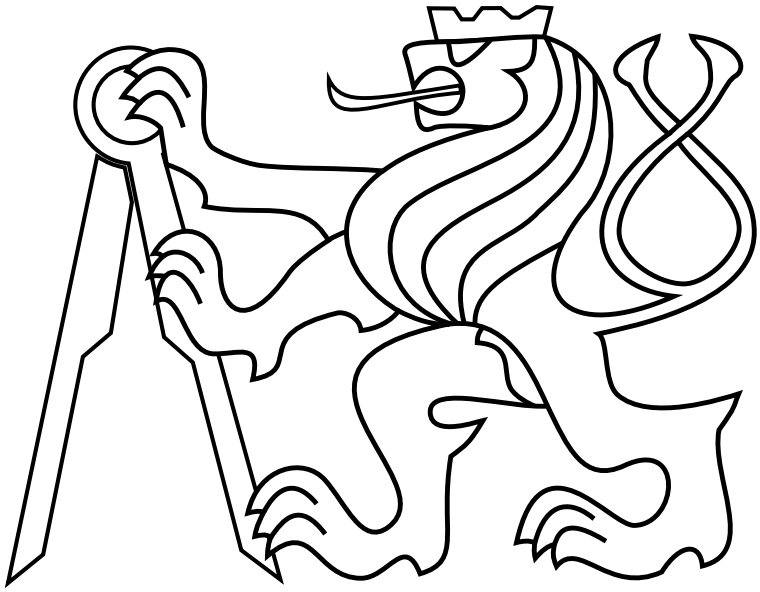
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Aleš Novotný

Software for Analysis of Automotive Ethernet Communication

Department of Cybernetics

Thesis supervisor: **doc. Ing. Jiří Novák, Ph.D.**

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne. . . . . Podpis . . . . . .

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Novotný Aleš** Personal ID number: **457209**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Software for Analysis of Automotive Ethernet Communication

Master's thesis title in Czech:

Programové vybavení pro analýzu komunikace v síti Automotive Ethernet

Guidelines:

Design and develop a software for analysis of Automotive Ethernet communication, satisfying the following requirements:
1. Frames capturing at Ethernet interface of common personal computer.
2. Filtering of the captured data by address at least..
3. Logging of captured data into the file format suitable for further processing.
4. Import of description database in arxml format.
5. Basic presentation of the captured data in form of table and graph.
6. Implementation of API for external program control.

Bibliography / sources:

[1] Bučkovskij, D.: Využití sítí Ethernet v osobních automobilech. Bakalářská práce ČVUT FEL, Praha 2016
[2] Correa, C.: Automotive Ethernet - The Definitive Guide. Interpids Control Systems 2014, ISBN: 978-0990538806

Name and workplace of master's thesis supervisor:

doc. Ing. Jiří Novák, Ph.D., K 13138 - katedra měření

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **30.01.2020** Deadline for master's thesis submission: _____

Assignment valid until:
by the end of summer semester 2020/2021

doc. Ing. Jiří Novák, Ph.D.	Head of department's signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Acknowledgements

First of all, I would like to thank doc. Ing. Ing. Jiří Novák, Ph.D. for his supervision in this project. I would like to thank my colleagues from Digiteq Automotive for their great advices and information. Also I would like to thank my family, especially my wife and my little son for their never-ending support, they have big understanding for researcher job.

Abstract

The aim of this work is to implement 100/1000BASE-T1 Automotive Ethernet packet interpreting system for a common personal computer connected to media converter from Automotive Ethernet device to conventional Ethernet. Especially, methods for Automotive packet capturing, filtering incoming communication, interpreting data with Automotive Open System Architecture Extensible Markup Language (AUTOSAR XML or ARXML), and logging of captured data to various formats are proposed. The designed system is modular, and it can be used from Graphical User Interface (GUI), Windows Command Prompt interface, Tcl console, or another program. The functionality of the system has been tested in several simulations using captured data from the Electronic Control Unit (ECU).

keywords:
[100/1000BASE-T1 Automotive Ethernet, packet interpreting, media converter, Automotive Open System Architecture Extensible Markup Language, logging, Graphical User Interface, Electronic Control Unit]

Abstrakt

Cílem této práce je implementovat systém, který interpretuje pakety z 100/1000BASE-T1 Automotive Ethernetu za pomocí běžného osobního počítače připojenému k převodníku médií z Automotive Ethernetu na běžný Ethernet. Především byly navrženy metody pro zachytávání paketů Automotive ethernetu, filtrování příchozí komunikace, interpretaci dat za pomocí Automotive Open System Architecture Extensible Markup Language a logování zachycených dat do různých formátů. Navržený systém je modulární a může být využit pro zpracování dat z grafického rozhraní, příkazového řádku Windows, Tcl konzole, nebo z jiného programu. Funkčnost systému byla testována v několika simulacích za použití simulátoru vestavěné řídicí jednotky.

klíčová slova:
[100/1000BASE-T1 Automotive Eternet, interpretace paketů, převodník médií, Automotive Open System Architecture Extensible Markup Language, logování, grafické rozhraní, vestavěná řídicí jednotka]

Contents

List of Figures

1 Introduction

In recent years, there has been significant progress in Automotive industry car technologies and features. A lot of cars produced nowadays has embedded systems as Adaptive Cruise Control (ACC) [1], [2] which enables a flexible response to traffic situation thanks to embedded radar. The driver can adjust the car speed, and it is maintained up until the ACC evaluates it as safe. If it is calculated by a Front Assist (part of ACC) that the car or object in front is too close, the car will automatically brake (see Figure 1). Another example is Parking Assist (PLA) [3] which helps the driver to park quickly, safely, and easily in every possible parking place. The PLA uses for its functionality the Optical Parking System (OPS), which consists of about 12 parking sensors. The number of units and sensors in the automobiles increases with many new features added to new cars, which makes the car more complex and leads to an increasing volume of data transmitted and thus placing more emphasis on communication Data Transfer Rates (DTR).

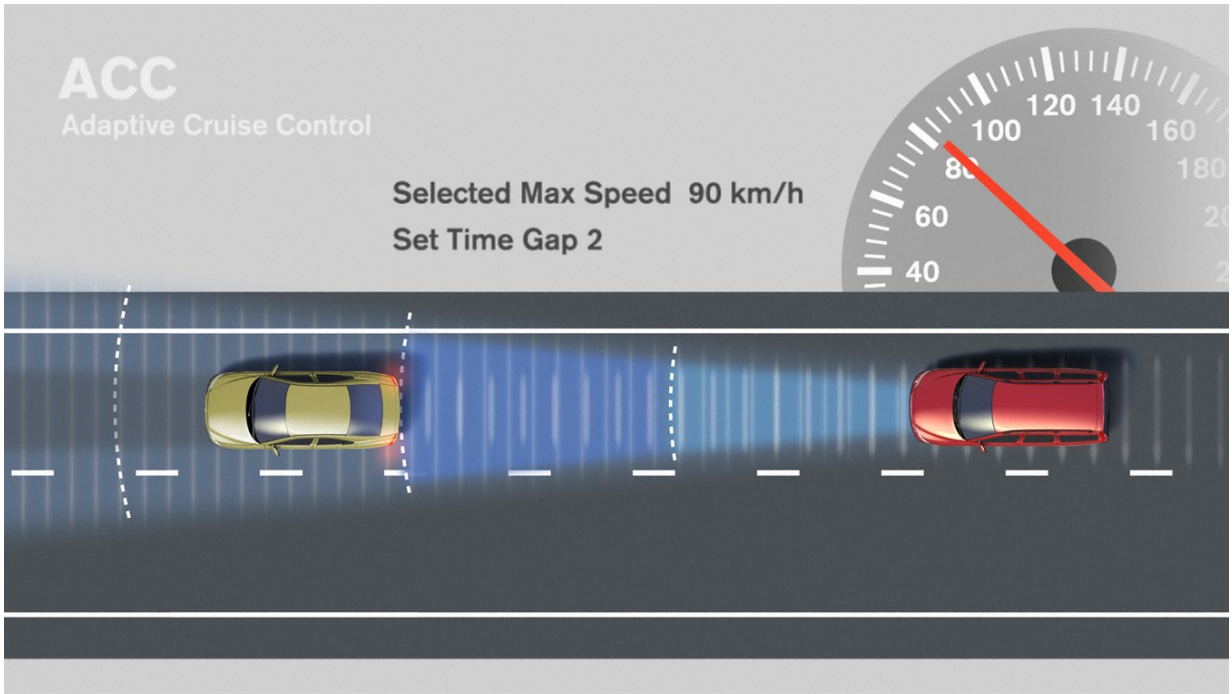


Figure 1: Adaptive Cruise Control principle [2].

