AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Institute of Electronics

MASTER'S THESIS

6TiSCH network sniffer

Sniffer sieci 6TiSCH

Author: Artur Nowak
Field of study: Electronics and Telecommunication
Promoter: Łukasz Krzak, PhD

Kraków, 2021/2022

Contents

Chapter 1

Introduction

In the summer of 2022 IoT Analytics published the quarterly report titled *"State of IoT—Spring 2022"*. This report states that the number of globally active IoT connections in 2021 reached 12.2 billion. This report also forecasts that this number is going to increase up to 27 billion by 2025[1]. With the number of connections steadily increasing, it becomes necessary to have an insight into the network's traffic. This can be achieved with devices called data sniffers. Data sniffers can be used for debugging and maintenance purposes. By capturing and analyzing the packets in an existing network, one can verify whether a specific network is working properly or not. By analyzing network traffic one can identify bottlenecks and drawbacks of the network. Another use case for data sniffers are security reasons. Today's hackers can exploit the network with a wide range of attacks, such as Man-in-the-middle attack or spoofing. Recognition of unusual traffic patterns or unusual packet types can identify the attacker. Data sniffers are already available on the market and are widely discussed in multiple research papers. Among the existing devices there are specialized devices such as *Open Sniffer* or simple USB (Universal Serial Bus) sticks such as *nrf52840*. Academic articles show an elaborate approach, such as a connection of multiple Single Board Computers or dedicated FPGA boards. This thesis makes another contribution in this field.

1.1 Scope of work

The objective of this research was to create a system that would be able to capture and extract data from packets transmitted within a 6TiSCH network. This thesis presents protocol description, information about the utilized hardware and software tools and results of conducted tests. Created system had to meet following criteria:

- be able to work in 863 - 870MHz radio frequency band
- be able to work in multiple channels simultaneously
- be based on the SDR (Software Defined Radio) technique
- be able to analyze packets in time and frequency domains

1.2 Thesis composition

This thesis has been divided into four chapters:

- Chapter 2 contains the description of the theory behind this research. It mainly focuses on the 6TiSCH protocol, its structure and adjustments made for a better fit within Low Power, Lossy Networks (LLN). Further, this chapter presents software and hardware utilized in this thesis. Dedicated sections contain descriptions of GNURadio environment - free and open-source software development toolkit allowing the development of complex and advanced radio

paths, STM32WL boards - boards programmed as a source of 6TiSCH packets, RTL28232U - an SDR receiver packed into small USB stick, SQLite - simple relational database engine providing storage of retrieved data and PC with its specification. Chapter ends with the state of the art section containing an overview of most recent solutions and designs in the field of data sniffing.

- Chapter 3 describes the prototyping process conducted in GNURadio. Chapter is divided into two subsections. First one describes creation of a RF (Radio Frequency) path that can conduct sniffing in a single channel. Second subsection expands the solution from the previous subsection by multiplying the number of RF paths. As a result this enhanced version is capable of data sniffing in multiple channels.
- Chapter 4 presents the results of conducted tests. Among them are Packet Delivery Ratio (PDR) tests, parameter tests showing the impact of parameters change on PDR value and scalability tests which push created system to the computational power limit by expanding the number of channels.
- Chapter 5 presents conclusions that can be drawn from this work along with future development possibilities.

Chapter 2

Theoretical background

2.1 SDR technique

Software Defined Radio is a technique in which elements of the radio communication path, such as filters, mixers, amplifiers, modulators, demodulators, detectors are implemented in software instead of hardware. The following picture demonstrates the boundary between software and hardware in the SDR technique.

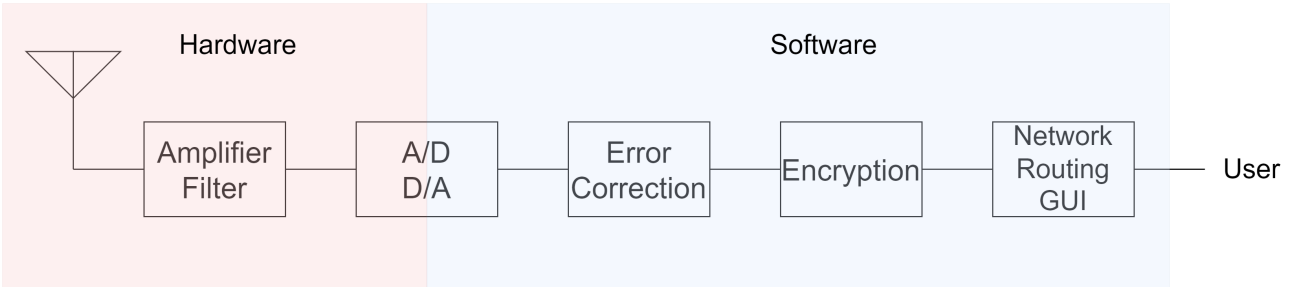

Figure 2.1: Division between hardware and software parts in SDR technique

The simplest Software Defined Radio would be any combination of Personal Computer (PC) or embedded system equipped with any analog to digital converter and preceded by RF front end.

Among the professional use cases can be mentioned[2]:

- *Military applications.* Military faces many challenges in the field of wireless communication. The key pieces of information exchanged between military units can be visual like videos and pictures. Others are simply just texts and voice commands. Different types of military agencies will communicate with its units in other ways. For example in the US army communication with submarines is conducted using Very Low Frequencies (VLF) while communication with aircrafts or satellites requires gigahertz frequencies[3][4]. In times of globalization countries militaries are cooperating in organizations like the United Nations, NATO or European Union. Each nation can use different means of hardware, software, encryption etc. One of the European Union agencies - European Defence Agency, which unites EU members militaries, uses and invests in SDR allowing enhancement of cohesion, cooperation and interoperability between different nations militaries[5][6].
- *Amateur radio.* Low costs of SDR hardware, combined with free, open source pieces of software are a great opportunity for all amateur radio enthusiasts. Amateur SDR applications range from simple, like listening to the broadcasting radio, through scientific and hobbystic ones, like listening to the conversations on International Space Station, capturing information from weather balloons or capturing signals from aircrafts and end on advanced projects like

triangulation[7][8][9][10][11]. Some comments can also be found stating that by using SDR it is possible to listen to drug cartels conducting illegal activities at southern US border. According to those comments, criminals are hiding their communication within legit signals and can be identified by long periods of silence and burst activity[12].

- *Mobile communications.* Before expanding on this item, a term of *cognitive radio* will be introduced. Cognitive radio is a type of radio communication scheme that can dynamically adjust to the changing propagation conditions in order to best utilize wireless channels in its vicinity by avoiding user interference and congestion. Transceivers are automatically detecting which communication channels are being occupied and which are free. Then the transceivers are changing their transmitting and receiving parameters to optimize the usage of spectrum and maximize the number of concurrent wireless communications to occur[13][14]. Cognitive radio can be implemented with the SDR technique and it plays a big role especially in cellular networks. Studies show that radio frequency bandwidths are inefficiently used[15]. Cellular network bands are overloaded whereas military or amateur radio frequencies are relatively unused. This disproportion however cannot be easily fixed, as radio spectrum usage is strictly limited by laws, and those inefficiently used bands are usually reserved for a reason.

2.2 6TiSCH protocol

IPv6 over Time Slotted Channel Hopping is the full name of the 6TiSCH protocol stack. This method of wireless communication utilizes multiple different protocols across all network layers. Most recent version of 6TiSCH protocol stack has been presented in the figure 2.2. The whole stack is standardized by two organisations. Physical and Link layers are standardized by The Institute of Electrical and Electronics Engineers (IEEE) in the IEEE 802.15.4 standard. Network, transport and application layers are standardized by The Internet Engineering Task Force (IETF). Since this thesis focuses on Software Defined Radio aspects, the main emphasis when describing the 6TiSCH protocol stack will be put on the Physical and Medium Access Control layers. Remaining upper layers will be described briefly.

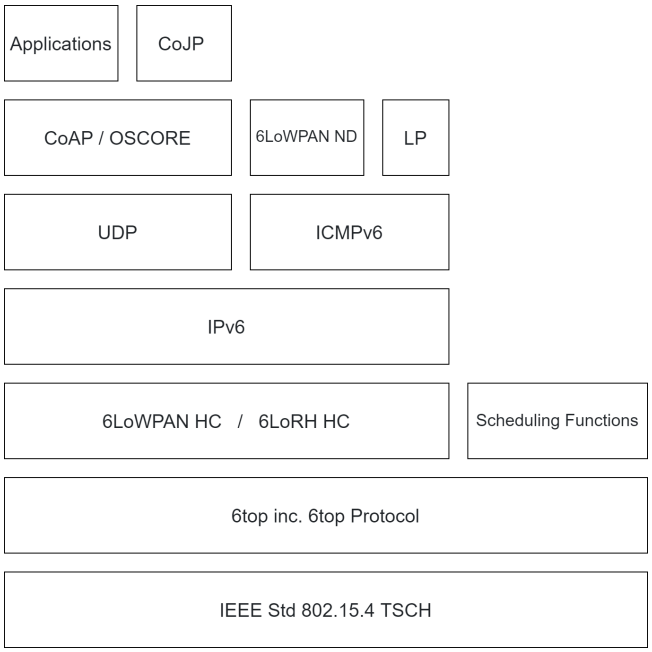

Figure 2.2: Illustration of the 6TiSCH protocol stack. Own elaboration based on RFC9030[16]

2.2.1 IEEE 802.15.4 standard - general description

The IEEE 802.15.4 standard defines the operation of Low-Rate Wireless Personal Area Networks (LR-WPAN). These networks are meant to be simple, low-cost and are characterized by low data rate and relatively low power consumption. The IEEE 802.15.4 standard is the basis for other protocols such as Zigbee or Thread. Typically this standard is used in application fields such as Internet of Things, home and industrial automation, smart grid etc.

This standard differentiates two types of topologies - star and peer-to-peer. In both of them devices can have two different roles - they are either a Fully Function Device (FFD) or a Reduced Function Device (RFD). RFD is a device that is meant to be extremely simple (e.g switch or sensor) and does not allow other devices in the network to communicate via itself. In other words, it's a device that is an edge of the network without routing capabilities. RFD can communicate with one FFD at most. FFD on the other hand is a device that allows other devices to communicate via itself. Moreover an FFD can communicate with multiple other FFDs and RFDs in the network. Additionally, one of the FFDs is selected as Personal Area Network coordinator (PAN coordinator). Main role of the PAN coordinator is initialization and synchronization of the network. In star topology all devices are communicating only with the PAN coordinator. In a peer-to-peer topology the devices can talk to each other directly. This elementary organization of communication leads to to more complex network topologies such as cluster tree or mesh network.

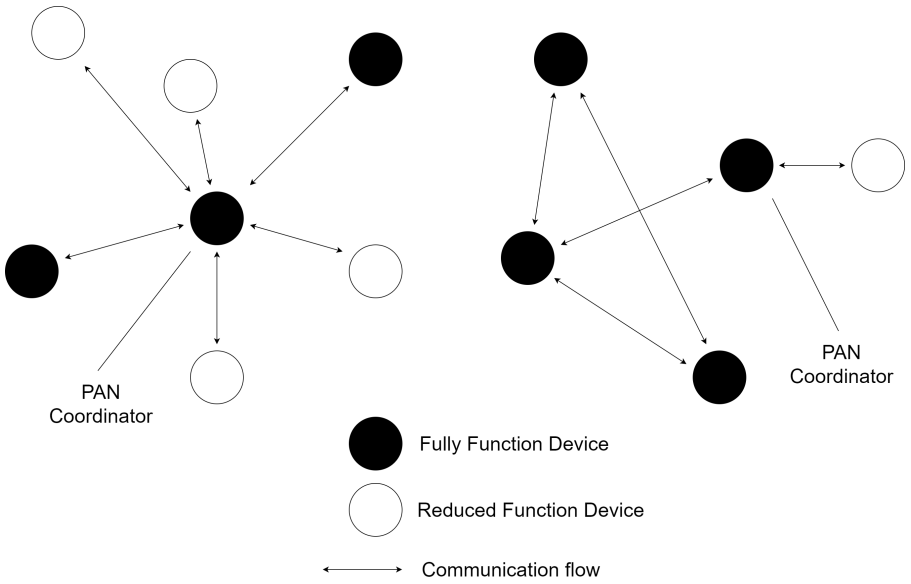

Figure 2.3: Possible topologies in IEEE 802.15.4 standard. Own elaboration based on the IEEE 802.15.4 standard

The IEEE 802.15.4 standard aims to address many diverse use cases and applications. Some of the application spaces were specifically defined and described in this standard. All of the specified applications possess a dedicated PHY in 802.15.4 standard.

- Smart Utility Networks (SUN) - SUNs are networks consisting of smart meters and other intelligent utility devices. Role of these meters is information gathering and forwarding this information to the applications which then store and analyze them. SUN devices are usually operating in a large-scale, cover great distances and require maximum available uplink throughput.
- Rail Communications and Control (RCC) - refers to the information exchanged or devices deployed as a part of a railway infrastructure. By this is meant:
  - communication within one vehicle or between many vehicles (trains)

  - communication between trackside, remote infrastructure and fixed infrastructure
  - communication between trains or any other mobile rolling stock to infrastructure

- Television White Space (TVWS) - these are unused spaces between active channels of Ultra High Frequency and Very High Frequency bandwidths. These unused channels can be used but only after the confirmation is made that those frequencies are free to take. This information is stored in geolocated databases. TVWS devices are querying those databases to determine whether in a given region the white space can be occupied or not[17].

- Radio Frequency Identification (RFID) - this term refers to the systems which allow wireless identification of objects (items, machines, people etc.). The RFID systems contain two types of devices - readers and tags. Readers are the devices which collect the information about identity of an object, whereas tags are the devices providing the identification. Tags can be active, which means they have their own source of power, or passive, which means they are powered by readers' incoming electro-magnetic wave which gives them enough power to send a response wave. RFID is used in inventory and livestock management, safety and accountability.

- Low-Energy, Critical Infrastructure Monitoring (LECIM) - some parts of the infrastructure are especially important for proper functioning of society and economy, such as water supplies, power plants, dams, bridges etc. Monitoring of those places increases safety, reliability and reduces maintenance costs. Monitoring devices are usually non-mains powered, widely dispersed and operating in challenging radio environments[18].

- Medical Body Area Network (MBAN) - is a wireless technology designed for sensing and collecting the information about vital signals of a person via tiny sensors located inside, outside or around the patients body[19].

2.2.2 IEEE 802.15.4 standard - physical layer

The foundation of the 6TiSCH protocol stack is based on the IEEE 802.15.4 standard physical layer. This layer conducts:

- typical radio transmission tasks such as:
  - radio transceiver activation and deactivation
  - channel frequency selection
  - transmission and reception of data
- additional tasks such as:
  - Energy Detection (ED) within the current channel
  - Link Quality Detection (LQI)
  - Clear Channel Assessment (CCA) for Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)
  - precision ranging for Ultra Wide Band (UWB)

Below is the description of the additional tasks.

Energy Detection (ED) within the current channel. In order to identify the proper channel for the transmission the device is estimating power of the received signals in the bandwidth. No attempts are made to identify or to decode the signals. Power is averaged every eight symbol periods.

Link Quality Detection (LQI). LQI is the characterization of the received signal quality when receiving a packet. This measurement can be implemented using receiver ED, a Signal-to-Noise Ration (SNR) estimation or a combination of both of these methods.

Clear Channel Assessment (CCA). It is used for Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) and serves two purposes. The first one is the situation when radio wants to initiate transmission. Before it happens, the carrier sensing mechanism makes sure that the channel is not already busy. The second situation is when the channel starts to be busy and inbound messages are being transmitted to the radio. Carrier sensing mechanism detects this change and the radio attempts to synchronize with the transmission. To appraise the RF medium, CCA mechanism is utilized. CCA is performed according to at least one of the following methods:

- *Energy above threshold.* CCA shall report a busy medium upon detecting any energy above the ED threshold.
- *Carrier sense only.* CCA shall report a busy medium only upon the detection of a signal compliant with this standard with the same modulation and spreading characteristics of the PHY that is currently in use by the device.
- *Carrier sense with energy above threshold.* CCA shall report a busy medium only upon the detection of a signal compliant with this standard with the same modulation and spreading characteristics of the PHY that is currently in use by the device. Power of the detected signal is above the ED threshold.

Precision ranging for Ultra Wide Band (UWB). Ultra Wide Band technology contains a mechanism which allows the calculation of the distance between the nodes. This measurement is based on the multiplication of time-of-flight and the speed of light.

Devices utilized in this research are implementing SUN FSK PHY. The standard defines three different operating modes for this PHY in bandwidth 863-870MHz. They differ in terms of data rate, modulation, modulation index and channel spacing. Table 2.1 presents those operating modes. The standard states that devices shall support operating modes 1 and 2 and they may additionally support operating mode 3. Utilized in this research devices will be operating in mode 1.

Frequency band (MHz)	Parameter	Operating mode #1	Operating mode #2	Operating mode #3
863 - 870	Data rate (kb/s)	50	100	150
	Modulation	2-FSK	2-FSK	2-FSK
	Modulation index	1.0	0.5	0.5
	Channel spacing (kHz)	100	200	200

Table 2.1: SUN FSK PHY modulations and channel parameters for 863-870MHz bandwidth

Table 2.2 presents the channel numbering in 863-870MHz bandwidth for different operating modes.

Frequency band (MHz)	Modulation	Channel spacing (MHz)	Total number of channels	Channel center frequency
863 - 870	SUN FSK operating mode #1	0.1	69	863.1
	SUN FSK operating modes #2 and #3	0.2	35	863.1
	SUN OFDM Option 4	0.2	35	863.1

	SUN O-QPSK	0.2	35	863.1

Table 2.2: Channel numbering for SUN PHYs in 863-870MHz bandwidth

2.2.3 IEEE 802.15.4 standard - medium access control sublayer

The IEEE 802.15.4 standard defines two different ways of accessing the medium. The first one uses a beacon mechanism in which beacon frames are sent periodically from the PAN coordinator to the nodes in order to synchronize and identify the network. In the second mode beacon frames are not used and access to the medium is realized by an unslotted CSMA/CA mechanism.

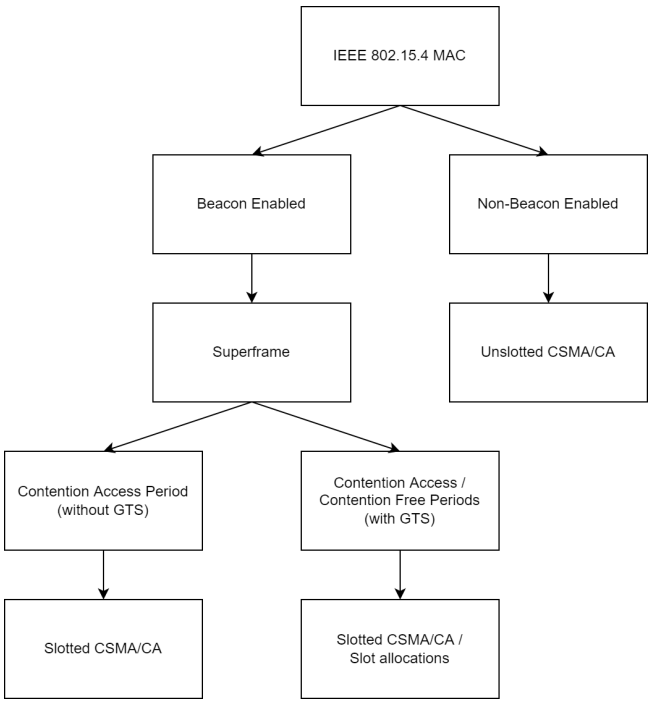

Figure 2.4: Components and interfaces of MAC. Own elaboration based on IEEE 802.15.4 standard

- Beacon Enabled - in beacon enabled mode beacon frames are sent periodically in order to synchronize and identify the network. Between the consecutive beacon frames, two periods can be differentiated - active and inactive. In the inactive period no communication occurs between the devices and the PAN coordinator can enter low-power mode. In the active period the communication between the devices can occur. Two consecutive beacons and periods between them are called a superframe.

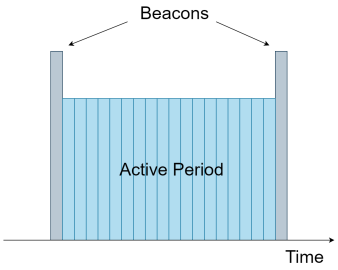

Figure 2.5: Superframe with active period only. Own elaboration based on IEEE 802.15.4 standard

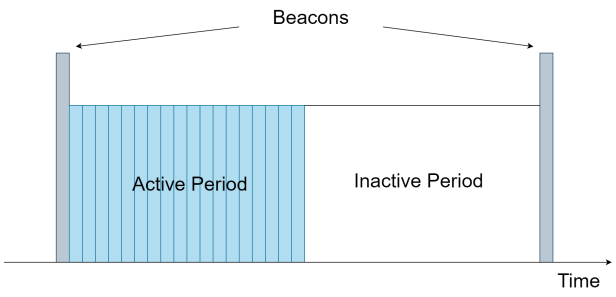

Figure 2.6: Superframe with both active and inactive periods. Own elaboration based on IEEE 802.15.4 standard

Active period can be divided into two different periods:

- In the first type of communication there is only one period called Contention Access Period (CAP) in which devices compete with each other for access to the medium. The CAP access to the medium is realized with CSMA/CA. CAP starts immediately after the beacon frame. Each CAP consists of time slots. Time slots consist of units of time called backoff periods. Boundaries of time slots align with backoff periods boundaries.
- The second type adds the Contention Free Period (CFP) in which devices have Guaranteed Time Slots (GTSs) for communication. CFP always appears at the end of the active period of the superframe and immediately follows the CAP. PAN coordinator allocates up to seven GTSs. GTS is allowed to occupy more than one time slot. This type of medium access is an implementation of Time Division Multiple Access (TDMA).

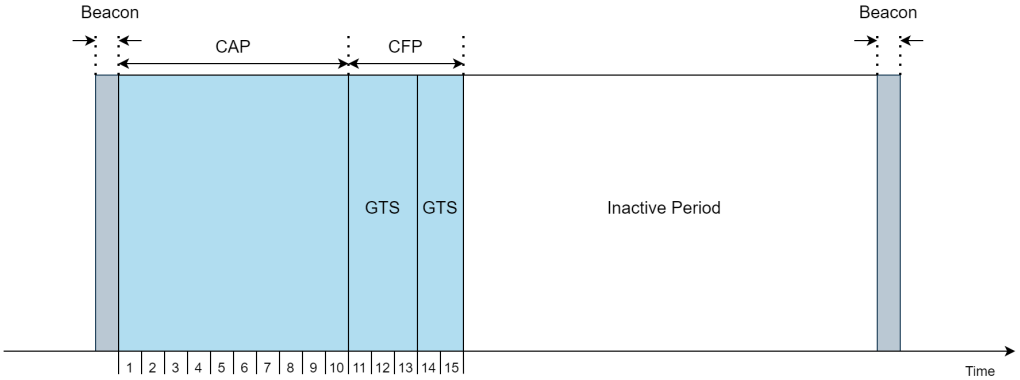

Figure 2.7: Structure of superframe with guaranteed time slots. Own elaboration based on IEEE 802.15.4 standard

- Non-Beacon Enabled - in this mode no beacon frames exist. Additionally GTSs are not allowed. Access to the medium is realized with an unslotted version of CSMA/CA, which is also based on backoff periods. However in this mode, different devices have backoff periods of different length.

2.2.4 6top inc. 6top Protocol

In Time Slotted Channel Hopping (TSCH), usage of the time slot in a certain channel can be represented as a matrix of cells. Width of the cell indicates time slot span and height equals to the bandwidth of the channel. 6TiSCH Operation Sublayer or 6top is an abstraction of an IP link over a TSCH MAC. This layer schedules packets over TSCH cells and exposes a management interface for this purpose[16]. In other words, 6top Protocol allows to allocate, move, or de-allocate cells. In the figure 2.8 each color represents an existing link between two nodes in the network.

2.2.5 6LoWPAN HC and 6LoRH HC

6LoWPAN Header Compression is a mechanism which reduces the overhead caused by IPv6 and UDP headers. In IEEE 802.15.4 standard the Maximum Transfer Unit (MTU) size is 127 bytes. IPv6 header is 40 bytes long. UDP header is 8 bytes long. Data is transferred with a throughput of 250 kbps or less[20]. To reduce the overhead caused by those headers, a Header Compression (HC) is introduced. In the best case scenario the 48 byte header can be compressed down to 6 bytes[21].

