FACULTY OF ENGINEERING AND ARCHITECTURE

The design of a CSI sensing authorisation mechanism using the open source Openwifi project

Seppe Dejonckheere
Student number: 01706815

Supervisors: Prof. dr. ir. Ingrid Moerman, Dr. Xianjun Jiao

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Information Engineering Technology

Academic year 2021-2022

GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

The design of a CSI sensing authorisation mechanism using the open source Openwifi project

Seppe Dejonckheere
Student number: 01706815

Supervisors: Prof. dr. ir. Ingrid Moerman, Dr. Xianjun Jiao

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Information Engineering Technology

Academic year 2021-2022

GHENT UNIVERSITY

Acknowledgements

I would not have been able to complete this thesis without the help and support of some people, whom I would like to thank:

First and foremost, I would like to thank my supervisor, prof. dr. ir. Ingrid Moerman. Her support, encouragement and assistance during our weekly meetings have proven to be invaluable, as well as her experience in conducting research and her reassurances when progress seemed slow.

I also would like to express my thanks to my other supervisor, dr. Xianjun Jiao, for his support and guidance on a technical level. During our weekly meetings, Dr. Jiao answered my technical questions promply, guided me in the right direction to proceed and provided me with additional insight into the matter.

In addition, I want to express my gratitude towards Annebeth Boudry for proofreading part of the thesis, which helped me to correct small mistakes, to find the right nuances and to rewrite vague or badly constructed sentences.

Finally, I would also like to express my gratitude to my parents and my girlfriend for the support and encouragement I received during this project and for reminding me that taking an occasional break is also important.

Admission to use

9 June 2022

"Deze masterproef vormt een onderdeel van een examen. Eventuele opmerkingen die door de beoordelingscommissie tijdens de mondelinge uiteenzetting van de masterproef werden geformuleerd, werden niet verwerkt in deze tekst."

"This master's dissertation is part of an exam. Any comments formulated by the assessment committee during the oral presentation of the master's dissertation are not included in this text."

The design of a CSI sensing authorisation mechanism for the open source Openwifi project

by
Seppe Dejonckheere

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Information Engineering Technology

Academic year 2021-2022

Supervisor: Prof. dr. ir. Ingrid Moerman, Dr. Xianjun Jiao

Faculty of Engineering and Architecture
Ghent University

Summary

The increasing availability of COTS Wi-Fi chips that support CSI extraction combined with advances made in the field of machine learning have resulted in machine learning models capable of performing localisation, sensing gestures and even inferring passwords. With the rise of these machine learning models also comes a security and privacy risk. At the moment of writing, it is possible for an unauthorised receiver to collect and abuse CSI. The open source Openwifi project recognises this threat and implements a CSI fuzzer. This fuzzer is able to actively change the CSI, rendering it useless for e.g. localisation and gesture sensing. In this work, a mechanism to undo the effects of the CSI fuzzer is researched, with view to implement it in a fuzzing-defuzzing protocol. First, a mechanism is proposed that predicts the fuzzer's effects based on parameters used to set up the fuzzer. During validation of the prediction, it becomes clear that the predictions do not match the observed effects. For this reason, an OFDM simulation is used to check if the mechanism works in a controlled environment. In this simulation, the predictions do match the effects applied by the fuzzer. Second, a learning based approach is discussed. Measurements show that this method is capable of undoing the applied effects, while the introduced error approaches the noise level of the dataset.

Keywords:

Openwifi, channel state information, CSI fuzzing

The design of a CSI sensing authorisation mechanism using the open source Openwifi project

Seppe Dejonckheere
Supervisors: Prof. dr. ir. Ingrid Moerman, Dr. Xianjun Jiao

***Abstract*—This document discusses two different approaches to undo the effects imposed by the CSI fuzzer included in the open source Openwifi projects.**

***Index Terms*—Openwifi, CSI, CSI fuzzing**

I. Introduction

Modern Wi-Fi systems perform channel estimations in order to provide high throughput and a robust channel. This channel estimation is an attempt to model the influence the physical environment has on a transmitted signal. An increase in common of the shelf (COTS) Wi-Fi chips that allow the extraction of this channel state information (CSI) combined with machine learning have resulted in machine learning models that are capable of performing human activity recognition [1, 2], gesture recognition [3, 4, 5] and even inferring phone passwords [6] based on the influence of the different hand positions around the phone.

The open source Openwifi project acknowledges this as a security and privacy threat and implements a CSI fuzzer in order to combat the unauthorised usage of CSI [7, 8]. This fuzzer will actively impose an artificial change to the signal en thus to the CSI [9]. Currently, no method exists that allows an authorised entity to regain the environmental part of the CSI. In this abstract, two possible methods and their performance are discussed.

II. Openwifi CSI fuzzer

A. Openwifi project

The Openwifi project is an open source full stack IEEE 802.11 design that is mac80211 compatible for software defined radio (SDR) boards[8]. The project provides field-programmable gate array (FPGA) hardware designs for several types of SDR boards and a mac80211 compatible driver to allow native Linux programs (ifconfig, wpa_supplicant, hostapd, etc.) to interact with the Openwifi FPGA. Besides this the Openwifi project also contains user space programs to control Openwifi specific functionalities. Two of these Openwifi specific functionalities are the capability to extract the CSI and the CSI fuzzer. The latter provides the capability to introduce an artificial component to the CSI [7, 8].

B. Fuzzer

The implementation of the fuzzer can be seen in figure 1. The fuzzer is implemented as a 3-tap filter that takes I/Q data as input right after the I/Q data has been transformed to the time domain and the cyclic prefix is added [7, 8]. c_1 and c_2 are

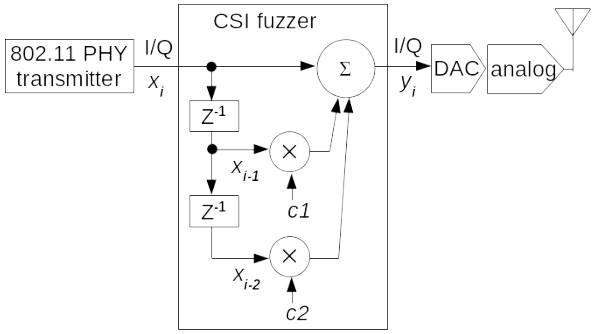


Fig. 1. CSI fuzzer implementation [8]

parameters derived from the parameters provided to the user space program and are restricted to the range of $[-0.5, 0.5)$ or $[-0.5i, 0.5i)$ for hardware simplicity reasons.

$$r(k) = \underbrace{H_{env}(k)H_{art}(k)}_{CSI(k)} s(k) + n(k) \tag{1}$$

$$H_{env}(k) = \frac{CSI(k)}{H_{art}(k)} \tag{2}$$

The effect of the fuzzer on the signal for a given subcarrier k can be described by equation 1. An entity that estimated $CSI(k)$ can thus regain the environmental part $H_{env}(k)$ as shown in equation 3, given that $H_{art}(k)$ is known and that the noise $n(k)$ is ignored.

III. H_{art} prediction

The key to regaining the environmental CSI part, H_{env}, is predicting H_{art}. The effect applied by the fuzzer can, theoretically, be predicted by calculating the n-point discrete Fourier transform of $[1, c_1, c_2]$ where n is equal to the amount of subcarrier used, which is 64 in the case of this work [9].

A. Validation

In order to validate the prediction, it is compared to the actual observed applied effect. This observed effect is calculated from a set of CSI samples where no fuzzing was applied and a set of samples where fuzzing was applied. In order to correctly calculate H_{art} the channel has to remain unchanged between both measurements and several samples are avereged out. H_{art} can be calculated from these datasets as following:

$$H_{art}(k) = \frac{CSI_{fuzzed}(k)}{CSI_{unfuzzed}(k)} \tag{3}$$

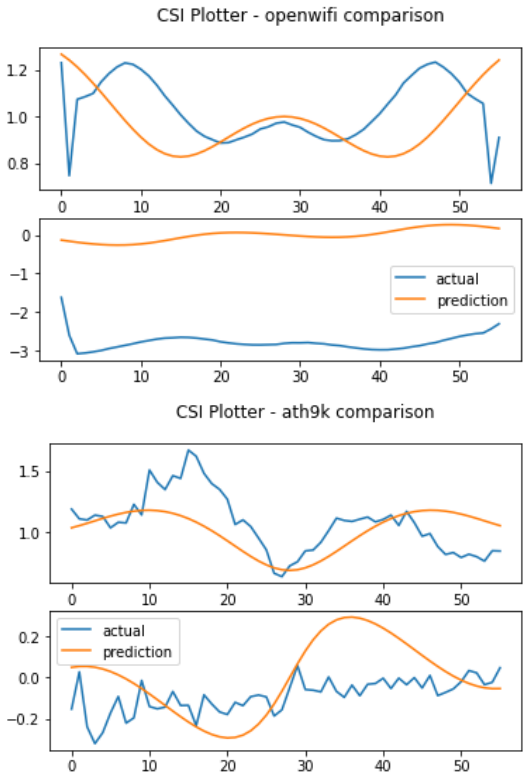


Fig. 2. Prediction vs. actual observed fuzzing pattern

When comparing these observations to the predictions, it becomes clear that they do not match. Figure 2 shows this comparison for CSI samples collected by the Openwifi board itself, as well as for CSI samples collected by a Linux node running the Atheros CSI tool [10]. It is clear in both cases that the predictions do not match the observed behaviour.

B. OFDM Simulation

To verify that the prediction based on the fuzzing parameters should be correct, a python OFDM simulation is used where the fuzzing is applied. Figure 3 shows the general flow of the simulation. Figure 4 shows the results of the simulation. The first graph shows the measured CSI and the prediction when no channel and noise were applied. This results in measured CSI where only the fuzzing effects are present. Here the fuzzing effect and the predictions do match. The second graph shows the result of H_{env} regained from fuzzed CSI, compared to CSI where no fuzzing was applied. It becomes clear that, when ignoring noise, the proposed method to predict H_{art} and use it to regain H_{env} is correct. This means that the difference between the prediction and the observed effects are caused by an unknown factor.

IV. Learning Based Method

As described in the previous section, it is possible to calculate the fuzzing effect from a set of CSI samples where no fuzzing was applied and a set where fuzzing was applied.

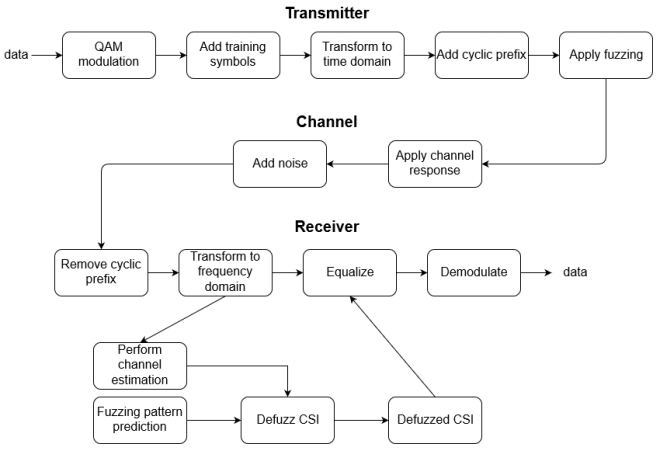


Fig. 3. OFDM simulation flowchart

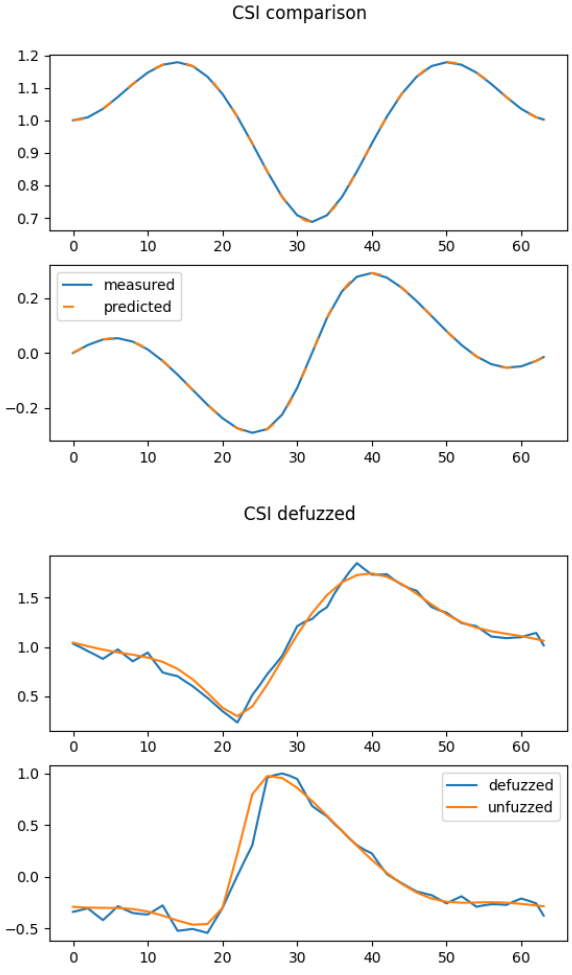


Fig. 4. Simulation results

In this section, the defuzzing based on the calculated fuzzing effect is discussed. This method is build up from two phases: the training phase and the defuzzing phase.

During the training phase, the channel has to remain static and consistent between two measurement windows. During these two measurement windows, several fuzzed and unfuzzed CSI samples should be collected respectively. Then, these collected samples should be averaged to reduce the noise. From these averaged samples, it is possible to calculate an estimation of the applied fuzzing effects, which can be used to regain the environmental part of the CSI. This happens in the same way as described in 3.

A. Performance

The performance of this method can be evaluated by calculating the error introduced by the defuzzing. The error, calculated as described by 4, represents the ratio of the average difference between the defuzzed sample x and a validation CSI sample y and the average magnitude of y, expressed in decibels. The validation sample is the average of several unfuzzed CSI samples.

$$error = 10 * \log_{10} \left(\frac{\frac{1}{s} \sum_{i=0}^{s} |y_i - x_i|}{\frac{1}{s} \sum_{i=0}^{s} |y_i|} \right) \tag{4}$$

with s as the number of subcarriers.

To visualise the performance of the defuzzing, figure 5 shows a defuzzed CSI sample compared to the averaged sample. The first graph shows this for CSI samples captured on the Openwifi board. It is very clear that both lines are very similar. The second graph shows the same but for CSI samples captured via the Atheros CSI tool. It is clear that these samples contain a lot more noise and it is harder to see how well the defuzzing performs. The third graph shows a single unfuzzed CSI sample compared to the averaged sample, to give an idea about the level of noise present in the CSI samples captures via the Atherso CSI tool.

Figure 6 shows the error for both CSI samples captured on the Openwifi board, as well as via the Atheros CSI tool, for two different sets of fuzzing parameters. In order to obtain these results, the fuzzing effect is calculated as described above, with x amount of samples averaged. The error itself is the average error of 100 separate defuzzed samples. The samples that were used to calculate the error are from a different dataset, where the channel was different then during the collection of the training datasets. The graph showing the results for the Openwifi samples varies roughly between $-12dB$ and $-14dB$. This means that the average error between the defuzzed sample and the validation CSI matrix is 15 to 25 times smaller then the average amplitude. For the Atheros CSI tool samples the results vary roughly between $-4dB$ and $-8dB$, meaning that the average error is about 2.5 to 6.5 times smaller then the average amplitude.

Better performance can be achieved by normalising all the CSI samples before calculating the fuzzing effect or before

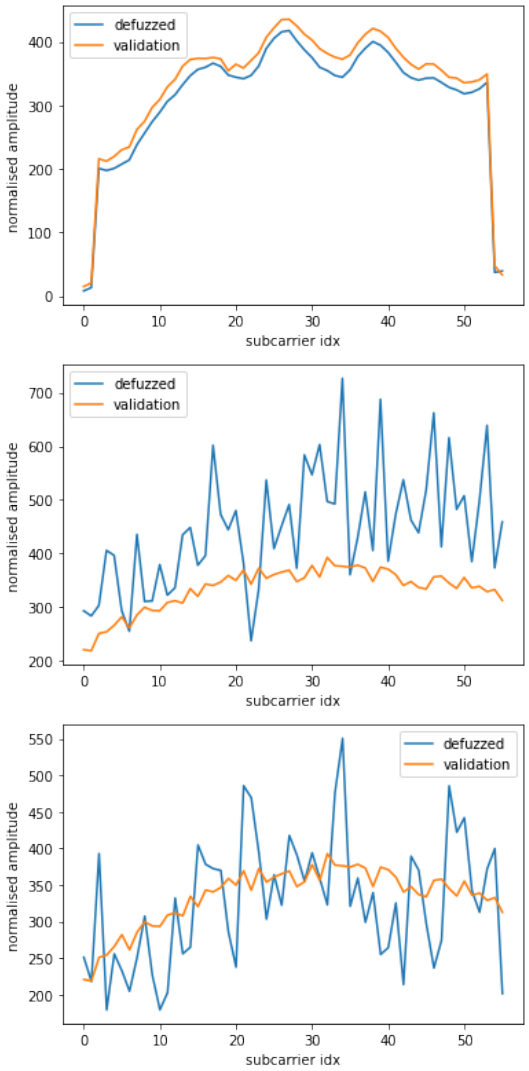


Fig. 5. Defuzzed CSI sample vs validation average

defuzzing the sample. The normalised CSI is calculated as following:

$$normalised\_csi = \frac{csi\_matrix}{\sqrt{\sum_{i=0}^{s} |csi\_matrix_i|^2}} \tag{5}$$

This has the result that:

$$\sum_{i=0}^{s} |csi\_matrix_i|^2 = 1 \tag{6}$$

Where csi_matrix is a single CSI matrix. In this case the matrix is a vector of length 56, one coefficient per subcarrier ignoring the guard band subcarriers. But it could have different shapes if other Wi-Fi versions or MIMO is used.

Figure 7 visualises the performance of the defuzzing after normalisation in the same way as 5 did. Again, the same observations can be made.

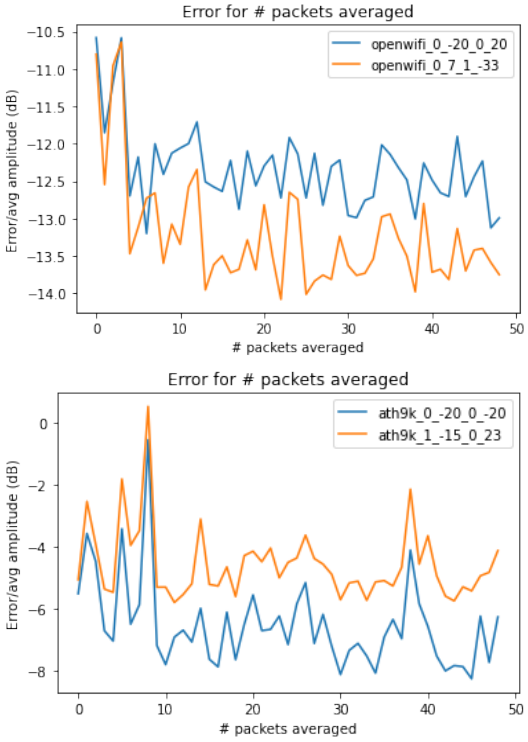


Fig. 6. Defuzzing performance in function of the # of packets averaged

The defuzzing results after normalisation can be seen in figure 8. When comparing both graphs to the previous graphs, where no normalisation was applied, it becomes clear that the results have improved. For the Openwifi samples, the average error for both fuzzing patterns fluctuates around $-15.5dB$ meaning that the mean error is about a 30 times smaller then the average amplitude. For the Atheros CSI tool samples, the mean error fluctuates around $7.4dB$ and is a lot more stable compared to the previous graph. This means that the mean error here is about a 5.5 times smaller then the average amplitude. In both graphs there is also a green line present. This green line is the result of calculating the error in the same way as described previously, but now for unfuzzed CSI samples. This means that the green line represents the expected error level due to noise.