At the beginnings of the transformation of a car composed of only mechanical parts to a smart car with a lot of sensors and Electronic Control Units (ECUs) was Controller Area Network (CAN). CAN is a multi-drop communication protocol that supports DTR up to 1 Mb/s. First cars with integrated CAN buses were produced in the 1990s, it meant revolution to the automotive industry because controllers, sensors, and actuators started

[2]Source: https://www.media.volvocars.com/global/en-gb/media/photos/8158

to be capable to communicate within one uniform network. With an increasing amount of transmitted data CAN throughput was ceased to be sufficient. This led to the development of new buses and protocols such as CAN-FD, MOST, Flexray [4]. The requirements for DTR in the car have been steadily increasing, as the number of devices and thus the buses. This has led to a demand for a low-cost bus that would have a large DTR. The solution came from Broadcom, which developed the Automotive Ethernet 100BASE-T1 capable of full-duplex communication over one pair of wires with originally DTR up to 100 Mb/s, and also 1 Gb/s [5], [6].

Bus	**DRT [Mb/s]**	**First applications [Year]**
CAN	1	1990s
CAN-FD	5	2015
FlexRay	10	2007
MOST	150	2001
Automotive Ethernet	1000	2013

Table 1: Comparision of CAN, CAN-FD, MOST, Flexray [4], [7], [8].

The aim of this thesis is to develop a modular system that is able to capture Automotive Ethernet packets, interpret their meanings, log results into various formats, and visualize results or track a signal. This solution brings an opportunity to analyze the Automotive Ethernet network with a media convertor and common PC or laptop. The designed system is able to provide for:

- Lossless capture of Automotive Ethernet frames from traffic using a common personal computer.
- Interpreting data with an appropriate ARXML file.
- Importing and working with large ARXML files in orders of 100 MB.
- Capturing data using these filters:
  - Interpreting all packets without filtering.
  - Protocol Data Unit (PDU) ID.
  - Source or destination IPv6 or both.
  - Signal name (specified in ARXML).
- Visualize results:
  - Table - allows a detailed view of packets composition and meanings of the data
  - Graph - allows to track one signal in time and view changes.

- Logging results into files formats:
  - .blf - one of the most widespread file systems in the automotive industry (used by CANoe).
  - .pcap - this format is used by many network sniffers like Wireshark.
  - .csv - this format can be opened on almost any computer by MS Excel, Open Office...
  - .txt - this format can be opened by almost every text editor.
- The system is modular and allows to implement various interfaces:
  - GUI interface implemented in C#.
  - Universal command-line C++ interface as a base for an unlimited number of interfaces like:
    - Tcl interface.
    - Windows Command Prompt interface.

It should be mentioned that this solution is adapted to Volkswagen AG standards and may not be compatible with other car manufacturers.

As already mentioned, the designed system can run on a common computer and use any media converter from Automotive Ethernet to the ethernet. In this case, a RAD-Moon (see Figure 2) device is chosen as a media converter. The RAD-Moon is a low cost, plug-and-play, small size, and high-performance device, which connects a 100BASE-T1 Automotive Ethernet physical layer to 4-wire 10/100 Ethernet (100Base-TX) and allows sending/receiving all data in both directions. In this thesis, this device is used as a network sniffer, when the RAD-Moon resends all received data to the PC for data packet analysis [9].

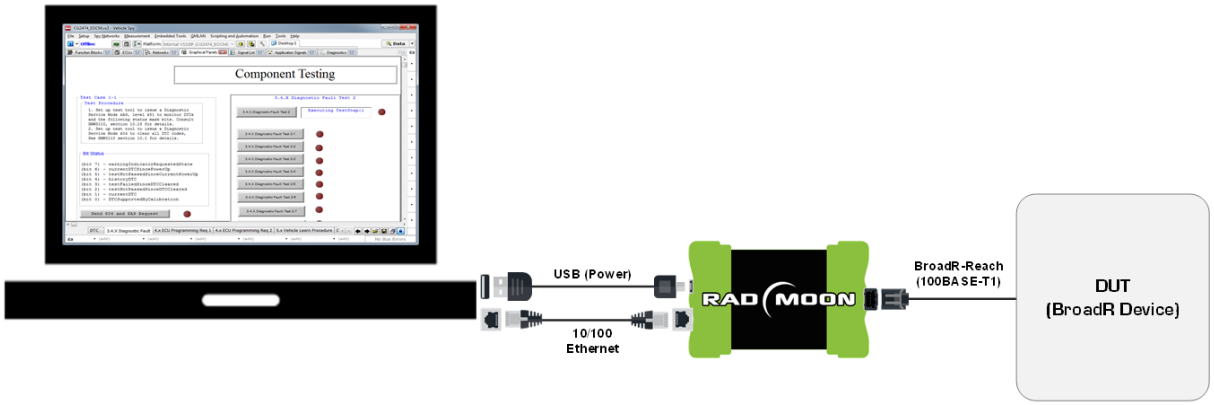


Figure 2: RAD-Moon [3].

As mentioned above, this thesis deals with the processing of received data from media converter connected to the PC with the use of the Automotive Open System Architecture Extensible Markup Language (ARXML) file. Firstly, Protocol Data Units (PDUs) from a data packet are detected, the PDUs are then parsed into signals, and finally, the meaning of signals is assigned. The designed solution also provides filtering data methods by IPv6 addresses, PDU ID, and signal name which improves readability of results and reduces the complexity of the program.

The thesis is structured as follows. Firstly in the Chapter 2, the AUTOSAR standard, the ARXML standard, used ISO/OSI model, and other essential information needed for understanding this thesis is explained. In the Chapter 3, the principle of system operation is shown. In the following Chapter 4, algorithms for searching data in ARXML are introduced. In the Chapter 5, the implemented principle of capturing is described. In the Chapter 6, the implementation of various interfaces with their features is described, and finally, in the Chapter 7, its functionality is verified.

[3]Source:https://www.intrepidcs.com/products/automotive-ethernet-tools/rad-moon/

2 Theoretical framework

2.1 Theoretical framework overview

In this chapter, the knowledge necessary for understanding the rest of the work is explained, the explanation is divided into two sub-chapters. In the first Chapter 2.2, the basics of signal communication in a car and ARXML standard is explained. In the second Chapter 2.3, the background of Automotive Ethernet is explained and the used ISO/OSI model is presented.

2.2 AUTOSAR standard

2.2.1 Introduction into AUTOSAR

AUTOSAR is an automotive standard, that was founded in 2003 as a product of partnership automotive manufacturers and suppliers. The AUTOSAR is providing a framework for development of uniform software independently on hardware components, thus accelerating the evolution of cars which leads to increasing demands on the complexity of software and hardware solutions that is still sustainable [12], [13].

The AUTOSAR standard separates software into hardware-independent (Application layer) and hardware-dependent Basic Software (BSW) layers. The layers communicate through Runtime Environment (RTE) layer which provides an input/output interface for applications (see Figure 3). Thus the development of applications and BSWs is independent. This allows the possibility to distribute different applications across ECUs due to car variants or integrate software modules from different suppliers [14], [15].

The Basic Software (BSW) is composed of three sublayers:

- Service Layer - provides services to upper layers (communication, memory, crypto services etc. (see Figure 3)).
- ECU abstraction Layer - software interface to electrical values of ECU.
- Microcontroller Abstraction Layer (MCAL) - standard microcontroller independent interface for BSW modules.

The BSW also incorporate operating system, complex device drivers etc. (see Figure 4) [12], [14].