6LoWPAN Routing Header (6LoRH) is a header dedicated for Low power and Lossy Networks (LLNs). One of the ways to achieve low power consumption is the control of the amount of data being transmitted[22]. Routing protocols require additional information in the packet for loop prevention

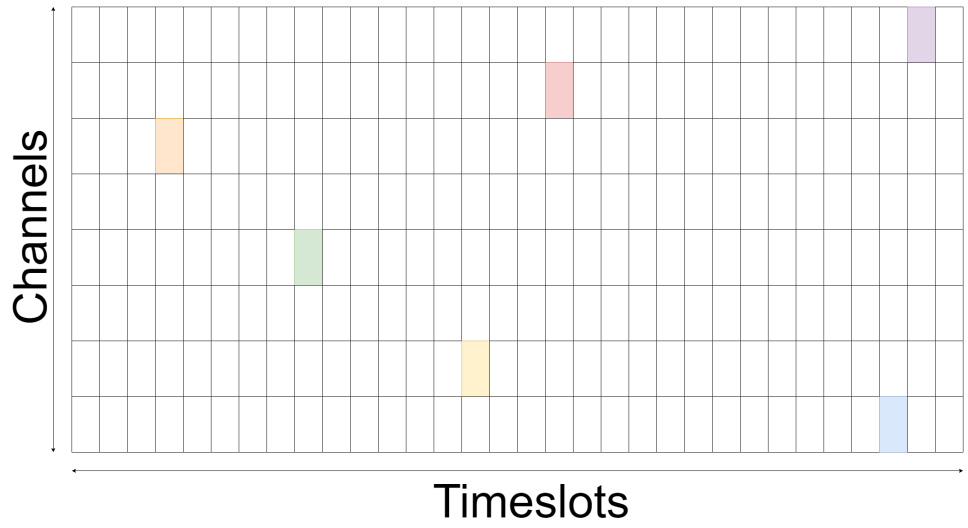

Figure 2.8: Matrix representation of Time Slotted Channel Hopping. Own elaboration based on RFC9030[16]

or for providing a source of routing information. However the original packet must not be changed in any way for reasons such as security or error reporting. That means that any additional data must be inserted in an extra IP-in-IP encapsulation. For this purpose 6LoRH was introduced. Its role is to carry the routing data and compress this data efficiently[16].

2.2.6 Scheduling Functions

6top Scheduling Function is a management entity that uses 6top protocol for dynamic creation or deletion of a cell based on a network requirements.

2.2.7 IPv6

Internet Protocol version 6 (IPv6) is a successor to the Internet Protocol version 4 (IPv4). Main reason for IPv6's creation and development is the arising problem of the IPv4 address exhaustion. In IPv4 the length of the address field is 32 bits which results in approximately 4.3 billion possible addresses. IPv6 resolves this problem by extending the address field to 128 bits. Number of possible addresses in this protocol is greater than 3.4e38.

2.2.8 ICMPv6

Internet Control Message Protocol (ICMP) is a network layer protocol used for reporting, finding and resolving errors in datagram communication. Based on this protocol is the well known *ping* command which is used for determination of hosts reachability. ICMPv6 is an implementation of ICMP over IPv6[23][24].

Even though the picture 2.2 represents the IMCPv6 on the same level as UDP it is not a transport layer protocol. ICMP and ICMPv6 are encapsulated as if they were above IP and IPv6 but they are an integral part of the IP's network layer.

2.2.9 UDP

User Datagram Protocol is a simple, connectionless transport layer protocol. In this protocol there is no acknowledgement information before or after data delivery. This protocol is thus less reliable than the standard Transmission Control Protocol (TCP). There is no certainty that the datagram was delivered. However its advantage is the small overhead of only 8 bytes. Its main use cases are

applications which prioritize fast data delivery over full data correctness such as video streaming, Voice over IP, Dynamic Host Configuration Protocol or Domain Name System.

2.2.10 CoAP and OSCORE

Constrained Application Protocol aims to integrate the so-called constrained devices (e.g., 8-bit microcontrollers with limited RAM and ROM) with the World Wide Web (WWW). CoAP is designed to easily translate to HTTP while meeting requirements posed by LLNs.

CoAP protocol mentions proxies as a tool for providing high scalability and efficiency. Additionally, in IoT, edge devices can use proxies for execution of asynchronous communication[25]. Proxies have access not only to data required to carry out proxies functionalities but they also have access to message payload and metadata. Proxies can eavesdrop, manipulate, corrupt and reorder the packets. Object Security for Constrained RESTful Environments (OSCORE) is a protocol providing end-to-end security between the endpoints[26]. For this purpose OSCORE uses symmetric key encryption, specifically Authenticated Encryption with Associated Data (AEAD) algorithm.

2.2.11 6LoWPAN ND

IPv6 network Neighbor Discovery (ND) mechanism serves number of functions:

- determination of the link-layer addresses for neighbors known to reside on attached links
- purge of cached addresses that become invalid
- hosts use this protocol in order to find the router willing to forward packets on their behalf
- determination of which neighbors are reachable and which are not
- detection of link-layer addresses changes

IPv6 ND is defined in RFC2461 and applies to all link-layers unless specified otherwise. The IPv6 ND for some of its services utilizes link-layer multicast[27]. Due to energy conservation, usage of multicast in 6LoWPANs is either highly limited or not present at all. RFC 6775 defines several changes to IPv6 ND that optimize the ND in 6LoWPANs[28]:

- complete elimination of multicast based address resolution
- introduction of new ND option in order to distribute 6LoWPAN HC to hosts

2.2.12 RPL

Routing Protocol for Low-Power and Lossy Networks (RPL) is a routing protocol defined specifically for networks working in constrained conditions. It is a distance vector type of routing protocol. RPL organizes a topology of Directed Acyclic Graph (DAG) and more specifically a Destination-Oriented DAG (DODAG). By the definition, DAG's possess at least one root - that is the node at which all paths terminate. DAG possessing a single root is called by the RFC6550 a DODAG. Each node has a rank assigned to it. The rank increases with increasing distance from the root. The nodes typically distribute packets based on lowest range as the path selection condition[29].

2.2.13 CoJP

Another layer of security in the 6TiSCH stack is Constrained Join Protocol (CoJP). While OSCORE provides end-to-end security, CoJP provides network access authentication, authorization and parameters distribution. CoJP introduces a "secure join" solution for new devices wanting to join the network, called a *pledge*. New devices are communicating with a central entity called Join Registrar/Coordinator or JRC. Both "pledge" and JRC share a unique symmetric cryptographic key called Pre-Shared Key (PSK). This PSK is used to configure the OSCORE protocol and to create a secure channel to Join Proxy. Join Proxy communicates with JRC and the new device is either authenticated or not[30] based on additional rules.

2.3 GNU Radio

GNURadio is a free and open-source software development toolkit for fast configuration of external SDR hardware devices. Similarly to MATLAB Simulink, projects in this environment are *flowgraphs* - collection of connected blocks. All blocks are connected to each other in such a way that they create a DAG. However, in contrast to Simulink, blocks provided by GNURadio are focused on radio typical computations and analysis. Filters, constellation and waterfall diagrams, clock synchronization blocks, OFDM blocks, modulators, demodulators form a majority of GNURadio tools. Except that custom blocks can also be created.

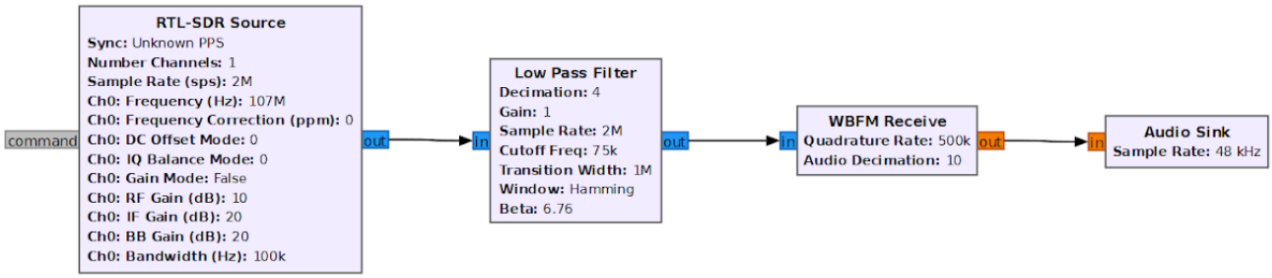

Figure 2.9: Exemplary GNURadio flowgraph. Implementation of the frequency modulated radio receiver

GNURadio blocks have been written in C++ however user tools such as GNURadio Companion (Graphical User Interface for GNURadio projects) or connections between the blocks have been written in Python. GNURadio follows an Object Oriented Programming paradigm. Flowgraphs and blocks are objects, instances of defined classes. Each class which represents a block must have a method named *work*, which contains implementation of the block.

```
/* -*- c++ -*- */
/*
 * Copyright 2004,2007,2013 Free Software Foundation, Inc.
 *
 * This file is part of GNU Radio
 *
 * SPDX-License-Identifier: GPL-3.0-or-later
 *
 */

#ifndef INCLUDED_GR_FILE_SINK_IMPL_H
#define INCLUDED_GR_FILE_SINK_IMPL_H

#include <gnuradio/blocks/file_sink.h>
```

```
namespace gr {
namespace blocks {

class file_sink_impl : public file_sink
{
private:
    const size_t d_itemsize;

public:
    file_sink_impl(size_t itemsize, const char* filename, bool append = false);
    ~file_sink_impl() override;

    int work(int noutput_items,
             gr_vector_const_void_star& input_items,
             gr_vector_void_star& output_items) override;

    bool stop() override;
};

} /* namespace blocks */
} /* namespace gr */

#endif /* INCLUDED_GR_FILE_SINK_IMPL_H */
```

LISTING 2.1: DECLARATION OF *File Sink* CLASS IN GNURADIO. SOURCE: GNURADIO GITHUB REPOSITORY[31]

This thesis utilizes GNURadio version 3.10.1.1.

2.4 STM32WL board

STM32WL boards are a combination of general purpose microcontroller and a sub-GHz radio on the same chip. They possess a single or dual core 32-bit ARM Cortex M4 or M0+ processors, 64kB of RAM and 256kB of Flash. Radio is based on Semtech SX126x transceiver which supports BPSK, (G)FSK and (G)MSK modulations. Additionally those modulations can be successfully used in legacy and proprietary protocols such as Sigfox or W-MBUS. Frequency coverage ranges from 150 to 960MHz. Security functions are also built in, such as 128/256-bit AES hardware encryption or a public-key accelerator with an elliptic curve cryptographic engine. Multiple energy saving modes are also available[32].

For the purpose of this research two STM32WL boards have been used, each with different software, courtesy of Institute of Electronics from the AGH University of Science and Technology. First software implements a mechanism allowing execution of Packet Error Ratio (PER) tests. Parameter setup is conducted via commands given in the SSH terminal. Parameters that can be specified are: number of packets in a test, packets length, transmission power and channel. This software was especially important in the debugging process and flowgraph parameters adjustments. Second software is an implementation of PAN coordinator. Board programmed with this software broadcasts packets in radio channels 13, 52 and 68 which are placed at frequencies 864.4 MHz, 868.3 MHz and 869.9 MHz. Packets are sent with a source address equal to aaaa:bbbb:cccc:dddd:0080:e115:0500:da7c.

Packets sent by the board programmed with PER test software are different from IEEE 802.15.4 standard packet format. The exact format of the packet has been presented in the figure 2.10.

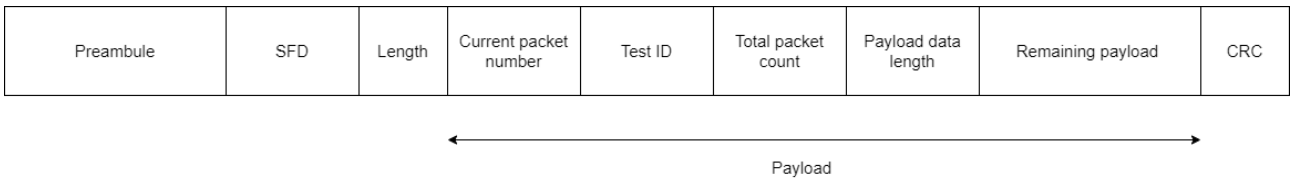

Figure 2.10: Structure of packet sent during PER tests

Fields description:

- Preamble - 64 bits of consecutive ones and zeros
- SFD - 0x904e sequence defined by IEEE 802.15.4 standard as SFD for uncoded 2-FSK packets
- Length - 8 bits describing packet length
- Current packet number - 16 bit field describing current packet number out of all packets in PER test. Starting from one incrementing by one until total packet count is reached
- Test ID - 32 bit ID of the PER test. The same for all the packets in a test. This parameter is provided via SSH by user
- Total packet count - 16 bit field with total number of packets in a given PER test
- Payload data length - 16 bit field with a length of payload
- Remaining payload - random number of zeroes and ones filling the remaining payload bits
- CRC - 16 bit Cyclic Redundancy Check field

Packet format for the PAN coordinator software is IEEE 802.15.4 standard compliant.

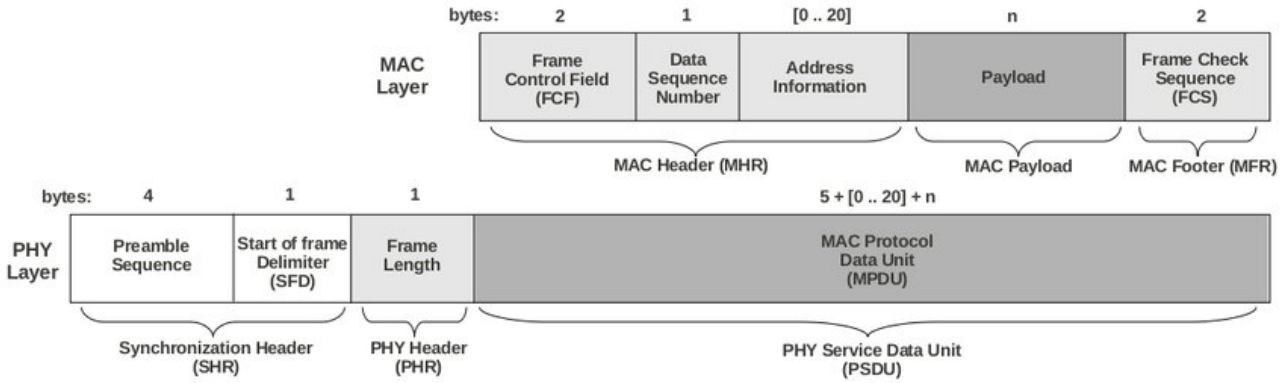

Figure 2.11: IEEE 802.15.4 packet format. Source: *The Collection Tree Protocol for the Castalia Wireless Sensor Networks Simulator*[33]

2.5 RTL2832U

RTL2832U or RTL-SDR is a simple SDR receiver. This piece of hardware is attached as a dongle via USB port to a PC or an embedded system. Originally, those receivers were designed and developed for the purpose of DVB-T HDTV (Digital Video Broadcasting – Terrestrial High-Definition Television) reception. However, they were found to be a good general purpose SDR receivers. Latest version of RTL2832U was redesigned with SDR users' needs in mind. RTL2832U can be described by the following parameters[34]:

- Bandwidth - 2.5MHz - maximum sample rate of RTL2832U is 3.2MS/s. However at the maximum rate, samples might be dropped. Highest sample rate at which samples are not dropped is 2.5 MS/s. Usage of USB 3.0 might increase this value to 2.8 or 3 MS/s
- ADC resolution - 8 bits - ADC resolution is 8 bits. However Effective Number Of Bits (ENOB) is approximated to 7 bits
- Input impedance - 50 Ohms - previous versions of RTL dongles were intended purely for television, hence they had 75 Ohms of input impedance. In recent version this parameter has been changed to 50 Ohms
- Frequency range - 500kHz to 1766MHz

2.6 PC

As stated in subsection 2.1 the SDR technique is a software implementation of some parts of the RF path. Hence to achieve the full view of the created system and provide the reproducibility, parameters of the PC to which SDR dongle is attached, must be presented. Especially crucial are the capabilities of the CPU. Originally the PC had Windows 10 installed. However for the purpose of this research PC has been reconfigured to possess two operating systems - Windows and Ubuntu (Dual Boot). All measurements and prototyping have been conducted on Ubuntu.

- CPU - Intel i7-9750H @ 2.60MHz - 6 physical cores, 12 logical cores
- USB - USB 2.0 port
- Operating System - Ubuntu 22.04.01

2.7 SQLite

For the purpose of captured packets storing the SQLite database engine has been utilized. SQLite is an open source, relational, lightweight, self-contained, highly-reliable and fully-featured database. SQLite is also an embedded database. It means that SQLite is tightly integrated into the application. In client-server database management systems all communication is conducted via a separate standalone process. In order to access a database, connection string is required. In non-embedded database engines connection string contains the IP address of the database, user credentials such as login and password, and the port on which the database is running. In SQLite no such process exists. Instead SQLite is linked into the application directly. Linking can be either static or dynamic. Connection string is simply a path to the file. This feature results in the high popularity of SQLite in IoT, mobile, automotive and desktop applications. According to SQLite's official website, there are over 1 trillion (1e12) active SQLite databases in use[35]. According to the StackOverflow 2022 Developer Survey, 30.83% of professional developers declared that they used SQLite in the last year or are going to use it within the next year, making it the 3rd most popular database engine[36].

2.8 State of the art

2.8.1 GSM Wireless Sniffer using Software Defined Radio

In 2017 an article titled *GSM Wireless Sniffer using Software Defined Radio* was published on Universitatea Tehnica Cluj-Napoca in Romania. This paper presents RTL-SDR with the R820T tuner attached to the PC with Kali Linux operating system. Creation of described data sniffing system starts with the installation of necessary software components, such as GNURadio and libraries which adjust the SDR dongle for GSM packets acquisition. After the setup was complete, GSM channels, specific

for the geographical location, were identified with *kal* command entered into the Linux terminal. After that channels were observed in GQRX software in order to make necessary RF and IF gains adjustments. In the last step, captured GSM packets were observed in Wireshark software. Captured in this paper packets are SMS (Short Message Service) messages. Authors present the way to utilize the extracted information from the packet in order to decrypt confidential information that might be in those SMS messages. Sniffing was conducted with a single SDR dongle on a single GSM downlink channel located at 952.2 MHz[37].

2.8.2 Multi Channel Wi-Fi Sniffer

Article in this subsection is the oldest out of the all presented solutions. Article titled *Multi Channel Wi-Fi Sniffer* is from November 2008 and at the time of writing this thesis is 14 years old. Authors of this article created a system based on multiple Single Board Computers (SBC) equipped with wireless cards. Those SBCs are connected to a PC via an Ethernet Switch.

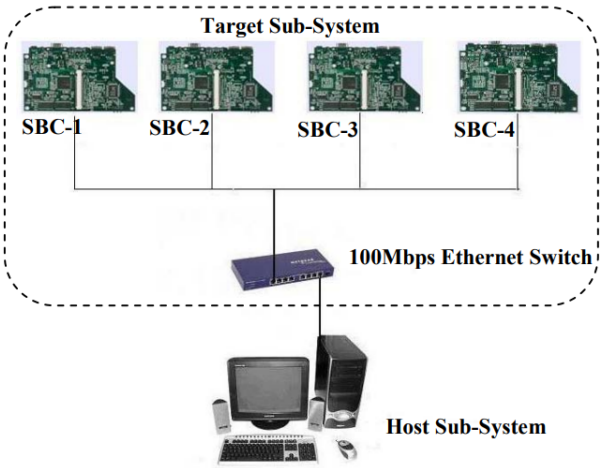

Figure 2.12: Architecture of the system presented in *Multi Channel Wi-Fi Sniffer* article. Source: *Multi Channel Wi-Fi Sniffer*[38]

SBCs were connected to the wireless cards via Mini PCI bus. Single SBC could have up to four Mini PCI buses, meaning that a single SBC could contain up to four wireless cards resulting in a total number of sixteen channels that could be sniffed at simultaneously. Each of the SBCs used a WLAN driver and Linux Network Stack for packet capturing. Those packets were then sent to the PC via server application. Authors emphasize the lack of computational power at high number rates - 6000 packets per second. Table 2.3 contains, presented by authors, typical parameters of a single SBC in 2008.

Component	Description
Processor	Processor Operating 533MHz
Memory	128MB SDRAM
Flash	16MB
Physical Ports	4 Mini PCI slots 100 Base-T Ethernet Port
DC Supply	3.3V
Typical Operating Power (@ DC=3.3V)	3.6 W (Without Mini PCI), 6.1 W (With 4 Mini PCI cards only with receiving mode enabled)

Table 2.3: Typical SBC specification in 2008. Source: *Multi Channel Wi-Fi Sniffer*[38]

2.8.3 Theoretical and Practical Approach to GNU Radio and LimeSDR Platform

The article titled *Theoretical and Practical Approach to GNU Radio and LimeSDR Platform* presents an SDR receiver implemented on FPGA board - LimeSDR. This article focuses on what exactly GNURadio and LimeSDR platforms are. As a proof of concept, the board is programmed as a simple FM receiver. Author conducts a comparison with other SDR receivers available on the market and emphasizes the LimeSDR capabilities[39].

Parameter	Hack RF One	RTL-SDR	LimeSDR
Frequency Range	1MHz - 6GHz	22MHz - 2.2GHz	100kHz - 3.8GHz
RF Bandwidth	20MHz	3.2MHz	61.44MHz
Sample Depth	8 bits	8 bits	12 bits
Sample Rate	20MSPS	3.2MSPS	61.44MSPS (Limited by USB 3.0 data rate)
Transmitter Channels	1	0	2
Receivers	1	1	2
Duplex	Half	Not available	Full
Interface	USB 2.0	USB 2.0	USB 3.0
Programmable Logic Gates	64 macrocell CPLD	Not available	40k
Chipset	MAX5864 MAX2837 RFFC5072	RTL2832U	LMS7002M
Open Source	Full	No	Full
Oscillator Precision	+/- 20ppm	Unknown	+/- 1ppm initial and +/- 4ppm stable
Transmit Power	-10 dBm+(15dBm @ 2.4GHz)	Not available	0 to 10 dBm (depending on frequency)
Price	~$299	~$10	~$299

Table 2.4: SDR platforms comparison. Source: *Theoretical and Practical Approach to GNU Radio and LimeSDR Platform*[39]

2.8.4 Open Sniffer For 802.15.4, Zigbee, 6lowpan

Open Sniffer is a device capable of conducting packet capture and packet analysis in networks based on standards such as IEEE 802.15.4, Zigbee, 6LoWPAN, Wireless Hart or ISA100.11a. *Open Sniffer* is a $20 \times 10 \times 10$ cm box equipped with two antennas. This box has to be connected to the PC via USB for powering and Ethernet for communication. Users can specify one of the 31 channels. One channel can be selected at the time of packet sniffing.

Frequency / Channel	**Modulation**
780/0	OQPSK-RC-250
782/1	OQPSK-RC-250
784/2	OQPSK-RC-250
786/3	OQPSK-RC-250
868/0	BPSK-20
906/1	BPSK-40/OQPSK-SIN-250
908/2	BPSK-40/OQPSK-SIN-250
910/3	BPSK-40/OQPSK-SIN-250

912/4	BPSK-40/OQPSK-SIN-250
914/5	BPSK-40/OQPSK-SIN-250
916/6	BPSK-40/OQPSK-SIN-250
918/7	BPSK-40/OQPSK-SIN-250
920/8	BPSK-40/OQPSK-SIN-250
922/9	BPSK-40/OQPSK-SIN-250
924/10	BPSK-40/OQPSK-SIN-250
2405/11	OQPSK-250
2410/12	OQPSK-250
2415/13	OQPSK-250
2420/14	OQPSK-250
2425/15	OQPSK-250
2430/16	OQPSK-250
2435/17	OQPSK-250
2440/18	OQPSK-250
2445/19	OQPSK-250
2450/20	OQPSK-250
2455/21	OQPSK-250
2460/22	OQPSK-250
2465/23	OQPSK-250
2470/24	OQPSK-250
2475/25	OQPSK-250
2480/26	OQPSK-250

Table 2.5: Channels and modulations available in *Open Sniffer*. Source: *Open Sniffer* datasheet[40]

This device can be bought from the producer for 379 euro, which is an equivalent of 373,77 dollars at the 22nd of September 2022[41].

2.8.5 nRF52840

Nordic Semiconductors (NI) provides a full tutorial which explains how to convert their nRF52840 dongle into the IEEE 802.15.4 standard packet sniffer. This tutorial is based on the Wireshark software and is provided specifically for ZigBee and Thread protocols. Configuration requires following steps:

- installation of Python v3.7
- programming of the RF52840 dongle with a specific firmware created by NI
- installation of Wireshark and Wireshark plugins

After the setup is complete, ZigBee and Thread packets can be captured on a specified channel[42].

The nRF52840 dongle can be bought for a price of 79.90 zł, which is an equivalent of 16.56$ at 22nd of September 2022[43].

Chapter 3

Technical background

3.1 Single channel demodulation

Before the creation of a fully fledged RF path could begun, one thousand packets were captured and saved into a binary file. Usage of this file instead of packets transmitted in real-time provided much needed convenience and reproducibility during the prototyping process. Figure 3.1 depicts five blocks. The first block is the *RTL-SDR Source* which is a representation of the SDR dongle within the GNURadio flowgraph. This block automatically detects whether the SDR dongle is plugged into any available USB port. The second most important block in this flowgraph is *File Sink*. This block saves the captured signals into the file. Remaining blocks are present in the flowgraph for debugging purposes. Block *QT GUI Waterfall Sink* creates a spectrogram in GNURadio Graphical User Interface (GUI) whereas *QT GUI Time Sink* creates a time domain chart on which demodulated signals are presented. Demodulation is conducted via *Quadrature Demod* block. Different colors on input and output of this block mean that the domain of the signal will change. Blue color means that the input signal is in the complex domain while orange color means that the output signal is in the real domain and numbers are of type float.