V. Conclusion

Because the predicted fuzzing pattern, based on the fuzzing parameters, does not match the observed fuzzing pattern, it is currently not possible to use this as a way to defuzz the CSI. This means that it is currently not possible to implement a simple protocol where the fuzzing parameters are send from the transmitter to the authorised receiver, allowing the defuzzing of the CSI. However, if the unknown factor could be included in the mathematical model, or if it could be removed altogether, such a protocol would be possible.

At the moment, if a fuzzing-defuzzing protocol is to be designed, the transmitter would have to start by transmitting

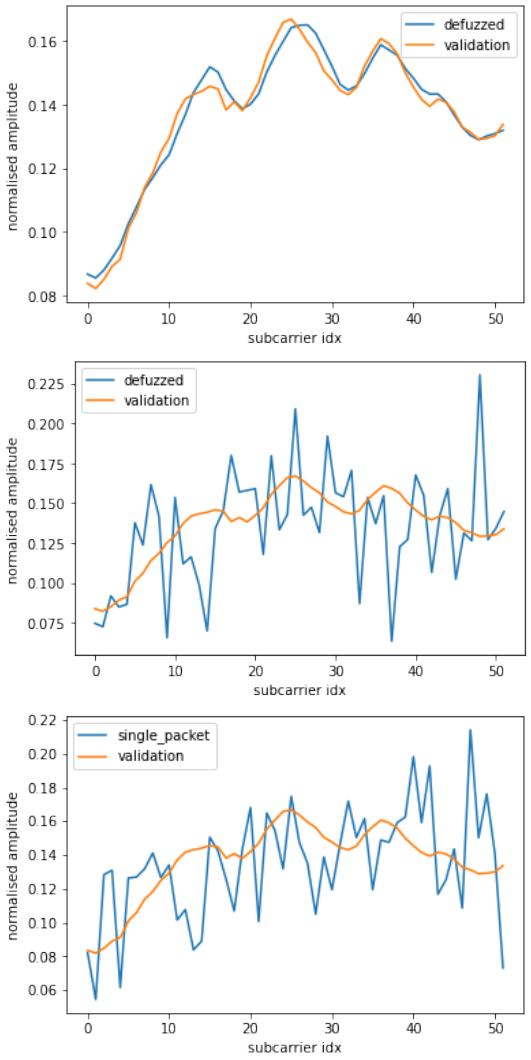


Fig. 7. Defuzzed CSI sample vs validation average

a certain amount of packets where the fuzzer is not active, followed by a certain amount of packets where the fuzzer is active. During the transmission of these packets, the training period, the channel has to remain static as the CSI has to be averaged. Once the fuzzing pattern has been calculated from the training data, the channel can be dynamic.

This setup, with the training period, also means that it is still possible for an unauthorised receiver to perform CSI sensing. The attack vector has become time sensitive, as the attacker has to collect the CSI during the training phase, but it is not impossible. For that reason, additional research into the cause of the unexpected behavior of the fuzzer is desirable.

References

[1] F. Adib and D. Katabi, "See through walls with wifi!" in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing

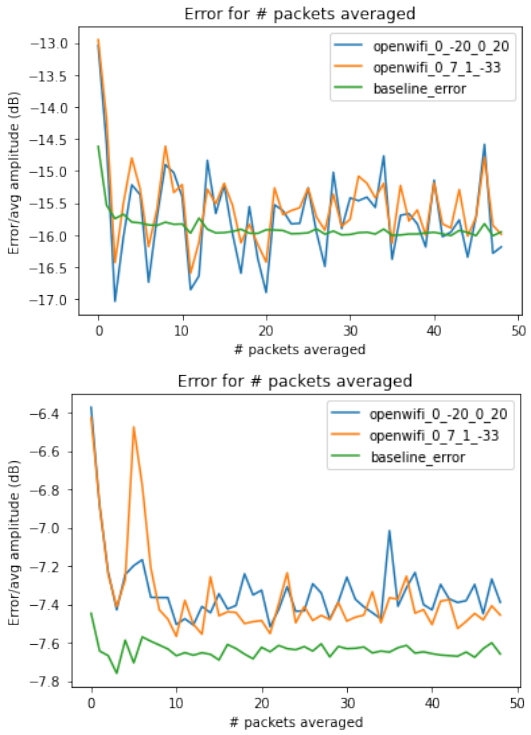


Fig. 8. Defuzzing performance in function of the # of packets averaged after normalisation

Machinery, 2013, p. 75–86. [Online]. Available: https://doi.org/10.1145/2486001.2486039

[2] S. Arshad, C. Feng, Y. Liu, Y. Hu, R. Yu, S. Zhou, and H. Li, "Wi-chase: A wifi based human activity recognition system for sensorless environments," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017, pp. 1–6.

[3] H. Li, W. Yang, J. Wang, Y. Xu, and L. Huang, "Wifinger: Talk to your smart devices with finger-grained gesture," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 250–261. [Online]. Available: https://doi.org/10.1145/2971648.2971738

[4] Y. Ma, G. Zhou, S. Wang, H. Zhao, and W. Jung, "Signfi: Sign language recognition using wifi," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, no. 1, mar 2018. [Online]. Available: https://doi.org/10.1145/3191755

[5] P. Melgarejo, X. Zhang, P. Ramanathan, and D. Chu, "Leveraging directional antenna capabilities for fine-grained gesture recognition," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 541–551. [Online]. Available: https://doi.org/10.1145/2632048.2632095

[6] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When csi meets public wifi: Inferring your mobile phone password via wifi signals," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1068–1079. [Online]. Available: https://doi.org/10.1145/2976749.2978397

[7] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source ieee802. 11 sdr implementation on soc," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–2.

[8] X. Jiao, W. Liu, and M. Mehari. (2019) open-source ieee802.11/wi-fi baseband chip/fpga design. [Online]. Available: https://github.com/open-sdr/openwifi

[9] X. Jiao, M. Mehari, W. Liu, M. Aslam, and I. Moerman, "Openwifi csi fuzzer for authorized sensing and covert channels," ser. WiSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 377–379. [Online]. Available: https://doi.org/10.1145/3448300.3468255

[10] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity wifi," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: ACM, 2015, p. 53–64. [Online]. Available: http://doi.acm.org/10.1145/2789168.2790124

Contents

List of Figures

List of acronyms

CSI channel state information

SDR software defined radio

COTS common of the shelf

OFDM orthogonal frequency division multiplexing

IDFT inverse discrete Fourier transform

DFT discrete Fourier transform

CP cyclic prefix

IBI inter block interference

ISI inter symbol interference

HT high throughput

MIMO multiple input, multiple output

SISO single input, single output

MMSE minimal mean squared error

HT-LTF high througput, long training field

FPGA field-programmable gate array

I/Q in-phase/quadrature

SDGs Sustainable Development Goals

1

Introduction

The widespread usage of modern Wi-Fi systems require them to be reliable and provide high throughput. One of the techniques used to provide these needs, is the usage of channel state information (CSI) to optimise the transmission of data [1]. The physical layers of the IEEE 802.11 standard allow for the estimation of the communication's channel state. This estimation, the channel state information, is a representation of the effects the physical environment has on the signal, like power decay over distance, scattering, interference, and so forth. Channel state information is used to adapt the transmission parameters to the current state of the channel allowing for a reliable communication channel with high throughput [2].

Because the CSI is a representation of the physical environment, it carries information about the environment itself. It is possible to train machine learning models capable of extracting this information about the environment and use it for further processing. They can be trained to detect human activity [3, 4] and recognise gestures [5, 6, 7]. They can even be used to infer phone passwords or pin codes by detecting the small changes in the physical environment made by the hand movements [8].

Initially the CSI had to be exposed by using specific hardware like a software defined radio (SDR), but in recent years there has been an increase in common of the shelf (COTS) Wi-Fi cards that support CSI extraction: like the Atheros chipset using the ath9k driver [9], devices using certain Broadcom Wi-Fi chips like the Raspberry Pi B3+/B4 and the Nexus 5/6P [10, 11] and the Intel Wi-Fi Wireless Link 5300 [12]. This rise in relatively cheap COTS Wi-Fi cards that support CSI extraction makes the extraction of CSI easier and more widely available.

The relative easily accessible CSI makes it a lot easier to train a machine learning model, but this trend also poses a privacy/security risk, because it is easier for unauthorised entities to collect CSI as well. An unauthorised entity could collect CSI and use it for e.g. localisation or password inferring. An attacker could, unknown to the victim, passively capture the CSI transmitted by devices already present (e.g., Wi-Fi access point), or actively transmit packets and measure the CSI [13]. The attacker can then use the captured CSI to extract information.

The Openwifi project recognises this privacy/security threat. It provides a mitigation for the passive attack vector. Mitigating the active attack vector would require the active jamming of Wi-Fi signals. The openwifi project contains a CSI fuzzer which changes the CSI at the transmission side. The effects of the fuzzer on the signal are combined with the environmental effects during transmission. On the receivers side, the estimated CSI is now a combination of the effects imposed by the fuzzer and the environment. Without knowing the exact modifications imposed by the fuzzer it is not possible to regain the environmental effects and so it is not possible to extract information about the environment [14]. This fuzzer prevents an unauthorised entity from extracting information from the CSI, but it also prevents any authorised entities from doing so. Currently no mechanism is in place to allow an authorised receiver to regain the environmental CSI.

In the coming chapters, the design of such a mechanism will be discussed, with view to implement the mechanism in a fuzzing-defuzzing protocol. But first, some basic key concepts about signal propagation and orthogonal frequency division multiplexing (OFDM) will be discussed. Then, chapter 3 will explain the core concept of CSI in greater detail. Chapter 4 digs deeper in the Openwifi project and its features that are used in this thesis. In chapter 5 the implementation of a mechanism allowing to regain the environmental CSI and its performance are discussed. Finally, chapter 6 contains the conclusions which can be drawn from this thesis, as well as some remarks and possibilities for future improvements.

2

Literature

In order to have a better understanding of the context of this thesis, it is necessary to have some knowledge about the basic concepts of signal propagation, as it are these effects that ultimately give shape to the CSI. Next, the principles of OFDM will be discussed, as this technique is used to combat some effects of wireless channel propagation.

2.1 Wireless signal propagation

As a wireless signal, an electromagnetic wave, travels from a transmitter to a receiver, the signal is exposed to several effects imposed by the physical environment that it is propagating through. There are several effects influencing the signal, each with their own characteristics. First, path loss will be discussed. Next, the effects of the signal interacting with objects are discussed. These are [15]:

- Reflection: interaction with large surfaces relative to the wave length
- Diffraction: interaction with edges of large surfaces relative to the wave length
- Scattering: interaction with surfaces of the same size order as the wave length

Finally, the derived effects that are the result of a combination of the previously mentioned ones will be discussed.

Path loss

Path loss is the result of the combination of power decay over distance and losses due to absorption while traveling through media. Power decay behaves according to the inverse square law. The initially transmitted energy is divided over the surface of a sphere. Conservation of energy dictates that as the travelled distance increases and thus the radius of the sphere, the energy has to be divided over a larger surface. The surface of a sphere is quadratic in relation to its radius. This means that when the distance between the transmitter and receiver doubles, the amount of energy received by the receiver is divided by four [15, 16]. The power decay over distance increases for increasing frequency [15].

The rate at which energy is lost or absorbed can be described by the path loss exponent. A path loss exponent of 2 is expected for free-space transmissions, as this is the result of the power decay over distance. The path loss exponent increases when the signal travels through non empty space.

Reflection

Reflection is the result of an electromagnetic wave colliding with an object or surface where any convexities or concavities are much bigger then the wavelength of the electromagnetic wave [15, 16]. The incoming magnetic wave collides with the object or surface and (partially) reflects back in a different direction. A part of the signal will be refracted or absorbed, causing some energy loss. The amount of energy that is reflected, absorbed or refracted depends on the materialistic properties of the object and the medium and on the frequency of the signal.

Diffraction

Diffraction happens at sharp edges of object, where electromagnetic waves will bend around those edges, as shown in figure 2.1. This effect can be explained by Huygen's principle, which states that each point on of a wave front will act as the source of a secondary wave front [17]. In the case of a sharp edge, the points near the edge will act as a secondary wave source. This causes the wave to bend around that edge. The figure shows how diffraction helps cellular coverage in cities.

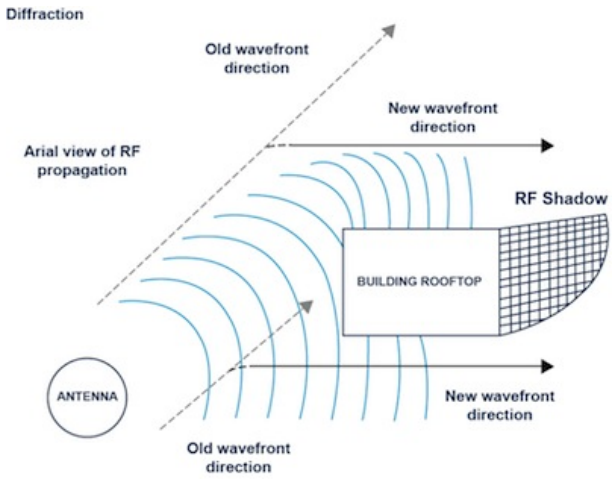


Figure 2.1: Diffraction

Source: [18]

Scattering

Scattering occurs when an electromagnetic wave impinges with an object or a surface with convexities or concavities that have roughly the same dimensions as the wavelength [15, 16]. The effects of scattering are similar to those of reflection, but here the incoming wave is reflected in multiple different directions instead of a single one, as can be seen in figure 2.2. The scattered parts generally have a much lower amplitude then the incoming wave because the energy of the incoming wave is distributed over the several outgoing ones.

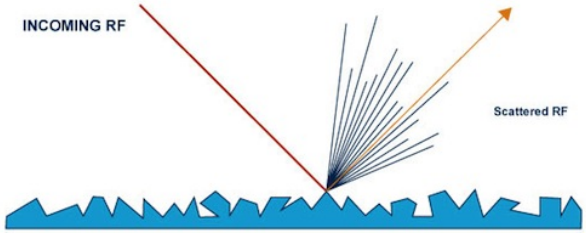


Figure 2.2: Scattering

Source: [18]

Visualisation

The combination of the previously discussed effects result in the original signal being reflected, scattered and diffracted and causes these signals to interfere with each other. It is possible to model these effects via the Helmholtz equation. Physicist and software engineer J. Cole solved this equation for his apartment as can be seen in 2.3. It's important to note that for this simulation a wavelength $\lambda = 30cm$ was used [19]. In the image, bright yellow represent a high power signal and the darker colors represent a low energy signal. The power decay over distance is clearly visible as the color gradually gets darker when moving away from the starting point. The typical destructive and constructive interference patterns are also easy to recognise.

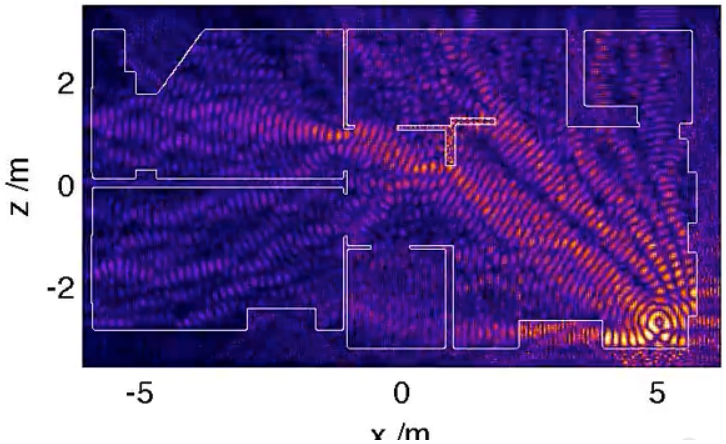
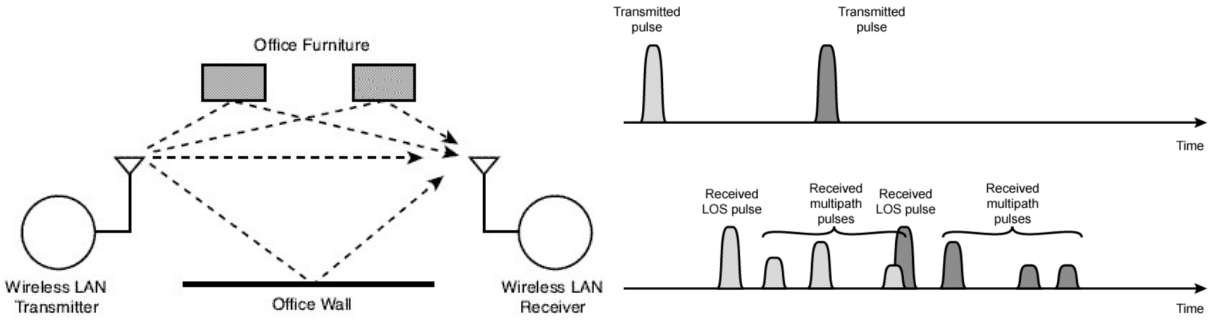


Figure 2.3: Signal propagation simulation
Source: [19]

2.1.1 Multipath

When there is a line of sight between the transmitter and the receiver, diffraction and scattering have no significant effects because of the lower energy these signals have. Reflection can have a significant effect on the received signal. When there is no line of sight, transmission is still possible mainly due to the effects of scattering, diffraction and reflection [15].

The effects of reflection can be significant even when there is a line of sight between the transmitter and the receiver. Reflection causes the signal to have multiple paths from the transmitter to the receiver, as is illustrated in 2.4a. This effect is called multipath. Because these reflected signals all take a different path, they all take a different amount of time to reach the receiver. This causes the received signal to be smeared out over time, as can be seen in figure 2.4b [16, 15]. The phase difference between the multiple received signals can cause interference which may lead to significant changes in received power for very small changes in position.



