cm.

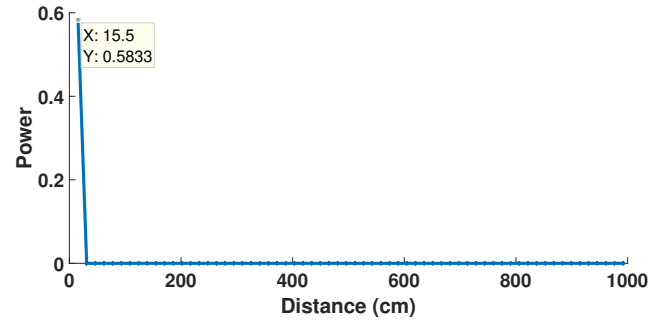
We have the 15.5 cm plot for reference, where there is only a peak coinciding with the exact result. For 23.25, the worst-case scenario, the first peak is still in 15.5 cm but is less powerful than before. Apart from that, we see two more peaks around 500 cm. If we look at 27 cm PDP, the first path has already moved to 31 cm but we still have an extra peak at 527 cm like in Figure 5.9b. We can see that these unexpected peaks appear when the exact solution is not available in the output time vector due to the spatial resolution of the algorithm. Therefore, the first peak will still point out to the closest possible solution but we will notice the presence of other peaks at a further distance that might even be stronger than the first one.

The only factor contributing to these results is the INDFT, as we use ideal data without delays. Therefore, we can assume that this is an effect triggered by using an approximation algorithm to compute the INDFT.

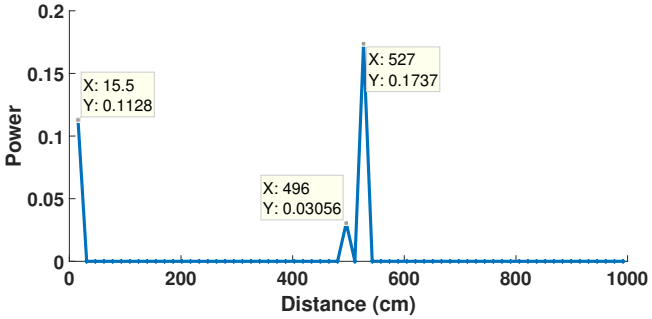
One of the scans of Figure 5.7c has two small first peaks far apart from the average solution. This effect looks like an isolated event for that particular scan. We know it cannot be an influence of the multipath, as we are using cable data. So we can only assume that either it is the data that for some reason in this scan was incorrectly logged, or it is another effect of the INDFT algorithm. However, we observe that for this scan there is also a tiny peak at the same distance (573.5 cm) as the rest of scans, which may suggest that it is an effect of the algorithm. We see that we should choose the solution based on the distance where more first peaks coincide, which in this case is 573.5 cm. In the following section, we will also apply this metric to choose the solution.

5.2.4 Wireless Data Results

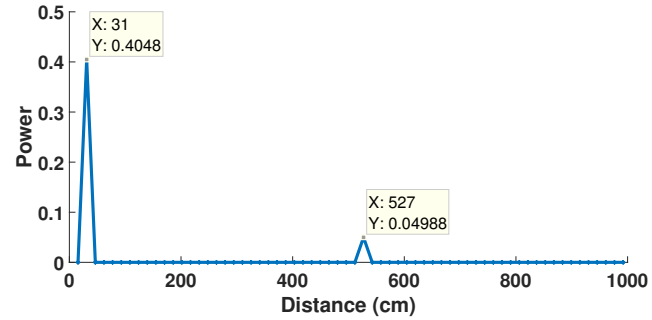
Finally, we are able to analyze the PDP results of the wireless data taking into account what we have seen until this point. We already talked about the



(a) 15.5 cm ideal cable PDP



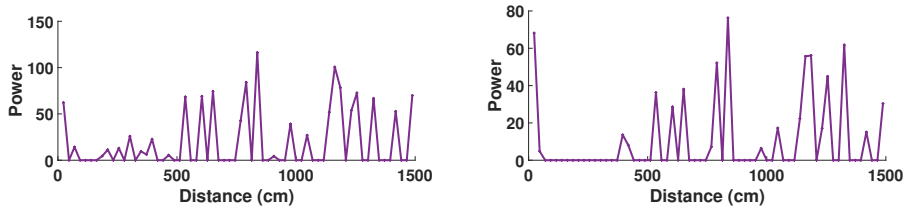
(b) 23.25 cm ideal cable PDP



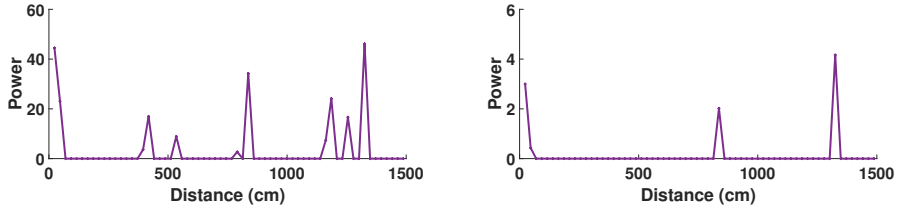
(c) 27 cm ideal cable PDP

Figure 5.9: **The effect of having data that does not coincide exactly with a possible output in the PDP.** (a) PDP output for ideal 15.5 cm cable, coinciding exactly with a possible distance output. (b) PDP output for ideal 23.25 cm cable, value with the highest possible error. (c) PDP output for ideal 27 cm cable, close to the next possible distance output at 31 cm.

setup and the distances at which we collect the data; now we have to discuss the sparsity parameter α for the INDFT algorithm. As explained in Section 4.2.3, depending on the value assigned to this parameter α we will get a result with a higher or lower number of dominant paths. However, it needs



(a) 29 cm PDP with α = 1000. (b) 29 cm PDP with α = 2000.



(c) 29 cm PDP with α = 3000. (d) 29 cm PDP with α = 4000.