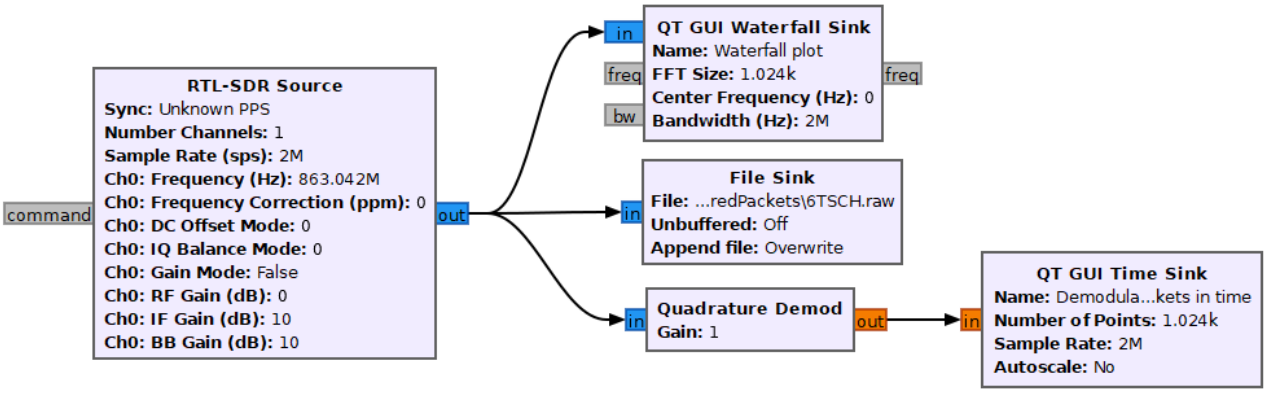

Figure 3.1: This flowgraph saves the captured packets into the binary file

Figure 3.2 represents results yielded by *QT GUI Waterfall Sink* and *QT GUI Time Sink* blocks when the STM32WL board has been switched into transmission mode but packets are not being sent. In other words, figure 3.2 shows the presence of carrier frequency.

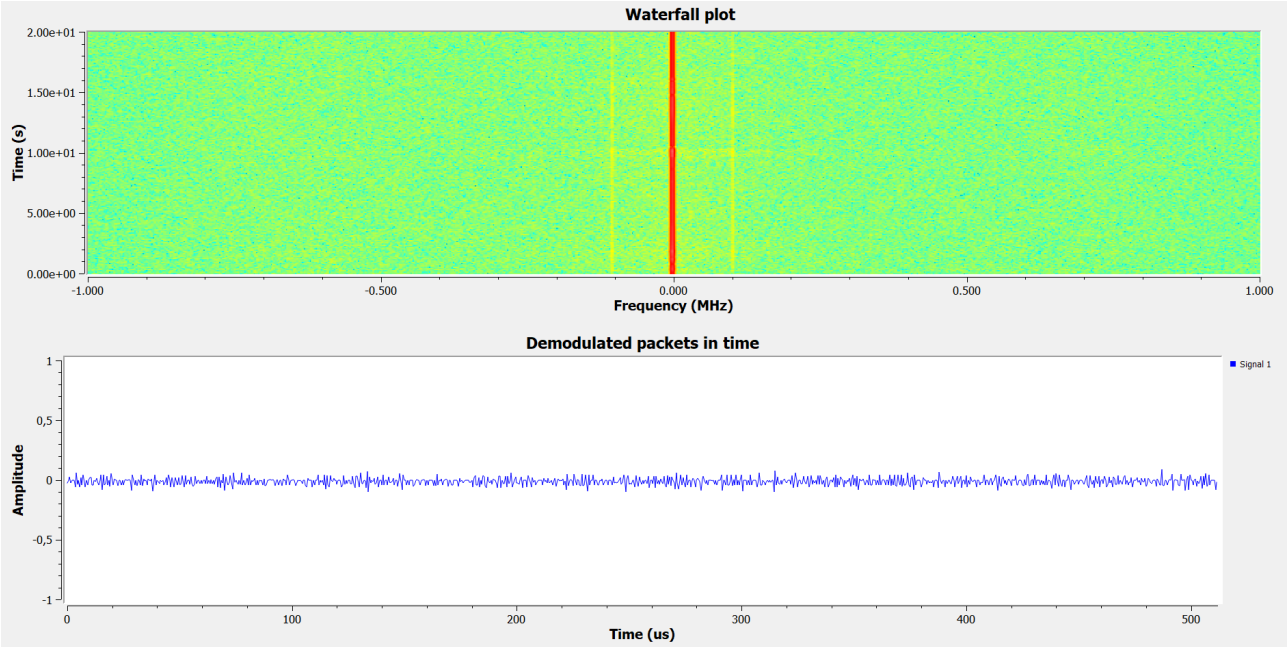

Figure 3.2: Results yielded by *QT GUI Waterfall Sink* and *QT GUI Time Sink* blocks when only carrier frequency is present

Figure 3.2 presents results yielded by *QT GUI Waterfall Sink* and *QT GUI Time Sink* blocks when the STM32WL board is sending PER test packets. The screenshot has been made between two consecutive packets. This can be concluded from a time domain chart. The difference in amplitude of packet and the signal when neither packet or carrier wave is present is much more visible in the figure 3.5.

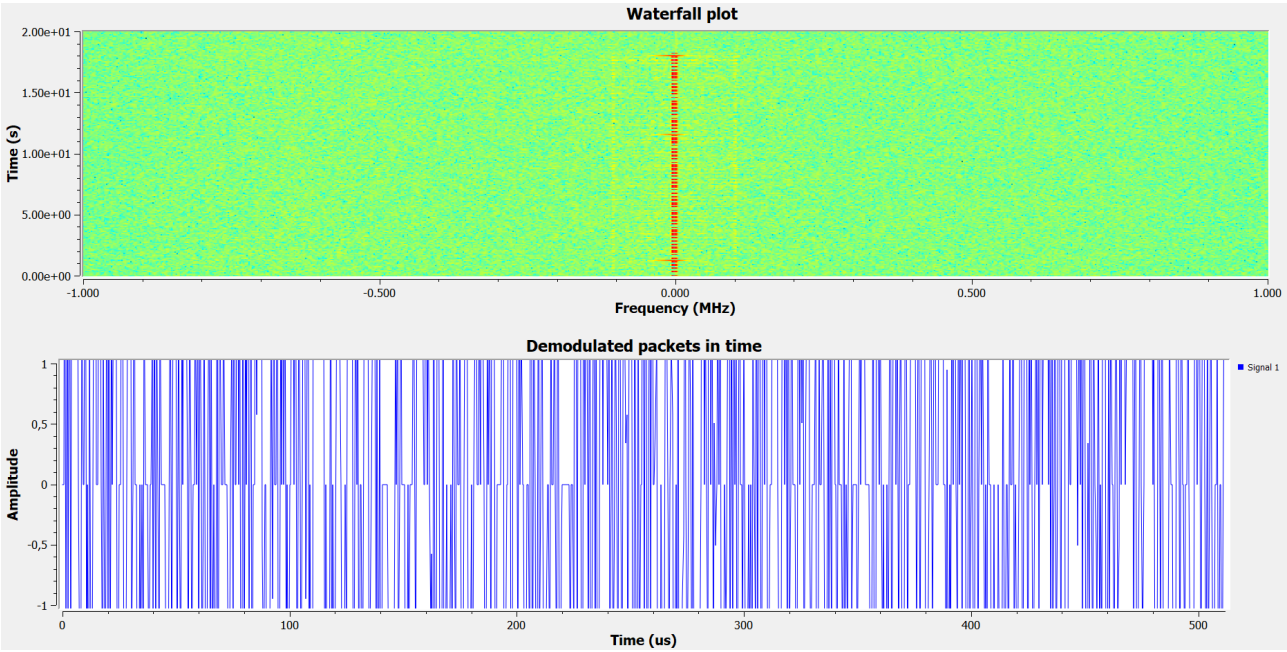

Figure 3.3: Results yielded by *QT GUI Waterfall Sink* and *QT GUI Time Sink* blocks when PER test is conducted

Figure 3.4 presents a full RF path used for reception of 6TiSCH packets. Flowgraph starts with *File Source* block. This block loads files with ".raw" extensions that were obtained with the flowgraph presented in the figure 3.1. Next two blocks are *Skip Head* and *Head* blocks. As the name suggests

those two blocks omit and copy from input to the output specified number of samples. Thanks to that the stationary period of samples of interest can be observed.

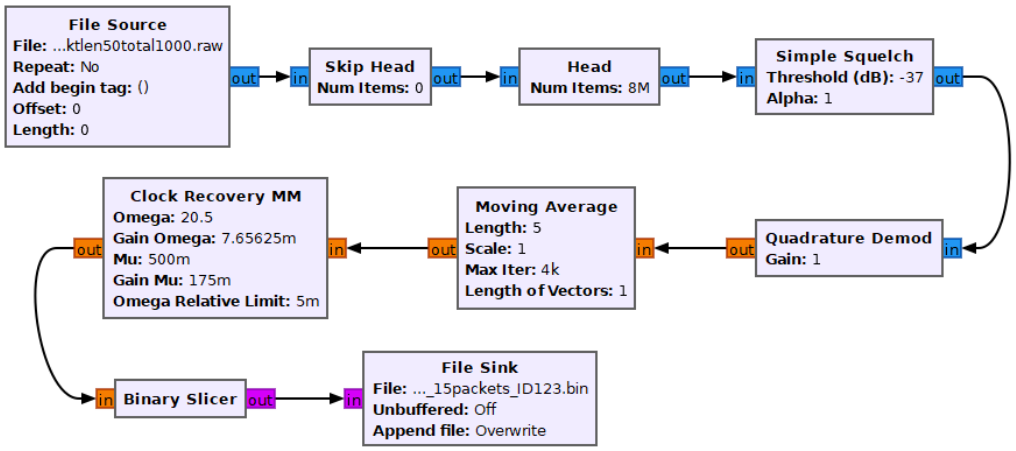

Figure 3.4: Prototype of single channel receiver

Next in the graph is *Simple Squelch* filter. This filter computes average signal power using a Single Pole IIR filter. Magnitude of signal is squared, filtered by IIR and compared with threshold value. The output of this block is equal to input if average signal power exceeds threshold. Otherwise output equals zero.

- Threshold - threshold used for muting
- Alpha - gain parameter for averaging filter

Figure 3.5 presents the beginning of the PER test when the *Simple Squelch* block is not present in the RF path. Beginning of the test is clearly visible as the transmitted packets have significantly smaller amplitude than the received noise.

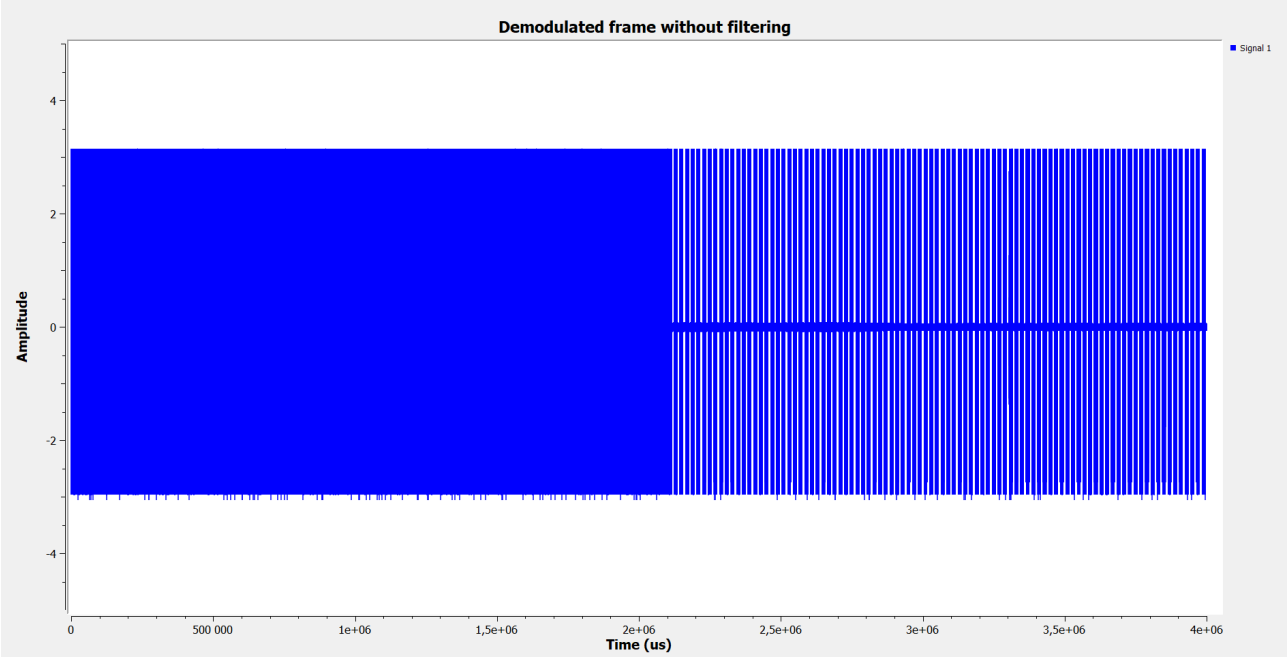

Figure 3.5: Beginning of the PER test. *Simple Squelch* filter is not present in the flowgraph

When zoomed in on a packet, packet ones and zeros are clearly visible. On each side of the plot, high amplitudes of noise can be observed.

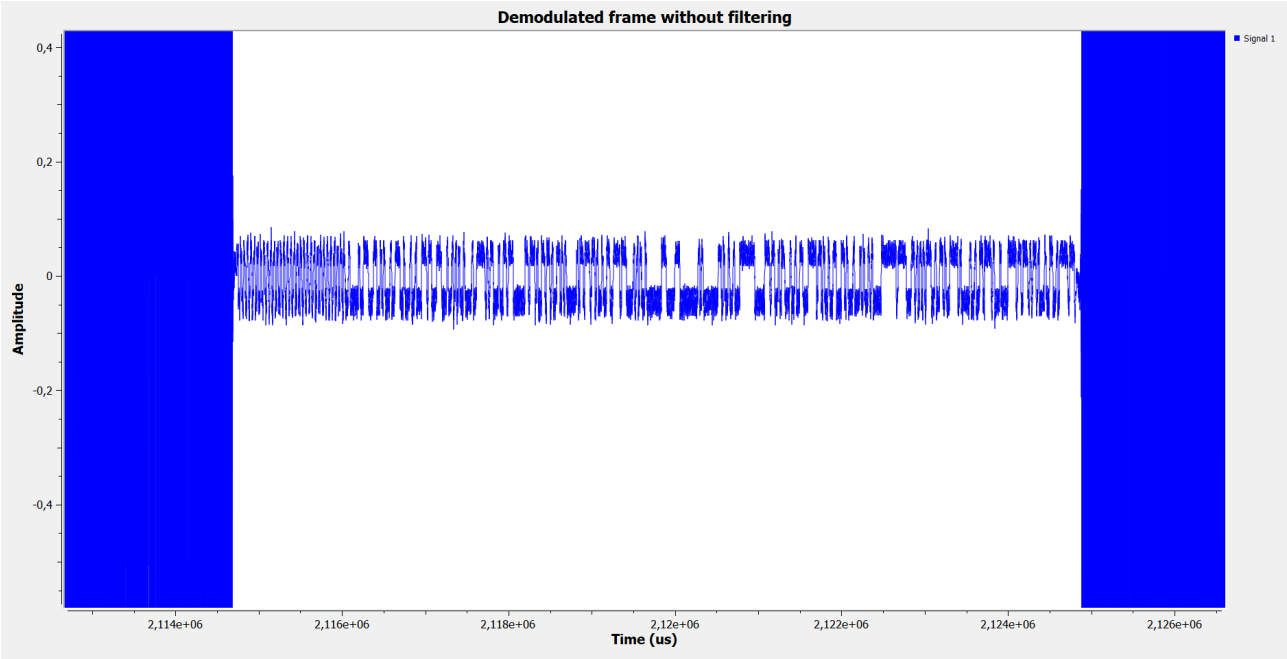

Figure 3.6: Zoom on a PER test packet in flowgraph without *Simple Squelch* filter

Figure 3.7 presents the beginning of the PER test when the *Simple Squelch* block is present in the RF path. Anything but PER test packets has been removed from the signal.

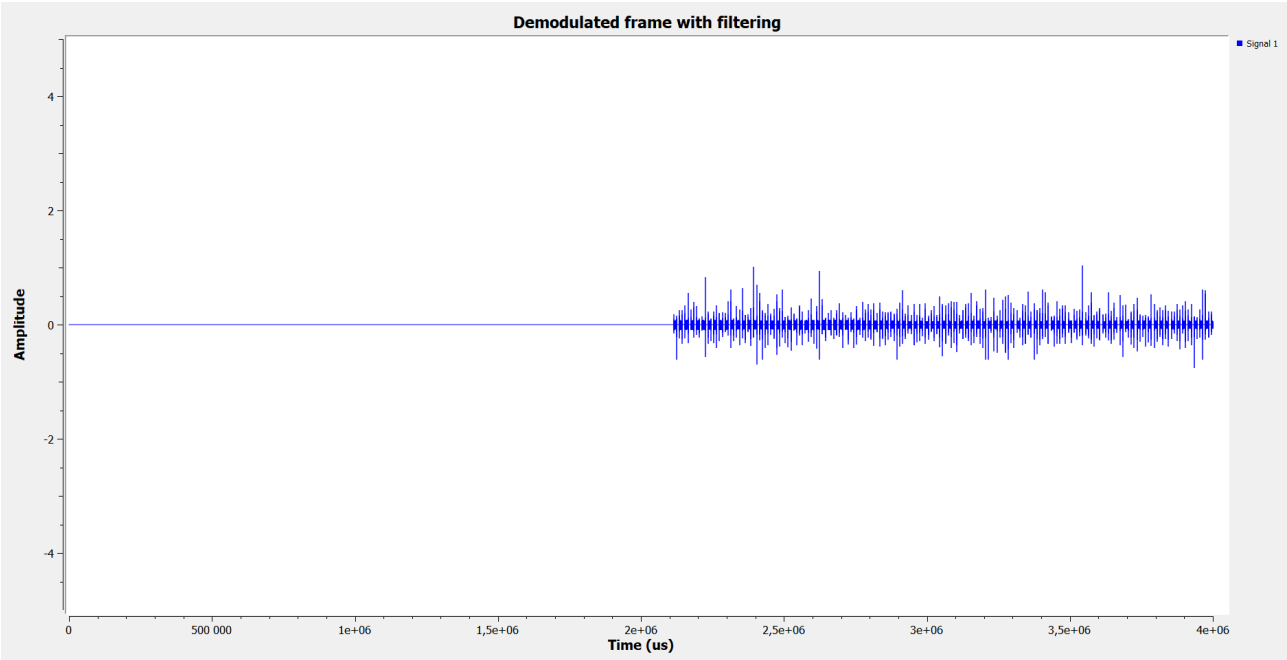

Figure 3.7: Beginning of the PER test. *Simple Squelch* filter is present in the flowgraph

When zoomed in on a packet, packet ones and zeros are clearly visible with no noise signal on each side of the figure. Compared to figure 3.6 packet envelope was not affected in any way.

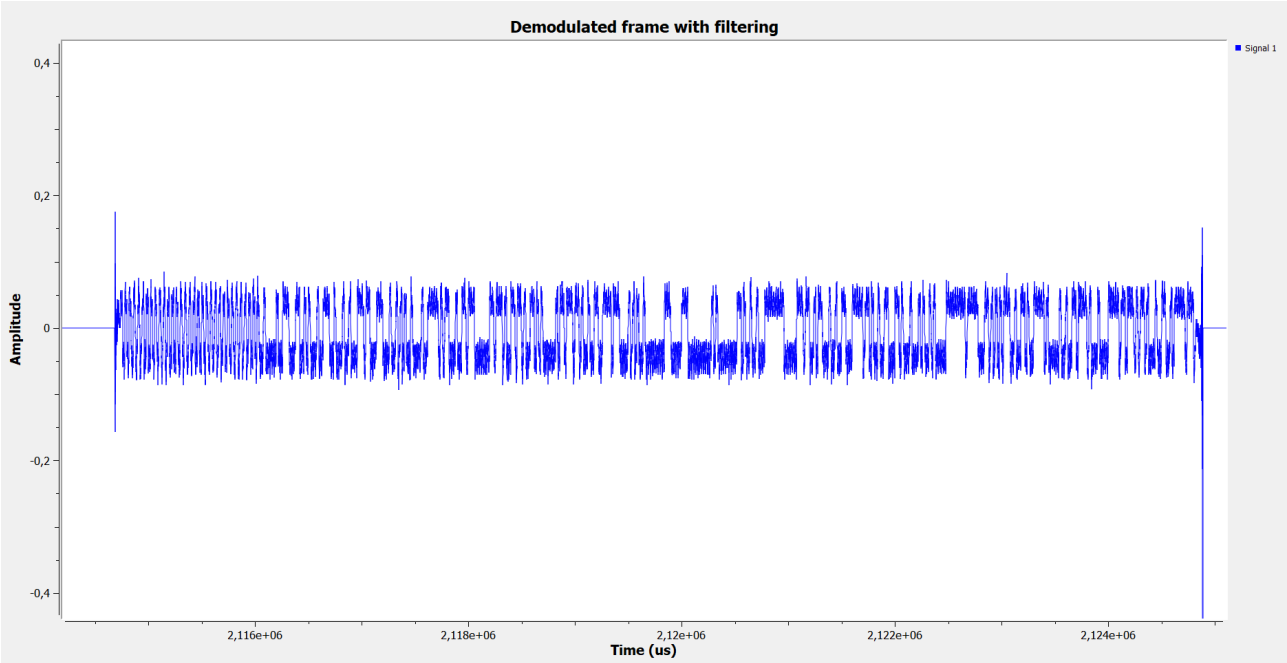

Figure 3.8: Zoom on a PER test packet in flowgraph with *Simple Squelch* filter

The next block is called *Moving Average*. As the name suggests moving average operation is conducted on the specified number of samples. Figure 3.9 presents the output of the averaging operation. It functions as a low-pass filter removing excessive signal fluctuations.

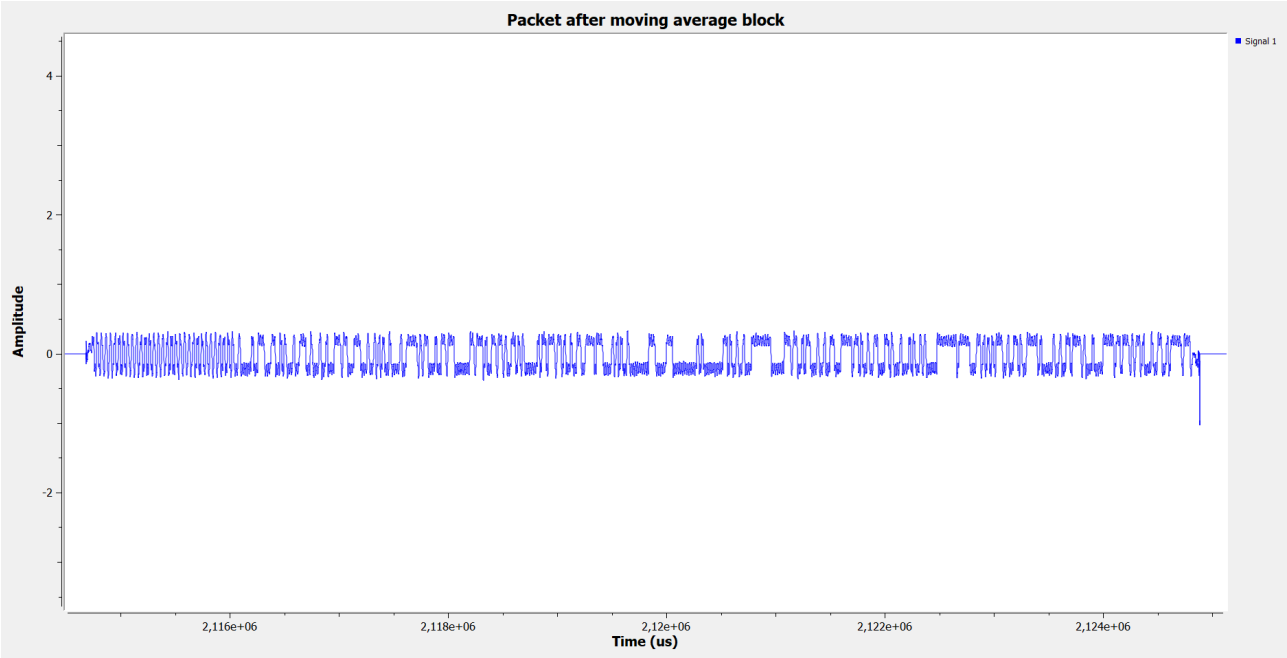

Figure 3.9: Zoom on a PER test packet after moving average operation has been conducted

At this moment packets with clearly distinguishable ones and zeros can be seen. Conversion of those analog packets to digital ones will be executed with two blocks: *Clock Recovery MM* and *Binary Slicer*. *Clock Recovery MM* is an implementation of timing synchronization algorithm proposed by Kurt H. Mueller and Markus Müller in 1976[44] and more precisely its optimisation from 1995. This algorithm is based on Phase Locked Loops (PLLs) and requires estimations of sample time, symbol time and the timing phase offset. Block consumes five parameters: Omega, Gain Omega, Mu, Gain

Mu and Omega Relative Limit. Omega parameter provides two of the needed estimations and it should be equal to a division of sample rate and data rate. The third estimation is satisfied by Mu parameter. However all of the parameters, except Omega, come with default values that can be used as a rule of a thumb. As a result, in this case:

$$Omega = \frac{SampleRate}{DataRate} = \frac{2MHz}{50kb/s} = 40 \tag{3.1}$$

In order to properly explain the calculations conducted within this block, a robust introduction to the subject of Digital Signal Processing would be necessary, which is beyond the scope of this Thesis. However on the web can be found a number of sources which provide the insight into the *Clock Recovery MM* block calculations. Among them can be mentioned:

- Andy Walls GNURadio Conference lecture, "Symbol Clock Recovery and Improved Symbol Synchronization Blocks", source: `https://www.youtube.com/watch?v=uMEfx_l5Oxk&ab`
- Qasim Chaudhari blog post, "Mueller and Muller Timing Synchronization Algorithm", source: `https://wirelesspi.com/mueller-and-muller-timing-synchronization-algorithm/`
- Tomaž Šolc blog post, "NOTES ON MM CLOCK RECOVERY", source `https://www.tablix.org/~avian/blog/archives/2015/03/notes_on_m_m_clock_recovery/`

Figure 3.10 depicts a packet at the output of *Clock Recovery MM* block. Each bit is now represented by only one sample. However, the signal is not yet binary.