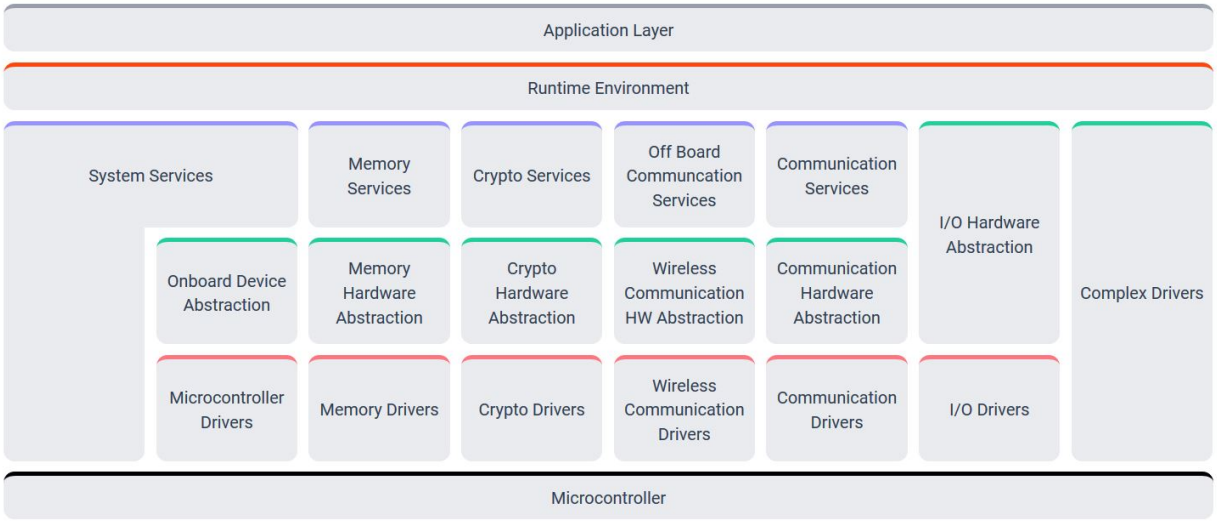


Figure 3: AUTOSAR Classic Platform software layers [4].

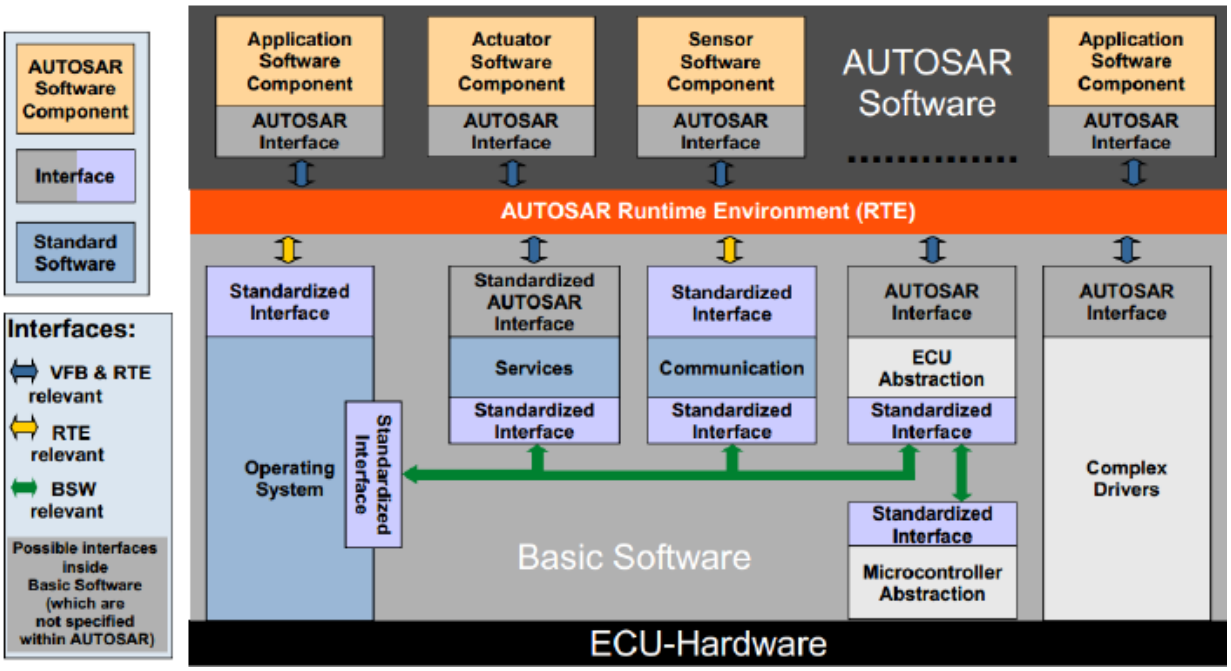


Figure 4: Connection of AUTOSAR Classic Platform software layers [16].

The communication within every single ECU or between ECUs is based on Virtual Functional Bus (VFB). The virtual bus strictly splits applications from the infrastructure. There are defined specifically dedicated ports that are used by an application to handle communication, these ports are then mapped onto local connections or network-technology

[4]Source: https://www.autosar.org/standards/classic-platform/

specific connections. Thus for application development detailed knowledge of lower-level technologies is not needed (see Figure 5) [14], [16].

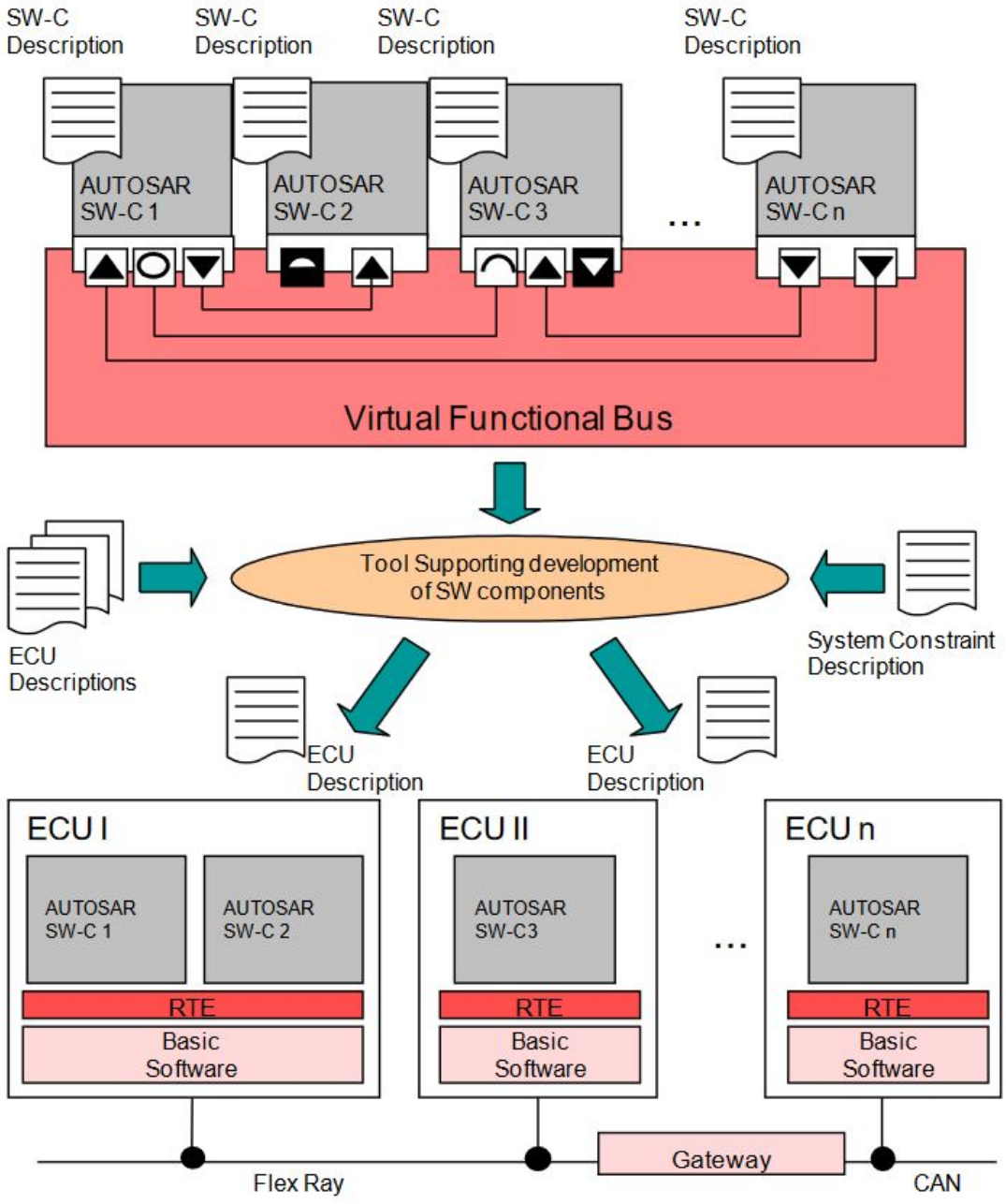


Figure 5: Principle of Virtual Functional Bus [16].