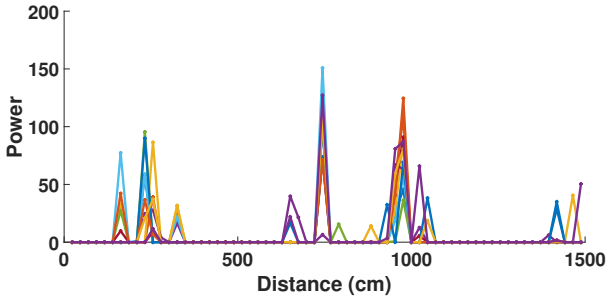
Figure 5.10: **Qualitative effect of different sparsity parameter α for the same wireless data.** The X axis represents the distances travelled by the di erent paths, while the Y axis shows their intensity. (a) 29 cm PDP with α = 1000. (b) 29 cm PDP with α = 2000. (c) 29 cm PDP with α = 3000. (d) 29 cm PDP with α = 4000.

to be adjusted empirically. In Figure 5.10a - 5.10d, we can qualitatively see the e ect of increasing the value of this parameter. The number of paths gets reduced as the α increases. We will not always get the same number of peaks for the same α. In a transmission where the multipath is weak, a high α value will output a low number of peaks. If, on the contrary, the intensity of the re ected signals is high, we will obtain more peaks than before for the same sparsity parameter.

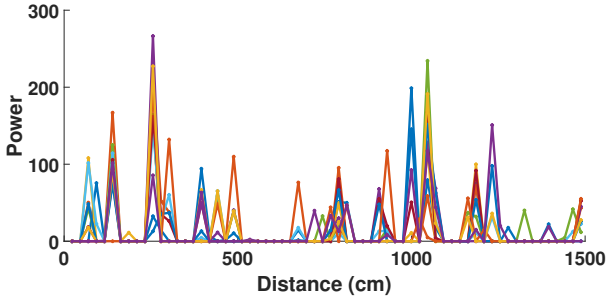
In *Chronos* ve is the average number of predominant paths they obtain after multiple experiments with an unknown α. Following the opposite process, we choose α = 3000 to get an output with approximately ve main peaks in each wireless scan. We have to note that for the previous three sections, as there was no multipath involved, we use a really low α = 10 parameter.

Moving on to the results for the wireless data at distances of 29 cm, 59 cm, 89 cm, 119 cm, 149 cm, 179 cm and 209 cm, we expect to obtain the rst peak of the PDP at 23.3 cm, 69.8 cm, 93 cm, 116.3 cm, 139.5 cm, 186 cm and 209.3 cm. These are the closest possible outputs to the true distance with a spatial resolution of 23.3 cm.

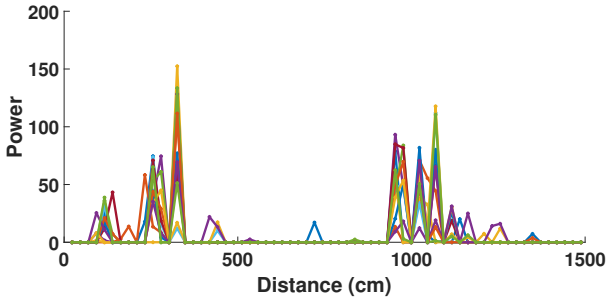
From Figure 5.11a to 5.11g we can observe the PDPs for several scans. As it is not easy to appreciate in the plot where is the rst peak of each scan, the one that determines the time and distance of the direct path, we summarize the most relevant features of the results in Table 5.2.



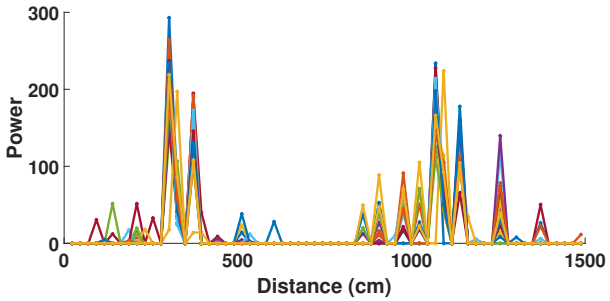
(a) Wireless data PDP at 29 cm



(b) Wireless data PDP at 59 cm

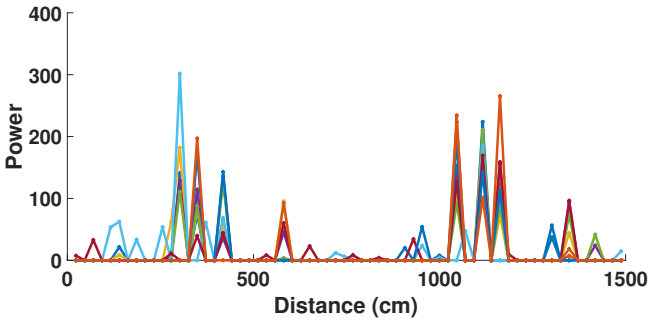


(c) Wireless data PDP at 89 cm

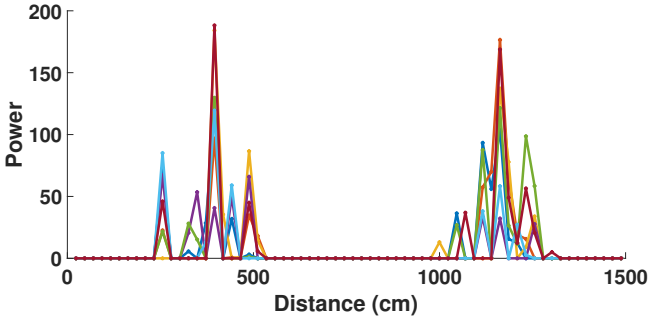


(d) Wireless data PDP at 119 cm

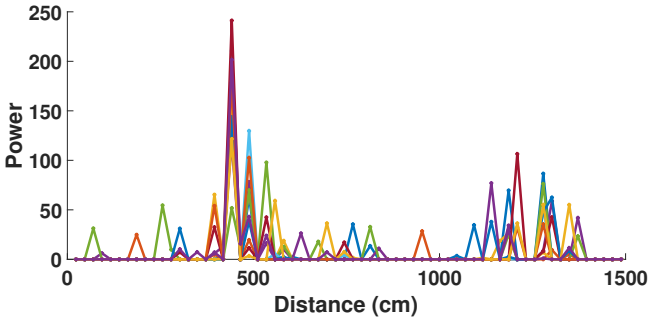
Figure 5.11: **[Part I] PDP for the wireless data at different distances:** (a) For 29 cm. (b) For 59 cm. (c) For 89 cm. (d) For 119 cm.