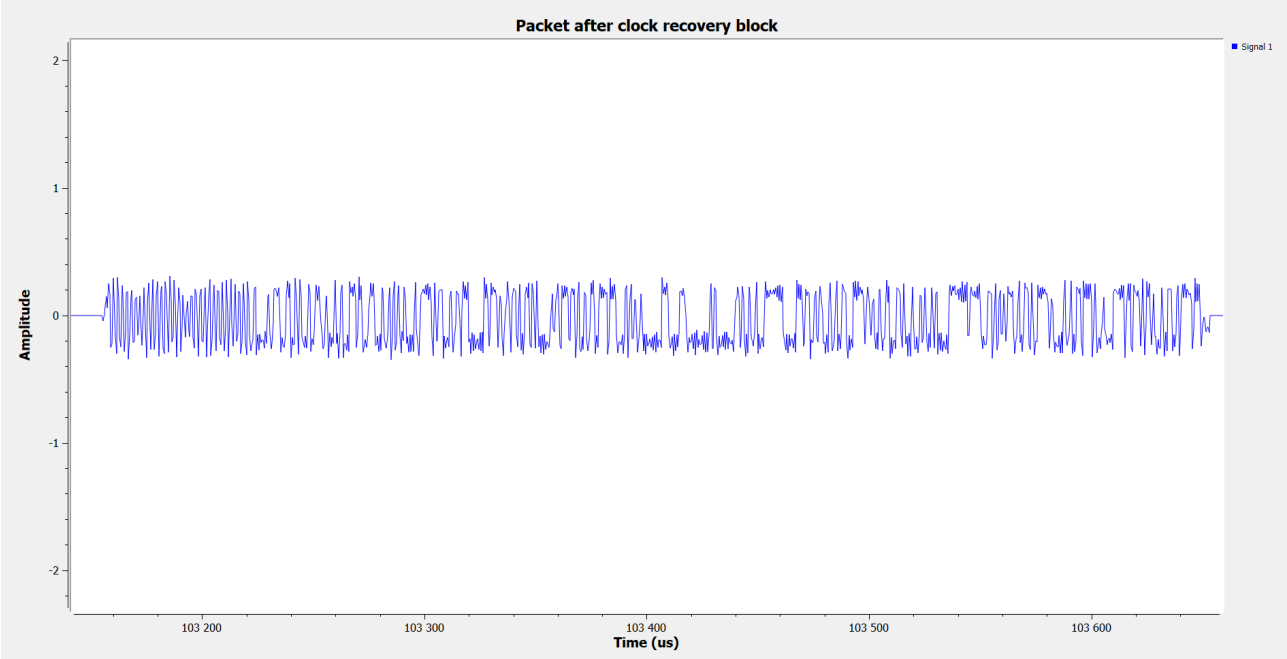

Figure 3.10: Zoom on a PER test packet at the output of *Clock Recovery MM* block

The last thing that needs to be done is the classification of each sample as either one or zero. This operation is done by *Binary slicer* block. As the figure 3.11 depicts, this block outputs a purely digital signal.

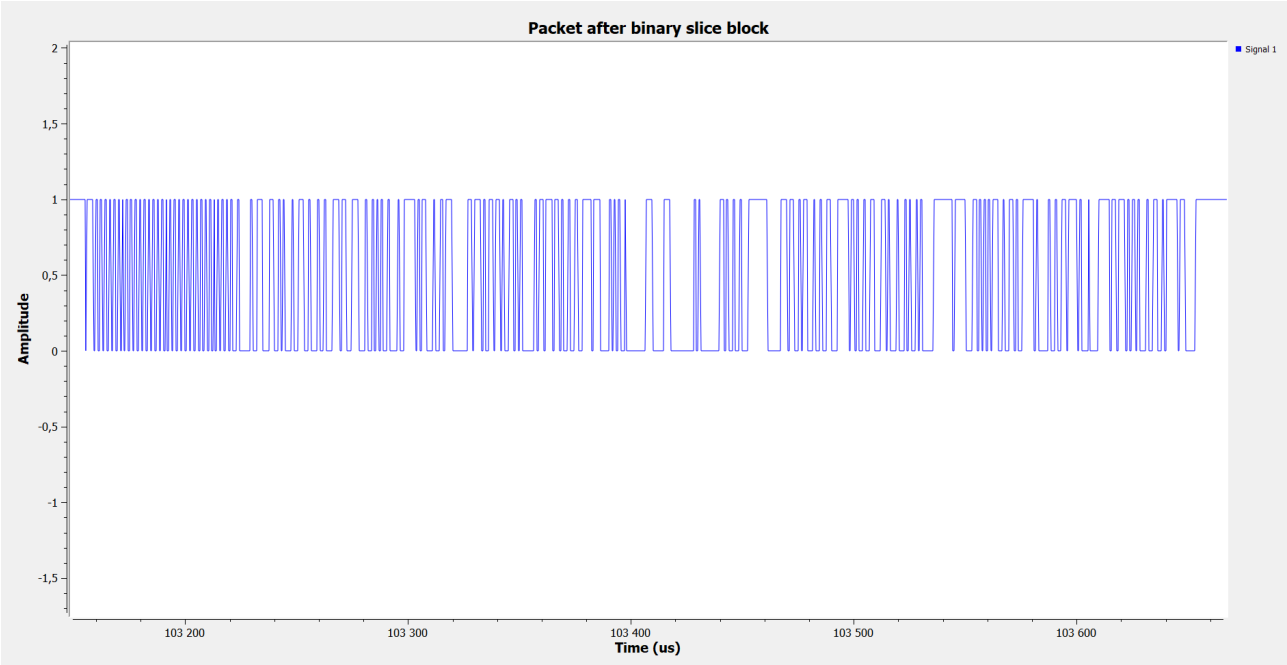

Figure 3.11: Zoom on a PER test packet after ones and zeros were differentiated

At this point the packets were saved into the file with *File Sink* block. Created file could be opened with a hexdump reader such as a popular *xxd* command that can be invoked in windows and linux shells. It was a temporary solution before proper payload scrapping was implemented.

3.2 Multichannel demodulation

After the prototyping process was over the attempt of sniffing on multiple channels could be made. For this purpose the RF path from figure 3.4 has been expanded with three additional blocks and duplicated.

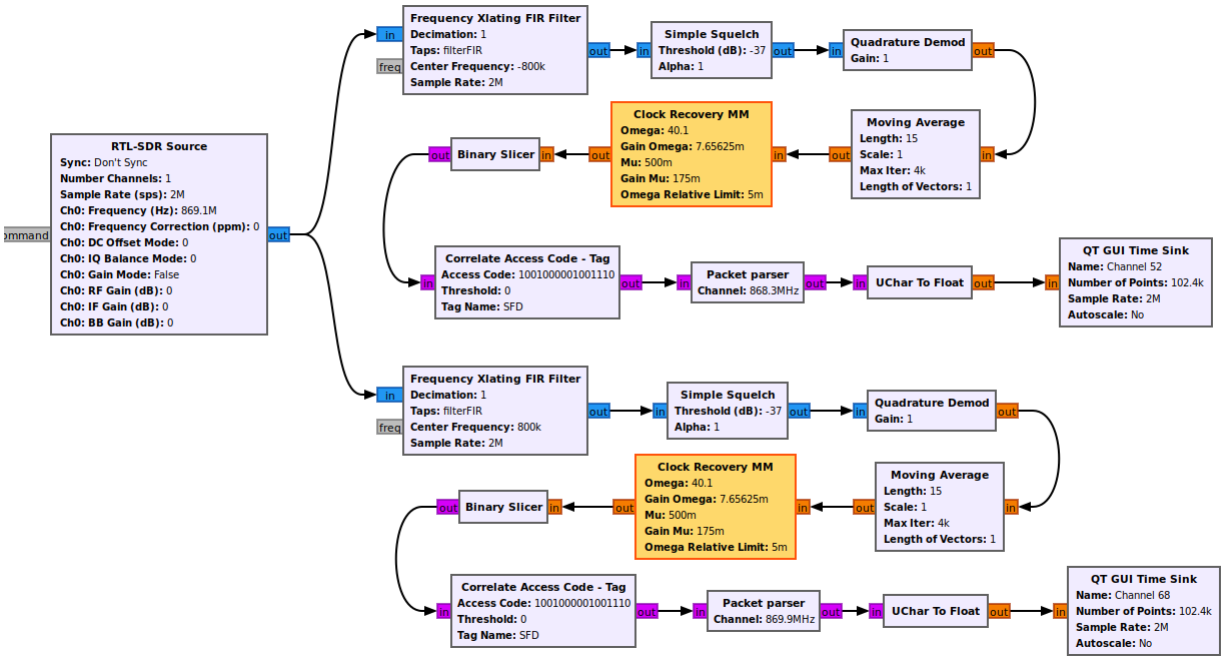

Figure 3.12: Fully fledged two channel SDR packet sniffer

As stated in section 2.4 one of the STM32WL boards is a PAN coordinator which periodically sends packets on channels 13, 52 and 68 which are positioned at frequencies 864.4 MHz, 868.3 MHz and 869.9 MHz respectively. As stated in the section 2.5 the bandwidth of the RTL2832U receiver is 2.5MHz. The span between channels 13 and 52 is equal to 3.9MHz. Span between channels 53 and 68 is equal to 1.5MHz. Together both of these spans equal to 5.4MHz, that means that only channels 53 and 68 can be reliably received simultaneously. *RTL-SDR source* channel zero frequency parameter has been moved exactly in between the channels 53 and 68.

The first block that has been added is a *Frequency Xlating FIR Filter*. This block performs a frequency translation on the signal which can be used to achieve channelization.

Following parameters are consumed by this block:

- Decimation - integer ratio between the input and the output signal's sampling rate
- Taps - vector of complex or real taps for the FIR filter. These taps can be inserted manually as a collection of values or generated using *firdes* class. This class represents different types of filters that can be created in GNURadio. In this case *firdes.low_pass()* method has been invoked. As the name suggests, a low pass filter has been instantiated. *firdes.low_pass()* consumes four parameters: gain, sampling rate, cut off frequency and transition width. Parameters were provided respectively with given values: 1, 2MHz, 80kHz, 20kHz.
- Center Frequency - frequency translation value
- Sample Rate - sample rate of the input signal

Resulting change in processed spectrum is commented in chapter 4 before the figures 4.2 and 4.3.

Remaining two newly added blocks are placed at the ends of flowgraph branches. Both of them in conjunction allow scrapping of the captured packets in real time. The first one is *Correlate Access Code - Tag*. This block takes a stream of bits as an input. Whenever a sequence of bits in the input stream matches the pattern specified in the *Access Code* parameter, the *Tag* object is added into the stream. *Tags* are Key-Value pairs which are used as markers in the GNURadio environment. In this flowgraph *Correlate Access Code - Tag* will add *Tag* with key set to "SFD" and value set to zero whenever an SFD sequence occurs in the stream of bits. For the purpose of this thesis value of tag is meaningless. Just the presence of the tag object will allow to implement proper packet scrapping. Figures 4.4 and 4.5 present streams of bits with tag objects.

The second block is *Packet Parser*. It is a custom python block designed specifically for scrapping of payloads. Whenever a tag object occurs in an input stream of bits, it means that an SFD has been matched. After that the following bits are saved into the buffer. When the buffer reaches specified limit, it is divided into separate collections which correspond to different fields in the packet. Those collections are then converted from zeroes and ones into decimal form in case of PER packets and into hexadecimal form in case of PAN coordinator packets. In this human friendly form values are saved into the database. Buffer is then cleared and the whole process repeats again.

Full implementation for PER test packets can be seen here: PERPacketParser.py

Chapter 4

Test results

4.1 Methodology

This chapter contains the results of tests and benchmarks conducted with the usage of *STM32WL* boards. In each of the sections the board has been placed at a distance of 20cm from the SDR receiver. Packet length in PER tests was equal to 20 bytes. Output power of the STM32WL was equal to 22dBm. The SNR value in the channels has been constantly monitored. Values of SNR during the tests equaled to 49dB +/- 3dB, indicating a relatively strong signal.

Figure 4.1: PC setup

4.2 PER tests and PAN coordinator packets acquisition

Pictures 4.2 and 4.3 display the waterfall plots during the PER tests being conducted on channels 52 and 68. On each of the figures there are three waterfall plots. First *Waterfall sink* has been attached right to the output of *RTL-SDR source* block. As a result this waterfall displays whole unfiltered 2MHz spectrum. Its 0MHz value corresponds to the center frequency of the *RTL-SDR*

source block - 869.1MHz. Second and third *Waterfall sinks* were attached right at the output of *Xlating FIR filters*. Impact of FIR filters is clearly visible. Frequencies that are passed by FIR filters are marked with bright green color whereas frequencies that are cut off are marked with teal color. Limitations of digital signal processing can also be noticed. If transmission occurs on channel 52, waterfall of channel 68 displays transmitted packets as if the transmission occurred on frequency 870.3MHz (beyond 863-870MHz bandwidth). If transmission occurs on channel 68, waterfall of channel 52 displays transmitted packets as if transmission occurred on frequency 867.9MHz (channel 48).

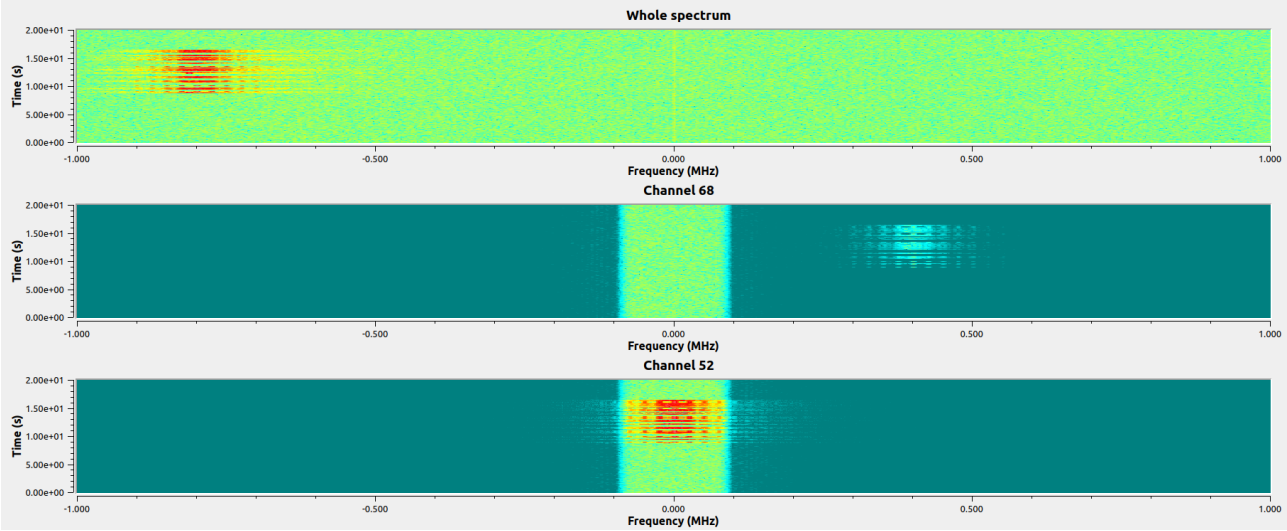

Figure 4.2: Waterfall plot during the PER test being conducted on channel 52

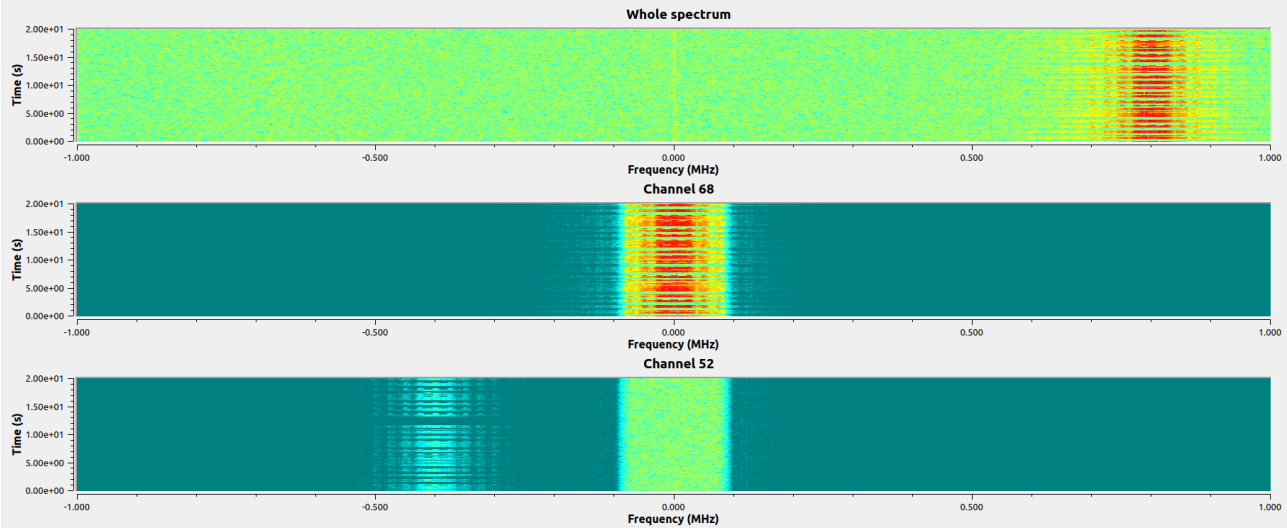

Figure 4.3: Waterfall plot during the PER test being conducted on channel 68

Pictures 4.4 and 4.5 present transmission of packets in the time domain. Whenever transmission occurs on channel 52, channel 68 displays value one. Whenever transmission occurs on channel 68, channel 52 displays value one.

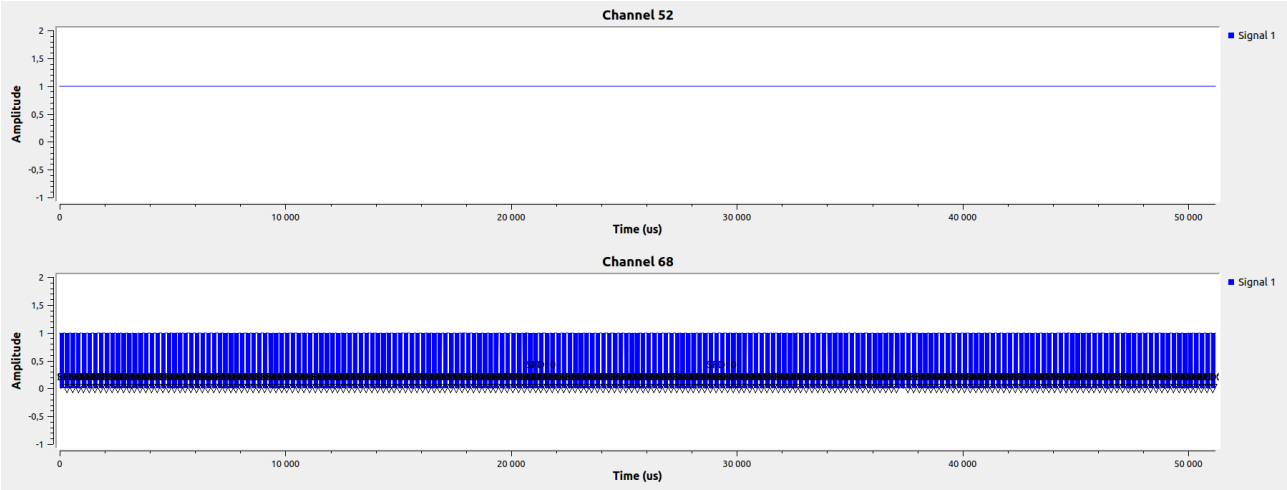

Figure 4.4: Stream of PER test packets in the time domain

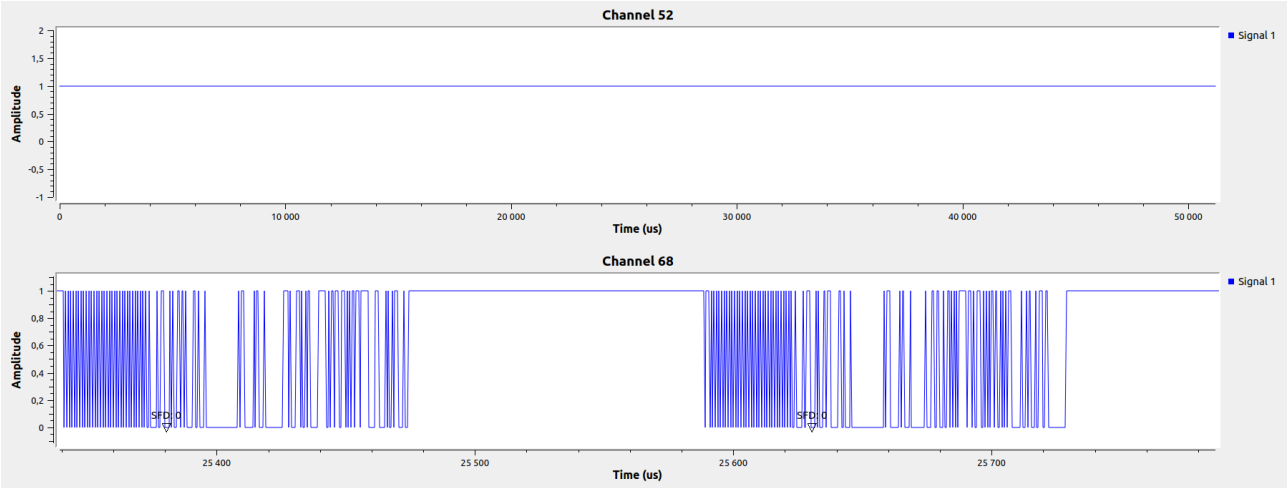

Figure 4.5: Single PER test packets in the time domain

Captured and scrapped packets are saved into the SQLite database. Figure 4.6 presents the records with scrapped payloads of captured packets.

	Id	Timestamp	Channel	Length	CurrentPacketNumber	TestId	TotalPacketCount	PayloadDataLength
	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr
1	57909	2022-09-11 20:20:12.534460	868.3MHz	20	1	123	1000	8
2	57910	2022-09-11 20:20:12.549461	868.3MHz	20	2	123	1000	8
3	57911	2022-09-11 20:20:12.550459	868.3MHz	20	3	123	1000	8
4	57912	2022-09-11 20:20:12.551485	868.3MHz	20	4	123	1000	8
5	57913	2022-09-11 20:20:12.552484	868.3MHz	20	5	123	1000	8
6	57914	2022-09-11 20:20:12.552484	868.3MHz	20	6	123	1000	8
7	57915	2022-09-11 20:20:12.553484	868.3MHz	20	7	123	1000	8
8	57916	2022-09-11 20:20:12.554484	868.3MHz	20	8	123	1000	8
9	57917	2022-09-11 20:20:12.630460	868.3MHz	20	12	123	1000	8
10	57918	2022-09-11 20:20:12.645461	868.3MHz	20	13	123	1000	8
11	57919	2022-09-11 20:20:12.647461	868.3MHz	20	14	123	1000	8
12	57920	2022-09-11 20:20:12.648461	868.3MHz	20	15	123	1000	8
13	57921	2022-09-11 20:20:12.649461	868.3MHz	20	16	123	1000	8
14	57922	2022-09-11 20:20:12.649461	868.3MHz	20	17	123	1000	8
15	57923	2022-09-11 20:20:12.650461	868.3MHz	20	18	123	1000	8
16	57924	2022-09-11 20:20:12.723458	868.3MHz	20	20	123	1000	8
17	57925	2022-09-11 20:20:12.735461	868.3MHz	20	21	123	1000	8
18	57926	2022-09-11 20:20:12.736461	868.3MHz	20	22	123	1000	8
19	57927	2022-09-11 20:20:12.737461	868.3MHz	20	23	123	1000	8
20	57928	2022-09-11 20:20:12.737461	868.3MHz	20	24	123	1000	8
21	57929	2022-09-11 20:20:12.738462	868.3MHz	20	25	123	1000	8
22	57930	2022-09-11 20:20:12.739461	868.3MHz	20	26	123	1000	8
23	57931	2022-09-11 20:20:12.740460	868.3MHz	20	27	123	1000	8

Figure 4.6: Scrapped payloads of packets sent during PER tests

Figures 4.7, 4.8 and 4.9 present exactly the same waterfall plots, time plots and database table but this time for the packets sent by PAN coordinator.