The VFB is used by applications to access microcontroller peripherals and ECUs electronics, data from sensors and to control actuators (see Figure 6). An ECU abstraction provides a software interface for accessing physical ECU safely from higher-level software. Below this is Microcontroller Abstraction Layer (MCAL), this hardware-dependent

layer provides hardware-independent commands for accessing microcontroller registers from higher-level software indirectly [16].

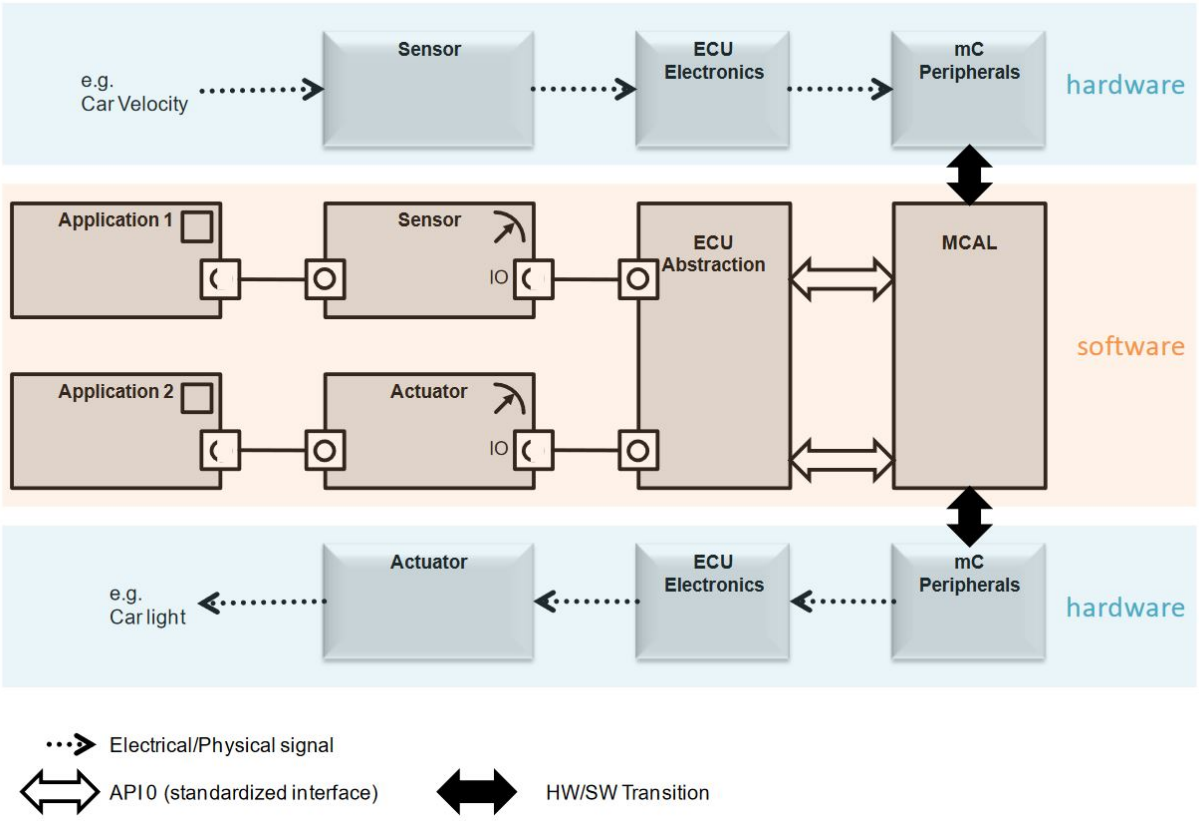


Figure 6: Conversion from physical to software signals and vice versa [16].

The above described between ECUs or within ECU communication and ECU descriptions (shown in Figure 5) are all included in AUTOSAR UML2.0 meta-model which graphically describes all information in AUTOSAR system. From the UML2.0 meta-model, the compliant XML schema can be compiled (see Figure 7) [17]. This AUTOSAR XML (ARXML) is used in this thesis for the acquisition of AUTOSAR systems and is described below.

2.2.2 AUTOSAR XML

AUTOSAR XML (ARXML) is derived from AUTOSAR UML2.0 meta-model (see Figure 7), that describes all AUTOSAR systems [17]. The ARXML file holds important information about application interfaces (messages periods, signals meanings) in standardized format [18].

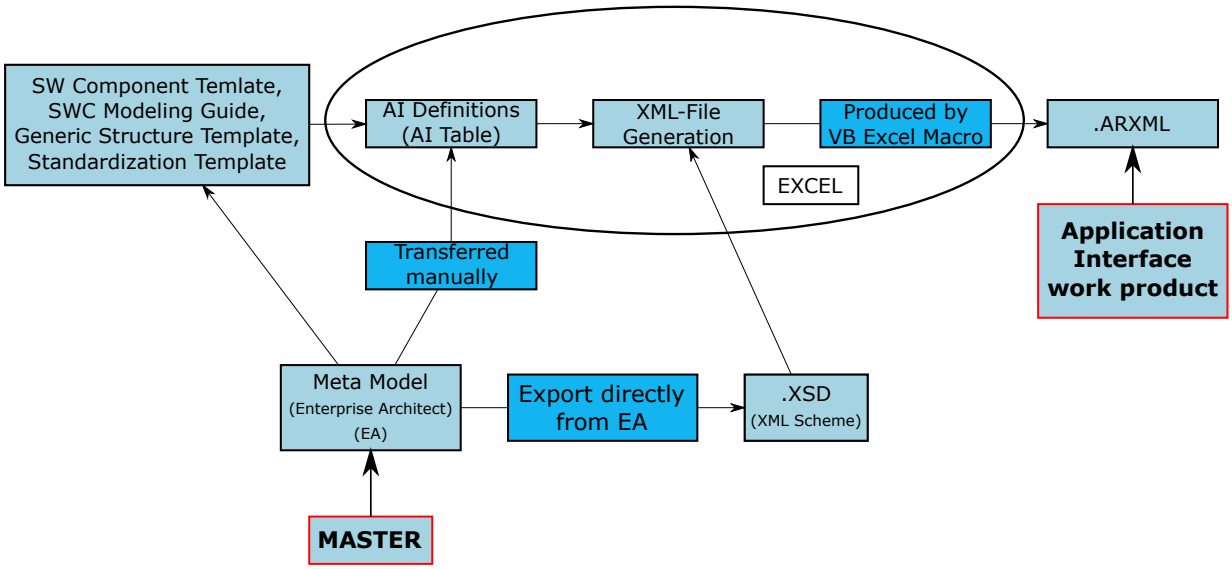


Figure 7: Process of ARXML creation from AUTOSAR meta-model [18].

The AUTOSAR systems specification contains a huge amount of information. Each subsystem is also described in a comprehensive way. For example CAN message specification includes information about message ID, message length, cycle time, signals sent within a message, physical dimensions of signal, byte order, etc. To avoid storing information into very complex structures, the AUTOSAR splits information thematically into groups and subgroups, which are then linked using relative path references [18].

The ARXML provides a great tool for searching data interpretations from packets due to the comprehensiveness of the description. Likewise, it is preferable that the data is stored in a common XML format, that can be opened without using specialized software and it is human readable. The only disadvantage is the computational complexity due to the low ratio of useful (data, values) and useless (element and node names, special characters) data.