(e) Wireless data PDP at 149 cm



(f) Wireless data PDP at 179 cm



(g) Wireless data PDP at 209 cm

Figure 5.11: **[Part II] PDP for the wireless data at different distances:** (e) For 149 cm. (f) For 179 cm. (g) For 209 cm.

We can interpret these tables as a summary of the output of a histogram that accounts for the number of times the rst peak appears at a certain distance in each scan. We count the number of times a rst peak appears at a given distance in each scan (in orange). We select the most recurrent one as the solution and compute the error in distance (in red) and in time (in gray) to the true location of the AP. Finally, we also calculate the percentage of appearance of that error among the total number of scans (in blue).

For example, for the results of 29 cm in Table 5.2a we expect a result of

23.3 cm shown in green. Adding all the first peak appearances (in orange), we know we performed a total of 11 scans, where the most repeated output was 162.8 cm with 5 appearances, followed by 232.5 cm with 3. Therefore, we take 162.8 cm as the output for this distance. We can see that for the expected output, there was not even a single result. Then, we show in red the error in cm of our output to the true 29 cm distance, and in gray the time delay in ns that this error implies which are 133.8 cm and 4.46 ns respectively. Finally, in blue we show the percentage of that solution, this is, of that error, among the total number of scans. In the other columns we can see the results for the other distances that are not the selected output.

For the range of 0m to 2m, *Chronos* achieves a median distance error below 10 cm and a median time error under the 0.3 ns for measurements in LoS. In our case, looking at the expected outputs and the true distances, the minimum theoretical error we can get is 5.7 cm, 10.8 cm, 4 cm, 2.7 cm, 9.5 cm, 7 cm and 0.3 cm for each distance from 29 cm to 209 cm respectively. We can see that only for the distance of 59 cm (10.8 cm error) we expect to have an error above the threshold that *Chronos* results indicate. For any distance, we can compute the worst-case error as half of the spatial resolution, this will happen with a location that is just in the middle of two values that we can resolve. For example, in our case, as we have outputs every 23.3 cm, 11.65 cm is our expected worst-case error.

As we could expect from the results of Section 5.2.3, there is a delay in almost all the PDPs. In cases like 29 cm, 119 cm and 149 cm, the error is higher than the distance itself, 133.8 cm, 183.3 cm and 153.3 cm respectively. As before, these delays do not fit a linear regression.

Summarizing, we can conclude about this delay of unknown origin that:

It can be too large to be acceptable for an indoor localization environment.

It is neither constant nor can be adjusted in a linear regression, or follows a clear relation with the true location of the AP.

The linear fit of the cable delay does not help to determine its quantity.

5.3 The New Delay

The main objective of this section is to search for a pattern in the delay found in the previous section. We will try to find this pattern by observing the outcomes of two different delay compensations that we will apply, also based on the results of Section 5.2.4. First, we will check if the outcome of the delay removing methods improves the poor results of Table 5.2. If that is the case, we will check if the applied time compensation has a bound or follows a regression.

Table 5.2: [**Part I**] **PDP analysis of the first peak in order of appearance.** These tables summarize the histogram representing the appearances as the rst peak of the possible output distances. The rst row shows the distances at which we nd the rst peak. In green the expected output. The second row indicates the number of times that the rst peak appears at the given distance. In orange the distance with more appearances. The third row shows the distance error in cm to the true distance of the device. In red the error in cm of the distance with more appearances. In the fourth row, we compute the percentage of appearance of that distance over the total number of scans. In blue the percentage of appearance of the output with more presences. The last row shows the time error in ns of that distance. In gray the time error in ns to compensate for the peak with more appearances.

(a) 29 cm **PDP analysis.**

Distance (cm)	23.3	162.8	209.3	232.5	255.8
1^{st} Peak Appearances	0	5	1	3	2
Error (cm)	-5.7	133.8	180.3	203.5	226.8
%Error Appearance	0	45.5	9.1	27.3	18.2
Time Error (ns)	0.19	4.46	6.01	6.783	7.56

(b) 59 cm **PDP analysis.**

Distance (cm)	69.8	93	116.3	139.5	255.8
1^{st} Peak Appearances	7	1	1	1	1
Error (cm)	10.8	34	57.3	80.5	196.8
%Error Appearance	63.6	9.1	9.1	9.1	9.1
Time Error (ns)	0.36	1.133	1.91	2.683	6.56

(c) 89 cm **PDP analysis.**

Distance (cm)	93	116.3	162.8
1^{st} Peak Appearances	4	7	1
Error (cm)	4	27.3	73.8
%Error Appearance	33.3	58.3	8.3
Time Error (ns)	0.133	0.91	2.46

We perform a delay compensation based on the results shown in Table 5.2 and another one looking at Figure 5.11. The rst compensation corresponds to the time error depicted in the tables (in gray) for the rst peak with more appearances. The second approach compensates the delay of what we will call the *main peak* to the true distance. We de ne the main peak as the rst output of the PDP that presents an intensity signi cantly higher than

Table 5.2: **[Part II] PDP analysis of the first peak in order of appearance.**

(d) 119 cm **PDP analysis.**

Distance (cm)	93	116.3	139.5	186	209.3	302.3
1^{st} Peak Appearances	1	2	1	2	4	7
Error (cm)	-26	-2.7	20.5	67	90.3	183.3
%Error Appearance	5.9	11.8	5.9	11.8	23.5	41.2
Time Error (ns)	0.867	0.09	0.683	2.233	3.01	6.11