(a) Multiple paths from transmitter to receiver
Source: [20]

(b) Received multipath pulses
Source: [15]

Figure 2.4: Signal multipath

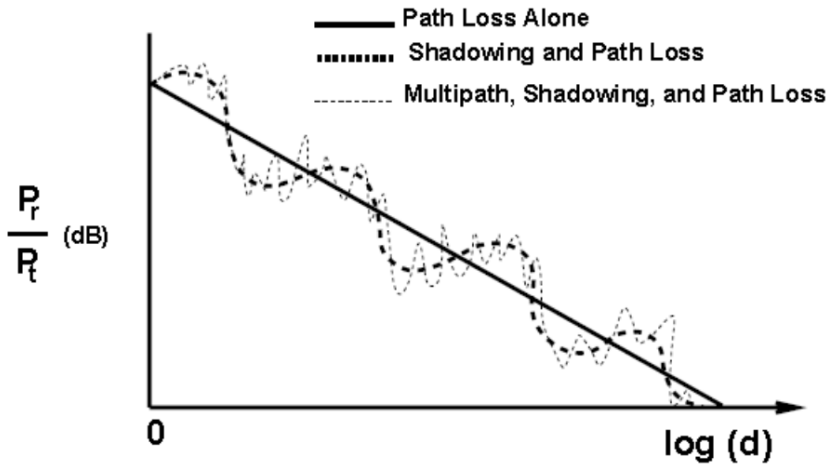


Figure 2.5: Received power P_r vs. distance d
Source: [15]

Figure 2.5 shows the expected received power in function of the distance. Because of the path loss, the logarithmic distance is used instead of the distance itself. When only path loss is considered, the average received power declines linearly. When, in addition to path loss, shadowing is considered, the large scale fluctuations can be explained. The fluctuations, called large scale fading, are the result large obstructions that are present on the paths between the transmitter and receiver. When all effects are considered, there are very small fluctuations present as well. These fluctuations, called small scale fading, are the result of the constructive and destructive interference caused by the multipath components. This means that a small change in the position of an antenna can have a significant effect on the received power level [15].

2.1.2 Frequency selective fading

The effects discussed previously are all frequency selective to some extend. Power decay over distance increases for higher frequencies. Reflection, scattering and diffraction are frequency selective as well causing multipath to be frequency selective too. This combined with a dynamic environment with moving objects, which cause Doppler shifts to the reflected and scattered signals, also cause a difference in received power depending on the frequency.

When no frequency dependency is present, the fading is uniform for the whole bandwidth, this is called flat fading. When plotting the received power for each frequency the difference between flat and selective fading becomes clear:

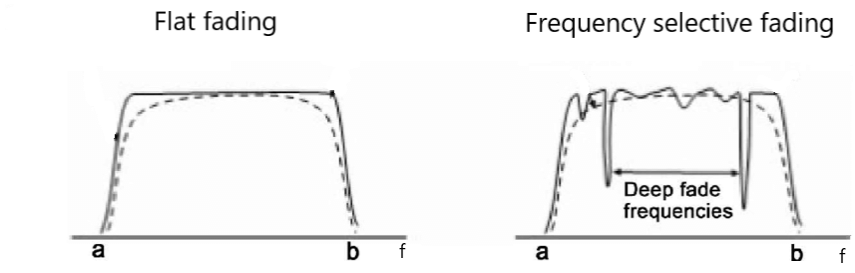


Figure 2.6: Flat fading vs. frequency selective fading
Source: [21]

2.2 Orthogonal frequency division multiplexing

To combat the frequency selective fading the 802.11a amendment to the original 802.11 standard came with the introduction of OFDM. This term is composed of 2 parts: the frequency division multiplexing part and the orthogonal part.

Frequency division multiplexing

Frequency division multiplexing is a way to combine multiple signals and send them over a shared medium. The available bandwidth is divided in multiple frequency bands, called carriers. Each of these bands carries a different signal. Because the bandwidth is now divided into several subcarriers, the frequency selective fading only effects a number subcarriers while the others remain fairly undistorted. To prevent interference between frequency bands guard bands are placed between the carriers. These are frequency bands that are not used and so leave space between the used frequency bands [22].

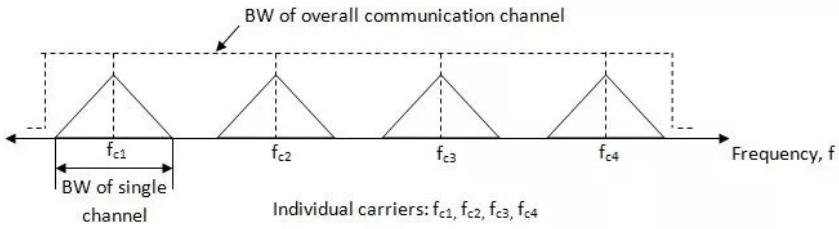


Figure 2.7: Frequency division multiplexing
Source: [22]

Orthogonal FDM

As can be seen in figure 2.7 quite some bandwidth is lost to guard bands. These unused frequency bands lower the amount of signals that can be multiplexed. This lowers the data rate by wasting bandwidth. This is where the orthogonal part of OFDM enters. By using specific, orthogonal, carrier signals guard bands are no longer necessary and the carriers can even overlap. This tight packing of the carriers allows to fit a lot more carriers in the available bandwidth [23].

The specific shape of the signal in frequency domain is described by equation as seen in 2.2. This is the Fourier transform of equation 2.1 where A defines the amplitude of the sinusoid , f_{sc} defines the frequency of the sinusoid and T the duration of the signal [24].

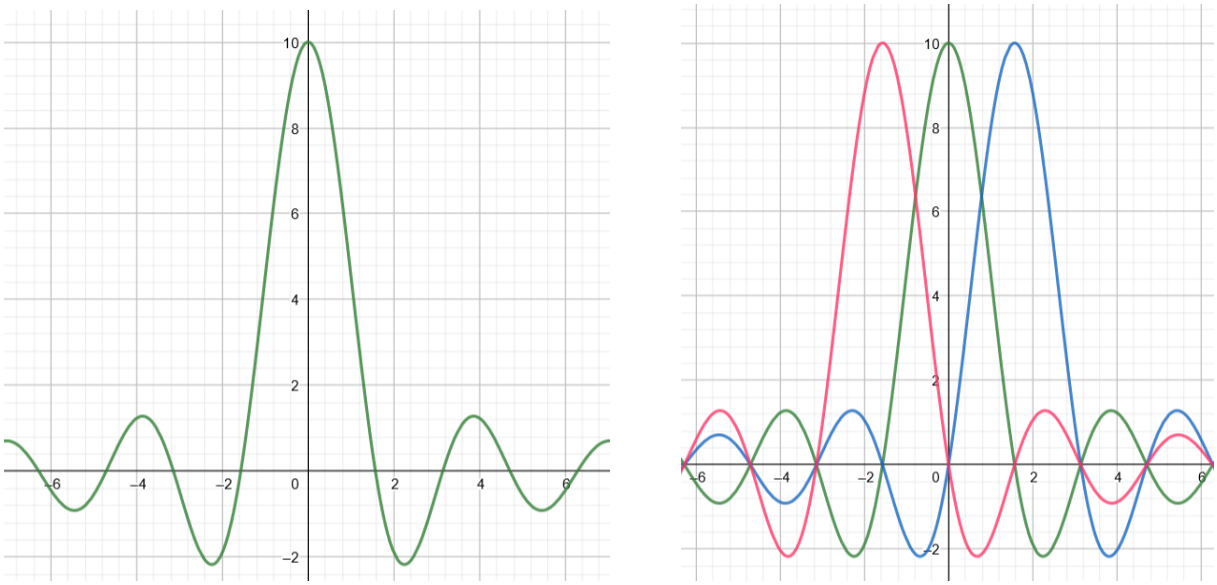
$$s(t) = A \sin(2\pi f_{sc} t) \text{ for a duration of } T \tag{2.1}$$

$$S(f) = A\frac{sin[T(f - f_{sc})]}{(f - f_{sc})} \tag{2.2}$$

Figure 2.8a shows an example of how this function looks in the frequency domain. When visualised, it becomes clear that it has several interesting properties. First of all, A defines the overall amplitude of the function. This parameter is used to modulate different symbols. Secondly, by increasing the value of parameter T the difference between the central peak and the other peaks increases, this means that T defines the frequency of the function and thus the location of the zero points. Third, f_{sc} can be used to translate the function in the horizontal direction and so defines the carrier frequency. In this case $f_{sc} = 0$.

Because the period and the location of the peak can be controlled, multiple of these functions can be smartly spaced. By making sure the peak of one function is above a zero point of other functions, the cross carrier interference is reduced to zero. Figure 2.8b shows some of these functions spaced in such a way. At the center of every sub-carrier, all other sub-carriers are zero. The figure makes it abundantly clear that there is a high peak to average ratio, resulting in minimal cross carrier interference.

Figure 2.9 shows how the frequency selective fading only affects a number of subcarriers and not the whole signal. Thus, providing a more robust communication channel overall while also providing high throughput. The downside of the tight packaging of subcarriers is the sensitivity to the frequency synchronisation. When the receiver doesn't sample at the exact peak of a sub-carrier, the cross carrier interference becomes significant [24].



(a) Frequency domain single signal

(b) Frequency domain multiple signals

Figure 2.8: OFDM subcarrier spacing

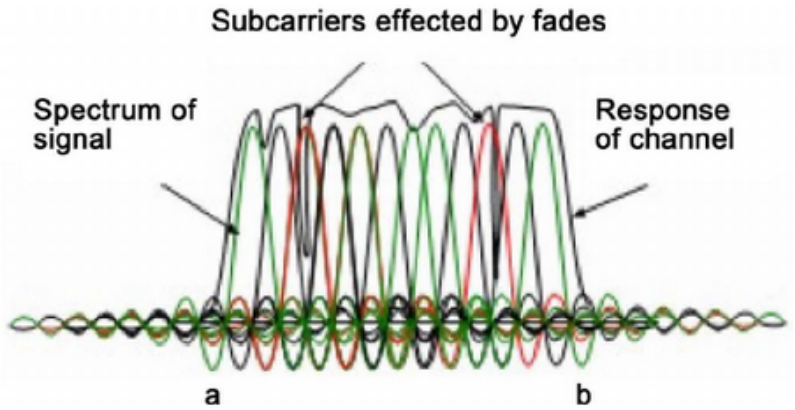


Figure 2.9: Frequency selective fading in an OFDM system

Source: [21]

2.2.1 OFDM Transmitter and Receiver

Figure 2.10 gives a schematic overview of the structure of an OFDM transmitter. $S(i)$ is a serial symbol stream, with symbols from a certain constellation e.g. 64QAM. This serial stream is converted to a parallel stream, here N represents the number of subcarriers. Then this parallel stream is converted to the time domain via the inverse discrete Fourier transform (IDFT). Next, the stream is converted back to a serial stream. Finally a cyclic prefix (CP) of length L is added.

At the receivers side, the CP is removed from the incomming signal. Next, the serial stream is converted to a parallel stream, in order to transfor this parallel stream to the frequency domain via the discrete Fourier transform (DFT). Finally the frequency domain stream is converted back to a serial stream. It has to be noted that the $S(i)$ at the receivers side is not necessarily the same as $S(i)$ on the transmitters side.

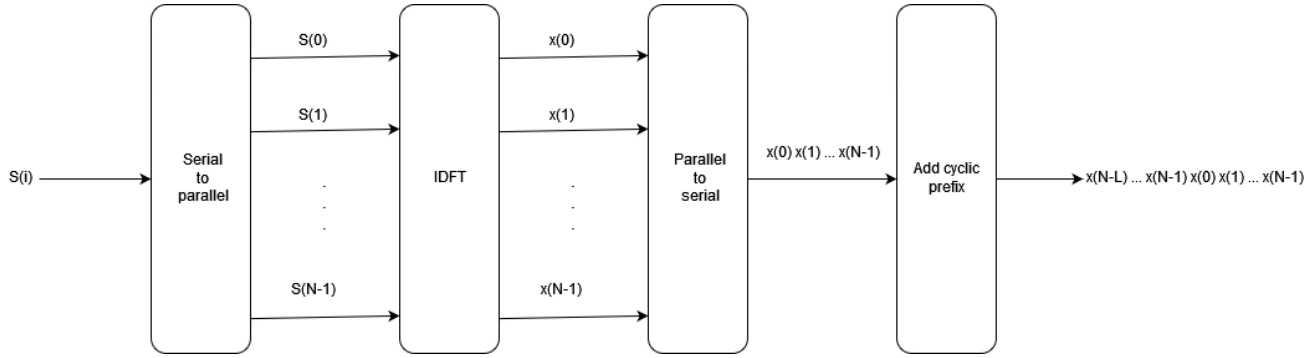


Figure 2.10: OFDM transmitter schema

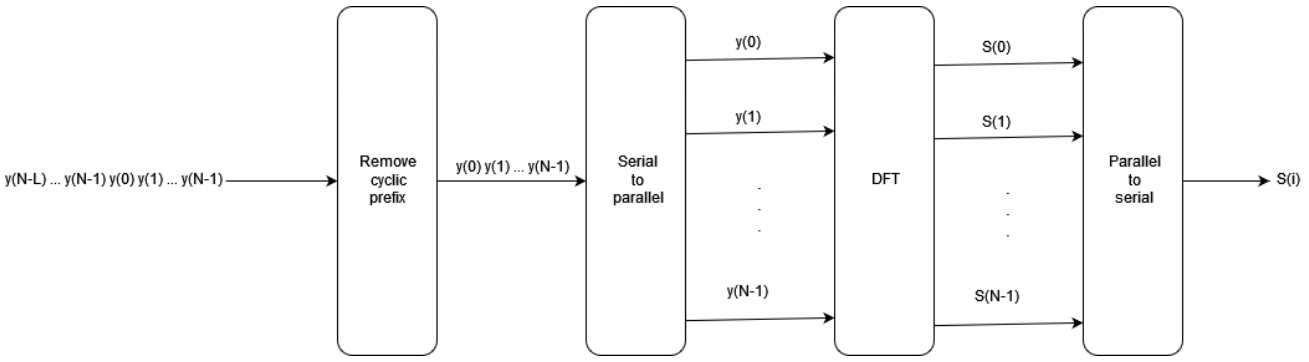


Figure 2.11: OFDM receiver schema

While the CP seemingly has no function, as it is added and then immediately removed at the receiver. However, the CP tries to prevents inter block interference (IBI). When multiple OFDM blocks, the serial stream at the end if 2.10, are send consecutively, multipath may cause these blocks to interfere with each other. These blocks interfering with each other is called IBI. By introducing a CP of length L, the CP acts as a guard interval. When multipath causes delay, the interference is now restricted to samples from the same block. This means that no IBI occurs at the cost of inter symbol interference (ISI) and a reduction of throughput, as long as the delay is smaller then the transmission time of the L prefix samples.

In this thesis, the IEEE 802.11n standard is used in 20MHz high throughput (HT) mode. This mode uses 64 sub-carriers per channel, from which 52 are used to send data. There are 4 pilot carriers used to reduce the effects of frequency offset and phase noise, 7 guard carriers to prevent cross channel interference and the centre DC or null carrier, which is not used since measurements on this subcarrier interfere with the hardware's DC [25, 26].

3

Channel state information

This chapter will discuss the concept of CSI and how this is used in modern Wi-Fi systems to provide a high throughput channel. First the general usage of CSI will be discussed. Next several methods to perform a channel estimation are discussed.

3.1 Channel coefficients

A general modern Wi-Fi system may have multiple antennas at the receiver and transmitter's side. Such a system is called a multiple input, multiple output (MIMO) system. MIMO systems are added in the IEEE 802.11n adoption to the Wi-Fi standard [25]. MIMO systems use the multipath effect to its advantage by using the multiple paths for spatial multiplexing or to provide a more robust channel [27]. Such a system can be seen in figure 3.1.

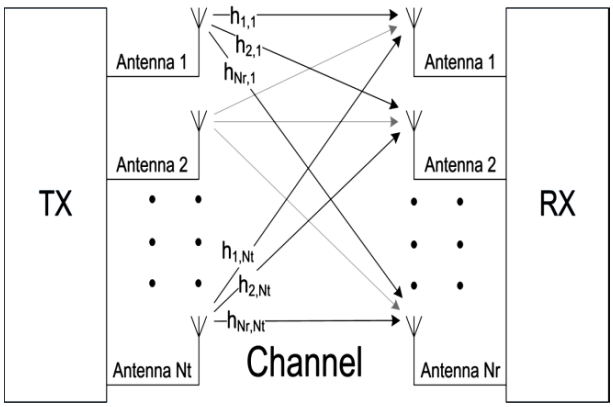


Figure 3.1: MIMO diagram
Source: [28]

A transmitter with t antennas, as in figure 3.1, can be represented as a t-dimensional vector:

$$T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \end{bmatrix}$$

The same applies to the receiver with r antennas:

$$R = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix}$$

Because the input is a t-dimensional vector and the output is a r-dimensional vector, the channel itself can be modeled as a $r \times t$-dimensional matrix. When noise is taken into account, the channel can be modeled as [27]:

$$R_{r\times 1} = H_{r\times t}T_{t\times 1} + N_{r\times 1} \tag{3.1}$$

Each element h_{ij} from the CSI matrix H is the channel coefficient between the j-th transmitter antenna and the i-th receiver antenna. Channel coefficients are complex numbers modelling the influence of the channel on the signal. When the transmitted and received signal are mapped onto a constellation diagram, the channel coefficient is the displacement factor between the transmitted and received symbol.

At the receiver's side, the received signal on a certain antenna is a combination of the signals send from all antennas at the transmitter's side. For the receiver to regain the the send signals, it has to perform some signal processing. This signal processing boils down to solving a set of linear equations with r equations and t unknowns [29]:

$$\begin{aligned} r_1 &= h_{11}t_1 + h_{12}t_2 + \ldots + h_{1t}t_t + n_1 \\ r_2 &= h_{21}t_1 + h_{22}t_2 + \ldots + h_{2t}t_t + n_2 \\ &\vdots \\ r_r &= h_{r1}t_1 + h_{r2}t_2 + \ldots + h_{rt}t_t + n_r \end{aligned} \tag{3.2}$$

Solving this set of equations is only trivial when $r = t$ and H is an invertible matrix. In all other cases, it is not possible to exactly solve this system of linear equations. Instead the goal is to minimize the error while estimating T [29]. This estimation of T is called equalisation and there are multiple techniques to this like: zero-force equalisation, minimal mean squared error (MMSE) equalisation and more recently machine learning based equalisation [30, 31]. In order to perform equalisation, matrix H has to be known. This is where channel estimation comes into play.

Because Wi-Fi uses OFDM, the actual CSI matrix of the system above is a $r \times t \times s$-dimensional matrix, with s the number of subcarriers. This matrix can still be interpreted as the channel coefficients between each pair of transmitting and receiving antenna, but instead of a single value per pair, there is one coefficient per subcarrier.

The setup used in this thesis is a single input, single output (SISO) system and thus has one transmitting and one receiving antenna. Because of this, the CSI matrices have a dimension of $H_{1\times1\times s}$ with s equal to the number of subcarriers, 64 in this case, or 56 if the guard bands are ignored. Thus the each CSI matrix is effectively a vector of length s.