2.3 Description of Automotive Ethernet ISO/OSI model

2.3.1 Introduction to Automotive Ethernet

OABR (Open Alliance BroadR-Reach) Ethernet is a technology that was developed by Broadcom and it was released in 2011 within OPEN Alliance. The OPEN Alliance SIG is a non-profit group (formation in 2011) of the main automotive industry and technology providers who collaborate together to support widespread acceptance of Ethernet as a standard for automotive network communication [19], [20].

The BroadR-Reach is a technology that allows point-to-point Ethernet communication only over one pair of unshielded twisted pair (see Figure 10b) [21] [6]. Cabling is the 3rd most expensive and also 3rd heaviest component in the car. This technology has led to a significant reduction in cabling prices and weight which has a direct impact on fuel economy [8].

The OABR was transformed by the OPEN Alliance SIG into IEEE standard IEEE 802.3bw also known as 100BASE-T1 in 2015 [22]. Year later in 2016 the standard IEEE 802.3bp was published for 1Gb/s Physical Layer (PHY) also known as 1000BASE-T1 [23].

2.3.2 ISO/OSI model using Automotive Ethernet

Layer	Purpose
7. Application	Application/Services
6. Presentation	Application/Services
5. Session	AUTOSAR PDU, SOME/IP, ViWi, DoIP
4. Transport	UDP, TCP
3. Network	IPv6 (ICMPv6)
2. Data Link	Ethernet MAC + VLAN
1. Physical	100/1000Base-T1

Figure 8: ISO/OSI model of Automotive Ethernet.

ISO/OSI model of Automotive Ethernet is shown in Figure 8. The first two layers are administrated by an application that sends data to Automotive Ethernet. The session layer describes the organization of data in a packet. Bellow this in Transport layer data packet is transformed to TCP packet if the communication is reliable and into UDP if not. In the network layer, the packet is extended with information about the source and

target IP addresses and other network layer fields. Below this is the data link layer which extends packet with Medium Access Control (MAC) addresses and other link layer fields. Finally, in the physical layer, a packet is sent over 100/1000Base-T1 ethernet. An opposite sequence is applied when receiving packets.

2.3.2.1 Physical layer

The physical layer of the above mentioned ISO/OSI model defines physical and electrical features. It includes Physical Coding Sublayer (PCS), Physical Medium Attachment (PMA) sublayer, cable, connectors, etc. (see Figure 9).

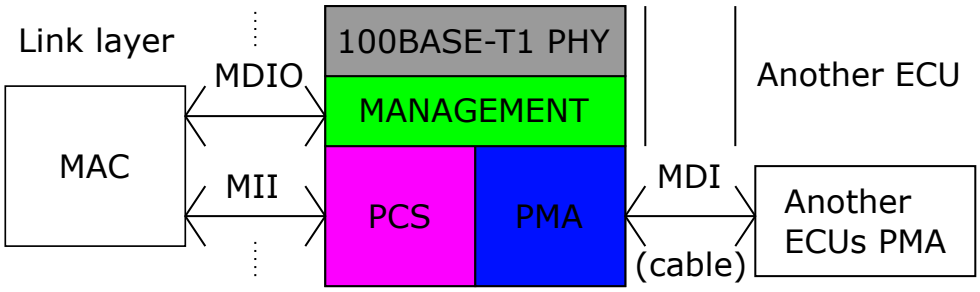
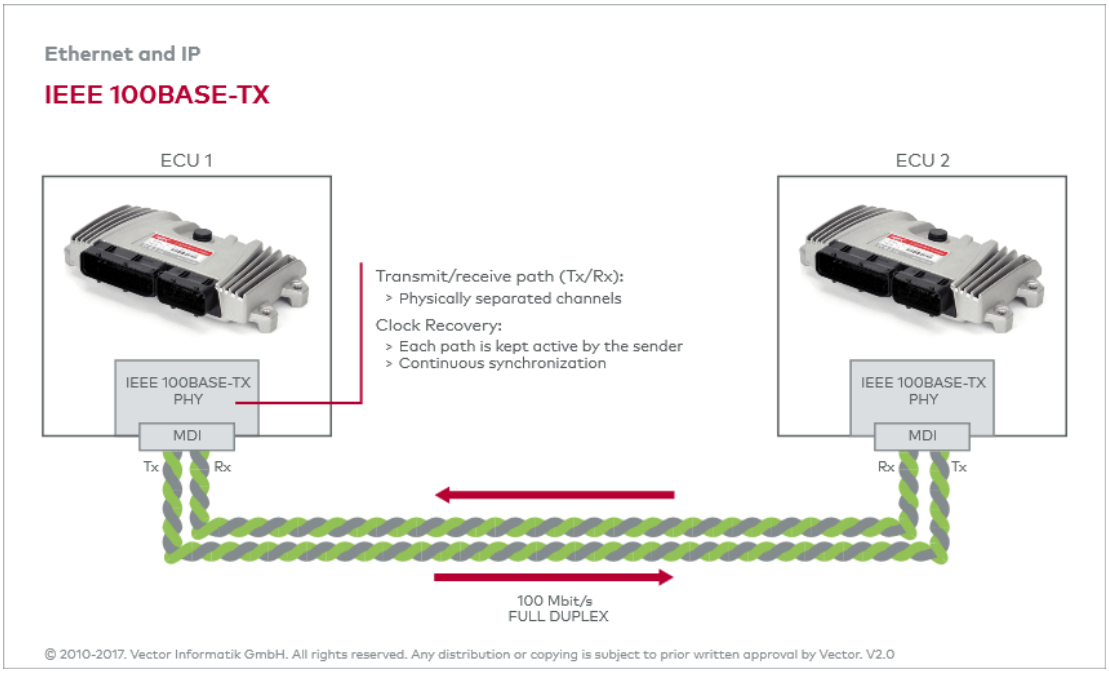
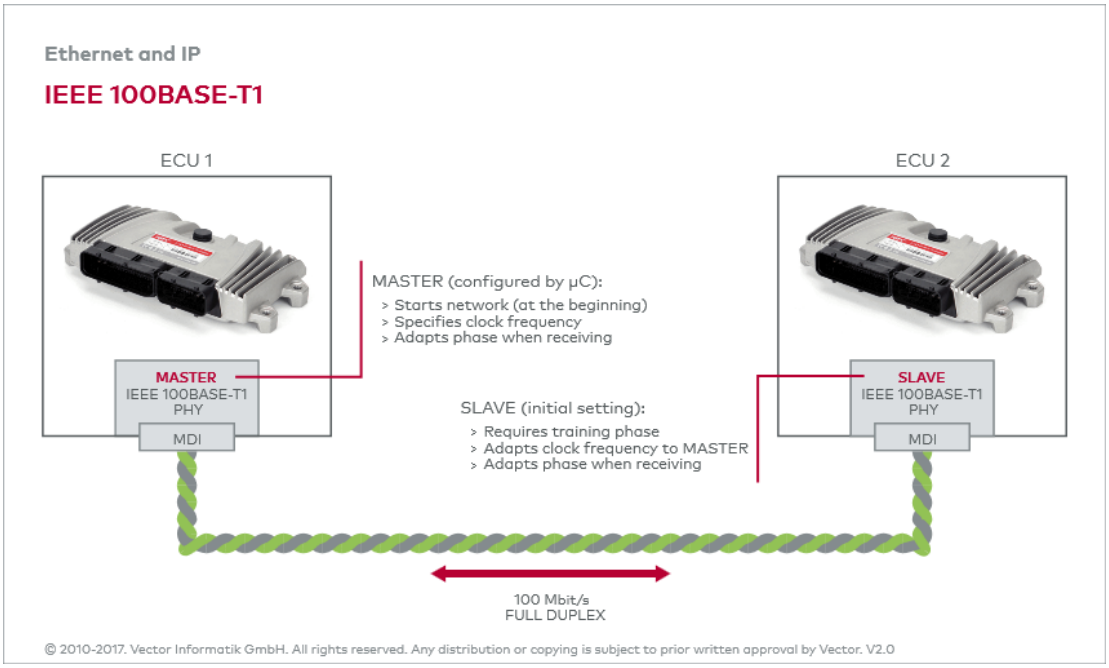


Figure 9: 100BASE T1 physical layers and its sublayers.