(e) 149 cm **PDP analysis.**

Distance (cm)	23.3	116.3	139.5	186	279	302.3	348.8
1^{st} Peak Appear.	1	3	3	2	1	5	2
Error (cm)	-125.7	-32.7	-9.5	37	130	153.3	199.8
%Error Appear.	5.9	17.6	17.6	11.8	5.9	29.4	11.8
Time Error (ns)	4.19	1.09	0.317	1.233	4.333	5.11	6.66

(f) 189 cm **PDP analysis.**

Distance (cm)	116.3	139.5	186	255.8	302.3	348.8	395.3
1^{st} Peak Appear.	2	1	2	6	4	1	1
Error (cm)	-62.7	-39.5	7	76.8	123.3	169.8	216.3
%Error Appear.	11.8	5.9	11.8	35.3	23.5	5.9	5.9
Time Error (ns)	2.09	1.317	0.233	2.56	4.11	5.66	7.21

(g) 209 cm **PDP analysis.**

Dist.	69.8	93	116.3	139.5	186	209.3	302.3	395.3	418.5	441.8
1^{st} P.	1	1	1	1	2	0	6	2	2	1
Err.	-139.2	-116	-92.7	-69.5	-23	0.3	93.3	186.3	209.5	232.8
%Er.	5.9	5.9	5.9	5.9	11.8	0	35.3	11.8	11.8	5.9
T Er.	4.64	3.867	3.09	2.317	0.767	0.01	3.11	6.21	6.983	7.76

the rst peak. Therefore, we choose the main peak based on the intensity that they show in the graph, on two conditions. First, the peak needs to be in the vicinity of the main peak in Table 5.2. If the outputs are far from the rst peak, it could be a result of using CSI^2 as input data (Equation (3.15)), or the e ect of the algorithm like in Figure 5.9. Second, more than one scan must have a solution at that distance. A single and solitary peak can be a result of an isolated case where the data of the scan does not follow the behavior of the rest, as it was the case of the small peaks in Figure 5.7c. For example, for the distance of 89 cm the histogram outputs a

result of 116.3 cm, and if we take a look at the PDP in Figure 5.11c we can appreciate several small peaks gathering around that position. However, there is a peak at 325.5 cm that stands out above all the surrounding ones. In this case, we choose this distance because it is close to the first peak solution, and multiple scans show a peak in this position, so it satisfies both conditions.

The purpose of this compensation is to ignore the smaller peaks that might remain even after choosing a high sparsity parameter and focus on a peak that has a bigger chance of being the direct path in LoS because of its higher intensity.

We will start analyzing the results of applying the first approach where we compensate the delay of the first peak in Table 5.3. We compare them with the original results in Table 5.2 (now in in orange). We use the same metric as in the previous section to evaluate the result; this is, choose the output with more appearances as the first peak.

In general terms, we can appreciate a noticeable reduction of the error. In fact, in cases where the result was especially inaccurate, like 29 cm or 149 cm, there is an error reduction of more than 1m. We have three distances with an error below the 10 cm. In turn, we observe that the lowest error of the previous results, for 59 cm, is now more than 20 cm higher than before. For the distances of 119 cm, 179 cm and 209 cm we see in the tables more than one result for the error. In these cases, multiple distances have the same number of appearances as the first peak in the histogram, and each of these presents a different error to the true distance that is shown in the table. We will refer to them as *possible solutions*. All three distances, 119 cm, 179 cm and 209 cm , have one *possible solution* under the 10 cm error threshold.

The reason for the appearance of multiple possible solutions comes from the fact that the peaks will not be moving as a whole when we apply the time compensation. We have already seen in Figure 5.9 that the outputs will fit the closest possible output upwards or downwards. When we shift the data to compensate the delay, some of the peaks that gathered around the same output might not end up now in the vicinity of the same result. Therefore, after the delay compensation, we cannot expect to have the same first peak appearances distribution as before but with a distance shift. As a result, a distance can end up having multiple *possible solutions*.

If we now observe the time compensation applied to obtain these results, in gray in Table 5.3, we see that the values are quite scattered. It is not possible to fit them in a linear regression. Even if we have seen a significant improvement in the results, we would need a non-linear approach to find a general and scalable solution to compute the delay for all distances with this method.

In Table 5.4 we see the results for the second delay compensation method. Here we choose the main peak for each plot of Figure 5.11 based on the two conditions mentioned before and compensate the delay to the true distance.

Table 5.3: **Results for delay compensation using the values of Table 5.2a-5.2g.** In green the error in cm after the application of this method. In orange the error in cm of the comparative method in Table 5.2a-5.2g.

Real Distance (cm)	**29**	**59**	**89**	**119**	**149**	**179**	**209**
Time compensation (ns)	4.46	0.36	0.91	6.11	5.11	2.56	3.11
1^{st} Peak Distance (cm)	23.3	93	93	23.3 116.3	139.5	46.5 116.3 186	116.3 209.3 302.3
1^{st} Peak Appearances	10 (11)	5 (11)	7 (12)	6 (17)	6 (17)	3 (17)	3 (17)
%Appearances	90.9	45.5	58.3	35.3	35.3	17.7	17.7
Error (cm)	-5.7	34	4	-95.7 -2.7	-9.5	-142.5 -72.7 -3	-92.7 0.3 93.3
Previous Error (cm)	133.8	10.8	27.3	183.3	153.3	76.8	93.3

In the second row of this table, we indicate the distance of the main peak in each case. We compare the results with the ones in Table 5.3 (in orange).

In general terms, the results are better than for the first method. We improve the output for the scans at 29 cm, 149 cm and 209 cm significantly, reducing the error around 100 cm. Five distances are now close to or under the error threshold of 10 cm marked by *Chronos*. Only two other distances are still over the 90 cm error.

Overall, we can conclude that this method works better than using the histogram results. However, again it is not possible to fit the time compensation applied in this method in a linear regression, we can only appreciate that the delay is bounded between 5 ns and 8 ns.