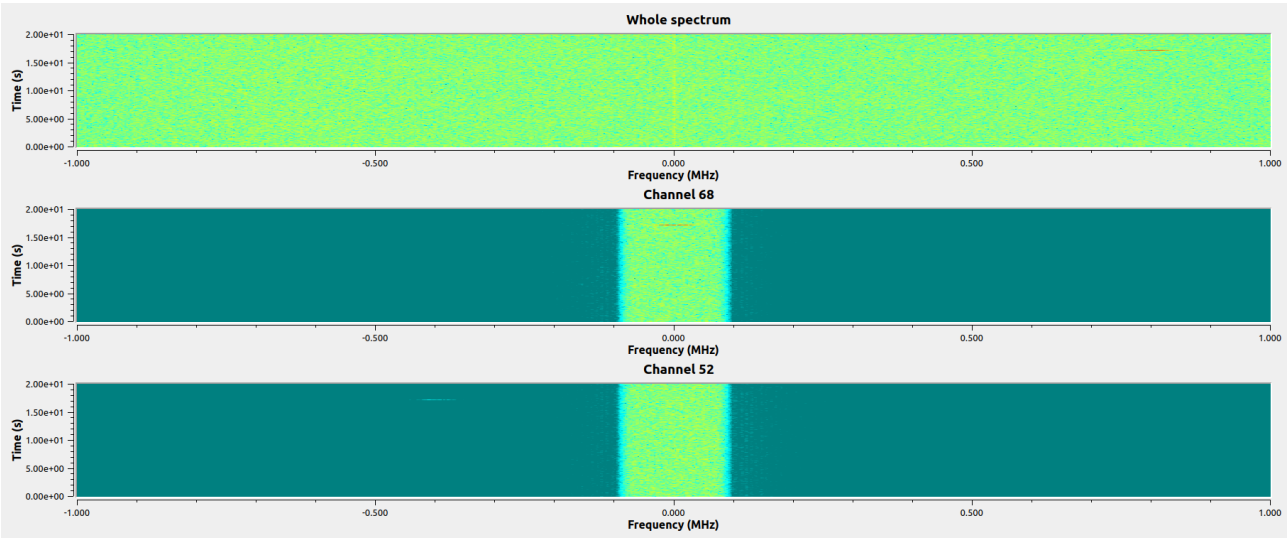

Figure 4.7: Waterfall plot during the broadcast of packets by PAN coordinator on channel 68

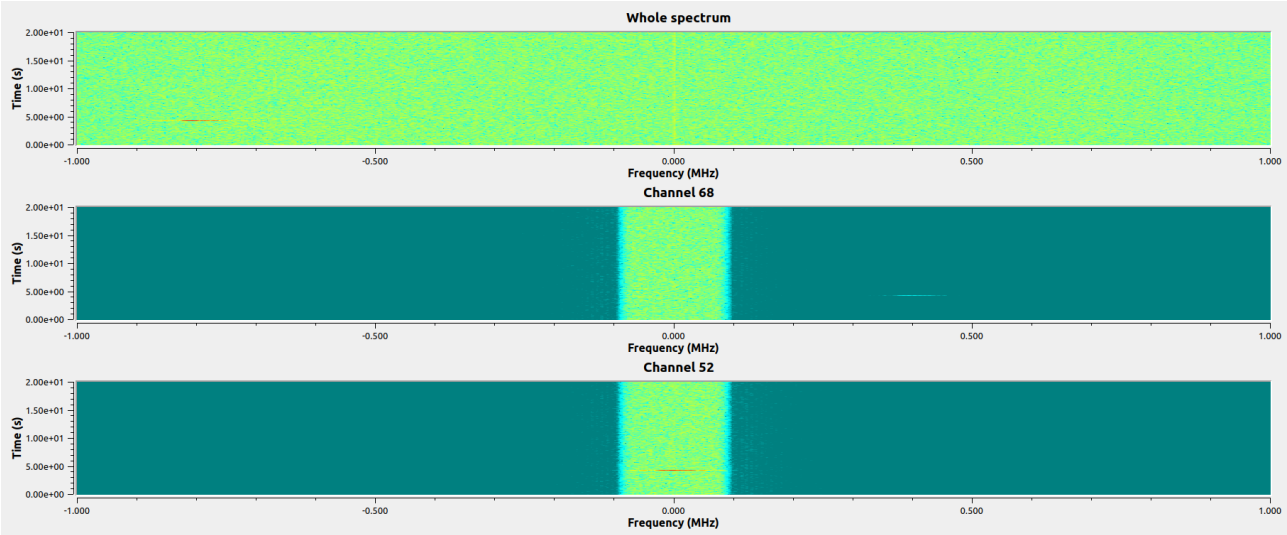

Figure 4.8: Waterfall plot during the broadcast of packets by PAN coordinator on channel 52

	Id	Timestamp	Channel	SFD	MiddleBytes	Address
	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr
1	34	2022-09-08 21:14:12.500272	869.9MHz	904e	6449e8fabee3ffff	7dda000721c30000
2	35	2022-09-08 21:14:38.454021	869.9MHz	904e	3748ea0dbee3ffff	7cda000515e18000
3	36	2022-09-08 21:14:50.761889	869.9MHz	904e	3f49ea13bee3ffff	7cda000515e18000
4	37	2022-09-08 21:14:58.495452	868.3MHz	904e	37486a1936e3bbfd	5cda000115e18000
5	38	2022-09-08 21:14:58.898830	869.9MHz	904e	3748ea1abee3ffff	7cda000515e18000
6	39	2022-09-08 21:15:02.967573	869.9MHz	904e	3748ea1dbee3ffff	7cda000515e18000
7	40	2022-09-08 21:15:07.146564	869.9MHz	904e	6449e820bee3ffff	7cda000515e18000
8	41	2022-09-08 21:15:08.471370	869.9MHz	904e	3748ea22bee3ffff	7cda000515e18000
9	42	2022-09-08 21:15:09.787973	869.9MHz	904e	6449e824bee3ffff	7cda000515e18000
10	43	2022-09-08 21:15:11.213521	869.9MHz	904e	3748ea25bee3ffff	7cda000515e18000
11	44	2022-09-08 21:15:13.454251	868.3MHz	904e	3748ea27bee3ffff	fcd6000a2bc30000
12	45	2022-09-08 21:15:14.879898	868.3MHz	904e	3748ca28bee3fffe	f9b4000a2bc30000
13	46	2022-09-08 21:15:30.344375	869.9MHz	904e	3748ea2fbee3ffff	7cda000515e18000
14	47	2022-09-08 21:15:31.665609	869.9MHz	904e	6449e831bee3ffff	7cda000515e18000
15	48	2022-09-08 21:15:33.895919	868.3MHz	904e	3748ea32bee3ffff	7cda000d1de30000
16	49	2022-09-08 21:15:35.329308	868.3MHz	904e	6449e835bee3ffff	7cda000515e18000
17	50	2022-09-08 21:15:37.162026	869.9MHz	904e	3748ea37bee3ffff	7cda000515e18000
18	51	2022-09-08 21:15:39.913277	869.9MHz	904e	6449e839bee3ffff	7cda000515e18000
19	52	2022-09-08 21:15:42.158591	868.3MHz	904e	6449e83cbee3ffff	7cda000515e18000
20	53	2022-09-08 21:15:45.307334	869.9MHz	904e	3748ea3ebee3ffff	7cda000515e18000
21	54	2022-09-08 21:15:48.054572	869.9MHz	904e	3748ea40bee3ffff	7cda000515e18000
22	55	2022-09-08 21:15:53.547359	869.9MHz	904e	6449e843bee3ffff	7cda000515e18000
23	56	2022-09-08 21:16:01.280291	868.3MHz	904e	6449e846bee3ffff	7cda000515e18000

Figure 4.9: Scrapped payloads of packets sent by PAN coordinator

4.3 Performance characteristics

GNURadio contains a built-in performance control mechanism called *Control Port*. *Control Port* is an interface introduced to GNURadio in order to standarize communication with the framework via

Remote Procedure Calls (RPC). This mechanism comes with a generic utility which allows monitoring of the performance in the flowgraph. This RPC is conducted with *gr-ctrlport-monitor* command which takes IP or host address and port as arguments. After connection has been established GNURadio periodically provides information per each block. These informations are based on a term called *item*. In this thesis a single *item* will correspond to a single sample. However in other flowgraphs that might not be the case. GNURadio documentation states that by an *item* can be meant for a example a pack of bits or set of filter coefficients. *Control Port* provides following informations about the *items*:

- noutput items - this variable is set by GNURadio during runtime. Its an indication of how many output items block might produce in a certain call to work
- items produced - this is the number of items the block produced in a certain call to work
- item throughput - number of items flowing through block per second
- clock cycles in call to work - number of CPU ticks spent in work method calls
- percentage of how full input buffers are
- percentage of how full output buffers are

Performance characteristics were gathered from the two channel flowgraph. Same flowgraph as one presented in the figure 3.12. Following pictures contain all characteristics that are provided by GNURadio. Figures are in descending order of distance from the RTL-SDR source block. Before the statistics were gathered 10000 packets were transmitted on channel 52.

rtl_source_c0::avg input % full	[]		Average of how full input buffers are
rtl_source_c0::avg noutput_items	3654.467041015625		Average noutput items
rtl_source_c0::avg nproduced	3654.467041015625		Average items produced
rtl_source_c0::avg output % full	[0.5252465009689331]		Average of how full output buffers are
rtl_source_c0::avg throughput	1997657.5	items/s	Average items throughput in call to work
rtl_source_c0::avg work time	12701.0791015625		Average clock cycles in call to work
rtl_source_c0::input % full	[]		how full input buffers are
rtl_source_c0::noutput_items	4096.0		noutput items
rtl_source_c0::nproduced	4096.0		items produced
rtl_source_c0::output % full	[0.5]		how full output buffers are
rtl_source_c0::total work time	4611957248.0		Total clock cycles in calls to work
rtl_source_c0::var input % full	[]		Var. of how full input buffers are
rtl_source_c0::var noutput_items	1016267.0625		Var. noutput items
rtl_source_c0::var nproduced	1016267.0625		Var. items produced
rtl_source_c0::var output % full	[1.446557575945917e-06]		Var. of how full output buffers are
rtl_source_c0::var work time	128888008.0		Var. clock cycles in call to work
rtl_source_c0::work time	9169.0		clock cycles in call to work

Figure 4.10: *RTL-SDR source* performance characteristics

waterfall_sink_c0::avg input % full	[0.07375943660736084]		Average of how full input buffers are
waterfall_sink_c0::avg noutput_items	4521.748046875		Average noutput items
waterfall_sink_c0::avg nproduced	3953.90625		Average items produced
waterfall_sink_c0::avg output % full	[]		Average of how full output buffers are
waterfall_sink_c0::avg throughput	2287171.75	items/s	Average items throughput in call to work
waterfall_sink_c0::avg work time	10879.4365234375		Average clock cycles in call to work
waterfall_sink_c0::input % full	[0.05481626093387604]		how full input buffers are
waterfall_sink_c0::noutput_items	1027.0		noutput items
waterfall_sink_c0::nproduced	578.0		items produced
waterfall_sink_c0::output % full	[]		how full output buffers are
waterfall_sink_c0::total work time	3930280960.0		Total clock cycles in calls to work
waterfall_sink_c0::var input % full	[2.0430279334959778e-07]		Var. of how full input buffers are
waterfall_sink_c0::var noutput_items	700453.75		Var. noutput items
waterfall_sink_c0::var nproduced	828667.25		Var. items produced
waterfall_sink_c0::var output % full	[]		Var. of how full output buffers are
waterfall_sink_c0::var work time	4651714048.0		Var. clock cycles in call to work
waterfall_sink_c0::work time	1679.0		clock cycles in call to work

Figure 4.11: *Waterfall sink* performance characteristics

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg input % full	[0.45375266671180725]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg noutput_items	3851.955078125		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg nproduced	3851.955078125		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg output % full	[0.00506945513188839]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg throughput	1997644.0	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg work time	514527.28125		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::input % full	[0.029300451278686523]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::noutput_items	240.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::nproduced	240.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::output % full	[0.0001220703125]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::total work time	155918843904.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var input % full	[1.4973458064559964e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var noutput_items	457460.25		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var nproduced	457460.25		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var output % full	[1.672877658620564e-08]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var work time	99999145984.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::work time	31933.0		clock cycles in call to work

Figure 4.12: *Frequency Xlating FIR Filter* performance characteristics on channel 52

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg input % full	[0.4596967101097107]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg noutput_items	3854.108642578125		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg nproduced	3854.108642578125		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg output % full	[0.004743591882288456]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg throughput	1997620.25	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg work time	515586.90625		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::input % full	[0.029300451278686523]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::noutput_items	240.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::nproduced	240.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::output % full	[0.0001220703125]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::total work time	157252370432.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var input % full	[1.5072122323545045e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var noutput_items	453243.9375		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var nproduced	453243.9375		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var output % full	[1.5552862109302623e-08]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var work time	99012329472.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::work time	25326.0		clock cycles in call to work

Figure 4.13: *Frequency Xlating FIR Filter* performance characteristics on channel 68

simple_squelch_cc0::avg input % full	[0.0014646891504526138]		Average of how full input buffers are
simple_squelch_cc0::avg noutput_items	3847.0791015625		Average noutput items
simple_squelch_cc0::avg nproduced	3847.0791015625		Average items produced
simple_squelch_cc0::avg output % full	[0.017966551706194878]		Average of how full output buffers are
simple_squelch_cc0::avg throughput	1997515.25	items/s	Average items throughput in call to work
simple_squelch_cc0::avg work time	19438.4453125		Average clock cycles in call to work
simple_squelch_cc0::input % full	[0.0]		how full input buffers are
simple_squelch_cc0::noutput_items	240.0		noutput items
simple_squelch_cc0::nproduced	240.0		items produced
simple_squelch_cc0::output % full	[0.4708251953125]		how full output buffers are
simple_squelch_cc0::total work time	6783220224.0		Total clock cycles in calls to work
simple_squelch_cc0::var input % full	[4.197286074258955e-09]		Var. of how full input buffers are
simple_squelch_cc0::var noutput_items	470135.78125		Var. noutput items
simple_squelch_cc0::var nproduced	470135.78125		Var. items produced
simple_squelch_cc0::var output % full	[5.148584492076225e-08]		Var. of how full output buffers are
simple_squelch_cc0::var work time	145257008.0		Var. clock cycles in call to work
simple_squelch_cc0::work time	1719.0		clock cycles in call to work

Figure 4.14: *Simple Squelch* performance characteristics on channel 52

simple_squelch_cc1::avg input % full	[0.0014281950425356627]		Average of how full input buffers are
simple_squelch_cc1::avg noutput_items	3849.560791015625		Average noutput items
simple_squelch_cc1::avg nproduced	3849.560791015625		Average items produced
simple_squelch_cc1::avg output % full	[0.01783638633787632]		Average of how full output buffers are
simple_squelch_cc1::avg throughput	1997507.375	items/s	Average items throughput in call to work
simple_squelch_cc1::avg work time	19086.201171875		Average clock cycles in call to work
simple_squelch_cc1::input % full	[0.0]		how full input buffers are
simple_squelch_cc1::noutput_items	240.0		noutput items
simple_squelch_cc1::nproduced	240.0		items produced
simple_squelch_cc1::output % full	[0.500244140625]		how full output buffers are
simple_squelch_cc1::total work time	6736927232.0		Total clock cycles in calls to work
simple_squelch_cc1::var input % full	[4.046142088043325e-09]		Var. of how full input buffers are
simple_squelch_cc1::var noutput_items	464796.96875		Var. noutput items
simple_squelch_cc1::var nproduced	464796.96875		Var. items produced
simple_squelch_cc1::var output % full	[5.053129825682845e-08]		Var. of how full output buffers are
simple_squelch_cc1::var work time	142243280.0		Var. clock cycles in call to work
simple_squelch_cc1::work time	2287.0		clock cycles in call to work

Figure 4.15: *Simple Squelch* performance characteristics on channel 68

quadrature_demod_cf0::avg input % full	[0.0015635393792763352]		Average of how full input buffers are
quadrature_demod_cf0::avg noutput_items	2252.2607421875		Average noutput items
quadrature_demod_cf0::avg nproduced	2252.2607421875		Average items produced
quadrature_demod_cf0::avg output % full	[0.10605209320783615]		Average of how full output buffers are
quadrature_demod_cf0::avg throughput	1997514.0	items/s	Average items throughput in call to work
quadrature_demod_cf0::avg work time	29079.80078125		Average clock cycles in call to work
quadrature_demod_cf0::input % full	[0.00012208521366119385]		how full input buffers are
quadrature_demod_cf0::noutput_items	2.0		noutput items
quadrature_demod_cf0::nproduced	2.0		items produced
quadrature_demod_cf0::output % full	[0.26544189453125]		how full output buffers are
quadrature_demod_cf0::total work time	16723592192.0		Total clock cycles in calls to work
quadrature_demod_cf0::var input % full	[2.7180360184786423e-09]		Var. of how full input buffers are
quadrature_demod_cf0::var noutput_items	3926793.5		Var. noutput items
quadrature_demod_cf0::var nproduced	3926793.5		Var. items produced
quadrature_demod_cf0::var output % full	[1.8435954984852287e-07]		Var. of how full output buffers are
quadrature_demod_cf0::var work time	1045511232.0		Var. clock cycles in call to work
quadrature_demod_cf0::work time	1703.0		clock cycles in call to work

Figure 4.16: *Quadrature Demod* performance characteristics on channel 52

quadrature_demod_cf1::avg input % full	[0.0015013230731710792]		Average of how full input buffers are
quadrature_demod_cf1::avg noutput_items	2252.247314453125		Average noutput items
quadrature_demod_cf1::avg nproduced	2252.247314453125		Average items produced
quadrature_demod_cf1::avg output % full	[0.10584399849176407]		Average of how full output buffers are
quadrature_demod_cf1::avg throughput	1997510.5	items/s	Average items throughput in call to work
quadrature_demod_cf1::avg work time	28137.931640625		Average clock cycles in call to work
quadrature_demod_cf1::input % full	[0.00012208521366119385]		how full input buffers are
quadrature_demod_cf1::noutput_items	2.0		noutput items
quadrature_demod_cf1::nproduced	2.0		items produced
quadrature_demod_cf1::output % full	[0.26544189453125]		how full output buffers are
quadrature_demod_cf1::total work time	16328359936.0		Total clock cycles in calls to work
quadrature_demod_cf1::var input % full	[2.5862540997678707e-09]		Var. of how full input buffers are
quadrature_demod_cf1::var noutput_items	3925392.5		Var. noutput items
quadrature_demod_cf1::var nproduced	3925392.5		Var. items produced
quadrature_demod_cf1::var output % full	[1.8233215826057858e-07]		Var. of how full output buffers are
quadrature_demod_cf1::var work time	998822784.0		Var. clock cycles in call to work
quadrature_demod_cf1::work time	1002.0		clock cycles in call to work

Figure 4.17: *Quadrature Demod* performance characteristics on channel 68

moving_average0::avg input % full	[0.0039478931576013565]		Average of how full input buffers are
moving_average0::avg noutput_items	2575.074462890625		Average noutput items
moving_average0::avg nproduced	2531.690185546875		Average items produced
moving_average0::avg output % full	[0.09128758311271667]		Average of how full output buffers are
moving_average0::avg throughput	2035683.625	items/s	Average items throughput in call to work
moving_average0::avg work time	11591.8505859375		Average clock cycles in call to work
moving_average0::input % full	[0.0008545443415641785]		how full input buffers are
moving_average0::noutput_items	336.0		noutput items
moving_average0::nproduced	336.0		items produced
moving_average0::output % full	[0.245361328125]		how full output buffers are
moving_average0::total work time	5455483904.0		Total clock cycles in calls to work
moving_average0::var input % full	[8.388100880551974e-09]		Var. of how full input buffers are
moving_average0::var noutput_items	3566227.25		Var. noutput items
moving_average0::var nproduced	3382068.25		Var. items produced
moving_average0::var output % full	[1.9395901063035126e-07]		Var. of how full output buffers are
moving_average0::var work time	118006224.0		Var. clock cycles in call to work
moving_average0::work time	2450.0		clock cycles in call to work

Figure 4.18: *Moving Average* performance characteristics on channel 52

moving_average1::avg input % full	[0.0038418276235461235]		Average of how full input buffers are
moving_average1::avg noutput_items	2578.151123046875		Average noutput items
moving_average1::avg nproduced	2535.649169921875		Average items produced
moving_average1::avg output % full	[0.09056852012872696]		Average of how full output buffers are
moving_average1::avg throughput	2034972.5	items/s	Average items throughput in call to work
moving_average1::avg work time	11561.18359375		Average clock cycles in call to work
moving_average1::input % full	[0.0008545443415641785]		how full input buffers are
moving_average1::noutput_items	95.0		noutput items
moving_average1::nproduced	95.0		items produced
moving_average1::output % full	[0.24639892578125]		how full output buffers are
moving_average1::total work time	5560412672.0		Total clock cycles in calls to work
moving_average1::var input % full	[7.987648764640198e-09]		Var. of how full input buffers are
moving_average1::var noutput_items	3554695.75		Var. noutput items
moving_average1::var nproduced	3376512.25		Var. items produced
moving_average1::var output % full	[1.8830348835763289e-07]		Var. of how full output buffers are
moving_average1::var work time	116655896.0		Var. clock cycles in call to work
moving_average1::work time	1170.0		clock cycles in call to work

Figure 4.19: *Moving Average* performance characteristics on channel 68

clock_recovery_mm_ff0::avg input % full	[0.03207385540008545]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	30.365467071533203		Average noutput items
clock_recovery_mm_ff0::avg nproduced	30.365467071533203		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.0018174160504713655]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	49960.5078125	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	6504.38720703125		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0013428553938865662]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	1.0		noutput items
clock_recovery_mm_ff0::nproduced	1.0		items produced
clock_recovery_mm_ff0::output % full	[0.00030517578125]		how full output buffers are
clock_recovery_mm_ff0::total work time	5797249024.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[3.5987092417144595e-08]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	669.8028564453125		Var. noutput items
clock_recovery_mm_ff0::var nproduced	669.8028564453125		Var. items produced
clock_recovery_mm_ff0::var output % full	[2.0391535127117777e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	48853796.0		Var. clock cycles in call to work
clock_recovery_mm_ff0::work time	1017.0		clock cycles in call to work

Figure 4.20: *Clock Recovery MM* performance characteristics on channel 52

clock_recovery_mm_ff1::avg input % full	[0.03265376016497612]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	31.185455322265625		Average noutput items
clock_recovery_mm_ff1::avg nproduced	31.185455322265625		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.0019813147373497486]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	49879.40625	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	5498.46435546875		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.00079350546002388]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	1.0		noutput items
clock_recovery_mm_ff1::nproduced	1.0		items produced
clock_recovery_mm_ff1::output % full	[0.00628662109375]		how full output buffers are
clock_recovery_mm_ff1::total work time	4839973376.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[3.7096892668841974e-08]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	661.5498657226562		Var. noutput items
clock_recovery_mm_ff1::var nproduced	661.5498657226562		Var. items produced
clock_recovery_mm_ff1::var output % full	[2.250908348599978e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	30189046.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	1105.0		clock cycles in call to work

Figure 4.21: *Clock Recovery MM* performance characteristics on channel 68

binary_slicer_fb0::avg input % full	[0.0008758499170653522]		Average of how full input buffers are
binary_slicer_fb0::avg noutput_items	32.10028076171875		Average noutput items
binary_slicer_fb0::avg nproduced	32.10028076171875		Average items produced
binary_slicer_fb0::avg output % full	[0.0006361612376309931 3]		Average of how full output buffers are
binary_slicer_fb0::avg throughput	49922.08984375	items/s	Average items throughput in call to work
binary_slicer_fb0::avg work time	1808.0849609375		Average clock cycles in call to work
binary_slicer_fb0::input % full	[0.0]		how full input buffers are
binary_slicer_fb0::noutput_items	30.0		noutput items
binary_slicer_fb0::nproduced	30.0		items produced
binary_slicer_fb0::output % full	[0.0010223388671875]		how full output buffers are
binary_slicer_fb0::total work time	1462386560.0		Total clock cycles in calls to work
binary_slicer_fb0::var input % full	[1.0829620533669981e-09]		Var. of how full input buffers are
binary_slicer_fb0::var noutput_items	580.6356201171875		Var. noutput items
binary_slicer_fb0::var nproduced	580.6356201171875		Var. items produced
binary_slicer_fb0::var output % full	[7.80969733327197e-10]		Var. of how full output buffers are
binary_slicer_fb0::var work time	2070917.75		Var. clock cycles in call to work
binary_slicer_fb0::work time	4538.0		clock cycles in call to work