3.2 Channel estimation

To allow the receiver to regain the original transmitted signals,as discussed in the previous section, it has to estimate the channels state, and thus the channel coefficients. In the previous section the channel coefficients were depicted as an $r \times t \times s$-dimensional matrix H. There are multiple different ways for the receiver in an OFDM system to estimate this matrix H, but they all can be put into one of three broad categories [32]:

- Training or pilot based channel estimation
- Blind channel estimation
- Semi-blind channel estimation

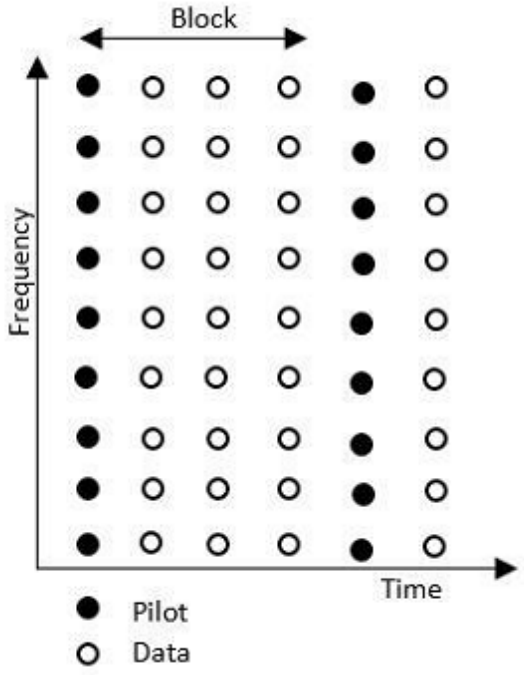
Training based channel estimation

Training based channel estimation techniques are based on the insertion of training symbols, also referred to as pilot symbols. The receiver knows what these pilot symbols should be and how they are received, thus it can estimate the channel state. The channel estimation is performed based on the pilot symbols and is then extended for the data symbols [32]. There are 2 ways for these pilot symbols to be inserted [33]:

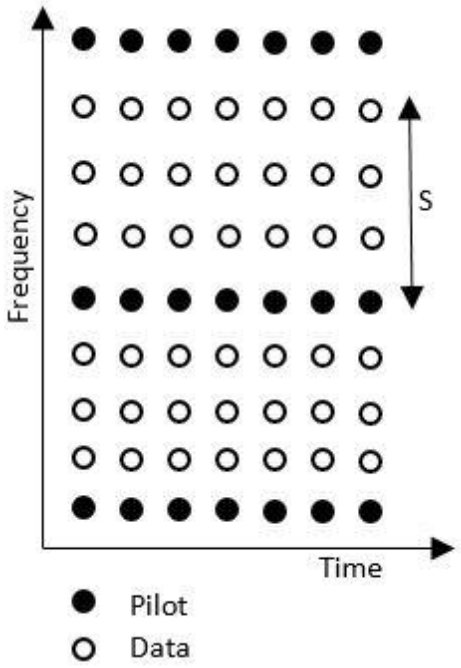
- Block type pilot channel estimation
- Comb type pilot channel estimation

Block type channel estimation happens by periodically sending pilot symbols on all subcarriers, as can be seen in figure3.2a. Channel estimation happens based upon these symbols and is extended for the following data symbols until new pilot symbols are send. As long as the channel doesn't change in between no channel estimation error is made [34]. Comb type channel estimation uses a selection of the subcarriers to continuously use these subcarriers for pilot symbols, as can be seen in figure 3.2b. The estimation for the data subcarriers is obtained via interpolation [34]. Techniques using training based channel

estimation use the available bandwidth less efficient because of the overhead introduced by injecting the training symbols.



(a) Block type channel estimation
Source: [33]



(b) Comb type channel estimation
Source: [33]

Figure 3.2: Pilot based channel estimation

Blind channel estimation

Blind channel estimation techniques base their estimation on the (higher order) statistical properties of the received signal without pilot symbols as a reference. No overhead is introduced by pilot symbols so the available bandwidth is used more efficiently, but blind estimation techniques have slow convergence rates and thus require a large number of received symbols to perform accurate estimations [32]. They also require increased processing at the receiver to perform the statistical calculations.

Semi-blind channel estimation

Semi-blind channel estimation techniques combine the use of training symbols and statistical properties to perform a channel estimation. These techniques chose the middle ground in the trade of between bandwidth efficiency and processing at the receivers side. So they have a lower overhead due to a smaller amount of training symbols being used then pilot based estimation techniques, but the require more processing to perform the blind estimation [32, 35].

3.2.1 Channel estimation in IEEE802.11n

The IEEE802.11n amendment to the original standard implements a block type structure for the pilot symbols. In HT mode, the pilot symbols are introduced across all 52 data subcarriers, as well as the four pilot subcarriers, in the high througput, long training field (HT-LTF) block. As this block is located in the beginning of the packet, the channel estimation based on these pilot symbols is used for the equalisation of all the data in the packet. If the channel would change mid transmission, This means that the channel estimation may be invalid at the end of the packet [36].

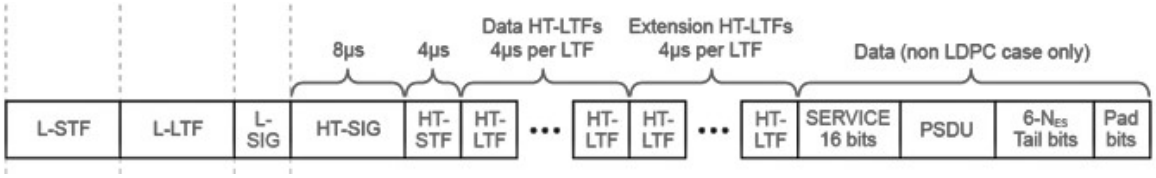


Figure 3.3: OFDM block structure in IEEE802.11n HT mode

Source: [37]

4

Openwifi

In this the chapter, the setup used for this thesis will be discussed. The most important part of the setup is an SDR board running an Openwifi image. The Openwifi project is an open source full stack IEEE 802.11 design that is mac80211 compatible for SDR boards[14]. The project provides field-programmable gate array (FPGA) hardware designs for several types of SDR boards and a mac80211 compatible driver to allow native linux programs (ifconfig, wpa_supplicant, hostapd, etc.) to interact with the Openwifi FPGA. Besides this the Openwifi project also contains a user space program to control Openwifi specific functionalities. Two of these Openwifi specific functionalities are the capability of CSI extraction and the CSI fuzzing. The latter provides the capability to introduce an artificial component to the CSI [14, 38]. Detailed information about the Openwifi project can easily be found on the github page: https://github.com/open-sdr/openwifi.

For this thesis an ADRV1CRR-FMC SDR module carrier board is used with an ADRV9361-Z7035 SDR module, as shown in figure 4.1. The board has one transmission antenna and one receiving antenna. It is used as an access point for this thesis and a COTS client will be used to connect to the board.

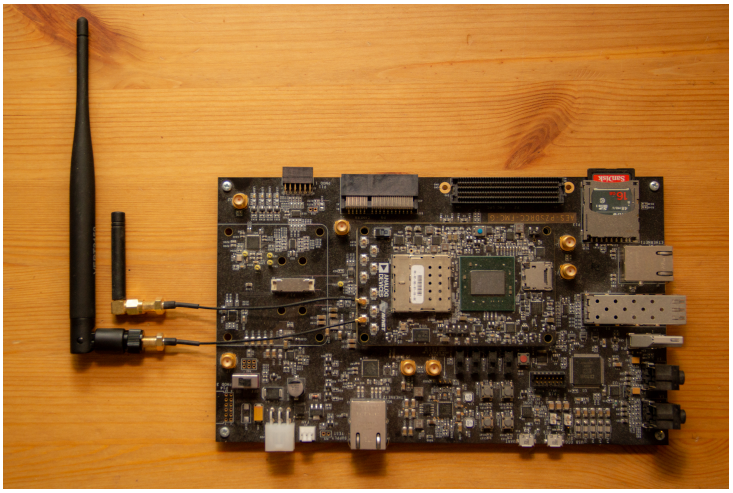


Figure 4.1: SDR board used in this thesis

4.1 CSI extraction

The Openwifi project provides a user space program that enables the extraction of CSI data from the chip and a program to visualise the captured data. The Openwifi board has full duplex capabilities. Normally, during transmission the receiving antenna is muted, but it is possible to unmute the receiving antenna. This allows the board to receive it's own transmitted signal . As the board has the original signal, as well as the received signal, it is possible to calculate the CSI. This is different from the methods discussed in the previous chapter as it can be seen as training based channel estimation, for every subcarrier and every packet but without the need for training symbols. To reduce noise, the CSI is also smoothed by calculating a moving average for every subcarrier [38, 14].

In addition to the program to collect CSI, there is also a program provided to visualise the captured data. The top part show the absolute value of the CSI per subcarrier, calculated as $A = \sqrt{a^2 + b^2}$ for a complex number $a + bi$. The bottom graph shows the phase per subcarrier, calculated as $\phi = \arctan \frac{b}{a}$ for a complex number $a + bi$. This program also writes the received data to a file, for later processing.

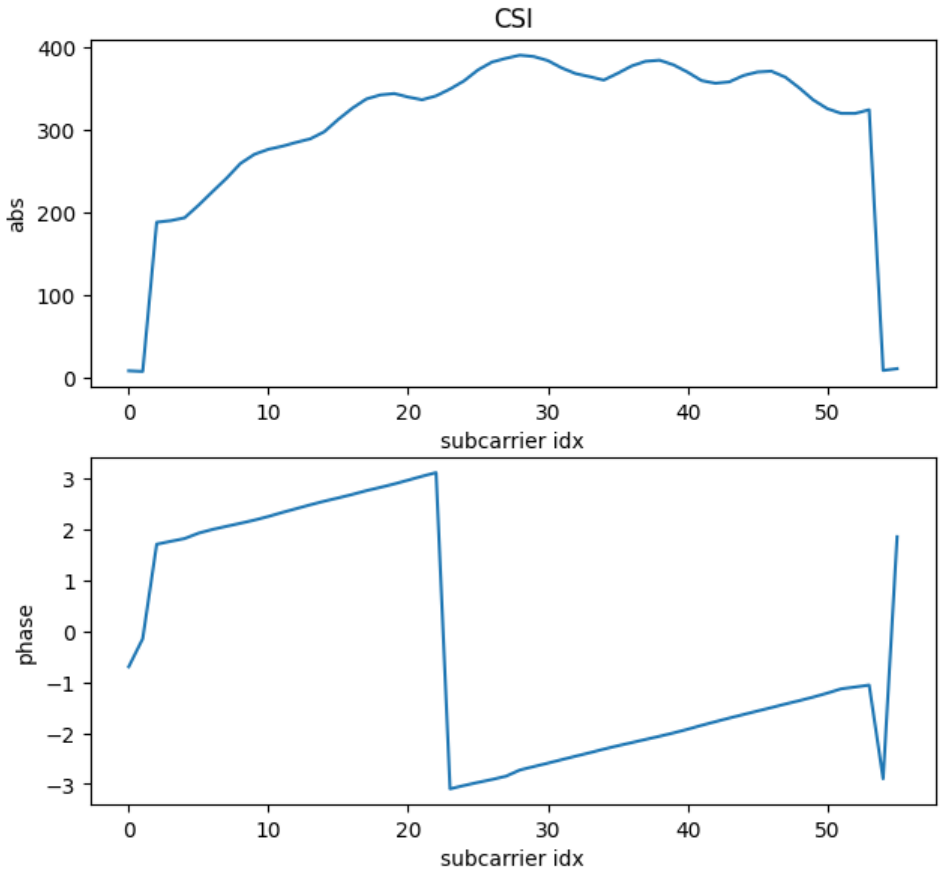


Figure 4.2: Example of captured CSI

4.1.1 Client with COTS Wi-Fi chip

In order to validate the collected CSI, there was also a client used that has a COTS Wi-Fi chip which exposes its CSI. First, a Raspberry Pi B4 was used. This Raspberry Pi had the Nexmon firmware patching framework installed, in order to install the Nexmon Channel State Information Extractor patch [11, 10]. There were some problems with this approach. To begin, the extracted CSI was very unstable, the amplitude had fluctuations that were orders of magintude bigger then the average amplitude, as can be seen in figure 4.3. Additionally, the Wi-Fi chip had to be set to monitor mode in order to extract the CSI. This is not feasible as this research is performed with view on a protocol build on top of a Wi-Fi connection.

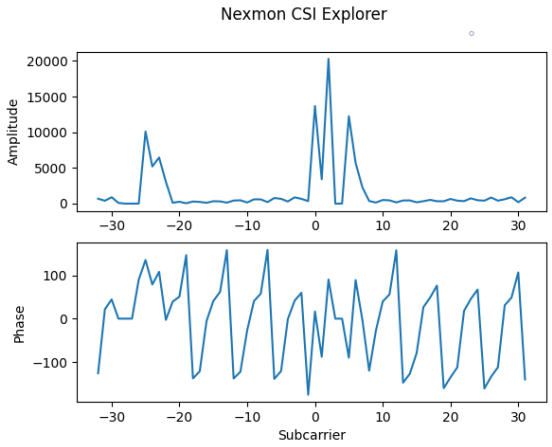


Figure 4.3: Example of CSI captured on the Raspberry Pi

Next, an Intel NUC running Linux was used. This client was running the Atheros CSI tool. This is an open source tool which allows the extraction of information of the physical layer of a Wi-Fi chip, including CSI. The tool is built upon the open source Ath9k kernel driver for Atheros 802.11n chips [9]. This tool allows for the extraction of CSI while maintaining a connection to an access point. The CSI from this node is used to check if measurements and predictions are similar for both this client and for the CSI collected on the Openwifi board. In order to process the captured data the CSIKit python library was used [39].

4.2 CSI fuzzer

The Openwifi project also has the capability to introduce an artificial factor to the CSI. When no artificial component is added, the CSI depicts the influence of the environment on the signal. For a system as shown in figure 4.4 the received signal for a given subcarrier k can be modeled as [40]:

$$r(k) = \underbrace{H_{env}(k)}_{CSI(k)} s(k) + n(k) \tag{4.1}$$

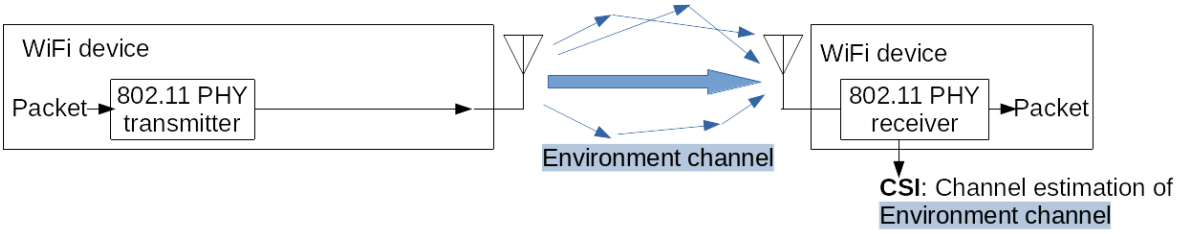


Figure 4.4: Channel estimation of environmental channel

Source: [14]

When the fuzzer is activated, an artificial component is added to the CSI, as can be seen in figure 4.5. For entities that do not know this artificial component, it is not possible to regain the environmental CSI. This mitigates the problem that an unauthorised receiver could use the CSI for environmental sensing. With the fuzzer activated, the signal for a given subcarrier k can now be modeled as [40]:

$$r(k) = \underbrace{H_{env}(k)H_{art}(k)}_{CSI(k)} s(k) + n(k) \tag{4.2}$$

To regain the environmental component, the receiver has to know $H_{art}(k)$. When ignoring the noise, the environmental component can be estimated as following [40]:

$$H_{env}(k) = \frac{CSI(k)}{H_{art}(k)} \tag{4.3}$$

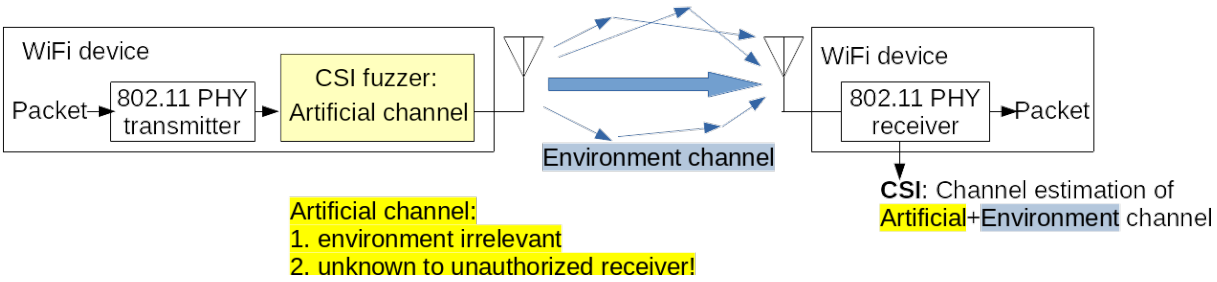


Figure 4.5: Channel estimation of environmental + artificial channel

Source: [14]

An example of fuzzed CSI can be seen in figure 4.6. The difference between this graph and figure 4.2 is very clear.

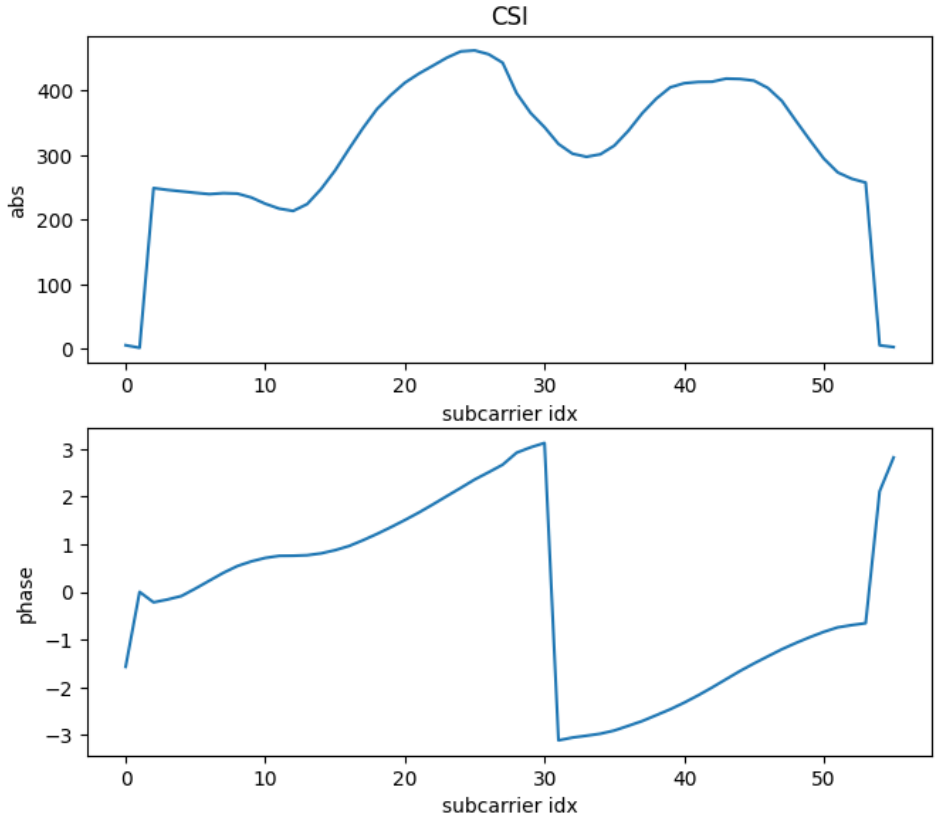


Figure 4.6: Example of fuzzed CSI

4.2.1 Fuzzer implementation

According to the Openwifi documentation the fuzzer is implemented as a 3-tap filter. The fuzzer takes in in-phase/quadrature (I/Q) data, right after the data is transformed to the time domain and the cyclic prefix is added. The fuzzer performs the operation shown in figure 4.7. c_1 and c_2 are the fuzzing parameters and define the fuzzing pattern that will be applied. The filter imposes $[1, c_1, c_2]$ as channel impulse response on the signal. To keep the fuzzer implementation simple, c_1 and c_2 are restricted to the range of $[-0.5, 0.5)$ or $[-0.5i, 0.5i)$ [40, 14].