In a final approach, we will see if there is a chance to obtain deterministic error results; this is, only one output per distance. We will not try to find a pattern for the delay, as it is noticeable its non-linear nature. We will also try to improve the results of Table 5.4 and get them closer to what *Chronos* reports. To do so, we will modify the sparsity parameter α maintaining the same delay compensation of Table 5.4, the time compensation of the second method. This approach will cause the reduction or disappearance of the smaller peaks. As a result, the appearances in the histogram of the most powerful ones will increase, as those are the only ones left. In other words, now the strongest peaks will determine the error to the true distance.

Table 5.4: **Delay compensation using the PDP graph.** In green the error in cm after the application of this method. In orange the error in cm of the comparative method in Table 5.2a-5.2g.

Real Distance (cm)	**29**	**59**	**89**	**119**	**149**	**179**	**209**
Distance of the main peak (cm)	232.5	255.8	325.5	302.3	302.3	395.3	441.8
Time compensation (ns)	6.78	6.56	7.88	6.11	5.11	7.21	7.76
1^{st} Peak Distance (cm)	23.3	46.5	23.3 93	23.3	139.5	46.5	209.3
1^{st} Peak Appearances	9 (11)	11 (11)	5 (12)	6 (17)	6 (17)	7 (17)	4 (17)
%Appearances	81	100	41.6	35.3	35.3	41.2	23.5
Error (cm)	-5.7	-12.5	-65.5 4	-95.7	-9.5	-132.5	0.3
Previous Error (cm)	133.8	10.8	27.3	183.3	153.3	76.8	93.3

We can increase α until a certain amount. All scans should have at least one peak that indicates the path of the incoming signal. The more α increases, the less peaks we get in the PDP. The first scan that ends up with a single peak delimits the threshold of the sparsity parameter. In this case, α = 4800. In Table 5.5 we compare the results of the different parameters, in green for α = 4800, in orange for α = 3000. Overall, the distance error improves and gets closer to what we can consider acceptable for indoor environments. For 29 cm, 59 cm and 209 cm we maintain the good results, for 119 cm and 179 cm we reduce the error almost 70 cm, and for 89 cm we obtain a result that lies between the two possible solutions of 4 cm and 65.5 cm error. We find the only drawback for 149 cm, where we add an error of 23.2 cm.

In this section, we have confirmed the existence of a delay that is not constant and does not adjust to a linear fit. We have tried two different approaches to remove the error that appears in Table 5.2. Although both of them improve the accuracy considerably, even reducing all errors below 65 cm and most of them below 30 cm, we have seen that the time compensation that they apply is non-linear, and we cannot extend this hardcoded approach to any other distance. Therefore, it will be necessary to know the origin of this non-linear delay as *Chronos* does for the PDD, CFO, and PLL

Table 5.5: **Delay compensation using the PDP graphs with α = 4800 maintaining the time compensation of Table 5.4.** In green the error in cm after the application of this method. In orange the error in cm of the comparative method in Table 5.4 for α = 3000. Finally, the last row shows the results for the wireless data with no delay compensation of Table Table 5.2a-5.2g.

Real Distance (cm)	**29**	**59**	**89**	**119**	**149**	**179**	**209**
Time compensation (ns)	6.78	6.56	7.88	6.11	5.11	7.21	7.76
1^{st} Peak Distance (cm)	23.3	46.5	116.3	93	116.3	116.3	209.3
1^{st} Peak Appearances	5 (11)	10 (11)	11 (12)	13 (17)	7 (17)	9 (17)	11 (17)
%Appearances	45.5	90.9	94.1	75	41.2	52.9	64.7
Error (cm)	-5.7	-12.5	27.3	-26	-32.7	-62.7	0.3
Previous Error (cm)	-5.7	-12.5	-65.54	-95.7	-9.5	-132.5	0.3
Original Error (cm)	133.8	10.8	27.3	183.3	153.3	76.8	93.3

O set to remove it accordingly and make it scalable for all distances.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work, we have analyzed the ranging method of the paper *Chronos: Decimeter-Level Localization with a Single WiFi Access Point*. It is one of the research papers with a signi cant impact in the past few years on the topic of indoor localization using the Channel State Information with a reduced number of Access Points. In the course of this Thesis, we have encountered di erent di culties relative to both the hardware installation and software implementation required to extract a critical parameter in WiFi communications like CSI. We have obtained results that di er from our expectations and reached some conclusions that can be summarized as follows:

- Implementation -

We tried to use a di erent chipset for the implementation, instead of the IWL5300 like in *Chronos*, to see if there was a possibility of using real APs. We found out that there are issues with the new chipsets of Qualcomm Atheros, that prevent us from extracting the CSI from the physical layer.

We developed a similar program to *Chronos* for the channel hopping procedure, except that our system does not work at the kernel level but in the userspace. We have performed all the measurements keeping both AP and Client static to reduce the constraint of the channel coherence time. We found that the communication was only working correctly for the 5 GHz band channels, and accordingly, we adjusted the spatial resolution of the INDFT to the bandwidth reduction.

On the side of the data processing algorithm, we discovered that the INDFT algorithm had an error that was preventing it from conver-

ging to a solution. In Section 4.2.3, we modified the necessary step size parameter to achieve the convergence.

- Evaluation -

After implementing the channel hopping protocol, we have tested the system with different experiments. We have taken as reference the mathematically generated ideal data plus the added delays explained in *Chronos*. In addition to this, we have performed a test with a cable transmission to obtain data without multipath, and finally, the complete system validation for wireless data. Thereby, we have gathered the following conclusions:

We have validated that the INDFT algorithm outputs the closest solution to the real distance as its first peak. However, we also found that we can see extra peaks appearing at a further distance that do not correspond to a path that the signal has traversed. The limited spatial resolution and the fact that the INDFT algorithm is an approximation algorithm are the cause of this effect.

With the cable experiment, we confirmed the presence of an extra uncompensated delay. The hardware is the source of this delay, as the signal is not affected by anything else in a wired transmission. We found that this delay is not constant and it does not adjust well to a linear regression.