The 100BASE-T1 uses one symmetric twisted pair cable (see Figure 10b) in contrast with classic 100BASE-TX Ethernet which uses 2 pairs (see Figure 10a). The 100BASE-T1 is full-duplex and point-to-point (at the physical layer). On the one hand, 100BASE-TX allows the higher distance between endpoints, 100 m instead of 15 m, but on the other hand 100BASE-T1 PHY is cheaper because it requires only one unshielded twisted pair for full-duplex communication. Another advantage can be seen at higher speeds, whereas 1000BASE-T1 is still unshielded twisted pair cable, but classic 1 Gb/s Ethernet used in automotive must be shielded, which increases the price even more [4].



(a) IEEE 802.3u (100BASE-TX) communication [5].



(b) IEEE 802.3bw (100BASE-T1) communication [6].

Figure 10: Differences between 100BASE-TX and 100BASE-T1 in physical layer.

[5]Source: https://elearning.vector.com/mod/page/view.php?id=153
[6]Source: https://elearning.vector.com/mod/page/view.php?id=152

The full-duplex communication over a twisted pair cable is made possible by the fact that there is only one receiver and sender and both know what they are sending (see Figure 11). The calculation of the received signal is based on the assumption that both the transmitted signal voltage and the actual voltage (sum of the transmitted and received signal) on the bus are known. Under these conditions, the received signal can be determined easily by this formula 1.

$$RX[V] = Bus\_status[V] - TX[V] \tag{1}$$

Physical Coding Sublayer (PCS) is part of the physical layer (PHY). The PCS is responsible for the preparation of data between the Medium Access Control (MAC) sublayer of Microcontroller Unit (MCU) and Physical Medium Attachment (PMA) sublayer. For communication between these sublayers is used Media-Independent Interface (MII) (see Figure 9) [5], [6].

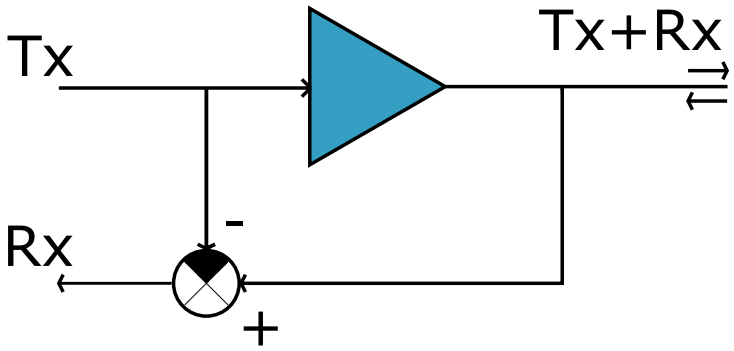


Figure 11: Simplified principle of full duplex communication over one twisted pair cable.

The data from MII is first converted by a 4B3B converter when the sequence of the quadruple bits are changed to the sequence of triplets [6].

Then the bits are mixed using Scrambling, which has the benefit of reducing the occurrence of spectral lines that arise when concatenating more the same data bit patterns. It allows the pseudo-random distribution of energy on the conductor. Removing the DC component reduces the Root Mean Square (RMS) of the signal [6].

Subsequently, each triad of bits is mapped to a pair of ternary symbols (TA_n and TB_n) using a conversion table. Finally, data is serialized and entered into the PMA sublayer(see Figure 12). This is called Pulse-amplitude modulation-3 (PAM-3) [19].

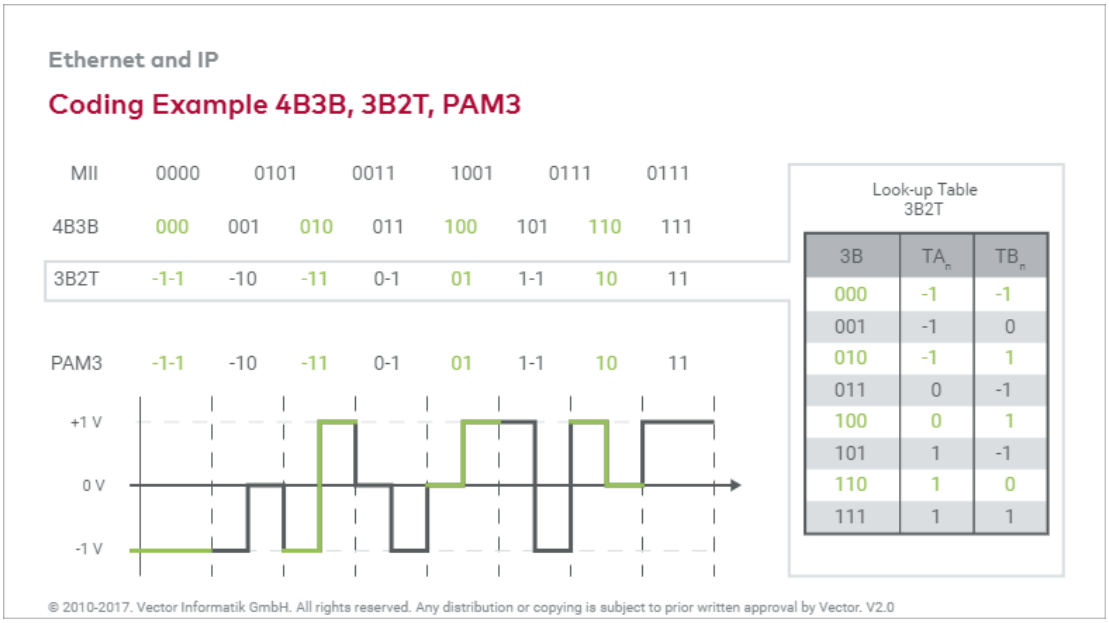


Figure 12: Example of 4B3B, 2B2T, PAM3 coding without Scrambler.[7].

The Physical Medium Attachment (PMA) sublayer cares for controlling, PAM3 modulation/demodulation of data to/from Medium-Dependent Interface (MDI) (see Figure 13).

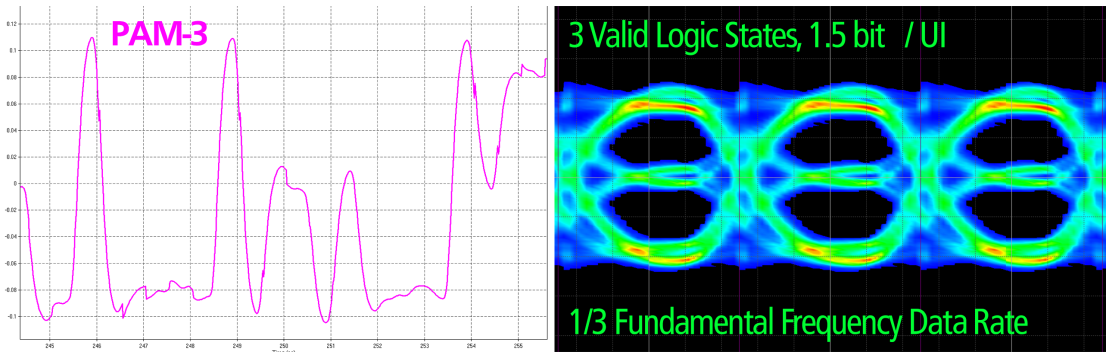


Figure 13: PAM3 modulation example.[8].

The PHY is managed via Media Data Input/Output (MDIO) from MAC (see Figure 9). The management registers are read and written via MDIO. Typically, this is accomplished by communicating using two signals (Data and Clock).

[7]Source: https://elearning.vector.com/mod/page/view.php?id=152
[8]Source: https://community.cadence.com/cadence_blogs_8/b/spi/posts/pam2

2.3.2.2 Data link layer

The data link layer cares about forwarding between adjacent nodes in the same network (WAN, LAN). It includes Media Access Control (MAC) [24].