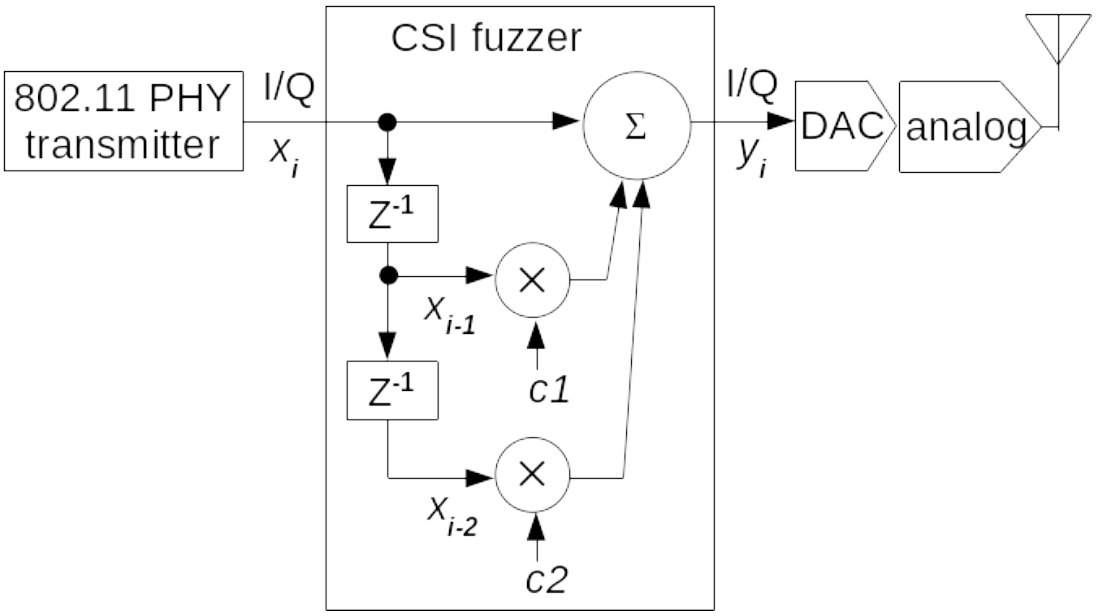


Figure 4.7: CSI fuzzer implementation
Source: [14]

c_1 and c_2 are derived from the parameter supplied to the user space program as following [14]:

```
csi_fuzzer.sh c1_rot90_en c1_raw c2_rot90_en c2_raw
```

Where

$$c_1 = \begin{cases} (c_1\_raw/128), c1_rot90_en = 0 \\ (c_1\_raw/128) * i, c1_rot90_en = 1 \end{cases} \quad (4.4)$$
$$c_1\_raw = -64, -63, \ldots, 0, 1, \ldots, 62, 63$$

The same applies for c_2 [14].

5

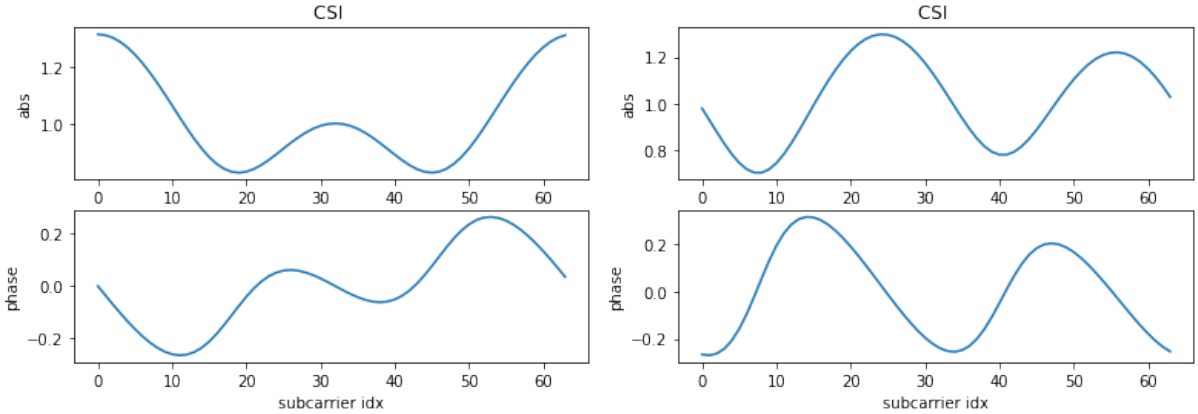
Defuzzing

The main goal of this thesis is to predict the influence the fuzzer has on the CSI and to undo this influence, regaining the environmental CSI. In this chapter, a method based on the prediction of the effects via the fuzzer's parameters is discussed. Next an OFDM-simulation is discussed. This simulation allows for the testing of the previously mentioned method in a controlled environment. Finally a learning based method is devised and the results are discussed.

5.1 Fuzzing pattern prediction

In order to regain the environmental CSI from the measured CSI, the influence of the fuzzer has to be predicted. Once this influence is known, the environmental CSI can be regained as shown in equation 4.3. According to signal processing theory, the influence of the fuzzer is the DFT of $[1, c_1, c_2, 0, \ldots]$ where the total length of the array from which the DFT is calculated is equal to the number of subcarriers used, thus 64 in this case. This will result in 64 complex numbers, where each one is the artificial channel coefficient for the corresponding subcarrier.

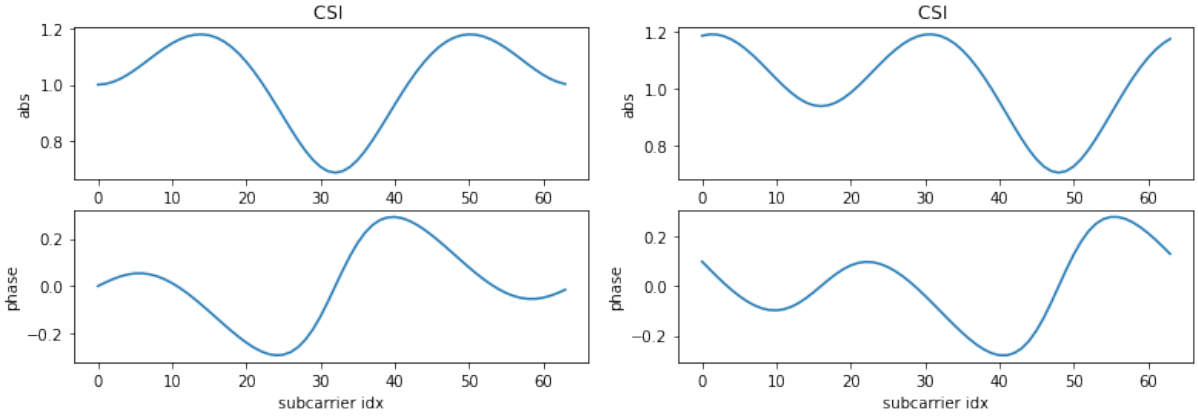
Figure 5.1 shows the predicted patterns for the Openwifi CSI samples shown in 5.3b and 5.3c



(a) Fuzzing pattern prediction, fuzzing parameters $0\ -20\ 0\ 20$ (b) Fuzzing pattern prediction, fuzzing parameters $0\ 7\ 1\ -33$

Figure 5.1: Openwifi fuzzing predictions

Figure 5.2 shows the predicted patterns for the Atheros CSI tool samples shown in 5.4b and 5.4c.



(a) Fuzzing pattern prediction, fuzzing parameters $0\ -20\ 0\ -20$ (b) Fuzzing pattern prediction, fuzzing parameters $1\ -15\ 0\ 23$

Figure 5.2: Atheros CSI tool fuzzing predictions

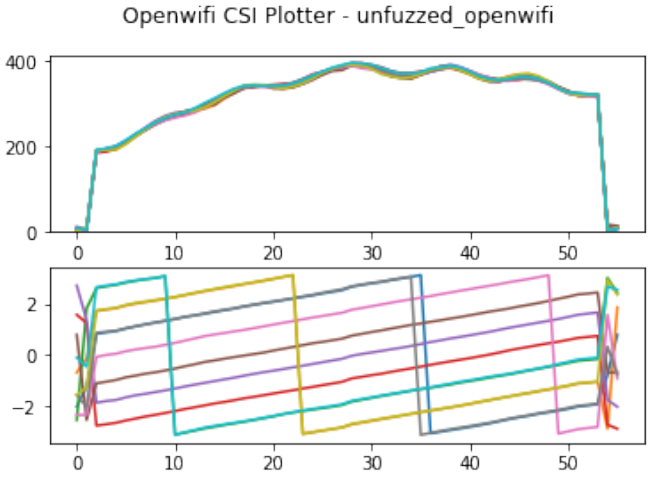
5.1.1 Estimation of the actual applied fuzzing pattern

In order to validate the prediction, it is necessary to compare it to the actual observed fuzzing patterns. In order to do this, $H_{art}(k)$, the actual applied fuzzing pattern should be estimated. This can be done as following:

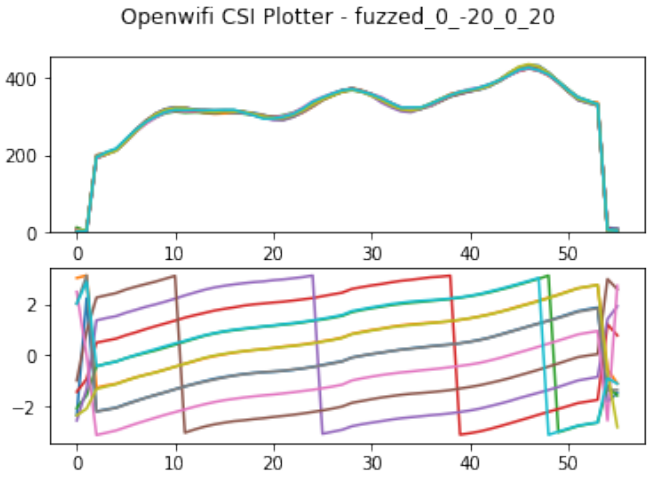
$$H_{art}(k) = \frac{CSI_{fuzzed}(k)}{CSI_{unfuzzed}(k)} \tag{5.1}$$

$H_{art}(k)$ can be estimated when $CSI(k)$ as well as $H_{env}(k)$ are known. To achieve this, CSI was measured when no fuzzing was applied, this data serves as H_{art}. Then several data was collected where different fuzzing parameters were used. Figure 5.3 shows CSI from 10 consecutive CSI samples with the fuzzer turned off and for two different sets of fuzzing parameters. The numbers in the title represent the fuzzing parameters used and are in the same order as they should be provided to the user space program. The $x - axis$ in the graphs represents the subcarrier index, the top sub-graph shows the amplitude and the bottom one the phase. The amplitudes of the different samples are all very similar, but the phase is quite different.

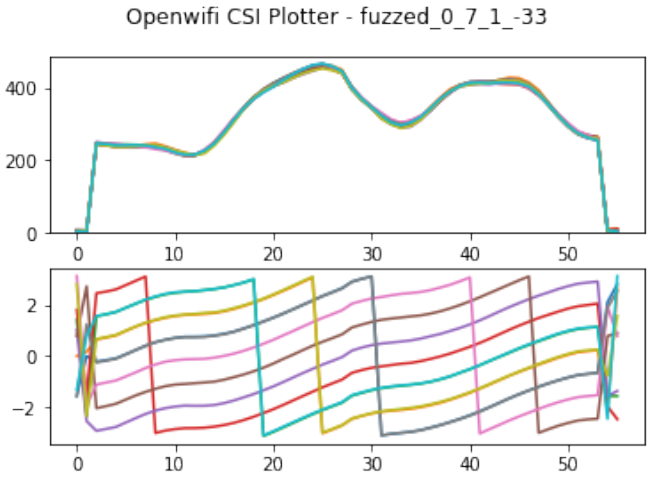
Figure 5.4 shows similar graphs for CSI captured with the client running the Atheros CSI tool. It is clear from the graphs that the CSI is a lot noisier here, but in contrary to the CSI collected on the Openwifi board, the phase remains quite constant for consecutive CSI samples.



(a) Unfuzzed CSI

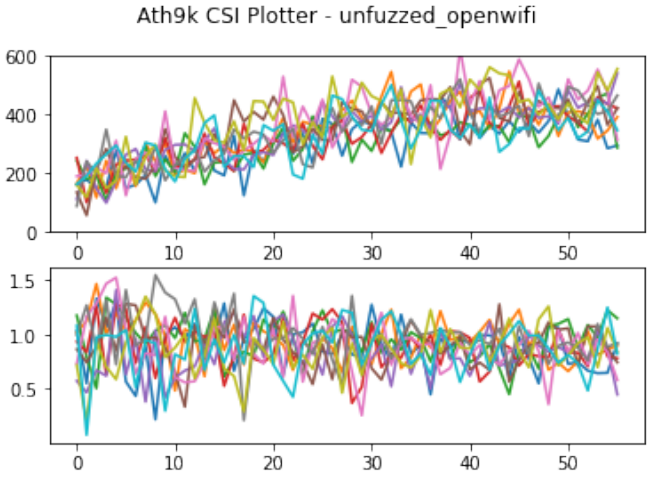


(b) Fuzzed CSI, fuzzing parameters $0\ -20\ 0\ 20$

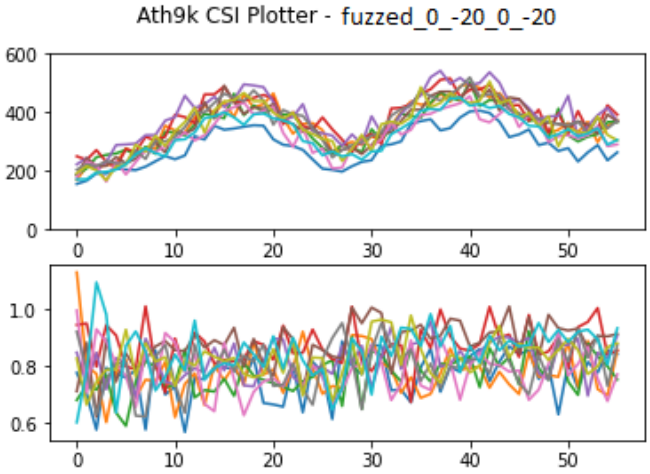


(c) Fuzzed CSI, fuzzing parameters $0\ 7\ 1\ -33$

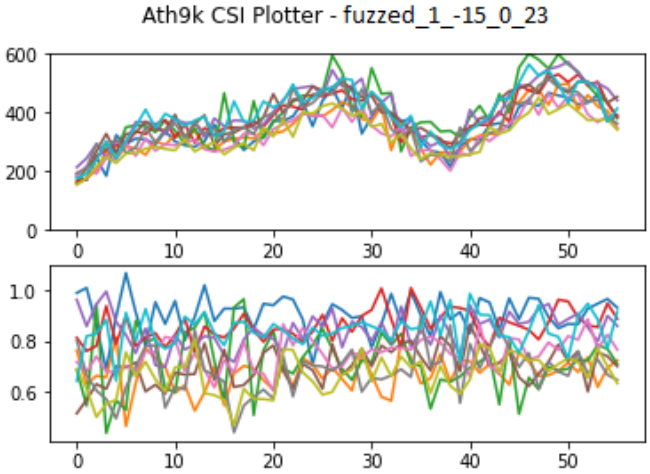
Figure 5.3: CSI graphs



(a) Unfuzzed CSI



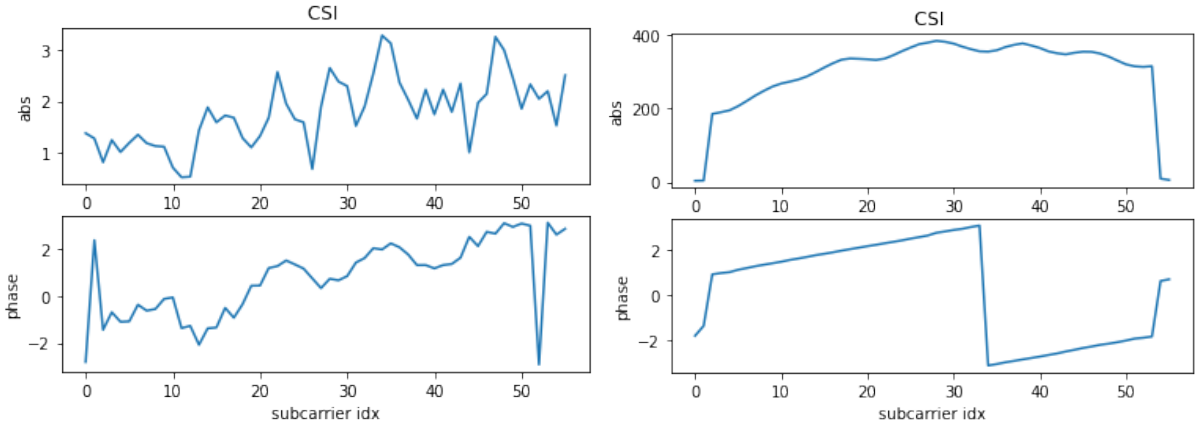
(b) Fuzzed CSI, fuzzing parameters $0\ -20\ 0\ -20$



(c) Fuzzed CSI, fuzzing parameters $1\ -15\ 0\ 23$

Figure 5.4: CSI graphs

In order to reduce the noise, several CSI samples can be averaged out. In case of the CSI collected on the openwifi board, it is important that CSI samples with similar phase are averaged. Otherwise the different phases will cause the different samples to cancel each other out and average near 0, as can be seen in 5.5a. When phase is taken into account while averaging, the result is as expected. The same caution applies when averaging fuzzed CSI samples.