With the wireless data, we corroborated the presence of the non-linear delay. We have experienced a median error of 96.9 cm. For some distances, this deviation gets close to two meters, while for others, we obtain results near the 10 cm threshold that *Chronos* determines.

Finally, we have applied different delay compensation methods that allow us to reduce the median error around 50 cm. However, none of these methods is applicable to compute the time compensation extensible for all distances in a generic way. We could observe there is not a pattern in the delay, it is not linear or quantifiable, and therefore it is necessary to know its origin to find a scalable method to remove it.

6.2 Future Work

Indoor localization is a well-known topic, studied from a varied range of perspectives and methods. Perhaps, this last approach of using a physical parameter and mathematical techniques to obtain the distance between two devices may have a future due to its robustness. For that, there are a couple of aspects that we can propose as future research to improve this approach

and make the most of it so that it can become a mainstream method for the indoor localization in the coming years:

The hardware delays: An important topic that the papers about indoor localization using CSI research is the e ect of the signal processing stage on the nal result. Using hardware that is not designed to obtain precise phase and amplitude measurements can lead to errors. A small delay in time can be translated to a considerable error in distance for indoor environments. In this work, we have detected the presence of a non-linear delay that can cause an error greater than one meter in some cases. Finding the source of this delay and achieving a method that systematically removes it can be a research topic to take into consideration.

The dedicated software to extract the CSI: Nowadays we know about two options to extract the CSI which are the Atheros and the IWL5300 chipset tool. However, if we want to extend this indoor localization technique, we need to have more hardware choices. Almost all current WiFi chipsets can compute the CSI, as IEEE802.11n is widespread. It is necessary to research the possibility of developing software that allows the user to extract that information from more than only those two pieces of hardware.

Bibliography

[1] Paramvir Bahl, Venkata N Padmanabhan and Anand Balachandran. `Enhancements to the RADAR user location and tracking system'. In: *Microsoft Research* 2.MSR-TR-2000-12 (2000), pp. 775{784.

[2] Paramvir Bahl, Venkata N Padmanabhan et al. `RADAR: An in-building RF-based user location and tracking system'. In: *IEEE infocom*. Vol. 2. 2000. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE). 2000, pp. 775{784.

[3] Dean Banerjee. *PLL Performance, Simulation and Design*. URL: http://www.ti.com/lit/ug/snaa106c/snaa106c.pdf (visited on 25/11/2018).

[4] Ernesto G Birgin, Jose Mario Mart nez, Marcos Raydan et al. `Spectral projected gradient methods: review and perspectives'. In: *J. Stat. Softw* 60.3 (2014), pp. 1{21.

[5] Mauro Brunato and Roberto Battiti. `Statistical learning theory for location ngerprinting in wireless LANs'. In: *Computer Networks* 47.6 (2005), pp. 825{845.

[6] Qasim Chaudhari. *Wireless Communications From the Ground Up. What is Carrier Frequency O set (CFO) and How It Distorts the Rx Symbols*. URL: https://wirelesspi.com/what-is-carrier-frequency-offset-cfo-and-how-it-distorts-the-rx-symbols/ (visited on 25/11/2018).

[7] Hao Chen et al. `ConFi: Convolutional neural networks based indoor wi- localization using channel state information'. In: *IEEE Access* 5 (2017), pp. 18066{18074.

[8] Zhenghua Chen et al. `Fusion of WiFi, smartphone sensors and landmarks using the Kalman lter for indoor localization'. In: *Sensors* 15.1 (2015), pp. 715{732.

[9] OpenWRT Community. *OpenWRT Repository*. URL: https://github.com/openwrt/openwrt (visited on 25/03/2018).

[10] OpenWRT Community. *OpenWRT Wireless Freedom*. URL: https://openwrt.org/docs/start (visited on 25/03/2018).

[11] Technical University of Denmark (DTU). *Lecture on Algorithms for large-scale convex optimization: Proximal gradient method*. URL: https://people.eecs.berkeley.edu/~elghaoui/Teaching/EE227A/lecture18.pdf (visited on 28/12/2018).

[12] Frederic Evennou and Francois Marx. `Advanced integration of WiFi and inertial navigation systems for indoor mobile positioning'. In: *Eurasip journal on applied signal processing* 2006 (2006), pp. 164{164.

[13] My Digital Life Forums. *BIOS Mods*. URL: https://forums.mydigitallife.net/ (visited on 03/05/2018).

[14] Patrick R Gill, Albert Wang and Alyosha Molnar. `The in-crowd algorithm for fast basis pursuit denoising'. In: *IEEE Transactions on Signal Processing* 59.10 (2011), pp. 4595{4605.

[15] Daniel Halperin. *iwlagn rate n ags bit elds*. URL: https://github.com/dhalperi/linux-80211n-csitool/blob/csitool-3.13/drivers/net/wireless/iwlwifi/dvm/commands.h#L245-L334 (visited on 01/12/2018).

[16] Daniel Halperin. *Linux 802.11n CSI Tool. Frequently Asked Questions*. URL: https://dhalperi.github.io/linux-80211n-csitool/faq.html (visited on 16/12/2018).

[17] Daniel Halperin. *Linux 802.11n CSI Tool. Installation Instructions*. URL: https://dhalperi.github.io/linux-80211n-csitool/installation.htm (visited on 28/09/2018).

[18] Daniel Halperin et al. `Tool Release: Gathering 802.11n Traces with Channel State Information'. In: *ACM SIGCOMM CCR* 41.1 (2011), p. 53. URL: http://www.halper.in/pubs/halperin_csitool.pdf.

[19] Society for Industrial and Applied Mathematics. *Chapter 10: The Proximal Gradient Descent Method*. URL: https://my.siam.org/books/mo25/mo25_ch10.pdf (visited on 28/12/2018).

[20] Manikanta Kotaru et al. `Spot : Decimeter level localization using wi '. In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 269{282.

[21] Lenovo. *BIOS update utility - ThinkPad R60, R60i (Types: 94xx)*. URL: https://support.lenovo.com/nl/es/downloads/ds001506 (visited on 03/05/2018).