The MAC is realized in a network adapter of the Microcontroller Unit (MCU) and identified by the MAC address saved in EEPROM. The purpose of the MAC sublayer is controlling data flow, sending/receiving Ethernet frames, CRC, checking the integrity of the received frame [25]. Ethernet frame structure is shown in Figure 14

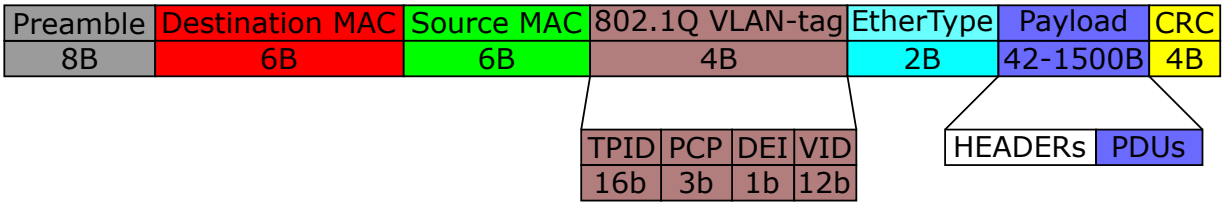


Figure 14: Data link layer (Ethernet) frame structure [26], [27].

A structure of a data link layer packet is shown in a Figure 14. At the start of the packet, there is a Preamble 8B data field. The first 56 bits are filled with alternating "1" and "0" bits (7 bytes filled with 0xAA) which allows devices to synchronize their clocks at bit-level. The last byte is called Start Frame Delimiter (SFD) and is used for byte synchronization and signalize the start of a packet with "1" bit (0xAB) [26].

The next source and destination MAC addresses are specified in MAC address fields. Then the 802.1Q tag (VLAN-tag) field follows. The IEEE 802.1Q is a networking standard that supports Virtual LANs (VLANs) on the Ethernet network. This field is divided into subfields Tag Protocol Identifier (TPID), Priority Code Point (PCP), Drop Eligible Indicator (DEI), and VLAN Identifier (VID). The TPID is at the same position as the EtherType field in untagged frames and in this case, the TPID is set to value 0x8100 to identify the 802.1Q-tagged frame. Then is the PCP field describes frame priority level. The DEI is used to indicate frames, which can be dropped when congestion occurs. The last subfield VID is specifying VLAN of the frame. The 0x000 value is reserved for frames which does not carry a VLAN ID, the 0xFFF is reserved for implementation use [26], [27].

Then the EtherType field follows, which is used to specify the size of bytes if the value is ≤ 1500. If the value is ≥ 1536, then this data field is used to indicate which protocol is encapsulated in the payload. In this case, the length of a frame is determined by the Interpacket Gap (IPG). The next field is the Payload field which contains upper ISO/OSI headers and PDUs, the headers are described in the following Chapters 2.3.2.3 and 2.3.2.4, the structure of PDU is described in Chapter 4. Finally, there is the Cyclic Redundancy Check (CRC) field that is used for detecting corrupted data [26].

2.3.2.3 Network layer

The Network layer is responsible for forwarding packets to destination network (LAN) and thus determining the source and target networks. According to the above mentioned ISO/OSI model, the IPv6 protocol is used [28].

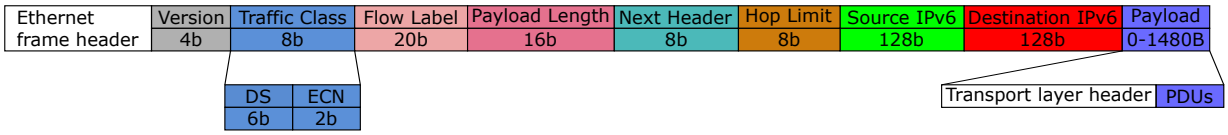


Figure 15: Network layer (IPv6) packet structure [29].

At the beginning the IPv6 packet is encapsulated into Ethernet frame packet (see Figure 15) described in Chapter 2.3.2.2. The first field is Version which has value 0x6, this describes that the packet is IPv6. Next field is the Traffic Class which is divided into subfields Differentiated Services (DS) and Explicit Congestion Notification (ECN). The DS is used for classification packets and the ECN describes if the source provides for congestion control or not [29].

The Flow Label field is used as an identifier of a packet flow (a group of packets as media stream) between source and destination. The next field is Payload Length which defines the size of the Payload field in bytes. The following field is the Next Header which specifies the transport header type. Then there is the Hop Limit field, this replaces Time To Live field in IPv4, and is decremented by every forwarding. When the Hop Limit has value 0, the packet is discarded. Next, there are Source Address and Destination Address fields that describe sending and receiving node IPv6 addresses [29]. Finally, there is the payload field which is composed of the transport layer header described in the following Chapter 2.3.2.4, and the structure of PDUs is described in Chapter 4.

2.3.2.4 Transport layer

The Transport layer is responsible for host-to-host communication and provides services for application as flow control, multiplexing, etc. [30] According to the above mentioned ISO/OSI model, it is expected both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) protocols.

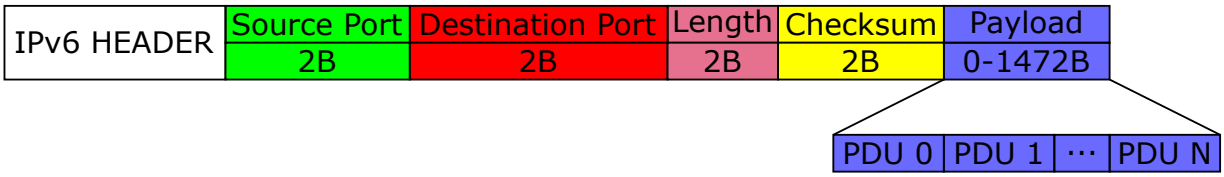


Figure 16: Transport layer (UDP) packet structure [31].

The UDP protocol is a very simple, unreliable communication protocol that is good for transferring large amounts of data that are not critical. The UDP packet header is formed up from 4 fields (see Figure 16). The first two fields specify the Source and Destination port. Next, the field Length value describes the total size of the whole UDP packet. Finally, there is a Checksum field which is mandatory in IPv6 and it is used for checking errors in header and data. The payload of the UDP packet may be composed of PDUs described in Chapter 4, [31].

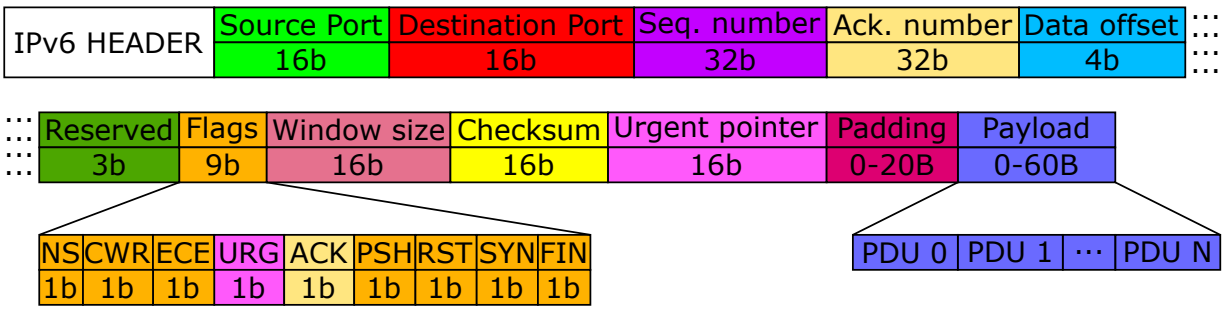


Figure 17: Transport layer (TCP) packet structure [32].

Unlike of UDP communication protocol, the TCP is reliable and connection-oriented (needs establishing communication), thus TCP packet is more complex. The TCP packet structure is shown in Figure 17. Similarly to the UDP protocol, the first 2 fields are ports. Next field is the Sequence number, this field holds information about the first data byte number. The following field is the Acknowledge number field which holds information about how many data bytes were received. Then follows the field Reserved, which is here for future purposes. The next field is Data offset which specifies the size of the header in 32-bit words. The minimum size of the header is 5 and the maximum is 15 words [32].

The next field is Flags. The most important flags for communication are the Acknowledgement (ACK) flag which indicates that the Acknowledgement number field has a valid value. Then Push (PSH) flag, which is set when sending data. The Reset (RST) flag is set to reset a connection. The Synchronize (SYN) is for establishing connections and Finish (FIN) signals the last packet from the sender. The URG flag indicates that the Urgent pointer field is valid. The CWR and ECE flags are used for Explicit Congestion Notification (ECN), the ECN is an optional feature and allows end-to-end notification of network congestion without dropping packets. Finally, the NS flags are used for signalizing concealment protection [32].