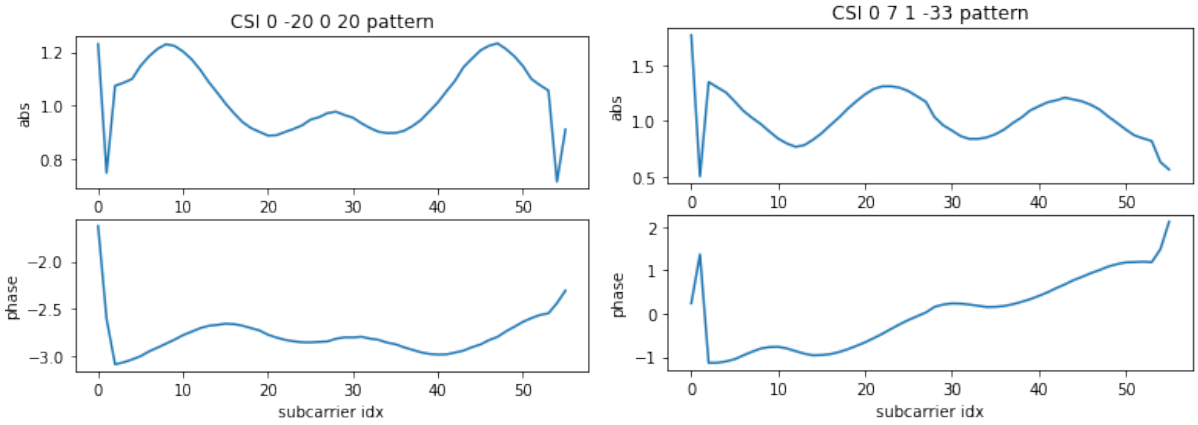
(a) Wrongly calculated CSI average

(b) Rightly calculated CSI average

Figure 5.5: Wrong and right CSI average

Figure 5.6 shows the fuzzing patterns applied for the 5.3b and 5.3c samples respectively. These patterns are calculated by dividing averaged fuzzed CSI samples by averaged unfuzzed CSI samples. This results in an estimation of H_{art}.

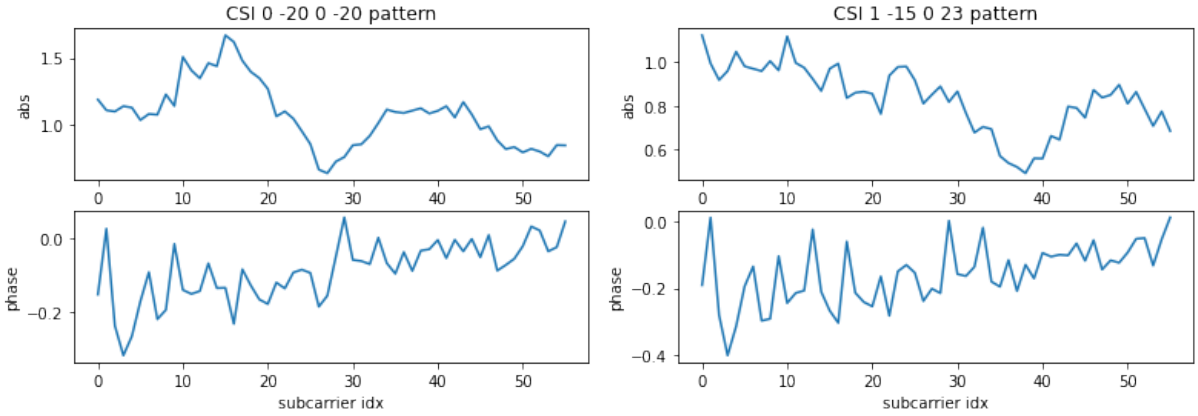
For the CSI samples collected by the Atheros CSI tool, the averaging is even more important, as these samples are a lot more noisy. Here it is not necessary to select samples with a similar phase pattern, as all samples have a similar phase pattern. Figure 5.7 shows the fuzzing patterns derived in the same way as for the Openwifi CSI samples, but now for CSI samples collected by the Atheros CSI tool. Again, it is clear that the CSI collected by the Atheros CSI tool is a lot noisier, even after averaging several samples.



(a) Fuzzing pattern, fuzzing parameters $0\ -20\ 0\ 20$

(b) Fuzzing pattern, fuzzing parameters $0\ 7\ 1\ -33$

Figure 5.6: Fuzzing patterns from Openwifi CSI samples



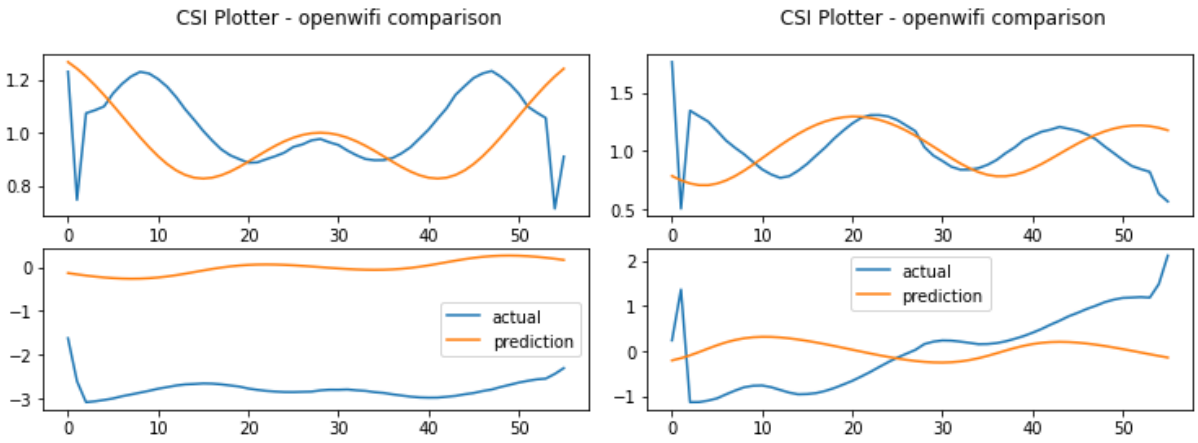
(a) Fuzzing pattern, fuzzing parameters $0\ -20\ 0\ -20$

(b) Fuzzing pattern, fuzzing parameters $1\ -15\ 0\ 23$

Figure 5.7: Fuzzing patterns from Atheros CSI tool samples

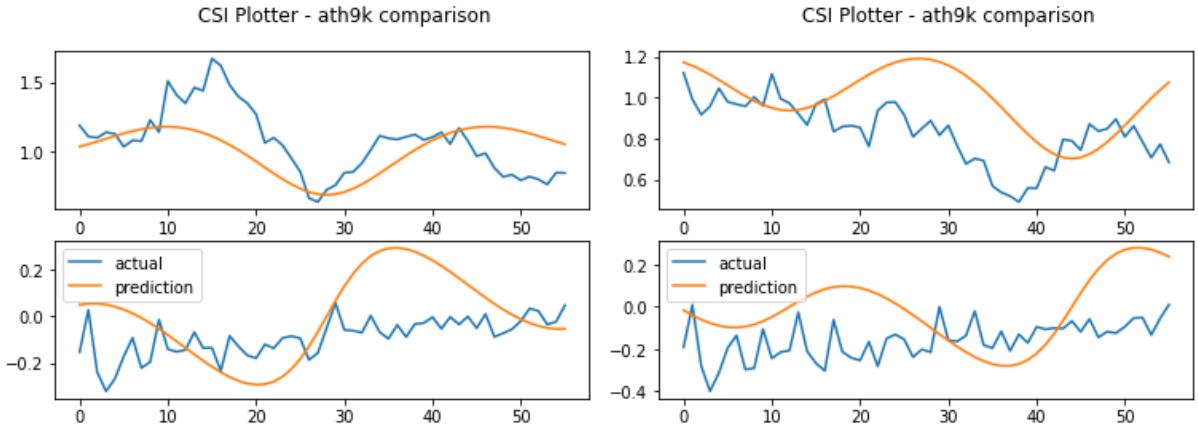
5.1.2 Validation

Now that an estimation of the actual applied fuzzing pattern is calculated, it can be compared to the predicted fuzzing pattern. Figure 5.8 and figure 5.9 each show the actual observed fuzzing pattern compared to the prediction for both sets of fuzzing parameters. When comparing the prediction with the actual observed pattern, it is immediately clear that they do not match.



(a) Fuzzing pattern comparison, fuzzing parameters $0\ -20\ 0\ 20$ pattern (b) Fuzzing pattern comparison, fuzzing parameters $0\ 7\ 1\ -33$

Figure 5.8: Prediction vs. actual observed fuzzing pattern for Openwifi CSI samples



(a) Fuzzing pattern comparison, fuzzing parameters $0\ -20\ 0\ -20$ pattern (b) Fuzzing pattern comparison, fuzzing parameters $1\ -15\ 0\ 23$

Figure 5.9: Prediction vs. actual observed fuzzing pattern for Atheros CSI tool samples

5.1.3 Simulation

To verify that the prediction should match the applied fuzzing pattern in theory, an OFDM simulation is used where the fuzzing is applied in a controlled environment. Appendix A contains the python code used for the simulation, the code is based on [41]. Figure 5.10 shows the general flow of the simulation.

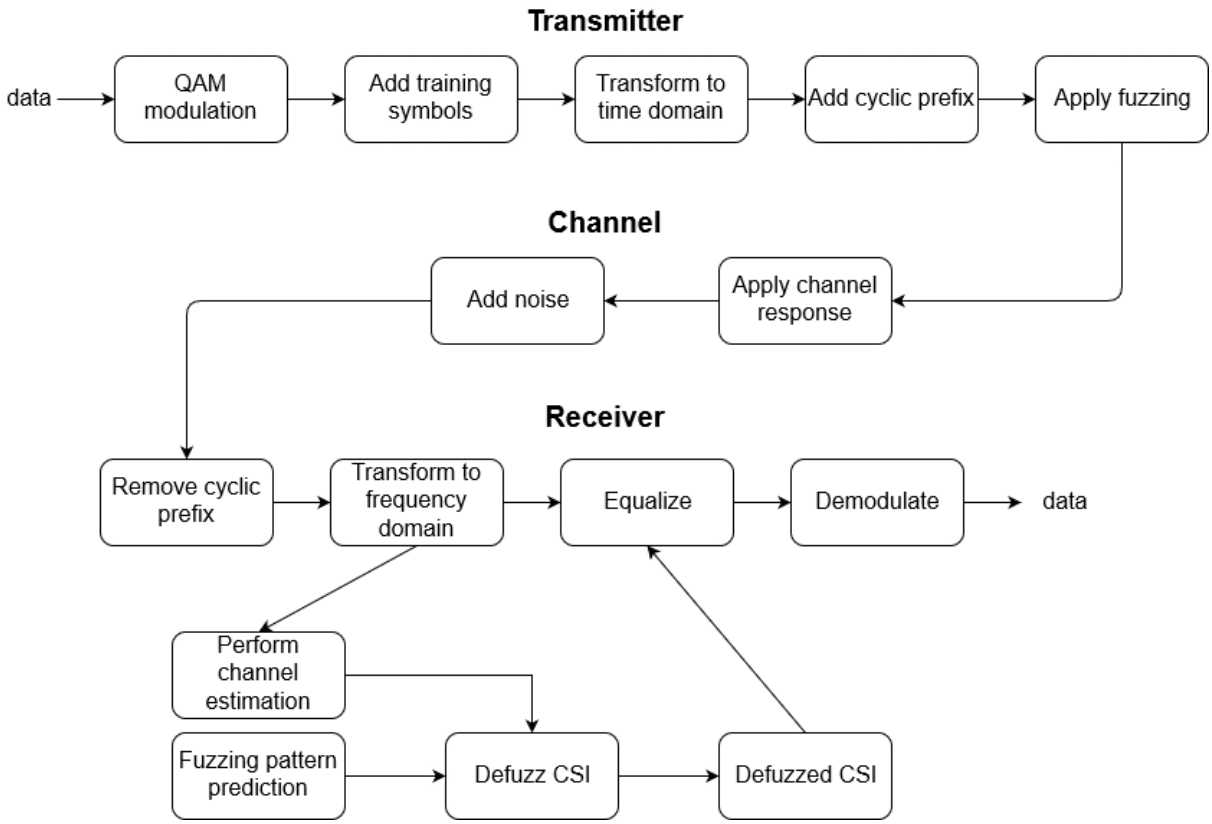
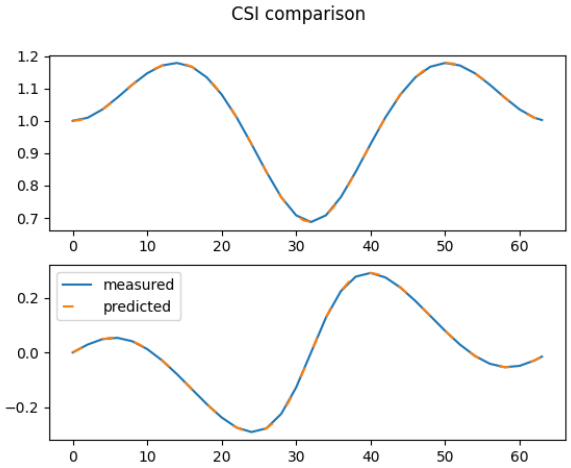


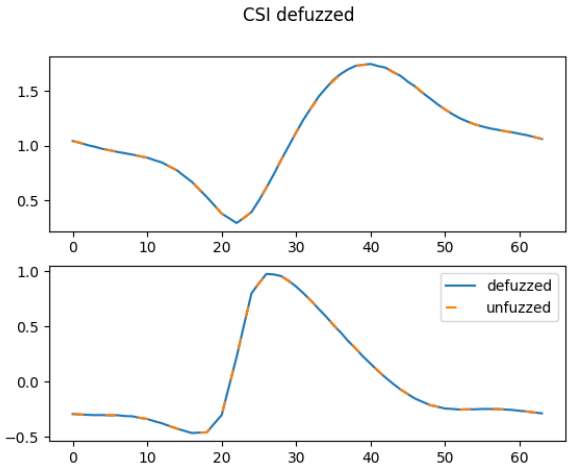
Figure 5.10: OFDM simulation flowchart

Figure 5.11 shows the results of the simulation for fuzzing parameters $0\ -20\ 0\ -20$. Graph 5.11a shows the measured CSI when no channel or noise were applied as well, as the fuzzing pattern predicted via the DFT of the fuzzing parameters. The only difference between both is due to the comb type channel estimation used for estimating the CSI and the interpolation used in that process. Graphs 5.11b and 5.11c show the result of defuzzing the measured CSI, via the method described in equation 4.3, when the channel response is applied and both the channel response and random noise are applied, respectively. Figure 5.12b shows the same results, but now for fuzzing parameters $1\ -15\ 0\ 23$. Again, the prediction and the observed fuzzing effect are almost an exact match.

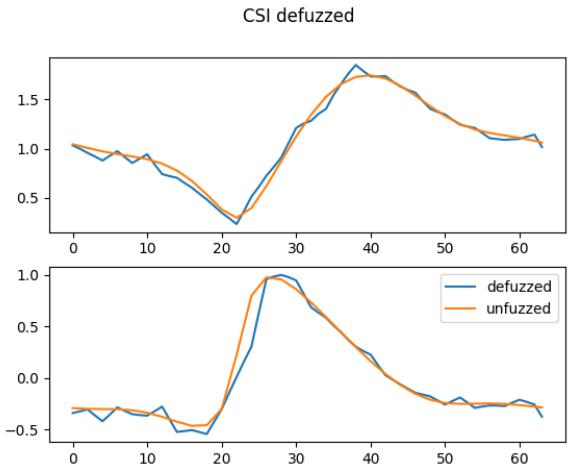
The difference between the predicted and observed fuzzing pattern is not present in the simulation. This means that there is another factor to the CSI fuzzer that is currently not documented. Due to this unknown factor, causing the discrepancy, it is not possible to predict the fuzzing pattern and use it for the defuzzing. The simulation is entirely software based, it could be that the fuzzer's hardware has an additional unknown effect on the CSI.



(a) Fuzzing pattern

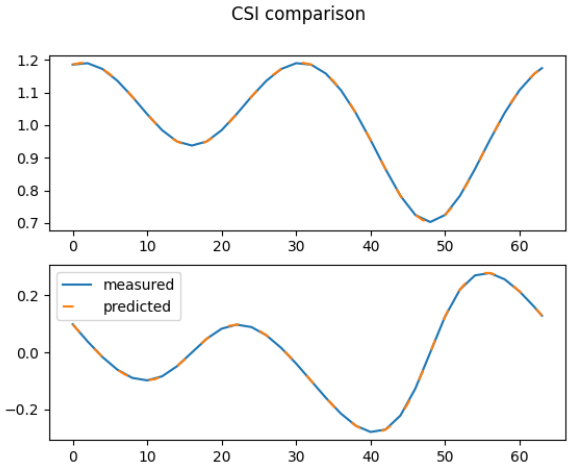


(b) Defuzzed CSI, no noise applied applied

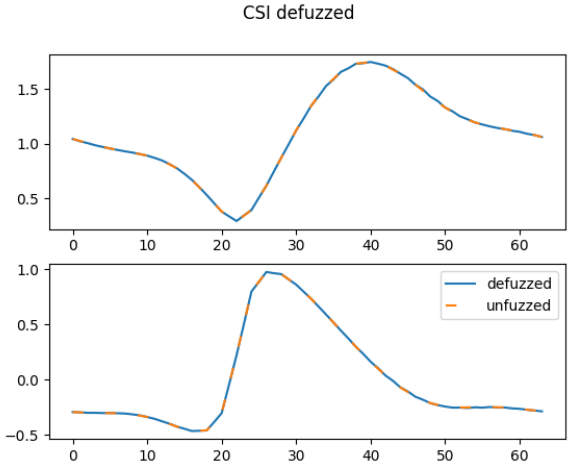


(c) Defuzzed CSI with noise applied

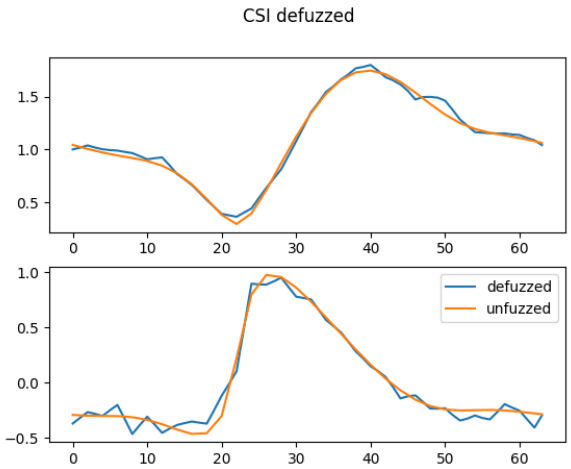
Figure 5.11: Simulation results



(a) Fuzzing pattern



(b) Defuzzed CSI, no noise applied applied



(c) Defuzzed CSI with noise applied

Figure 5.12: Simulation results

5.2 Learning based defuzzing

Predicting the fuzzing pattern is not feasible without knowing the additional, currently unknown, factor. A different approach, which circumvents that problem, is to generate the fuzzing pattern from a set of unfuzzed and a set of fuzzed CSI samples. This method was already used in the previous section to check if the predictions match the observed fuzzing pattern.