[22] Alex T Mariakakis et al. `Sail: Single access point-based indoor localization'. In: *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM. 2014, pp. 315{328.

[23] MathWorks. *spline Documentation*. URL: https://nl.mathworks.com/help/matlab/ref/spline.html (visited on 20/12/2018).

[24] Wolfram MathWorld. *Gradient*. URL: http://mathworld.wolfram.com/Gradient.html (visited on 28/12/2018).

[25] *MCS.Constellations and Modulation*. URL: http://ecee.colorado.edu/~liue/teaching/comm_standards/2015S_IEEE_802.11n/Webpages/constellation.html (visited on 01/12/2018).

[26] OpenWRT. *OpenWRT Table of Supported Hardware*. URL: https://openwrt.org/toh/start (visited on 25/03/2018).

[27] OpenWRT.org. *TP-Link TL-WA901ND v5*. URL: https://openwrt.org/toh/hwdata/tp-link/tp-link_tl-wa901nd_v5 (visited on 25/03/2018).

[28] Ian Poole. *OFDM Orthogonal Frequency Division Multiplexing Tutorial*. URL: https://www.radio-electronics.com/info/rf-technology-design/ofdm/ofdm-basics-tutorial.php (visited on 25/11/2018).

[29] Ian Poole. *Wi-Fi / WLAN Channels, Frequencies, Bands and Bandwidths*. URL: https://www.radio-electronics.com/info/wireless/wi-fi/80211-channels-number-frequencies-bandwidth.php (visited on 25/11/2018).

[30] Souvik Sen et al. `Avoiding multipath to revive inbuilding WiFi localization'. In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM. 2013, pp. 249{262.

[31] Yuxiang Sun, Ming Liu and Max Q-H Meng. `WiFi signal strength-based robot indoor localization'. In: *2014 IEEE International Conference on Information and Automation (ICIA)*. IEEE. 2014, pp. 250{256.

[32] ThinkWiki. *Problem with unauthorized MiniPCI network card*. URL: https://www.thinkwiki.org/wiki/Problem_with_unauthorized_MiniPCI_network_card (visited on 03/05/2018).

[33] Ryan Tibshirani. *Lecture 8 on Convex Optimization*. URL: http://www.stat.cmu.edu/~ryantibs/convexopt-S15/scribes/08-prox-grad-scribed.pdf (visited on 28/12/2018).

[34] Ubuntu. *Ubuntu 14.04.2 LTS (Trusty Tahr)*. URL: http://old-releases.ubuntu.com/releases/14.04.3/ (visited on 03/05/2018).

[35] Ubuntu. *Ubuntu Kernel Support and Schedules*. URL: https://wiki.ubuntu.com/Kernel/Support (visited on 01/12/2018).

[36] Deepak Vasisht, Swarun Kumar and Dina Katabi. `Decimeter-Level Localization with a Single WiFi Access Point.' In: *NSDI*. Vol. 16. 2016, pp. 165{178.

[37] Xuyu Wang, Xiangyu Wang and Shiwen Mao. `CiFi: Deep convolutional neural networks for indoor localization with 5 GHz Wi-Fi'. In: *2017 IEEE International Conference on Communications (ICC)*. IEEE. 2017, pp. 1{6.

[38] Xuyu Wang et al. `DeepFi: Deep learning for indoor ngerprinting using channel state information'. In: *2015 IEEE wireless communications and networking conference (WCNC)*. IEEE. 2015, pp. 1666{1671.

[39] Wikipedia. *IEEE 802.11n-2009. Data rates.* URL: https://en.wikipedia.org/wiki/IEEE_802.11n-2009#Data_rates (visited on 01/12/2018).

[40] Kaishun Wu et al. `CSI-based indoor localization'. In: *IEEE Transactions on Parallel and Distributed Systems* 24.7 (2013), pp. 1300{1309.

[41] Yaxiong Xie. *Accepted Hardware for the Atheros CSI Tool*. URL: http://wands.sg/research/wifi/AtherosCSI/Hardware.html (visited on 25/03/2018).

[42] Yaxiong Xie. *Atheros CSI Tool*. URL: http://wands.sg/research/wifi/AtherosCSI/ (visited on 25/03/2018).

[43] Yaxiong Xie. *Atheros CSI Tool Functioning Instructions*. URL: https://github.com/xieyaxiongfly/Atheros_CSI_tool_OpenWRT_src/wiki/Collect-CSI (visited on 25/03/2018).

[44] Yaxiong Xie. *OpenWRT Image Generation Instructions*. URL: https://github.com/xieyaxiongfly/Atheros_CSI_tool_OpenWRT_src/wiki/Install-OpenWRT-version-of-Atheros-CSI-tool (visited on 25/03/2018).

[45] Yaxiong Xie. *OpenWRT images with the Atheros CSI Tool*. URL: https://github.com/xieyaxiongfly/OpenWRT_firmware (visited on 25/03/2018).

[46] Yaxiong Xie, Zhenjiang Li and Mo Li. `Precise power delay pro ling with commodity Wi-Fi'. In: *IEEE Transactions on Mobile Computing* (2018).

[47] Jie Xiong and Kyle Jamieson. `Arraytrack: A ne-grained indoor location system'. In: *Presented as part of the 10th fUSENIXg Symposium on Networked Systems Design and Implementation (fNSDIg 13)*. 2013, pp. 71{84.

[48] Jie Xiong, Karthikeyan Sundaresan and Kyle Jamieson. `ToneTrack: Leveraging frequency-agile radios for time-based indoor wireless localization'. In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM. 2015, pp. 537{549.

[49] Ryota Yamasaki et al. `TDOA location system for IEEE 802.11 b WLAN'. In: *IEEE Wireless Communications and Networking Conference, 2005*. Vol. 4. IEEE. 2005, pp. 2338{2343.

[50] Xiuyan Zhu and Yuan Feng. `RSSI-based algorithm for indoor localization'. In: *Communications and Network* 5.02 (2013), p. 37.