Figure 4.22: *Binary Slicer* performance characteristics on channel 52

binary_slicer_fb1::avg input % full	[0.000929612317122519]		Average of how full input buffers are
binary_slicer_fb1::avg noutput_items	33.82304382324219		Average noutput items
binary_slicer_fb1::avg nproduced	33.82304382324219		Average items produced
binary_slicer_fb1::avg output % full	[0.0006408143672160804]		Average of how full output buffers are
binary_slicer_fb1::avg throughput	49880.65234375	items/s	Average items throughput in call to work
binary_slicer_fb1::avg work time	1830.2584228515625		Average clock cycles in call to work
binary_slicer_fb1::input % full	[0.0]		how full input buffers are
binary_slicer_fb1::noutput_items	22.0		noutput items
binary_slicer_fb1::nproduced	22.0		items produced
binary_slicer_fb1::output % full	[0.0013275146484375]		how full output buffers are
binary_slicer_fb1::total work time	1438800384.0		Total clock cycles in calls to work
binary_slicer_fb1::var input % full	[1.1825970203105385e-09]		Var. of how full input buffers are
binary_slicer_fb1::var noutput_items	607.12744140625		Var. noutput items
binary_slicer_fb1::var nproduced	607.12744140625		Var. items produced
binary_slicer_fb1::var output % full	[8.152055697152605e-10]		Var. of how full output buffers are
binary_slicer_fb1::var work time	2081988.0		Var. clock cycles in call to work
binary_slicer_fb1::work time	1021.0		clock cycles in call to work

Figure 4.23: *Binary Slicer* performance characteristics on channel 68

Packet parser0::avg input % full	[0.0017171596409752965]		Average of how full input buffers are
Packet parser0::avg noutput_items	84.39335632324219		Average noutput items
Packet parser0::avg nproduced	84.39335632324219		Average items produced
Packet parser0::avg output % full	[2.7135987693327 48e-05]		Average of how full output buffers are
Packet parser0::avg throughput	49947.00390625	items/s	Average items throughput in call to work
Packet parser0::avg work time	181227.953125		Average clock cycles in call to work
Packet parser0::input % full	[0.0018310826271772385]		how full input buffers are
Packet parser0::noutput_items	32.0		noutput items
Packet parser0::nproduced	32.0		items produced
Packet parser0::output % full	[1.52587890625e-05]		how full output buffers are
Packet parser0::total work time	67753193472.0		Total clock cycles in calls to work
Packet parser0::var input % full	[4.593104563355155e-09]		Var. of how full input buffers are
Packet parser0::var noutput_items	5780.62158203125		Var. noutput items
Packet parser0::var nproduced	5780.62158203125		Var. items produced
Packet parser0::var output % full	[7.258406375942883e-11]		Var. of how full output buffers are
Packet parser0::var work time	61788352512.0		Var. clock cycles in call to work
Packet parser0::work time	118986.0		clock cycles in call to work

Figure 4.24: *Packet Parser* performance characteristics on channel 52

Packet parser1::avg input % full	[0.0015753579791635275]		Average of how full input buffers are
Packet parser1::avg noutput_items	77.60182189941406		Average noutput items
Packet parser1::avg nproduced	77.60182189941406		Average items produced
Packet parser1::avg output % full	[2.6834566597244702e-05]		Average of how full output buffers are
Packet parser1::avg throughput	49877.72265625	items/s	Average items throughput in call to work
Packet parser1::avg work time	151571.96875		Average clock cycles in call to work
Packet parser1::input % full	[0.0]		how full input buffers are
Packet parser1::noutput_items	2.0		noutput items
Packet parser1::nproduced	2.0		items produced
Packet parser1::output % full	[1.52587890625e-05]		how full output buffers are
Packet parser1::total work time	62121508864.0		Total clock cycles in calls to work
Packet parser1::var input % full	[3.843789730950675e-09]		Var. of how full input buffers are
Packet parser1::var noutput_items	4261.81005859375		Var. noutput items
Packet parser1::var nproduced	4261.81005859375		Var. items produced
Packet parser1::var output % full	[6.547491715469533e-11]		Var. of how full output buffers are
Packet parser1::var work time	20395956224.0		Var. clock cycles in call to work
Packet parser1::work time	72919.0		clock cycles in call to work

Figure 4.25: *Packet Parser* performance characteristics on channel 68

time_sink_f0::avg input % full	[0.00026336568407714367]		Average of how full input buffers are
time_sink_f0::avg noutput_items	78.20947265625		Average noutput items
time_sink_f0::avg nproduced	78.15373229980469		Average items produced
time_sink_f0::avg output % full	[]		Average of how full output buffers are
time_sink_f0::avg throughput	49994.76953125	items/s	Average items throughput in call to work
time_sink_f0::avg work time	4770.35986328125		Average clock cycles in call to work
time_sink_f0::input % full	[6.103888154029846e-05]		how full input buffers are
time_sink_f0::noutput_items	8.0		noutput items
time_sink_f0::nproduced	8.0		items produced
time_sink_f0::output % full	[]		how full output buffers are
time_sink_f0::total work time	2096999168.0		Total clock cycles in calls to work
time_sink_f0::var input % full	[5.991357410195519e-10]		Var. of how full input buffers are
time_sink_f0::var noutput_items	5761.52490234375		Var. noutput items
time_sink_f0::var nproduced	5745.69970703125		Var. items produced
time_sink_f0::var output % full	[]		Var. of how full output buffers are
time_sink_f0::var work time	1993323392.0		Var. clock cycles in call to work
time_sink_f0::work time	2589.0		clock cycles in call to work

Figure 4.26: *Time Sink* performance characteristics on channel 52

time_sink_f1::avg input % full	[0.00028188974829390645]		Average of how full input buffers are
time_sink_f1::avg noutput_items	73.028076171875		Average noutput items
time_sink_f1::avg nproduced	72.98438262939453		Average items produced
time_sink_f1::avg output % full	[]		Average of how full output buffers are
time_sink_f1::avg throughput	49908.41015625	items/s	Average items throughput in call to work
time_sink_f1::avg work time	6975.96044921875		Average clock cycles in call to work
time_sink_f1::input % full	[0.00018311664462089539]		how full input buffers are
time_sink_f1::noutput_items	8.0		noutput items
time_sink_f1::nproduced	8.0		items produced
time_sink_f1::output % full	[]		how full output buffers are
time_sink_f1::total work time	3296085248.0		Total clock cycles in calls to work
time_sink_f1::var input % full	[5.96597105051444e-10]		Var. of how full input buffers are
time_sink_f1::var noutput_items	4721.89794921875		Var. noutput items
time_sink_f1::var nproduced	4708.96435546875		Var. items produced
time_sink_f1::var output % full	[]		Var. of how full output buffers are
time_sink_f1::var work time	19690311680.0		Var. clock cycles in call to work
time_sink_f1::work time	5938.0		clock cycles in call to work

Figure 4.27: *Time Sink* performance characteristics on channel 68

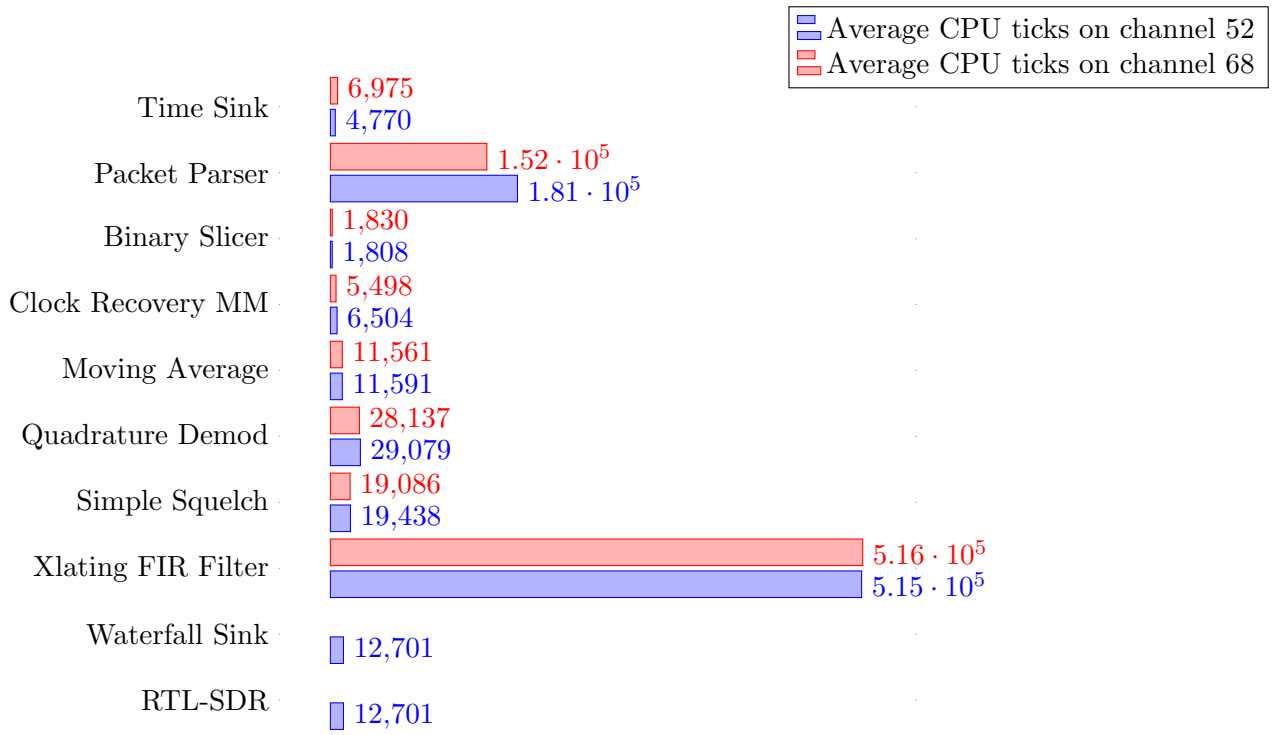

Figure 4.28: Average CPU ticks per each block. Remark: *RTL-SDR* and *Waterfall Sink* blocks are processing samples in whole spectrum not in any specific channels

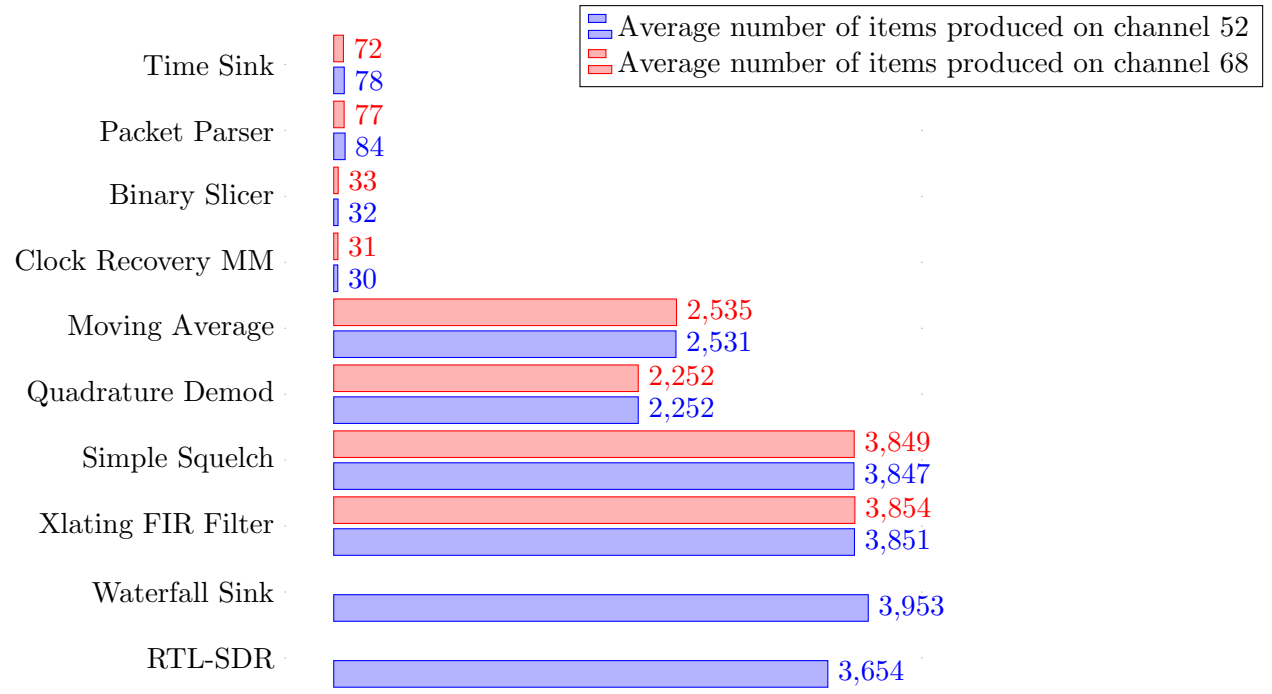

Figure 4.29: Average number of items produced per each block. Remark: *RTL-SDR* and *Waterfall Sink* blocks are processing samples in whole spectrum not in any specific channels

4.4 Impact of blocks parameters

Out of blocks in the final version of the flowgraph (figure 3.12) three blocks have the biggest impact on Packet Delivery Ratio (PDR) and correctness of the packet. Those blocks are *Frequency Xlating FIR Filter*, *Moving Average* and *Clock Recovery MM*. In case of *Frequency Xlating FIR Filter* impact of cut off frequency and transition bandwidth changes is presented in the subsection 4.4.1. In case of *Moving Average* impact of averaging window size is presented in the subsection 4.4.2. Impact of the Omega parameter changes in the *Clock Recovery MM* block is presented in the subsection 4.4.3. In each of these cases 10000 packets were transmitted on channel 52. After any change in parameters flowgraph has been restarted. As a reminder the original values of these parameters are:

- Cut off frequency - 80kHz
- Transition bandwidth - 20kHz
- Length - 15
- Omega - 40.1

Table 4.1 presents PDR value before any change in the flowgraph has been made.

	Number of packets	Percentage
Received	9808	98.08%
Correct	9795	97.95%
Corrupted	13	0.13%

Table 4.1: Number of correct, incorrect and all packets that were captured with original values of cut off frequency, transition bandwidth, omega and length

Performance characteristics are presented in the section 4.3 in the figures 4.12, 4.13, 4.18, 4.19, 4.20 and 4.21.

4.4.1 Xlating FIR Filter

	Number of packets	Percentage
Received	9581	95.81%
Correct	9527	95.27%
Corrupted	54	0.54%

Table 4.2: Number of correct, incorrect and all packets that were captured with cut off frequency = 30kHz and transition bandwidth = 70kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg input % full	[0.4422245919704437]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg noutput_items	3921.572509765625		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg nproduced	3921.572509765625		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg output % full	[0.0213795006275177]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg throughput	2001356.375	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg work time	289806.15625		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::input % full	[0.008301794528961182]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::noutput_items	68.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::nproduced	68.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::output % full	[0.49169921875]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::total work time	22957717504.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var input % full	[5.582657195191132e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var noutput_items	542129.125		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var nproduced	542129.125		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var output % full	[2.698954801871878e-07]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var work time	41894068224.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::work time	3537.0		clock cycles in call to work

Figure 4.30: Frequency Xlating FIR Filter performance characteristics on channel 52 with cut off frequency = 30kHz and transition bandwidth = 70kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg input % full	[0.4324629604816437]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg noutput_items	3920.437255859375		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg nproduced	3920.437255859375		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg output % full	[0.02045953646302232]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg throughput	2001354.0	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg work time	287361.0625		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::input % full	[0.008301794528961182]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::noutput_items	68.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::nproduced	68.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::output % full	[0.49169921875]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::total work time	22770589696.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var input % full	[5.4578413255512714e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var noutput_items	546632.125		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var nproduced	546632.125		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var output % full	[2.582068532319681e-07]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var work time	41276825600.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::work time	3582.0		clock cycles in call to work

Figure 4.31: Frequency Xlating FIR Filter performance characteristics on channel 68 with cut off frequency = 30kHz and transition bandwidth = 70kHz

	Number of packets	Percentage
Received	9850	98.50%
Correct	9812	98.12%
Corrupted	38	0.38%

Table 4.3: Number of correct, incorrect and all packets that were captured with cut off frequency = 50kHz and transition bandwidth = 50kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg input % full	[0.42251747846603394]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg noutput_items	3922.927001953125		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg nproduced	3922.927001953125		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg output % full	[0.015367347747087479]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg throughput	2000595.375	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg work time	342326.375		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::input % full	[0.49993896484375]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::noutput_items	4096.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::nproduced	4096.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::output % full	[0.0001220703125]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::total work time	27179395072.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var input % full	[5.32164676769753e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var noutput_items	481327.5625		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var nproduced	481327.5625		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var output % full	[1.9355317704139452e-07]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var work time	64770174976.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::work time	221177.0		clock cycles in call to work

Figure 4.32: Frequency Xlating FIR Filter performance characteristics on channel 52 with cut off frequency = 50kHz and transition bandwidth = 50kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg input % full	[0.4357506334781647]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg noutput_items	3921.78955078125		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg nproduced	3921.78955078125		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg output % full	[0.01606069691479206]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg throughput	2000633.75	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg work time	344708.375		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::input % full	[0.5117812156677246]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::noutput_items	3903.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::nproduced	3903.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::output % full	[0.0001220703125]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::total work time	27377381376.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var input % full	[5.486592272063717e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var noutput_items	485463.4375		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var nproduced	485463.4375		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var output % full	[2.0222229579758277e-07]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var work time	65876185088.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::work time	201283.0		clock cycles in call to work

Figure 4.33: Frequency Xlating FIR Filter performance characteristics on channel 68 with cut off frequency = 50kHz and transition bandwidth = 50kHz

	Number of packets	Percentage
Received	9804	98.04%
Correct	9776	97.76%
Corrupted	28	0.28%

Table 4.4: Number of correct, incorrect and all packets that were captured with cut off frequency = 65kHz and transition bandwidth = 35kHz

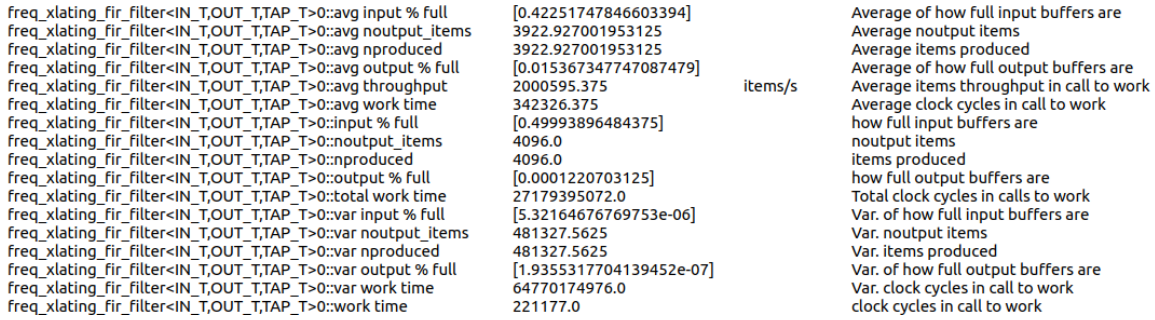

Figure 4.34: Frequency Xlating FIR Filter performance characteristics on channel 52 with cut off frequency = 65kHz and transition bandwidth = 35kHz

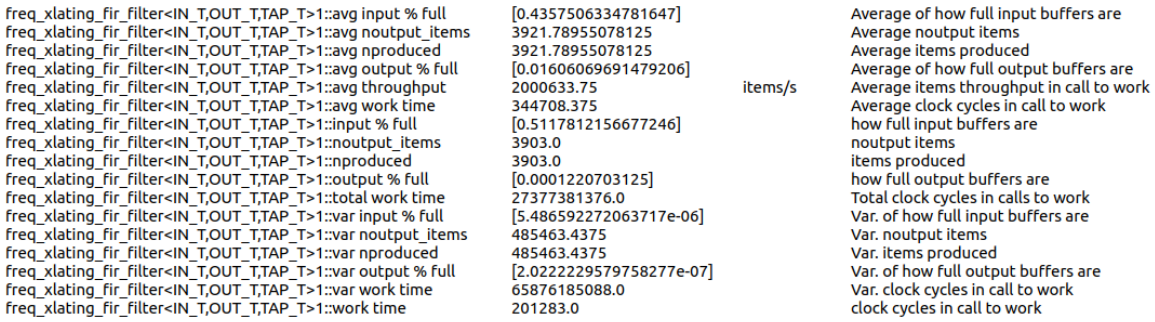

Figure 4.35: Frequency Xlating FIR Filter performance characteristics on channel 68 with cut off frequency = 65kHz and transition bandwidth = 35kHz

	Number of packets	Percentage
Received	9803	98.03%
Correct	9774	97.74%
Corrupted	29	0.29%

Table 4.5: Number of correct, incorrect and all packets that were captured with cut off frequency = 95kHz and transition bandwidth = 5kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg input % full	[0.5148124694824219]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg noutput_items	3544.63671875		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg nproduced	3544.63671875		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg output % full	[0.0009087737998925149]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg throughput	2000833.625	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::avg work time	1381833.375		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::input % full	[0.11744597554206848]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::noutput_items	962.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::nproduced	962.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::output % full	[0.000122070312 5]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::total work time	113078566912.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var input % full	[6.291166755545419e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var noutput_items	412748.90625		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var nproduced	412748.90625		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var output % full	[1.1105495190122383e-08]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::var work time	112040919040.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>0::work time	353364.0		clock cycles in call to work

Figure 4.36: Frequency Xlating FIR Filter performance characteristics on channel 52 with cut off frequency = 95kHz and transition bandwidth = 5kHz

freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg input % full	[0.5142645835876465]		Average of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg noutput_items	3545.521240234375		Average noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg nproduced	3545.521240234375		Average items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg output % full	[0.00084913498722016 81]		Average of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg throughput	2000824.75	items/s	Average items throughput in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::avg work time	1383096.25		Average clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::input % full	[0.11744597554206848]		how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::noutput_items	962.0		noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::nproduced	962.0		items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::output % full	[0.000122070312 5]		how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::total work time	113152548864.0		Total clock cycles in calls to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var input % full	[6.286007646849612e-06]		Var. of how full input buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var noutput_items	411049.125		Var. noutput items
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var nproduced	411049.125		Var. items produced
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var output % full	[1.0379227255441492e-08]		Var. of how full output buffers are
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::var work time	116108763136.0		Var. clock cycles in call to work
freq_xlating_fir_filter<IN_T,OUT_T,TAP_T>1::work time	376636.0		clock cycles in call to work

Figure 4.37: Frequency Xlating FIR Filter performance characteristics on channel 68 with cut off frequency = 95kHz and transition bandwidth = 5kHz

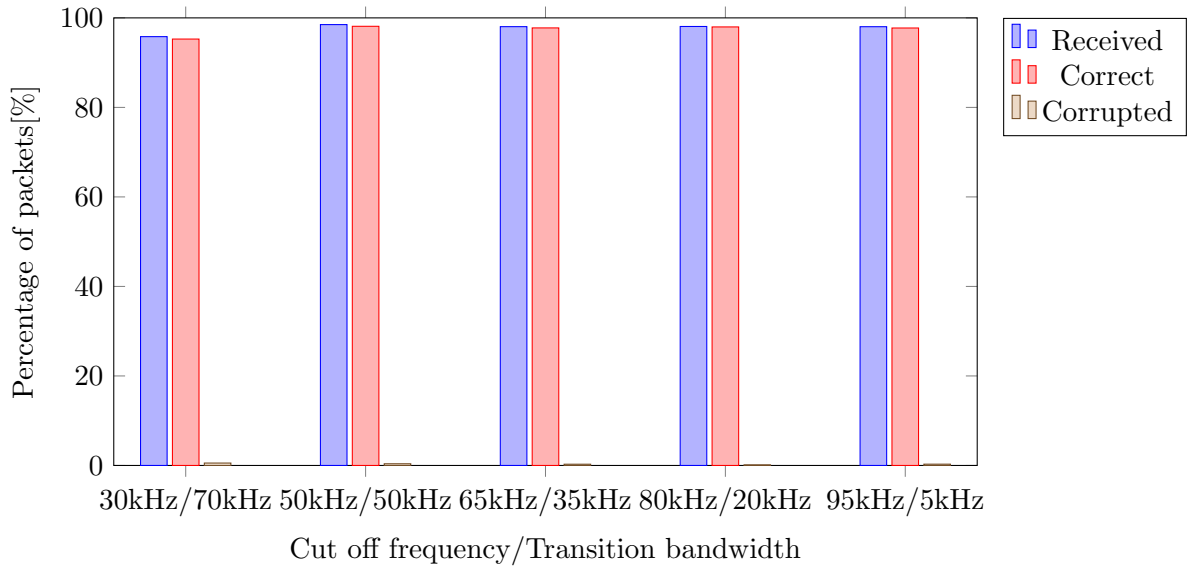

Figure 4.38: Percentage of received, correct and corrupted packets in different configurations of cut off and transition frequencies

4.4.2 Moving Average

	Number of packets	Percentage
Received	9703	97.03%
Correct	7592	75.92%
Corrupted	2111	21.11%

Table 4.6: Number of correct, incorrect and all packets that were captured without *Moving Average* block in the flowgraph

Block has been removed completely from the flowgraph so no performance characteristics could be gathered.