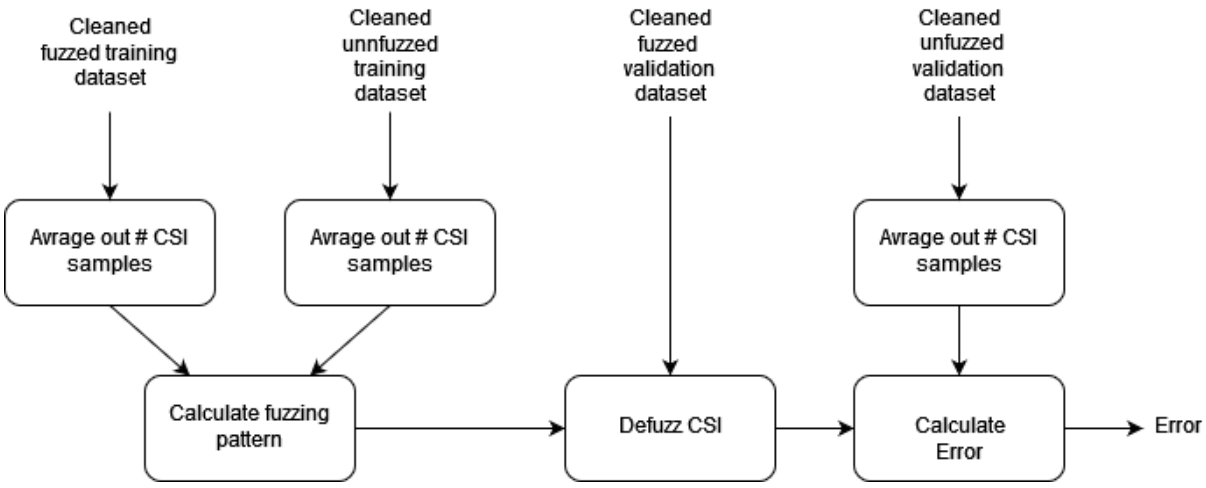


Figure 5.13: Learning based defuzzing flowchart

Figure 5.13 shows the general flow of the learning based defuzzing process. First, the fuzzing pattern is calculated based on the averaged out training samples. Next, this obtained defuzzing pattern is used to defuzz a packet of the validation dataset. After the defuzzing the result is checked against the averaged out unfuzzed validation dataset. The error is calculated as following:

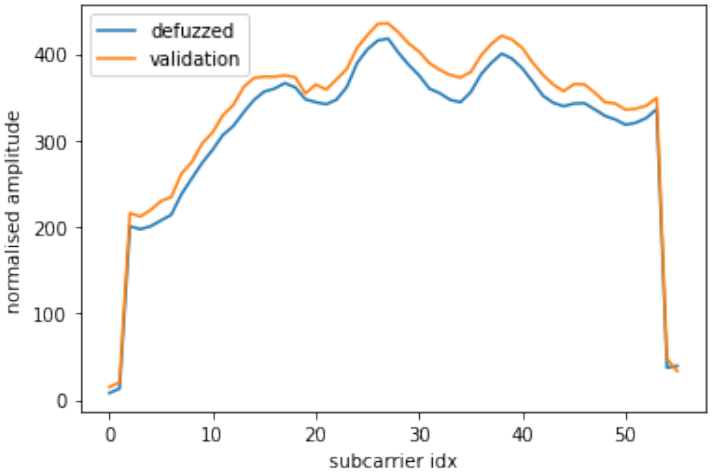
$$error = 10 * \log_{10}\left(\frac{\frac{1}{s}\sum_{i=0}^{s}|y_i - x_i|}{\frac{1}{s}\sum_{i=0}^{s}|y_i|}\right) = 10 * \log_{10}\left(\frac{\sum_{i=0}^{s}|y_i - x_i|}{\sum_{i=0}^{s}|y_i|}\right) \tag{5.2}$$

with s as the number of subcarriers.

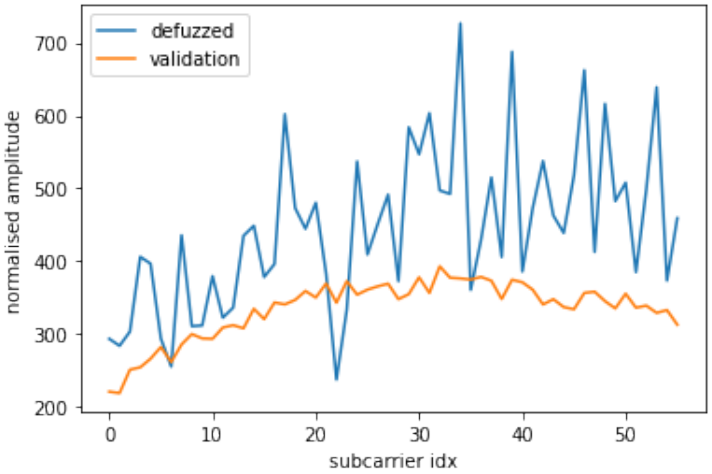
The error is the ratio of the average difference between the defuzzed sample x and the validation CSI sample y, and the average magnitude of y, expressed in decibels.

Appendix B contains the python code used to calculate this for 100 deffuzed Openwifi CSI samples. It does this multiple times, each time for an increased amount of averaged out CSI samples, starting with just a single sample. This will give an idea of how many packets should be averaged out to get a consistent result. For datasets collected by the Atheros CSI tool the code is completely analogous to the provided code, the only difference is how the data is read from the files.

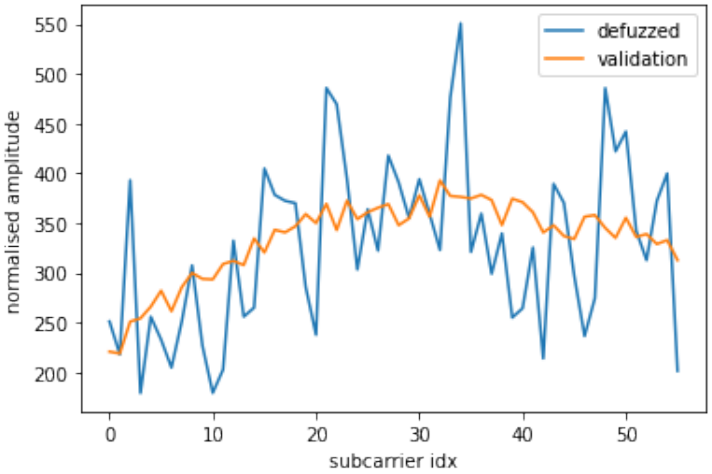
In order to visualise the performance of the learning based defuzzing, figure 5.14 shows the results of a single defuzzed CSI sample compared to the averaged validation sample. The first graph shows this for CSI collected on the Openwifi board. The similarities between both lines are very clear. The second graph shows the same but for CSI collected on via the Atheros CSI tool. Here it is very clear that there is a lot of noise present and it is hard to tell how well the defuzzing works. The third graph shows a unfuzzed CSI sample compared to the averaged sample, this gives an idea of the noise level present in the CSI, this is a bit less then the level of noise that can be observed in the second graph.



(a) Defuzzed Openwifi CSI sample vs validation average

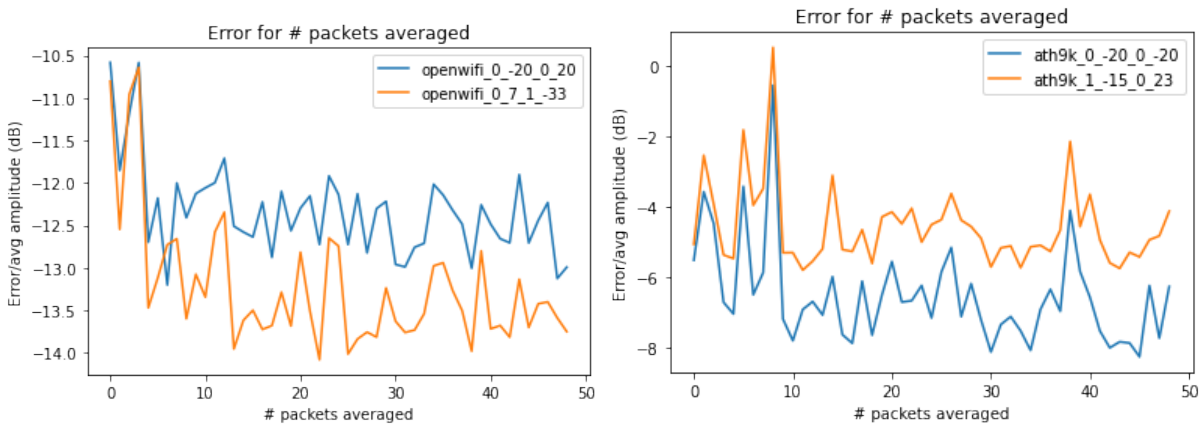


(b) Defuzzed Atheros CSI tool sample vs validation average



(c) Unfuzzed Atheros CSI tool sample vs validation average

Figure 5.14: Defuzzed CSI sample vs validation average



(a) Defuzzing performance of Openwifi samples

(b) Defuzzing performance of Atheros CSI tool samples

Figure 5.15: Defuzzing performance in function of the # of packets averaged

Figure 5.15 shows two graphs where the error, as calculated in 5.2, is plotted in function of the amount of CSI samples that were averaged together for two different sets of fuzzing parameters. When analysing the graphs, several things stand out. First of all, it seems that in both graphs, both lines follow a similar pattern. This can be explained by the fact that for both lines, the evaluation check is happens with the same validation CSI matrix, introducing some correlation to the results. The second thing that stands out is that the defuzzing is significantly better for Openwifi CSI samples then for Atheros CSI tool samples. This is according to the expectations, because the CSI collected by the Atheros CSI tool is a lot noisier. Thirdly, averaging out more then roughly 10 packets doesn't seem to increase the stability of the defuzzing any further.

The graph showing the results for the Openwifi samples varies roughly between $-12dB$ and $-14dB$. This means that the average error between the defuzzed sample and the validation CSI matrix is 15 to 25 times smaller then the average amplitude. For the Atheros CSI tool samples the results varies roughly between $-4dB$ and $-8dB$, meaning that the average error is about 2.5 to 6.5 times smaller then the average amplitude.

In order to increase the stability, and increase the general performance the CSI samples can be normalised. The code in appendix B does not normalise the CSI samples but the method to

do so is defined. The normalisation that is applied is the following:

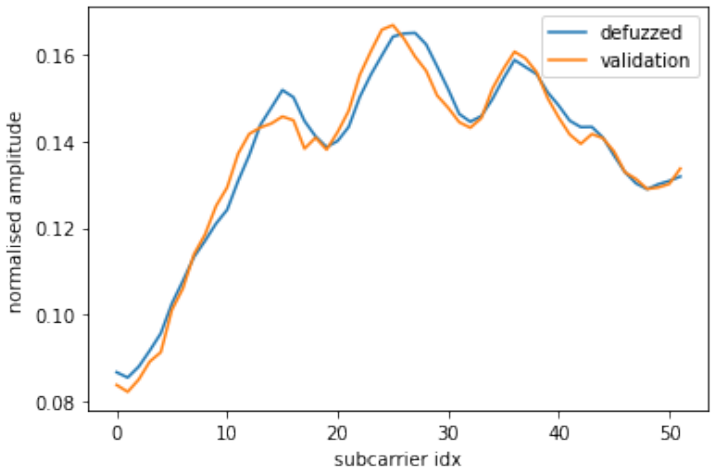
$$normalised\_csi = \frac{csi\_matrix}{\sqrt{\sum_{i=0}^{s} |csi\_matrix_i|^2}} \text{ with } s \text{ equal to the number of subcarriers} \tag{5.3}$$

This has the result that:

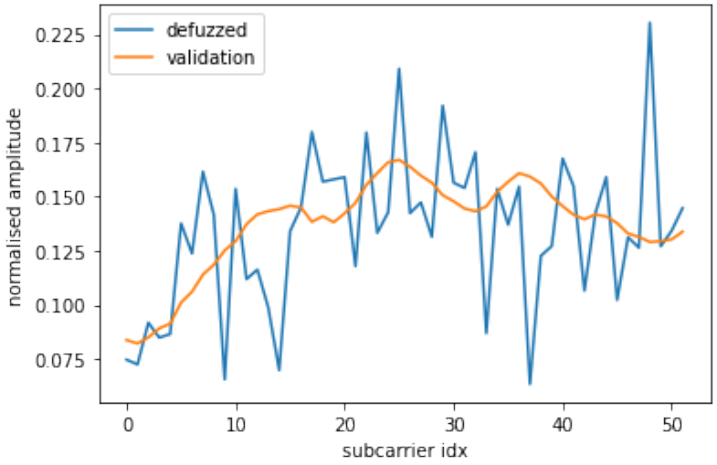
$$\sum_{i=0}^{s} |csi\_matrix_i|^2 = 1 \text{ with } s \text{ equal to the number of subcarriers} \tag{5.4}$$

Figure 5.16 shows a single normalised defuzzed packet against the normalised validation average for both the Openwifi and the Atheros CSI tool collected data. This helps to visualise the performance achieved via this method. For the Openwifi CSI sample it is easy to see that the defuzzing works very well, for the Atheros CSI tool sample this is not the case. However, the third graph shows an unfuzzed CSI sample compared to the validation average. It is clear from that graph the noise level very high and that it is comparable to the noise level of the defuzzed sample as shown in the second graph.

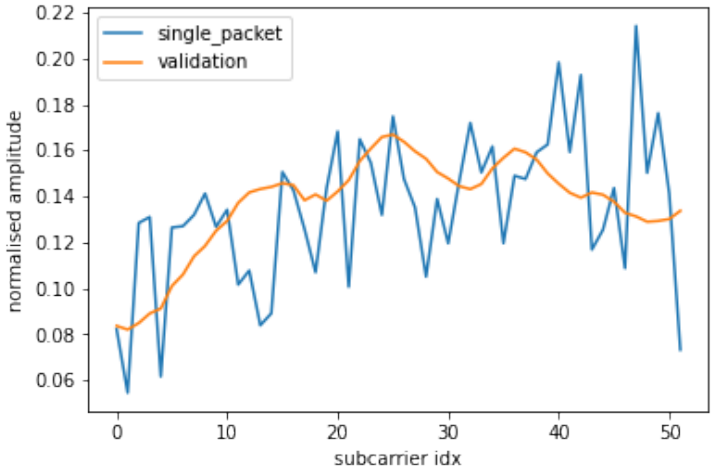
The defuzzing results after normalisation can be seen in figure 5.17. When comparing both graphs to the previous graphs where no normalisation was applied, it becomes clear that the results have improved. For the Openwifi samples, the average error for both fuzzing patterns fluctuates around $-15.5dB$ meaning that the mean error is about a 30 times smaller then the average amplitude. For the Atheros CSI tool samples, the mean error is a lot more stable and fluctuates around $7.4dB$. This means that the mean error here is about a 5.5 times smaller then the average amplitude. In both graphs there is also a green line present. This green line is the result of calculating the error in the same way as described previously, but now for unfuzzed CSI samples. This means that the green line represents the expected error level due to noise. It is clear that the error of the defuzzed samples approach the error level that is present due to noise. This confirms the observations from 5.16.



(a) Defuzzed Openwifi CSI sample vs validation average

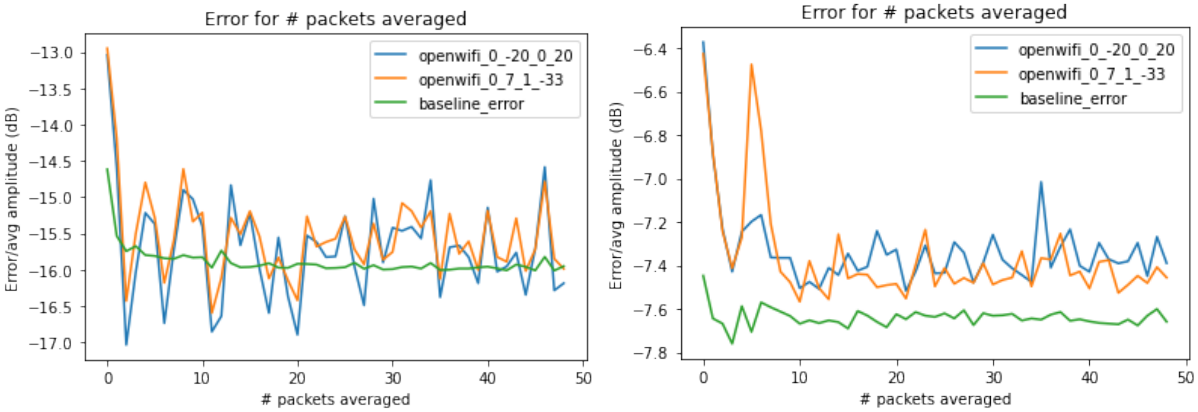


(b) Defuzzed Atheros CSI tool sample vs validation average



(c) Unfuzzed Atheros CSI tool sample vs validation average

Figure 5.16: Defuzzed CSI sample vs validation average



(a) Defuzzing performance of Openwifi samples after normalisation(b) Defuzzing performance of Atheros CSI tool samples after normalisation

Figure 5.17: Defuzzing performance in function of the # of packets averaged after normalisation

It has to noted that the 100 packets that are defuzzed are chosen randomly at runtime from the dataset, meaning that these and the previous do not represent the results for the same dataset with and without normalisation.

5.3 Remarks

Because of the unknown factor during the fuzzing, it is currently not possible to use the prediction based method in a simple protocol where the parameters are send from the transmitter to the authorised receiver. However, if the unknown factor can be taken into account in the mathematical model or if it can be removed altogether, this method would be preferable. At the momement however, this is not the case. Instead, if a fuzzing-defuzzing protocol is to be designed, the transmitter would have to start by transmitting a certain amount of packets where the fuzzer is not active, followed by a certain amount of packets where the fuzzer is active. During the transmission of these packets, the training period, the channel has to remain static as the CSI has to be averaged. Once the fuzzing pattern has been calculated from the training data, the channel can be dynamic.

This setup, with the training period, also means that it is still possible for an unauthorised receiver to perform CSI sensing. The attack vector has become time sensitive, as the attacker has to collect the CSI during the training phase, but it is not impossible. For that reason, additional research into the cause of the unexpected behavior of the fuzzer is desirable.

Conclusion

This thesis researched two mechanism to undo the effects imposed by the CSI fuzzer, included in the Openwifi project, with view to eventually implement such a method in a protocol. First the implementation of the fuzzer was discussed, in order to build a mathematical model of the imposed effects. Based on this mathematical model, a defuzzing method was proposed. This method only needs the fuzzing parameters in order to predict the applied effects. When validating the mathematical prediction against the observed effect, it became clear that these did not match. To rule out mistakes in the method, an OFDM simulation was used. This simulation showed that the method works in a controlled environment, meaning that there is an unknown factor causing the difference between the prediction and the observed effect. To circumvent this problem, a learning based defuzzing method is proposed, where the fuzzer's effects are calculated from collected CSI. This method performed quite good, as the introduced error approached the level of noise already present in the dataset.