Appendix A

Atheros CSI Tool

In this appendix, we will show the process that we followed for the installation and setup of the Atheros CSI tool [42] and the modi cations that we made to try to make it work. We will also explain the reasons why we deduce the tool is not working correctly for our hardware.

A.1 Tool Introduction

The Atheros CSI Tool was developed by Yaxiong Xie and presented in [46] to extract the CSI data from Atheros wireless chipsets. This tool has two variants, one for wireless devices like Access Points and another one for NIC cards in computers, both based on Atheros manufacturer chipsets. Depending on the choice, the software is slightly di erent. For APs, the tool works on top of the OpenWRT Operating System, an OS for this type of wireless devices. In the case of NIC cards for computers, as the IWL5300, it works for Linux OS.

We decided to get two AP and try the approach of the tool for OpenWRT. Most recent literature use laptops as APs ([20] [36] with the IWL5300 and [30] [22] with Atheros chipsets), so our aim was to re-implement the same system as *Chronos* with a di erent hardware. It could prove that the method works regardless of the hardware, and have a system close to what a real WiFi network looks like with actual APs.

A.2 Hardware Selection

The hardware needs to ful ll two requirements to be eligible for the installation of the tool:

> The wireless chipset must be from Atheros and preferably supported and previously tested by the tool. We can nd a list of tested chipsets in [41].

The AP must be able to run with OpenWRT OS. We can find a list of supported APs in [26].

The only device that we found still in the market fulfilling both requirements was the TP-Link TP-LINK WA901ND (v.5.0). According to [27] the main characteristics of this device are:

Atheros AR9380 chipset.

4Mb Flash/ 32Mb RAM.

TFTP server.

3x3 MIMO antennas.

A.3 OpenWRT Software Tool

This tool takes the open source software of OpenWRT operating system and adds modifications to extract the CSI from the physical layer and pipe it upwards to the userspace so that the user can log it. The transmitter needs the patched version of OpenWRT to be able to send IEEE802.11n HT packets that the receiver will use to compute the CSI.

This tool introduces a structure of modifications at driver level (`ath9k`), which are the following:

A patch to wireless drivers that allows recording the raw CSI data from incoming packets at the transmitter.

A patch to wireless drivers to modify all outgoing packets into HT packets.

A new kernel module (called `ar9003_csi`) that can access the raw CSI data and parse it.

Also, a series of additions at the equivalent userspace level of the AP:

A userspace program to log the parsed CSI data piped from the physical layer.

A userspace program to send HT packets at the transmitter.

After flashing the modified OpenWRT image, we can SSH into the AP and run the programs to send and log the received packets at the transmitter and receiver respectively. In [10] we can find all the information to establish a connection with the device once the OpenWRT OS is running.

A.4 Problems Found

We encountered two main problems, the first one is related to the OpenWRT image and flashing it in the AP. The second issue is related to the functioning of the logging tool itself.

A.4.1 Image Flashing

Yaxiong Xie provides a list of OS images in [45] with the Atheros Tool. However, there is no image for WA901ND v5. We need to generate our own by compiling the source code as explained in [44]. The problem is that the OpenWRT code of the repository is not the so-called \tiny distribution", build to work in 4MB flash memory devices like ours. *If we compile the image with the repository code, the AP will not boot correctly*.

The only way we found to build a valid image for the AP is to merge the additions of the tool with the \tiny distribution" of the OS at [9]. We have to make sure that we include all driver modifications and the userspace app to send packets and log the CSI. A code comparison software can help to locate all files to merge. Back in [44], we need to select in the OpwnWRT Configuration window `Subtarget(tiny)`. We should follow the rest of the instructions and compile to generate a valid image.

Finally, to flash the image, I recommend the option explained in the TL-WA901ND v4 support forum (`https://forum.archive.openwrt.org/viewtopic.php?id=60117&p=3`) of using a TFTP server as a back door. It is the less intrusive option as we need just a TFTP server at our computer with IP 198.168.0.66 with a file called `wa901ndv5_tp_recovery.bin` containing the image we want to flash in the device. When we boot the AP pressing the reset button, it will inquire the server for the file and install the new firmware.

A.4.2 Empty Log File

Once the system is running, we can start by setting the transmission at one AP and the logging at the receiver. In [43] we can find the instructions to set up the devices as AP and Client or in injector mode. We could only use AP-Client mode, as we could not compile the Lorcon source code in our image due to the reduced space in the flash memory. In [10] we can find a broader explanation to establish the connection between the AP setting one as a Client.

For the transmitter, we run the userspace app `sendData`, while in the receiver we start the logging running `recvCSI` and the file where to log the data as the input parameter. We can see that the Client is receiving packets when a message appears with the message count, rate and payload length of the incoming packet. If this packets correctly trigger the computation of

CSI data, the log file records the CSI information. In our case, we found the log file empty.

Diving into the driver code, we could figure out that the system detects the packets correctly as HT packets, but for some reason, the CSI data is not passing from the physical layer to the new kernel module that handles the logging. However, the lack of documentation about this chipset did not allow us to go further. We saw that the driver patch code changes a couple of bits in a reserved space of a register, which we assume is related to enabling the CSI computation or passing the information upstream.

On top of that, we found out that, according to the output traces of the bootloader of the AP this one integrates a QCA9561 from the new Qualcomm Atheros, and not the Atheros AR9380 as we thought and the documentation indicated. It might be the case that the way of enabling the CSI computation for this new manufacturer chipsets is not the same as the old Atheros chipsets.

A.5 Conclusion

We have seen that the Atheros CSI Tool is not working for the TP-Link AP WA901ND-v5. There is evidence that the CSI can be extracted from NIC cards with Atheros chipsets like in [30] with AR9390 and [22] with AR9590. However, we have to conclude that it is not extensible to the QCA9561. Therefore, we have to assume that the new Qualcomm Atheros chipsets have altered the way how they enable the extraction of the CSI from the physical layer [1].

[1] We notified the developer of the tool about it, but we received no feedback.