	Number of packets	Percentage
Received	9842	98.42%
Correct	9202	92.02%
Corrupted	640	6.40%

Table 4.7: Number of correct, incorrect and all packets that were captured with length = 5

moving_average0::avg input % full	[0.0025843423791229725]		Average of how full input buffers are
moving_average0::avg noutput_items	2576.703857421875		Average noutput items
moving_average0::avg nproduced	2541.177001953125		Average items produced
moving_average0::avg output % full	[0.0899699330329895]		Average of how full output buffers are
moving_average0::avg throughput	2028421.25	items/s	Average items throughput in call to work
moving_average0::avg work time	12360.3291015625		Average clock cycles in call to work
moving_average0::input % full	[0.00024415552616119385]		how full input buffers are
moving_average0::noutput_items	96.0		noutput items
moving_average0::nproduced	96.0		items produced
moving_average0::output % full	[0.24615478515625]		how full output buffers are
moving_average0::total work time	1214810624.0		Total clock cycles in calls to work
moving_average0::var input % full	[2.6297851007939244e-08]		Var. of how full input buffers are
moving_average0::var noutput_items	3511076.5		Var. noutput items
moving_average0::var nproduced	3396274.0		Var. items produced
moving_average0::var output % full	[9.155195357379853e-07]		Var. of how full output buffers are
moving_average0::var work time	180779728.0		Var. clock cycles in call to work
moving_average0::work time	4369.0		clock cycles in call to work

Figure 4.39: Moving Average performance characteristics on channel 52 with length = 5

moving_average1::avg input % full	[0.0026292430702596903]		Average of how full input buffers are
moving_average1::avg noutput_items	2576.85791015625		Average noutput items
moving_average1::avg nproduced	2540.816162109375		Average items produced
moving_average1::avg output % full	[0.08997885882854462]		Average of how full output buffers are
moving_average1::avg throughput	2028822.625	items/s	Average items throughput in call to work
moving_average1::avg work time	12378.111328125		Average clock cycles in call to work
moving_average1::input % full	[0.00024415552616119385]		how full input buffers are
moving_average1::noutput_items	3855.0		noutput items
moving_average1::nproduced	3855.0		items produced
moving_average1::output % full	[0.0018310546875]		how full output buffers are
moving_average1::total work time	1216784000.0		Total clock cycles in calls to work
moving_average1::var input % full	[2.6750424098054282e-08]		Var. of how full input buffers are
moving_average1::var noutput_items	3514650.25		Var. noutput items
moving_average1::var nproduced	3396609.0		Var. items produced
moving_average1::var output % full	[9.154614986073284e-07]		Var. of how full output buffers are
moving_average1::var work time	182167360.0		Var. clock cycles in call to work
moving_average1::work time	40758.0		clock cycles in call to work

Figure 4.40: Moving Average performance characteristics on channel 68 with length = 5

	Number of packets	Percentage
Received	9864	98.64%
Correct	9759	97.59%
Corrupted	105	1.05%

Table 4.8: Number of correct, incorrect and all packets that were captured with length = 10

moving_average0::avg input % full	[0.0029768417589366436]		Average of how full input buffers are
moving_average0::avg noutput_items	2572.231689453125		Average noutput items
moving_average0::avg nproduced	2535.60009765625		Average items produced
moving_average0::avg output % full	[0.09044627100229263]		Average of how full output buffers are
moving_average0::avg throughput	2029431.0	items/s	Average items throughput in call to work
moving_average0::avg work time	11011.767578125		Average clock cycles in call to work
moving_average0::input % full	[0.0005493499338626862]		how full input buffers are
moving_average0::noutput_items	96.0		noutput items
moving_average0::nproduced	96.0		items produced
moving_average0::output % full	[0.24554443359375]		how full output buffers are
moving_average0::total work time	1039892480.0		Total clock cycles in calls to work
moving_average0::var input % full	[3.152465666289572e-08]		Var. of how full input buffers are
moving_average0::var noutput_items	3525144.0		Var. noutput items
moving_average0::var nproduced	3402530.5		Var. items produced
moving_average0::var output % full	[9.578229764883872e-07]		Var. of how full output buffers are
moving_average0::var work time	136454528.0		Var. clock cycles in call to work
moving_average0::work time	4782.0		clock cycles in call to work

Figure 4.41: Moving Average performance characteristics on channel 52 with length = 10

moving_average1::avg input % full	[0.002898674923926592]		Average of how full input buffers are
moving_average1::avg noutput_items	2577.109130859375		Average noutput items
moving_average1::avg nproduced	2541.4853515625		Average items produced
moving_average1::avg output % full	[0.0898873507976532]		Average of how full output buffers are
moving_average1::avg throughput	2028552.25	items/s	Average items throughput in call to work
moving_average1::avg work time	11054.1279296875		Average clock cycles in call to work
moving_average1::input % full	[0.0005493499338626862]		how full input buffers are
moving_average1::noutput_items	96.0		noutput items
moving_average1::nproduced	96.0		items produced
moving_average1::output % full	[0.2457275390625]		how full output buffers are
moving_average1::total work time	1041467456.0		Total clock cycles in calls to work
moving_average1::var input % full	[3.0768230629973914e-08]		Var. of how full input buffers are
moving_average1::var noutput_items	3509666.25		Var. noutput items
moving_average1::var nproduced	3396058.0		Var. items produced
moving_average1::var output % full	[9.541168992655003e-07]		Var. of how full output buffers are
moving_average1::var work time	135439808.0		Var. clock cycles in call to work
moving_average1::work time	6449.0		clock cycles in call to work

Figure 4.42: Moving Average performance characteristics on channel 68 with length = 10

	Number of packets	Percentage
Received	9846	98.46%
Correct	9815	98.15%
Corrupted	31	0.31%

Table 4.9: Number of correct, incorrect and all packets that were captured with length = 20

moving_average0::avg input % full	[0.003806130029261112]		Average of how full input buffers are
moving_average0::avg noutput_items	2580.710693359375		Average noutput items
moving_average0::avg nproduced	2542.19677734375		Average items produced
moving_average0::avg output % full	[0.09051525592803955]		Average of how full output buffers are
moving_average0::avg throughput	2031251.875	items/s	Average items throughput in call to work
moving_average0::avg work time	11786.03515625		Average clock cycles in call to work
moving_average0::input % full	[0.0011597387492656708]		how full input buffers are
moving_average0::noutput_items	336.0		noutput items
moving_average0::nproduced	336.0		items produced
moving_average0::output % full	[0.244873046875]		how full output buffers are
moving_average0::total work time	882368192.0		Total clock cycles in calls to work
moving_average0::var input % full	[5.0841951093616444e-08]		Var. of how full input buffers are
moving_average0::var noutput_items	3529222.5		Var. noutput items
moving_average0::var nproduced	3395287.0		Var. items produced
moving_average0::var output % full	[1.2090947620890802e-06]		Var. of how full output buffers are
moving_average0::var work time	156134928.0		Var. clock cycles in call to work
moving_average0::work time	5541.0		clock cycles in call to work

Figure 4.43: Moving Average performance characteristics on channel 52 with length = 20

moving_average1::avg input % full	[0.003755127312615514]		Average of how full input buffers are
moving_average1::avg noutput_items	2587.962158203125		Average noutput items
moving_average1::avg nproduced	2549.932373046875		Average items produced
moving_average1::avg output % full	[0.08957472443580627]		Average of how full output buffers are
moving_average1::avg throughput	2030784.5	items/s	Average items throughput in call to work
moving_average1::avg work time	11828.8779296875		Average clock cycles in call to work
moving_average1::input % full	[0.0011597387492656708]		how full input buffers are
moving_average1::noutput_items	336.0		noutput items
moving_average1::nproduced	336.0		items produced
moving_average1::output % full	[0.24517822265625]		how full output buffers are
moving_average1::total work time	882890112.0		Total clock cycles in calls to work
moving_average1::var input % full	[5.031322203308264e-08]		Var. of how full input buffers are
moving_average1::var noutput_items	3514407.5		Var. noutput items
moving_average1::var nproduced	3386159.75		Var. items produced
moving_average1::var output % full	[1.200170459014771e-06]		Var. of how full output buffers are
moving_average1::var work time	157385728.0		Var. clock cycles in call to work
moving_average1::work time	5354.0		clock cycles in call to work

Figure 4.44: Moving Average performance characteristics on channel 68 with length = 20

	Number of packets	Percentage
Received	9798	97.98%
Correct	9782	97.82%
Corrupted	16	0.16%

Table 4.10: Number of correct, incorrect and all packets that were captured with length = 25

moving_average0::avg input % full	[0.0037923683412373066]		Average of how full input buffers are
moving_average0::avg noutput_items	2580.96142578125		Average noutput items
moving_average0::avg nproduced	2545.5107421875		Average items produced
moving_average0::avg output % full	[0.08960695564746857]		Average of how full output buffers are
moving_average0::avg throughput	2027698.75	items/s	Average items throughput in call to work
moving_average0::avg work time	14111.41015625		Average clock cycles in call to work
moving_average0::input % full	[0.001464933156967163]		how full input buffers are
moving_average0::noutput_items	3856.0		noutput items
moving_average0::nproduced	3856.0		items produced
moving_average0::output % full	[0.00091552734375]		how full output buffers are
moving_average0::total work time	1799885312.0		Total clock cycles in calls to work
moving_average0::var input % full	[2.9735435447264535e-08]		Var. of how full input buffers are
moving_average0::var noutput_items	3503966.0		Var. noutput items
moving_average0::var nproduced	3391602.0		Var. items produced
moving_average0::var output % full	[7.025957415862649e-07]		Var. of how full output buffers are
moving_average0::var work time	222357296.0		Var. clock cycles in call to work
moving_average0::work time	43545.0		clock cycles in call to work

Figure 4.45: Moving Average performance characteristics on channel 52 with length = 25

moving_average1::avg input % full	[0.003833852242678404]		Average of how full input buffers are
moving_average1::avg noutput_items	2586.322021484375		Average noutput items
moving_average1::avg nproduced	2550.3134765625		Average items produced
moving_average1::avg output % full	[0.08927792310714722]		Average of how full output buffers are
moving_average1::avg throughput	2028052.0	items/s	Average items throughput in call to work
moving_average1::avg work time	14158.5361328125		Average clock cycles in call to work
moving_average1::input % full	[0.001464933156967163]		how full input buffers are
moving_average1::noutput_items	95.0		noutput items
moving_average1::nproduced	95.0		items produced
moving_average1::output % full	[0.24566650390625]		how full output buffers are
moving_average1::total work time	1802447232.0		Total clock cycles in calls to work
moving_average1::var input % full	[3.01176186212615e-08]		Var. of how full input buffers are
moving_average1::var noutput_items	3500655.25		Var. noutput items
moving_average1::var nproduced	3384781.75		Var. items produced
moving_average1::var output % full	[7.01341150488588e-07]		Var. of how full output buffers are
moving_average1::var work time	225649440.0		Var. clock cycles in call to work
moving_average1::work time	5088.0		clock cycles in call to work

Figure 4.46: Moving Average performance characteristics on channel 68 with length = 25

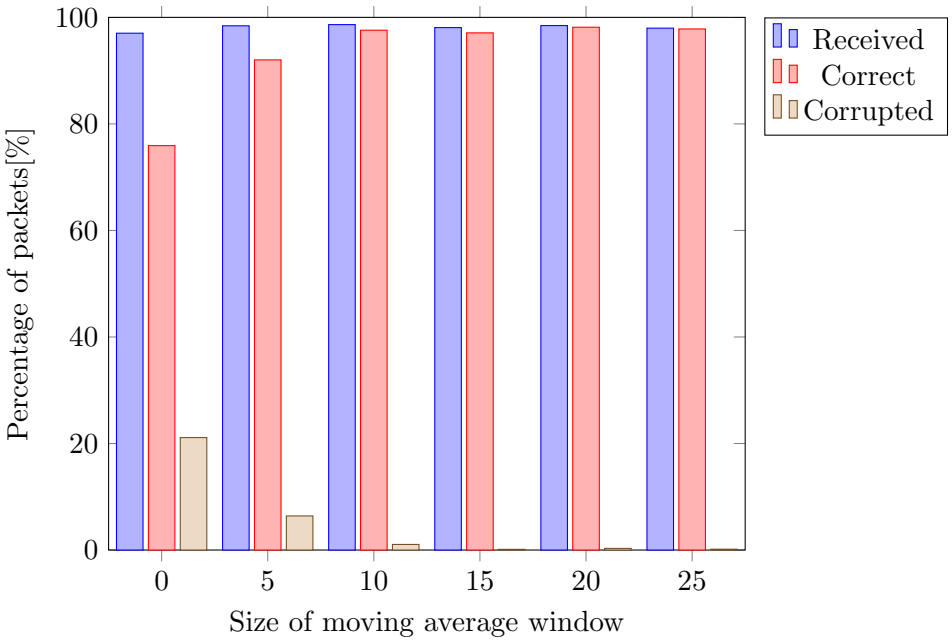

Figure 4.47: Percentage of received, correct and corrupted packets in different configurations of moving average window

4.4.3 Clock Recovery MM

	Number of packets	Percentage
Received	9464	94.64%
Correct	5367	53.67%
Corrupted	4097	40.97%

Table 4.11: Number of correct, incorrect and all packets that were captured with Omega = 39.9

clock_recovery_mm_ff0::avg input % full	[0.028660649433732033]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	26.356218338012695		Average noutput items
clock_recovery_mm_ff0::avg nproduced	26.356218338012695		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.001750827650539577]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	50253.3359375	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	5976.88037109375		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0005493499338626862]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	2.0		noutput items
clock_recovery_mm_ff0::nproduced	2.0		items produced
clock_recovery_mm_ff0::output % full	[0.00146484375]		how full output buffers are
clock_recovery_mm_ff0::total work time	1347773696.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[1.2710104613233852e-07]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	661.3343505859375		Var. noutput items
clock_recovery_mm_ff0::var nproduced	661.3343505859375		Var. items produced
clock_recovery_mm_ff0::var output % full	[7.764374920782302e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	47388252.0		Var. clock cycles in call to work

Figure 4.48: Clock Recovery MM performance characteristics on channel 52 with Omega = 39.9

clock_recovery_mm_ff1::avg input % full	[0.02920091338455677]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	27.189395904541016		Average noutput items
clock_recovery_mm_ff1::avg nproduced	27.189395904541016		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.0019852351397275925]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	50136.390625	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	4874.92626953125		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.0025636330246925354]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	4.0		noutput items
clock_recovery_mm_ff1::nproduced	4.0		items produced
clock_recovery_mm_ff1::output % full	[0.00640869140625]		how full output buffers are
clock_recovery_mm_ff1::total work time	1063176384.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[1.33896932652533e-07]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	663.6962890625		Var. noutput items
clock_recovery_mm_ff1::var nproduced	663.6962890625		Var. items produced
clock_recovery_mm_ff1::var output % full	[9.103033882240652e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	33446856.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	3806.0		clock cycles in call to work

Figure 4.49: Clock Recovery MM performance characteristics on channel 68 with Omega = 39.9

	Number of packets	Percentage
Received	9736	97.36%
Correct	9210	92.10%
Corrupted	526	5.26%

Table 4.12: Number of correct, incorrect and all packets that were captured with Omega = 40.0

clock_recovery_mm_ff0::avg input % full	[0.029760131612420082]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	27.74648094177246		Average noutput items
clock_recovery_mm_ff0::avg nproduced	27.74648094177246		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.0019639183301478624]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	50094.125	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	8334.4296875		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0005493499338626862]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	1.0		noutput items
clock_recovery_mm_ff0::nproduced	1.0		items produced
clock_recovery_mm_ff0::output % full	[6.103515625e-05]		how full output buffers are
clock_recovery_mm_ff0::total work time	3258757120.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[7.611267704987768e-08]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	668.2966918945312		Var. noutput items
clock_recovery_mm_ff0::var nproduced	668.2966918945312		Var. items produced
clock_recovery_mm_ff0::var output % full	[5.0227964010218 78e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	73819296.0		Var. clock cycles in call to work
clock_recovery_mm_ff0::work time	3402.0		clock cycles in call to work

Figure 4.50: Clock Recovery MM performance characteristics on channel 52 with Omega = 40.0

clock_recovery_mm_ff1::avg input % full	[0.030757619068026543]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	29.109912872314453		Average noutput items
clock_recovery_mm_ff1::avg nproduced	29.109912872314453		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.002220726804807782]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	50004.99609375	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	6181.736328125		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.0020753219723701477]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	4.0		noutput items
clock_recovery_mm_ff1::nproduced	4.0		items produced
clock_recovery_mm_ff1::output % full	[0.00640869140625]		how full output buffers are
clock_recovery_mm_ff1::total work time	2299651584.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[8.268288098634002e-08]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	663.0947265625		Var. noutput items
clock_recovery_mm_ff1::var nproduced	663.0947265625		Var. items produced
clock_recovery_mm_ff1::var output % full	[5.969775784819831e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	37124880.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	3791.0		clock cycles in call to work

Figure 4.51: Clock Recovery MM performance characteristics on channel 68 with Omega = 40.0

	Number of packets	Percentage
Received	9977	99.77%
Correct	9966	99.66%
Corrupted	11	0.11%

Table 4.13: Number of correct, incorrect and all packets that were captured with Omega = 40.2

clock_recovery_mm_ff0::avg input % full	[0.031194496899843216]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	29.447179794311523		Average noutput items
clock_recovery_mm_ff0::avg nproduced	29.447179794311523		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.0018412410281598568]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	49951.91015625	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	5443.01123046875		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0010376609861850739]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	1.0		noutput items
clock_recovery_mm_ff0::nproduced	1.0		items produced
clock_recovery_mm_ff0::output % full	[0.00592041015625]		how full output buffers are
clock_recovery_mm_ff0::total work time	1180880000.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[1.4378524326730258e-07]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	654.7947998046875		Var. noutput items
clock_recovery_mm_ff0::var nproduced	654.7947998046875		Var. items produced
clock_recovery_mm_ff0::var output % full	[8.48685921539527e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	34722960.0		Var. clock cycles in call to work
clock_recovery_mm_ff0::work time	3901.0		clock cycles in call to work

Figure 4.52: Clock Recovery MM performance characteristics on channel 52 with Omega = 40.2

clock_recovery_mm_ff1::avg input % full	[0.03267451003193855]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	30.651771545410156		Average noutput items
clock_recovery_mm_ff1::avg nproduced	30.651771545410156		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.0020023765973746777]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	49758.7578125	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	4792.41796875		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.002929866313934326]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	1.0		noutput items
clock_recovery_mm_ff1::nproduced	1.0		items produced
clock_recovery_mm_ff1::output % full	[0.003662109375]		how full output buffers are
clock_recovery_mm_ff1::total work time	995068480.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[1.5736893033135857e-07]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	642.2680053710938		Var. noutput items
clock_recovery_mm_ff1::var nproduced	642.2680053710938		Var. items produced
clock_recovery_mm_ff1::var output % full	[9.643965626082718e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	26985884.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	3823.0		clock cycles in call to work

Figure 4.53: Clock Recovery MM performance characteristics on channel 68 with Omega = 40.2

	Number of packets	Percentage
Received	9539	95.39%
Correct	7716	77.16%
Corrupted	1823	18.23%

Table 4.14: Number of correct, incorrect and all packets that were captured with Omega = 40.3

clock_recovery_mm_ff0::avg input % full	[0.03262867033481598]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	31.050230026245117		Average noutput items
clock_recovery_mm_ff0::avg nproduced	31.050230026245117		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.0017262781038880348]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	49839.79296875	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	6693.60205078125		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0023194774985313416]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	32.0		noutput items
clock_recovery_mm_ff0::nproduced	32.0		items produced
clock_recovery_mm_ff0::output % full	[0.00396728515625]		how full output buffers are
clock_recovery_mm_ff0::total work time	1313704832.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[1.6625820364879473e-07]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	642.960693359375		Var. noutput items
clock_recovery_mm_ff0::var nproduced	642.960693359375		Var. items produced
clock_recovery_mm_ff0::var output % full	[8.796186889981072e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	47934548.0		Var. clock cycles in call to work
clock_recovery_mm_ff0::work time	12305.0		clock cycles in call to work

Figure 4.54: Clock Recovery MM performance characteristics on channel 52 with Omega = 40.3

clock_recovery_mm_ff1::avg input % full	[0.030594229698181152]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	26.785480499267578		Average noutput items
clock_recovery_mm_ff1::avg nproduced	26.785480499267578		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.0019168677972629666]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	49636.0	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	4758.38037109375		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.0013428553938865662]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	32.0		noutput items
clock_recovery_mm_ff1::nproduced	32.0		items produced
clock_recovery_mm_ff1::output % full	[0.00396728515625]		how full output buffers are
clock_recovery_mm_ff1::total work time	1078042624.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[1.350452407677949e-07]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	647.2804565429688		Var. noutput items
clock_recovery_mm_ff1::var nproduced	647.2804565429688		Var. items produced
clock_recovery_mm_ff1::var output % full	[8.461199740850134e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	30095100.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	11114.0		clock cycles in call to work

Figure 4.55: Clock Recovery MM performance characteristics on channel 68 with Omega = 40.3

	Number of packets	Percentage
Received	8903	89.03%
Correct	4730	47.30%
Corrupted	4173	41.73%

Table 4.15: Number of correct, incorrect and all packets that were captured with Omega = 40.4

clock_recovery_mm_ff0::avg input % full	[0.031349942088127136]		Average of how full input buffers are
clock_recovery_mm_ff0::avg noutput_items	28.016407012939453		Average noutput items
clock_recovery_mm_ff0::avg nproduced	28.016407012939453		Average items produced
clock_recovery_mm_ff0::avg output % full	[0.0018045782344415784]		Average of how full output buffers are
clock_recovery_mm_ff0::avg throughput	49684.44140625	items/s	Average items throughput in call to work
clock_recovery_mm_ff0::avg work time	6492.98974609375		Average clock cycles in call to work
clock_recovery_mm_ff0::input % full	[0.0028077885508537292]		how full input buffers are
clock_recovery_mm_ff0::noutput_items	1.0		noutput items
clock_recovery_mm_ff0::nproduced	1.0		items produced
clock_recovery_mm_ff0::output % full	[0.000244140625]		how full output buffers are
clock_recovery_mm_ff0::total work time	1587334400.0		Total clock cycles in calls to work
clock_recovery_mm_ff0::var input % full	[1.2823845452203386e-07]		Var. of how full input buffers are
clock_recovery_mm_ff0::var noutput_items	653.502685546875		Var. noutput items
clock_recovery_mm_ff0::var nproduced	653.502685546875		Var. items produced
clock_recovery_mm_ff0::var output % full	[7.381714350884749e-09]		Var. of how full output buffers are
clock_recovery_mm_ff0::var work time	48087520.0		Var. clock cycles in call to work
clock_recovery_mm_ff0::work time	3213.0		clock cycles in call to work

Figure 4.56: Clock Recovery MM performance characteristics on channel 52 with Omega = 40.4

clock_recovery_mm_ff1::avg input % full	[0.02897857502102852]		Average of how full input buffers are
clock_recovery_mm_ff1::avg noutput_items	23.78371238708496		Average noutput items
clock_recovery_mm_ff1::avg nproduced	23.78371238708496		Average items produced
clock_recovery_mm_ff1::avg output % full	[0.002009134739637375]		Average of how full output buffers are
clock_recovery_mm_ff1::avg throughput	49518.765625	items/s	Average items throughput in call to work
clock_recovery_mm_ff1::avg work time	4434.60302734375		Average clock cycles in call to work
clock_recovery_mm_ff1::input % full	[0.0006714276969432831]		how full input buffers are
clock_recovery_mm_ff1::noutput_items	2.0		noutput items
clock_recovery_mm_ff1::nproduced	2.0		items produced
clock_recovery_mm_ff1::output % full	[0.0006103515625]		how full output buffers are
clock_recovery_mm_ff1::total work time	1272814976.0		Total clock cycles in calls to work
clock_recovery_mm_ff1::var input % full	[1.009674761576207e-07]		Var. of how full input buffers are
clock_recovery_mm_ff1::var noutput_items	635.1181640625		Var. noutput items
clock_recovery_mm_ff1::var nproduced	635.1181640625		Var. items produced
clock_recovery_mm_ff1::var output % full	[7.000250157318533e-09]		Var. of how full output buffers are
clock_recovery_mm_ff1::var work time	27226272.0		Var. clock cycles in call to work
clock_recovery_mm_ff1::work time	2354.0		clock cycles in call to work

Figure 4.57: Clock Recovery MM performance characteristics on channel 68 with Omega = 40.4

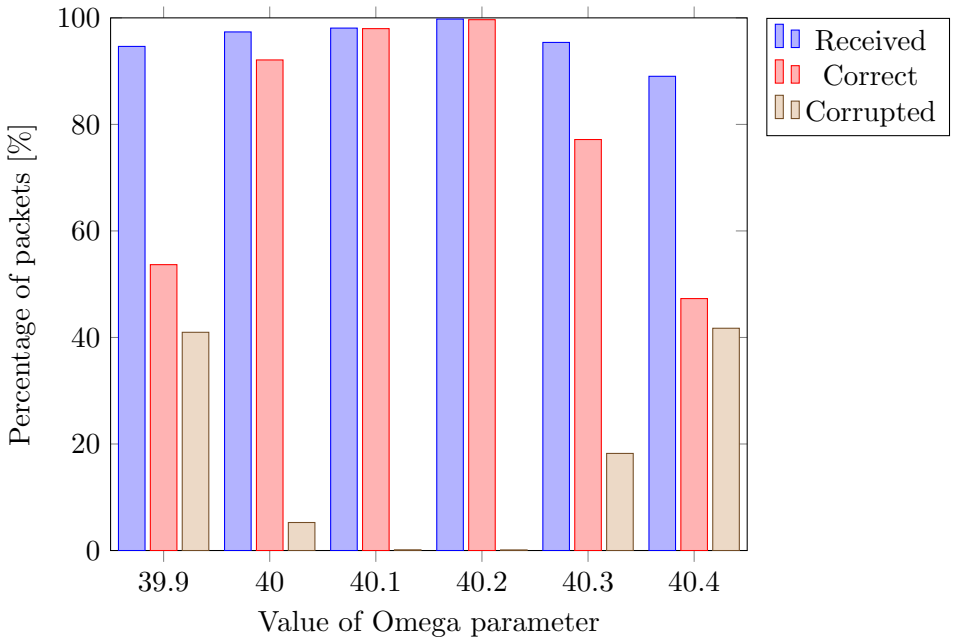

Figure 4.58: Percentage of received, correct and corrupted packets in different configurations of Omega parameter

4.5 Scalability testing

In this test flowgraph was extended with additional channels. Number of channels was increased up to seventeen.