Because there is an unknown factor during the fuzzing, it is currently not possible to design a simple protocol based on the exchange of the fuzzing parameters. However, if this unknown factor can be taken into account in the mathematical model, or if it can be removed altogether, such a protocol would be possible.

At the moment, if such a protocol were to be designed. It would need to have a learning phase, during which the channel has to remain static. As this training phase is used to generate the fuzzing pattern, it is possible for an unauthorised entity to generate the fuzzing pattern as well. The attack vector now has a time sensitive factor to it, but is not entirely mitigated.

Ethical and social reflection

The fuzzer provides enhanced privacy, by actively imposing a change in the I/Q data and thus the CSI, while being equally efficient in terms of throughput and bandwith. The fuzzer itself is a simple FIR implemented in hardware meaning it is much more energy efficient then any software equivalent. When taking energy consumption into account for the design of a fuzzing-defuzzing protocol, the defuzzing process based on the prediction via the fuzzing parameters should be preferred because he prediction of the fuzzing pattern happens via a DFT, which can be performed efficiently in hardware via the. The learning based approach requires a lot more processing and thus is harder to implement in hardware and would be less energy efficient.

If, after future research into the currently unknown factor, the prediction based method is used, the fuzzing-defuzzing protocol would be very energy efficient which is in line with the twelfth goal of the Sustainable Development Goals (SDGs), promoting a more efficient usage of energy and resources.

References

[1] Y. Ma, G. Zhou, and S. Wang, "Wifi sensing with channel state information: A survey," *ACM Comput. Surv.*, vol. 52, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3310194

[2] Z. Zhenghao, "Enhancing wi-fi communication with effective csi approximations," Oct 2019. [Online]. Available: https://researchoutreach.org/articles/enhancing-wi-fi-communication-with-effective-csi-approximations/

[3] F. Adib and D. Katabi, "See through walls with wifi!" in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 75–86. [Online]. Available: https://doi.org/10.1145/2486001.2486039

[4] S. Arshad, C. Feng, Y. Liu, Y. Hu, R. Yu, S. Zhou, and H. Li, "Wi-chase: A wifi based human activity recognition system for sensorless environments," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017, pp. 1–6.

[5] H. Li, W. Yang, J. Wang, Y. Xu, and L. Huang, "Wifinger: Talk to your smart devices with finger-grained gesture," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 250–261. [Online]. Available: https://doi.org/10.1145/2971648.2971738

[6] Y. Ma, G. Zhou, S. Wang, H. Zhao, and W. Jung, "Signfi: Sign language recognition using wifi," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, no. 1, mar 2018. [Online]. Available: https://doi.org/10.1145/3191755

[7] P. Melgarejo, X. Zhang, P. Ramanathan, and D. Chu, "Leveraging directional antenna capabilities for fine-grained gesture recognition," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 541–551. [Online]. Available: https://doi.org/10.1145/2632048.2632095

[8] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When csi meets public wifi: Inferring your mobile phone password via wifi signals," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1068–1079. [Online]. Available: https://doi.org/10.1145/2976749.2978397

[9] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity wifi," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: ACM, 2015, p. 53–64. [Online]. Available: http://doi.acm.org/10.1145/2789168.2790124

[10] M. Schulz, D. Wegemer, and M. Hollick. (2017) Nexmon: The c-based firmware patching framework. [Online]. Available: https://nexmon.org

[11] F. Gringoli, M. Schulz, J. Link, and M. Hollick, "Free your csi: A channel state information extraction platform for modern wi-fi chipsets," in *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, ser. WiNTECH '19, 2019, p. 21–28. [Online]. Available: https://doi.org/10.1145/3349623.3355477

[12] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: Gathering 802.11n traces with channel state information," *ACM SIGCOMM CCR*, vol. 41, no. 1, p. 53, Jan. 2011.

[13] F. Gringoli, M. Cominelli, and R. Lo Cigno, "Ans - advanced networking systems group," Aug 2020. [Online]. Available: https://ans.unibs.it/assets/documents/ORCA_CSI-MURDER_Report_v1.1.pdf

[14] X. Jiao, W. Liu, and M. Mehari. (2019) open-source ieee802.11/wi-fi baseband chip/fpga design. [Online]. Available: https://github.com/open-sdr/openwifi

[15] R. Jain, "Introduction to wireless signal propagation," 2014. [Online]. Available: https://www.cse.wustl.edu/~jain/cse574-14/ftp/j_04wsp.pdf

[16] R. W. Heath, *Introduction to wireless digital communication: A Signal Processing Perspective*. Prentice Hall, 2017.

[17] "Radio wave diffraction." [Online]. Available: https://www.electronics-notes.com/articles/antennas-propagation/propagation-overview/radio-em-wave-diffraction.php

[18] "How does wireless technology work: Learning center: Core wireless," Dec 2017. [Online]. Available: https://corecabling.com/5-signal-interference-network-connectivity-problems/

[19] J. Cole, "Helmhurts," Aug 2014. [Online]. Available: https://jasmcole.com/2014/08/25/helmhurts/

[20] "multipath propagation." [Online]. Available: https://sourcedaddy.com/networking/multipath-propagation.html

[21] R. Gasim and M. Zohdy, "Low complexity dynamic channel equalization in ofdm with high frequency mobile network technologies," *Journal of Signal and Information Processing*, vol. 8, pp. 17–41, 05 2017.

[22] A. Lavaa, "What is frequency division multiplexing - fdm? full guide," Sep 2021. [Online]. Available: https://www.linquip.com/blog/what-is-frequency-division-multiplexing-fdm/

[23] S. Sun. (2018, Oct) Ofdm - orthogonal frequency division multiplexing. Youtube. [Online]. Available: https://www.youtube.com/watch?v=KCH07zlU25Q

[24] D. Regev. (2019, May) Qam and ofdm basics. Youtube. [Online]. Available: https://www.youtube.com/watch?v=qsmYD_6AeIU

[25] "Ieee 802.11n wlan standard." [Online]. Available: https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/802-11n.php

[26] M. B. Khalilsarai, S. Stefanatos, G. Wunder, and G. Caire, "Wifi-based indoor localization via multi-band splicing and phase retrieval," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019, pp. 1–5.

[27] A. K. Jagannatham. Lecture 34: Multiple input multiple output (mimo) systems. Youtube. [Online]. Available: https://www.youtube.com/watch?v=g84f4mP0u30

[28] W. Abbas, N. Abbas, U. Majeed, and S. Khan, "Efficient stbc for the data rate of mimo-ofdma," *Science International Journal 1013-5316*, vol. 28, pp. 247–255, 01 2016.

[29] A. K. Jagannatham. Lecture 35: Mimo receivers. Youtube. [Online]. Available: https://www.youtube.com/watch?v=g5g4AvhsAns&t=2s

[30] Y. Jiang, M. K. Varanasi, and J. Li, "Performance analysis of zf and mmse equalizers for mimo systems: An in-depth study of the high snr regime," *IEEE Transactions on Information Theory*, vol. 57, no. 4, pp. 2008–2026, 2011.

[31] S. Spillane, K. H. Jung, K. Bowers, T. Peken, M. H. Marefat, and T. Bose, "Machine learning based mimo equalizer for high frequency (hf) communications," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[32] A. Ajibade, O. Folorunso, and A. Ojo, "Channel estimation in mimo -ofdm wireless communication systems," 06 2019.

[33] N. S. S. Srinivas, "Ofdm system implementation, channel estimation and performance comparison of ofdm signal," in *2015 13th International Conference on Electromagnetic Interference and Compatibility (INCEMIC)*, 2015, pp. 212–219.

[34] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel estimation techniques based on pilot arrangement in ofdm systems," *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, 2002.

[35] M. Al-Shoukairi and B. D. Rao, "Semi-blind channel estimation in mimo systems with discrete priors on data symbols," *IEEE Signal Processing Letters*, vol. 29, pp. 51–54, 2022.

[36] R. Hoefel, "Ieee 802.11n: On the performance of channel estimation schemes over ofdm mimo spatially- correlated frequency selective fading tgn channels," 09 2012.

[37] "Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016.

[38] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source ieee802. 11 sdr implementation on soc," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–2.

[39] G. Forbes. (2021) Csikit: Python csi processing and visualisation tools for commercial off-the-shelf hardware. [Online]. Available: https://github.com/Gi-z/CSIKit

[40] X. Jiao, M. Mehari, W. Liu, M. Aslam, and I. Moerman, "Openwifi csi fuzzer for authorized sensing and covert channels," ser. WiSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 377–379. [Online]. Available: https://doi.org/10.1145/3448300.3468255

[41] BetterBench, "Ofdm-simulation-python," https://github.com/BetterBench/OFDM-simulation-Python/blob/main/ofdm_simulation.py, 11 2021.

Appendices

Appendix A

Python OFDM simulation:

```
import commpy
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

K = 64
CP = K//4
P = 32
pilotValue = 3+3j
Modulation_type = 'QAM64'
channel_type ='random'
SNRdb = 25
allCarriers = np.arange(K)
pilotCarrier = allCarriers[::K//P]


pilotCarriers = np.hstack([pilotCarrier, np.array([allCarriers[-1]])])
P = P+1
m_map = {"BPSK": 1, "QPSK": 2, "8PSK": 3, "QAM16": 4, "QAM64": 6}
mu = m_map[Modulation_type]
dataCarriers = np.delete(allCarriers, pilotCarriers)
payloadBits_per_OFDM = len(dataCarriers)*mu

enable_fuzzing = True
enable_channel = False
channel_coefs =np.array([1, 0.5j, 0.3j, 0.1j])

def modulate(bits):
    QAM64 = commpy.QAMModem(64)
    symbol = QAM64.modulate(bits)
    return symbol

def demodulate(symbol):
    QAM64 = commpy.QAMModem(64)
    bits = QAM64.demodulate(symbol, demod_type='hard')
    return bits

def add_noise(x_s, snrDB):
    data_pwr = np.mean(abs(x_s**2))
    noise_pwr = data_pwr/(10**(snrDB/10))
```

```
    noise = 1/np.sqrt(2) * (np.random.randn(len(x_s)) + 1j *
                            np.random.randn(len(x_s))) * np.sqrt(noise_pwr)
    return x_s + noise, noise_pwr

def channel(in_signal, SNRdb):
    out_signal = np.convolve(in_signal, channel_coefs)

    return out_signal

def fuzzing(in_signal, c1_raw, c1_rot, c2_raw, c2_rot):
    if c1_rot == 0:
        c1 = c1_raw/128
    else:
        c1 = complex(0,c1_raw/128)

    if c2_rot == 0:
        c2 = c2_raw/128
    else:
        c2 = complex(0,c2_raw/128)

    fuzzing = np.array([1,c1,c2])
    out_signal = np.convolve(in_signal, fuzzing)
    return out_signal

def OFDM_symbol(QAM_payload):
    symbol = np.zeros(K, dtype=complex)
    symbol[pilotCarriers] = pilotValue
    symbol[dataCarriers] = QAM_payload
    return symbol

def IDFT(OFDM_data):
    return np.fft.ifft(np.fft.ifftshift(OFDM_data))

def addCP(OFDM_time):
    cp = OFDM_time[-CP:]
    return np.hstack([cp, OFDM_time])

def removeCP(signal):
    return signal[CP:(CP+K)]

def DFT(OFDM_RX):
    return np.fft.fftshift(np.fft.fft(OFDM_RX))

def channelEstimate(OFDM_demod):
```

```
    pilots = OFDM_demod[pilotCarriers]
    Hest_at_pilots = pilots / pilotValue

    Hest_abs = interpolate.interp1d(pilotCarriers, abs(
        Hest_at_pilots), kind='linear')(allCarriers)
    Hest_phase = interpolate.interp1d(pilotCarriers, np.angle(
        Hest_at_pilots), kind='linear')(allCarriers)
    Hest = Hest_abs * np.exp(1j*Hest_phase)
    return Hest

def equalize(OFDM_demod, Hest):
    return OFDM_demod / Hest

def csi_plot(array, title, legend):
    fig, axs = plt.subplots(2, figsize=(12.8,9.6))
    fig.suptitle('CSI ' + title)
    x_amp = np.arange(K)
    x_pha = np.arange(K)
    for CSI in array:
        axs[0].plot(x_amp, np.abs(CSI))
        axs[1].plot(x_pha, np.angle(CSI,deg=False))
    plt.legend(legend)
    plt.show()


def simulation():

    ### Transission side ###

    # Generate some random bits to be send
    bits = np.random.binomial(n=1, p=0.5, size=(payloadBits_per_OFDM, ))

    # Modulate the bits into OFDM symbols
    QAM_s = modulate(bits)
    OFDM_data = OFDM_symbol(QAM_s)

    # Tranform symbols to time domain and add cyclic prefix
    OFDM_time = IDFT(OFDM_data)
    OFDM_withCP = addCP(OFDM_time)


    if enable_fuzzing == True:
        # Apply fuzzing
        OFDM_fuzzed = fuzzing(OFDM_withCP,-20,0,-20,0)
```

```
    OFDM_TX = OFDM_fuzzed
else:
    OFDM_TX = OFDM_withCP

# 'send' it over the channel adding noise and channel effects
if enable_channel == True:
    OFDM_RX = channel(OFDM_TX, SNRdb)
    OFDM_RX = add_noise(OFDM_RX,SNRdb)[0]
else:
    OFDM_RX = OFDM_TX

### Receivers side ###

# Remove cp and transform to frequency domain
OFDM_RX_noCP = removeCP(OFDM_RX)
OFDM_demod = DFT(OFDM_RX_noCP)

# Perform channel state estimation
CSI = channelEstimate(OFDM_demod)


# Calculate the defuzzed_csi, to be used for equalisation
unfuzzed_csi = channelEstimate(DFT(removeCP(channel(OFDM_withCP, SNRdb))))
fft = np.fft.fftshift(np.fft.fft([1,-20/128,-20/128],64))
defuzzed_csi = CSI/fft


# Equalize and retrieve data
equalized_Hest = equalize(OFDM_demod, defuzzed_csi)
QAM_est = equalized_Hest[dataCarriers]

# Retrieve the original bits
bits_est = demodulate(QAM_est)

# Plot the CSI, and the defuzzed CSI
csi_plot([CSI,fft], "comparison",["measured","predicted"])
csi_plot([defuzzed_csi,unfuzzed_csi],"defuzzed",["defuzzed","unfuzzed"])

try:
    input()
    exit()
except KeyboardInterrupt:
    exit()
```

```
if __name__ == '__main__':
    simulation()
```

Appendix B

Python learning based defuzzing:

```
import numpy as np
import matplotlib.pyplot as plt
import random
# Imports for the openwifi self cap files
from openwifi_tools import plot, read_from_file


##########################################
########## Reading the datasets #########
##########################################

## Read the learning dataset ##
# The read_from_file function also performs data cleaning
fuzzed_openwifi = list()
fuzzed_openwifi.append(read_from_file("openwifi_csi_files/fuzzed_0_-20_0_20.txt"))
fuzzed_openwifi.append(read_from_file("openwifi_csi_files/fuzzed_0_7_1_-33.txt"))
unfuzzed_openwifi = read_from_file("openwifi_csi_files/unfuzzed.txt")


## Read the validation dataset
# The read_from_file function also performs data cleaning
fuzzed_openwifi_validation = list()
fuzzed_openwifi_validation.append(
    read_from_file("validation_csi_files/openwifi_0_-20_0_20.txt"))
fuzzed_openwifi_validation.append(
    read_from_file("validation_csi_files/openwifi_0_7_1_-33.txt"))
unfuzzed_openwifi_validation = read_from_file(
    "validation_csi_files/openwifi_unfuzzed.txt")


##########################################
############ Helper functions ############
##########################################

# Function to calculate the average error to average power ratio (dB)
def calculate_error(x, y):
    numerator = np.mean(np.abs(y-x))
    denominator = np.mean(np.abs(y))
    return 10*np.log10(numerator/denominator)
```

```
# Function to normalise the CSI samples
# Makes the sum of the squared absolute values equal to one
def normalise(csi_packets):
    normalised = list()
    for csi_matrix in csi_packets:
        norm = (csi_matrix[2:-2]) / np.sqrt((np.sum(np.abs(csi_matrix[2:-2])**2)))
        normalised.append(norm)
    return normalised

# Helper function to plot the errors
def plot_errors(nmse_array,legend):
    plt.plot(nmse_array)
    plt.title("Error for # packets averaged")
    plt.ylabel("Error")
    plt.xlabel("# packets averaged")
    plt.legend(legend)


########################################
####### Learning based defuzzing ########
########################################

## Main loop dictates how many packets will be averaged out ##

# Variable to keep the average error
openwifi_avg_error = [list() for _ in range(len(fuzzed_openwifi))]

for amount in range(1,50):
    # Helper variables used to calculate the averaged CSI
    openwifi_unfuzzed_total = np.zeros(56, dtype='complex128')
    openwifi_fuzzed_total = np.array(
        [np.zeros(56, dtype='complex128')] * len(fuzzed_openwifi))
    openwifi_unfuzzed_validation_total = np.zeros(56, dtype='complex128')

    # Sum up certain amount of packets
    for index in range(0, amount):
        openwifi_unfuzzed_total += unfuzzed_openwifi[
            random.randint(0,len(unfuzzed_openwifi)-1)]
        openwifi_unfuzzed_validation_total += unfuzzed_openwifi_validation[
            random.randint(0,len(unfuzzed_openwifi_validation)-1)]
        # Add packet to current total for every openwifi CSI file
        for item in range(0,len(fuzzed_openwifi)):
            openwifi_fuzzed_total[item] += fuzzed_openwifi[item][
                random.randint(0,len(fuzzed_openwifi[item])-1)]
```

```
    # Calculate the average CSI
    # Calculate the fuzzing pattern from the training dataset
    openwifi_unfuzzed_avg = openwifi_unfuzzed_total/amount
    openwifi_unfuzzed_validation_avg = openwifi_unfuzzed_validation_total/amount
    openwifi_defuzzing_patterns = list()
    for total in openwifi_fuzzed_total:
        avg = np.array(total)/amount
        openwifi_defuzzing_patterns.append(avg/openwifi_unfuzzed_avg)


    # Choose n random packets from the fuzzed validation dataset to be defuzzed
    n = 100
    openwifi_indices = list(np.random.permutation(
        np.arange(0,min([len(x) for x in fuzzed_openwifi_validation])))[:n])

    # Defuzz the packets and validate the result for Openwifi CSI
    for i,csi_matrix in enumerate(fuzzed_openwifi_validation):
        temp = list()
        for index in openwifi_indices:
            # Check the defuzzing against the average unfuzzed validation dataset
            defuzzed = csi_matrix[index][2:-2]/openwifi_defuzzing_patterns[i][2:-2]
            temp.append(calculate_error(defuzzed,openwifi_unfuzzed_validation_avg[2:-2]))
        # Save the mean error of n defuzzed packets
        openwifi_avg_error[i].append(np.mean(temp))



# Plot the mean error of the different openwifi CSI files
for file in openwifi_avg_error:
    plot_errors(file,["openwifi_0_-20_0_20","openwifi_0_7_1_-33"])
```