	Id	Timestamp	Channel	Length	CurrentPacketNumber	TestId	TotalPacketCount	PayloadDataLength
	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr	Filtr
1	111288	2022-09-18 21:37:46.853889	868.3MHz	20	1	111	3	8
2	111289	2022-09-18 21:37:46.901689	868.3MHz	20	2	111	3	8
3	111290	2022-09-18 21:37:46.903723	868.3MHz	20	3	111	3	8
4	111291	2022-09-18 21:37:53.973581	868.7MHz	20	1	222	3	8
5	111292	2022-09-18 21:37:53.992604	868.7MHz	20	2	222	3	8
6	111293	2022-09-18 21:37:54.026608	868.7MHz	20	3	222	3	8
7	111294	2022-09-18 21:38:01.410548	869.1MHz	20	1	333	3	8
8	111295	2022-09-18 21:38:01.437546	869.1MHz	20	2	333	3	8
9	111296	2022-09-18 21:38:01.439546	869.1MHz	20	3	333	3	8
10	111297	2022-09-18 21:38:09.755461	869.5MHz	20	1	444	3	8
11	111298	2022-09-18 21:38:09.766696	869.5MHz	20	2	444	3	8
12	111299	2022-09-18 21:38:09.768694	869.5MHz	20	3	444	3	8
13	111300	2022-09-18 21:38:14.344568	869.9MHz	20	1	555	3	8
14	111301	2022-09-18 21:38:14.356033	869.9MHz	20	2	555	3	8
15	111302	2022-09-18 21:38:14.358069	869.9MHz	20	3	555	3	8

Figure 4.59: Fifteen packets received on five different channels

However when the eighteenth channel was added, an overload occurred. Overload is signalized in GNURadio by the display of recurring letter "O" in log window.

```
Executing: /usr/bin/python3 -u /home/artur/Desktop/Workspace/6tschDataSniffer/
DataSniffer/DataSniffer.py

Warning: failed to XInitThreads()
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use
QT_QPA_PLATFORM=wayland to run on Wayland anyway.
gr-osmosdr 0.2.0.0 (0.2.0) gnuradio 3.10.1.1
built-in source types: file fcd rtl rtl_tcp uhd hackrf bladerf rfspace airspy airspyhf
soapy redpitaya freesrp
[INFO] [UHD] linux; GNU C++ version 11.2.0; Boost_107400; UHD_4.1.0.5-3
Detached kernel driver
Found Rafael Micro R820T tuner
Reattached kernel driver
libusb: warning [libusb_exit] device 2.1 still referenced
libusb: warning [libusb_exit] device 1.3 still referenced
libusb: warning [libusb_exit] device 1.5 still referenced
libusb: warning [libusb_exit] device 1.4 still referenced
libusb: warning [libusb_exit] device 1.2 still referenced
libusb: warning [libusb_exit] device 1.1 still referenced
libusb: warning [libusb_exit] device 4.1 still referenced
libusb: warning [libusb_exit] device 3.1 still referenced
Using device #0 Realtek RTL2838UHIDIR SN: 00000001
Detached kernel driver
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Exact sample rate is: 2000000,052982 Hz
[R82XX] PLL not locked!
controlport :info: Apache Thrift: -h artur-Nitro-AN515-54 -p 44139
Allocating 15 zero-copy buffers
OOOOOOOOOOOOOOOO
```

Figure 4.60: Overload signalized by GNURadio

Figure 4.62 presents the average CPU load over the last five minutes of work GNURadio flowgraph execution with different number of channels. This data has been gathered with htop program - Linux resource monitoring. Value 0 describes CPU load when GNURadio simulation was not running.

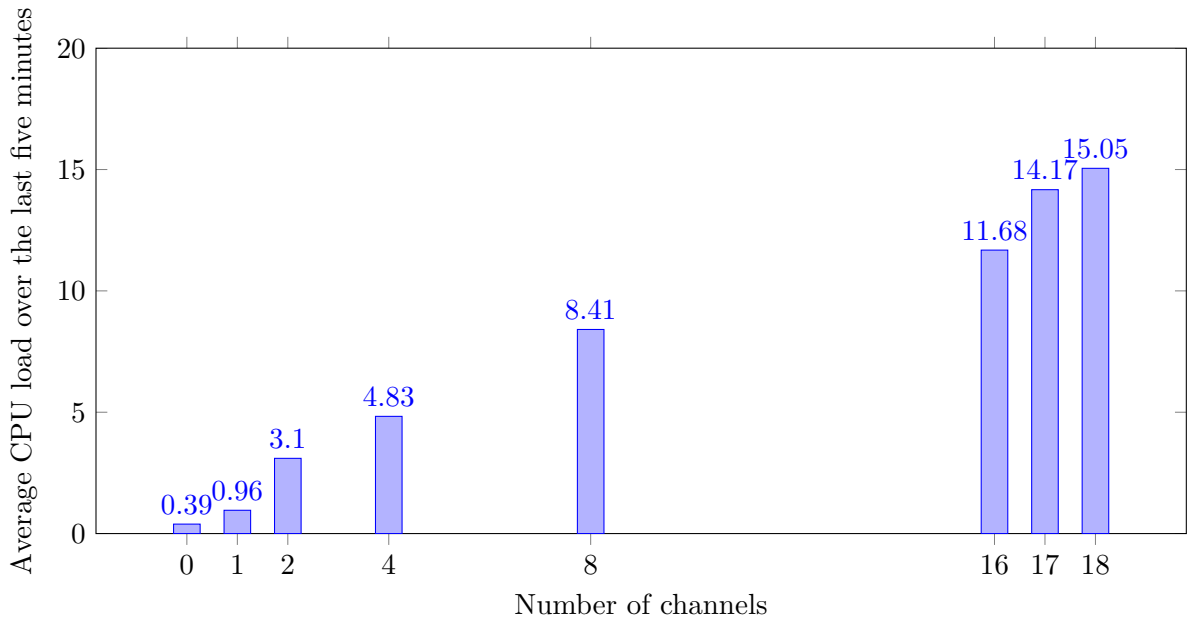

Figure 4.61: Average CPU load over the last five minutes of flowgraph execution with different number of channels present

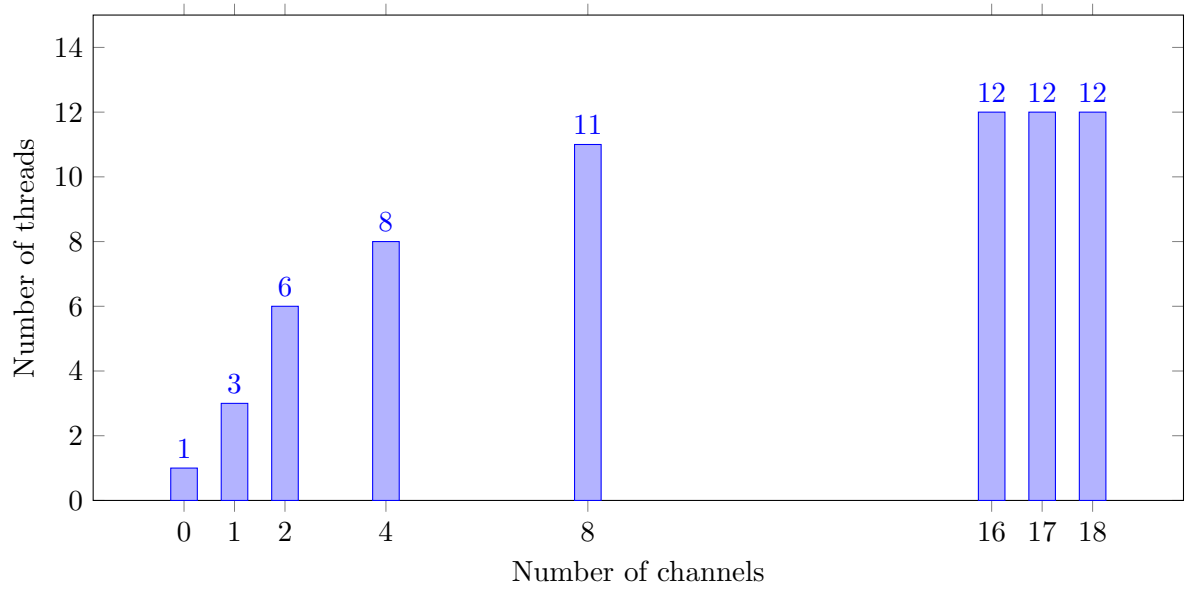

Figure 4.62: Number of threads utilized during the flowgraph execution with different number of channels present

Chapter 5

Summary

5.1 Conclusions

With increasing number of IoT connections this thesis makes a contribution in the field of wireless sensor networks. The central objective of the thesis was to create a data sniffer capable of capturing packets transmitted with 6TiSCH protocol in multiple radio channels. Subsequent objective was the performance and scalability analysis of the created system. Those objectives were accomplished fully. The created system is based on a single SDR dongle attached to a Personal Computer programmed with a python script created in the GNURadio environment.

This python script implements operations such as channelization, signal demodulation, signal averaging, clock recovery and scrapping of demodulated packets. All of these operations are CPU bound. Out of all of them, the channelization, implemented with *Xlating FIR Filter* blocks, proved to have the highest number of average clock cycles spent in the work method, measured in hundreds of thousands. Additionally, the average percentage of how full input buffers were was also the highest. GNURadio documentation states that the decrease of transition bandwidth of this filter will cause an increase in CPU usage[45]. Decrease in transition bandwidth from 20kHz to 5kHz joined with an increase of cut off frequency from 80kHz to 95kHz caused an increase in average clock cycles spent in work method from 514527 cycles to 2000833 clock cycles.

The system can capture packets in seventeen channels simultaneously. Available on the market packet sniffing solutions such as Open Sniffer using nRF52840 platform can capture packets in just one specified channel.

From 10000 packets transmitted at the distance of 20cm with SNR value equal to 49dB +/- 3dB created in this thesis data sniffer was able to capture 9808 of them. After scrapping them, 9795 possesed correct payloads, while 13 of them possesed incorrect values. By changing the Omega parameter from 40.1 to 40.2 additional 1.69% increase in Packet Delivery Ratio can be achieved.

Since the results of PDR tests reached highest percentage when the Omega was set to 40.2, that might suggest that either GNURadio sampling rate is in fact higher than provided in configuration 2MHz or the the data rate of transmitted packets is smaller than 50kB/s.

In the scalability tests, during which the CPU was working under high load, the GNURadio utilized all available threads. Any further optimization of the flowgraph with the usage of multithreading is thus limited. GNURadio utilizes all available computational power.

5.2 Further development possibilities

Created in this research system can be developed further. Among the possibilities can be mentioned:

- *coverage of whole 863-870MHz bandwidth* - single RTL-SDR 2832U dongle is not enough to cover the whole of the 863-870MHz bandwidth. One of the possible solutions could be usage of

Hack RF One dongle instead of RTL2832U. Since overload occurred when eighteenth channel was added, additional increase in computational power would also be needed. Another solution could be creation of a system similar to the one presented in the subsection 2.8.2. System would comprise of multiple SBCs, for example Raspberry Pi 4 placed together within a single Raspberry Rack. Instead of storing the data in the database placed on the board, the data could be saved into the universal database common for each board. Tests would have to be conducted in order to determine how many boards are necessary but based on the RTL-SDR 2832U bandwidth equal to 2.5MHz and 863-870MHz bandwidth span equal to 7MHz it means that at least three boards would be necessary.

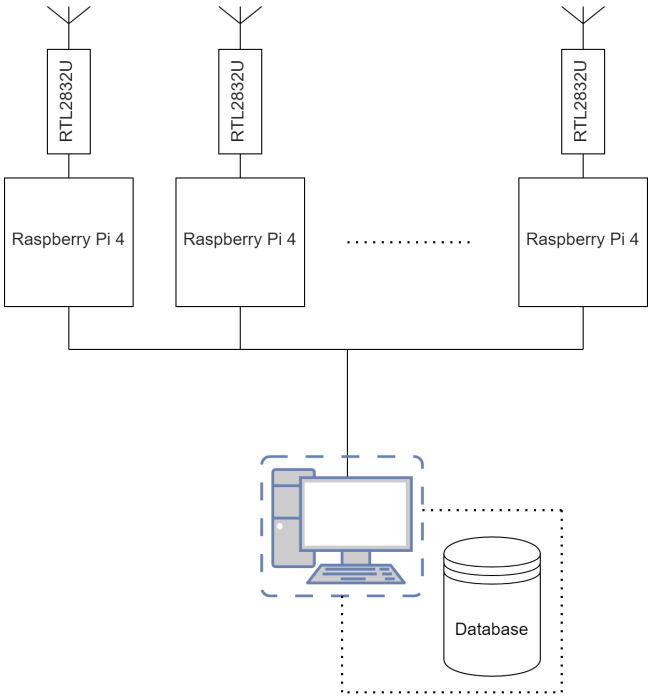

Figure 5.1: Possible further expansion of the presented system

- *implementation of dewhitening* - during the prototyping process an attempt was made to implement dewhitening operation. However this task proved to be highly time consuming and was omitted in the implementation.
- *implementation of 16 bit Cyclic Redundancy Check (CRC16)* - at the moment the recognition of the packet is conducted by tagging every encountered SFD. Calculation of CRC16 is another way to ensure that the packet has been received and its contents are not corrupted. After an SFD has been encountered, CRC16 can be calculated and compared with the last 16 bits of the packet.

List of Figures

List of Tables

Bibliography

[1] IoT Business News. State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally. https://iotbusinessnews.com/2022/05/19/70343-state-of-iot-2022-number-of-connected-iot-devices-growing-18-to-14-4-billion-global [Online; read at 22nd of September 2022].

[2] PRINCY A. J. All You Need to Know About Software Defined Radio. https://www.researchdive.com/blog/all-you-need-to-know-about-software-defined-radio, 2021. [Online; read at 9th of July 2022].

[3] The United States Navy. Extremely Low Frequency Transmitter SiteClam Lake, Wisconsin. https://web.archive.org/web/20061007110748/http://enterprise.spawar.navy.mil/UploadedFiles/fs_clam_lake_elf2003.pdf, 2003. [Online; read at 9th of July 2022].

[4] Federal Aviation Administration. Spectrum Engineering Policy - Radio Frequency Bands Supporting Aviation. https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/safety_ops_support/spec_management/engineering_office/rfb.cfm, 2020. [Online; read at 9th of July 2022].

[5] European Defence Agency. EDA supports work on interconnected secured European military software defined radio landscape. https://eda.europa.eu/news-and-events/news/2019/10/24/eda-supports-work-on-interconnected-secured-european-military-software-defined-radio-lands 2019. [Online; read at 9th of July 2022].

[6] European Defence Agency. EUROPEAN DEFENCE AGENCY: Perspectives on SDR Standards. https://www.wirelessinnovation.org/assets/Proceedings/2018Europe/11-%20EDA%20perspective%20on%20SDR_standards_2018.pdf, 2018. [Online; read at 9th of July 2022].

[7] https://www.rtl sdr.com/. SDR-J DECODING DAB RADIO IN SOFTWARE USING RTL-SDR . https://www.rtl-sdr.com/sdr-j-decoding-dab-radio-in-software-using-rtl-sdr/, 2013. [Online; read at 9th of July 2022].

[8] https://www.rtl sdr.com/. LISTENING TO AN ASTRONAUT TRANSMITTING FROM THE INTERNATIONAL SPACE STATION. https://www.rtl-sdr.com/listening-to-an-astronaut-transmitting-from-the-international-space-station/, 2016. [Online; read at 9th of July 2022].

[9] https://www.rtl sdr.com/. RTL-SDR TUTORIAL: RECEIVING WEATHER BALLOON (RADIOSONDE) DATA WITH RTL-SDR. https://www.rtl-sdr.com/receiving-weather-balloon-data-with-rtl-sdr/, 2013. [Online; read at 9th of July 2022].

[10] https://www.rtl sdr.com/. RTL-SDR TUTORIAL: CHEAP ADS-B AIRCRAFT RADAR. https://www.rtl-sdr.com/adsb-aircraft-radar-with-rtl-sdr/, 2013. [Online; read at 9th of July 2022].

[11] https://www.rtl sdr.com/. TRIANGULATION OF A VHF SIGNAL WITH RTLSDR-SCANNER . https://www.rtl-sdr.com/triangulation-vhf-signal-rtlsdr-scanner/, 2014. [Online; read at 9th of July 2022].

[12] wyccad2. Post on reddit forum uniting radio enthusiasts. https://www.reddit.com/r/RTLSDR/comments/hl3er0/i_want_to_get_into_sdr_got_any_good_use_cases/fwx3frl/?utm_source=reddit&utm_medium=web2x&context=3, 2020. [Online; read at 9th of July 2022].

[13] Rahul Awati. cognitive radio (CR). https://www.techtarget.com/searchnetworking/definition/cognitive-radio, 2021. [Online; read at 9th of July 2022].

[14] Jyoti Sekhar Banerjee. Fundamentals of Software Defined Radio and Cooperative Spectrum Sensing: A Step Ahead of Cognitive Radio Networks. https://www.igi-global.com/chapter/fundamentals-of-software-defined-radio-and-cooperative-spectrum-sensing/123579, year = 2014, note = "[Online; read at 9th of July 2022]".

[15] Geneviève Baudoin Martine Villegas Martha Suarez Fabien Robert Vaclav Valenta, Roman Marsalek. Survey on Spectrum Utilization in Europe: Measurements, Analyses and Observations. https://hal.archives-ouvertes.fr/file/index/docid/492021/filename/paper9220_valenta.pdf, year = 2010.

[16] IETF. An architecture for ipv6 over the tsch mode of ieee 802.15.4e draft-ietf-6tisch-architecture-07. https://datatracker.ietf.org/doc/rfc9030/.

[17] whizpace.com. All You Need to Know About Software Defined Radio. https://www.whizpace.com/technology. [Online; read at 16th of July 2022].

[18] mentor.ieee.org. Project: IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs). https://mentor.ieee.org/802.15/dcn/10/15-10-0528-00-leci-low-energy-critical-infrastructure-monitoring.pptx, 2010. [Online; read at 16th of July 2022].

[19] Gengfa Fang, Eryk Dutkiewicz, Mohammad A Huq, Rein Vesilo, and Yihuai Yang. Medical body area networks: Opportunities, challenges and practices. In *2011 11th International Symposium on Communications Information Technologies (ISCIT)*, pages 562–567, 2011. doi:10.1109/ISCIT.2011.6089699.

[20] IETF. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. https://datatracker.ietf.org/doc/html/rfc6282, 2011. [Online; read at 31st of July 2022].

[21] IETF. Thread Usage of 6LoWPAN. https://www.silabs.com/documents/public/white-papers/Thread-Usage-of-6LoWPAN.pdf, 2015. [Online; read at 31st of July 2022].

[22] IETF. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header. https://datatracker.ietf.org/doc/html/rfc8138, 2017. [Online; read at 31st of July 2022].

[23] IETF. INTERNET CONTROL MESSAGE PROTOCOL. https://datatracker.ietf.org/doc/html/rfc792, 1981. [Online; read at 26th of July 2022].

[24] IETF. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. https://datatracker.ietf.org/doc/html/rfc4443, 2006. [Online; read at 26th of July 2022].

[25] Martin Gunnarsson, Joakim Brorsson, Francesca Palombini, Ludwig Seitz, and Marco Tiloca. Evaluating the performance of the oscore security protocol in constrained iot environments. *Internet of Things*, 13:100333, 2021. URL: https://www.sciencedirect.com/science/article/pii/S2542660520301645, doi:https://doi.org/10.1016/j.iot.2020.100333.

[26] IETF. Object Security for Constrained RESTful Environments (OSCORE). https://datatracker.ietf.org/doc/html/rfc8613, 2021. [Online; read at 2nd of August 2022].

[27] IETF. Neighbor Discovery for IP Version 6 (IPv6). https://www.rfc-editor.org/rfc/rfc2461.html, 1998. [Online; read at 5th of August 2022].

[28] IETF. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). https://datatracker.ietf.org/doc/html/rfc6775, 2012. [Online; read at 5th of August 2022].

[29] IETF. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. https://datatracker.ietf.org/doc/html/rfc6550, 2012. [Online; read at 4th of August 2022].

[30] IETF. Constrained Join Protocol (CoJP) for 6TiSCH. https://datatracker.ietf.org/doc/html/rfc9031, 2021. [Online; read at 2nd of August 2022].

[31] GNURadio Github repository. file_sink_impl.h. https://github.com/gnuradio/gnuradio/blob/main/gr-blocks/lib/file_sink_impl.h. [Online; read at 22nd of October 2022].

[32] st.com. STM32WL Series. https://www.st.com/en/microcontrollers-microprocessors/stm32wl-series.html. [Online; read at 14th of August 2022].

[33] Ugo Colesanti and Silvia Santini. The collection tree protocol for the castalia wireless sensor networks simulator. 07 2011.

[34] RTL-SDT.com. ABOUT RTL-SDR. https://www.rtl-sdr.com/about-rtl-sdr/. [Online; read at 14th of August 2022].

[35] sqlite.org. What Is SQLite? https://www.sqlite.org/index.html. [Online; read at 16th of September 2022].

[36] survey.stackoverflow.co. 2022 Developer Survey. https://survey.stackoverflow.co/2022/#most-popular-technologies-database-prof. [Online; read at 16th of September 2022].

[37] Nicolae Crişan and Maria Condrea. Gsm wireless sniffer using software defined radio. *Carpathian Journal of Electronic and Computer Engineering*, 10:17–20, 01 2017.

[38] Pradeep Reddy, Hari Sharma, and Dominic Paulraj. Multi channel wi-fi sniffer. pages 1 – 6, 11 2008. doi:10.1109/WiCom.2008.727.

[39] Ritankar Sahu. Theoretical and practical approach to gnu radio and limesdr platform. 01 2020.

[40] sewio. Open Sniffer 3rd generation. https://www.sewio.net/wp-content/uploads/2018/09/OpenSniffer_gen3_v0.1.pdf. [Online; read at 22nd of September 2022].

[41] sewio. Open Sniffer v3.0. https://www.sewio.net/product/open-sniffer-kit/#. [Online; read at 22nd of September 2022].

[42] Nordic Semiconductors. nRF Sniffer for 802.15.4 User Guide v0.7.2. https://infocenter.nordicsemi.com/pdf/nRF_Sniffer_802154_User_Guide_v0.7.2.pdf. [Online; read at 22nd of September 2022].

[43] botland.com.pl. nRF52840 - moduł komunikacyjny Bluetooth, ZigBee, radiowy - USB. https://botland.com.pl/moduly-radiowe/14893-nrf52840-modul-komunikacyjny-bluetooth-zigbee-radiowy-usb-5904422301163.html. [Online; read at 22nd of September 2022].

[44] Kurt H. Mueller and Markus Müller. Timing Recovery in Digital Synchronous Data Receivers., 1976.

[45] GNURadio Community. GNURadio Wiki. `https://wiki.gnuradio.org/index.php/Frequency_Xlating_FIR_Filter`. [Online; read at 8th of November 2022].