The next field is Window size, this is used for defining how much data bytes can sender receive without sending acknowledge. Then the Checksum field is used for error-checking of a header or a payload. The following field is an Urgent pointer field, which is valid when the URG flag is set, and it is used for expressing the offset from a start of the data field. This creates a subfield in a range from the start of the data and up till Urgent pointer points up, this is then defined as urgent data. The last field of TCP header is Padding and it is used to ensure that the header size is multiple of 32, bigger than 20 bytes, and less than 60 bytes. The payload of the TCP packet may be composed of PDUs described in Chapter 4 or other protocol [32].

The example of TCP communication is shown in Figure 18. In this simple example of the TCP communication server and client establish a connection and each side sends a data packet. After this client close connection.

For establishing the connection, it is necessary to send a packet with a set SYN flag first, the client increments the internal next sequence counter. After the server gives the client an acknowledgement of sent by sending a message back to client with set ACK flag and SYN flag. After this client acknowledges the server that message is received and thus the connection is established.

When the connection is established each side can send any packet and after this should await an acknowledgement. The Acknowledgement number sent in response must increase by all packet data bytes number. The receiver shows this way that all sent data is transmitted and received successfully. When sending the last data from the buffer, the PSH flag is set.

For closing connection, one side sends a packet with a set FIN tag. Second node answer that packet is received by sending a packet with set FIN tag back and also await acknowledgement.

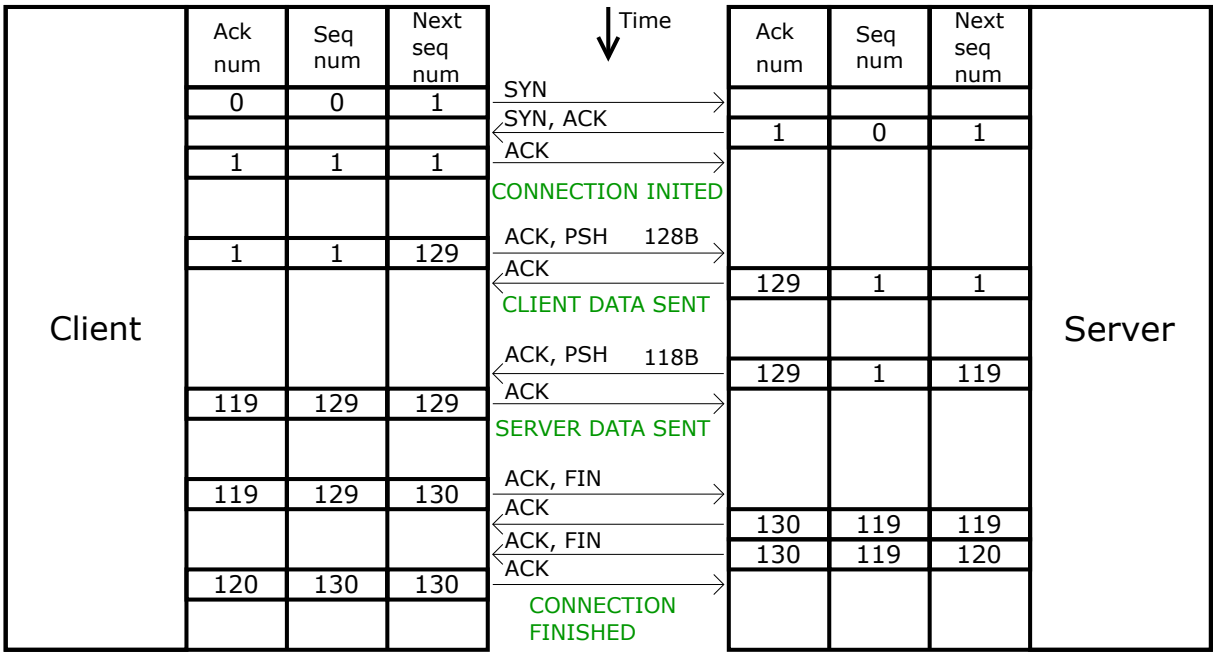


Figure 18: Example of TCP communication.

2.3.2.5 Session layer

In the session layer of the ISO/OSI model data may be composed of PDUs, which structure is described in Chapter 4, SOME/IP, ViWi, DoIP, or other protocol. In this thesis, it is dealt with data composed of PDUs.

3 Principle of system functionality

The core part of the designed system is a group of libraries (see Figure 19) encapsulated by the Network capture manager library, which provides an effective tool for capturing packets and their interpretation. The core of the system is connected by various interfaces, which allows different access to the library functions and gives different options to the user due to each selected interface.

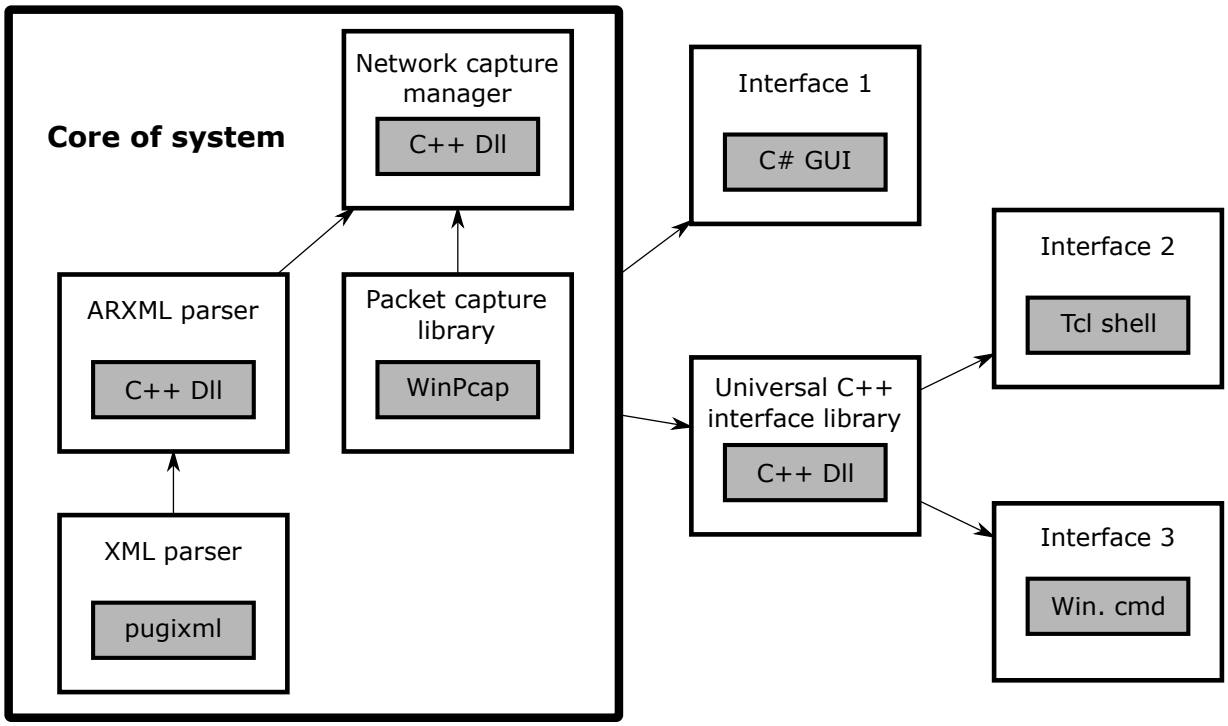


Figure 19: Structure of the system.

The core of system can be divided into 4 subsystems. First subsystem is third-party open-source pugixml library, which is extremely fast, light-weight C++ XML library which allows parsing a large XML file into Document Object Model (DOM) tree in the order of 100 MB in few seconds [10]. Second subsystem is ARXML parser library. This library encapsulates elementary functions of pugixml library and it is Swiss knife for searching in ARXML files.

The third subsystem is a third-party open-source library WinPcap which is an industry-standard tool for capturing and transmitting network packets [11]. The last subsystem is the Network manager library; this subsystem connects ARXML parser and WinPcap, creating a tool for receiving packets and its transformation into useful data.

The process of capturing is a complex routine, due to it is divided into several subroutines. Each subroutine can finish the job in a different time, to avoid slowing down to

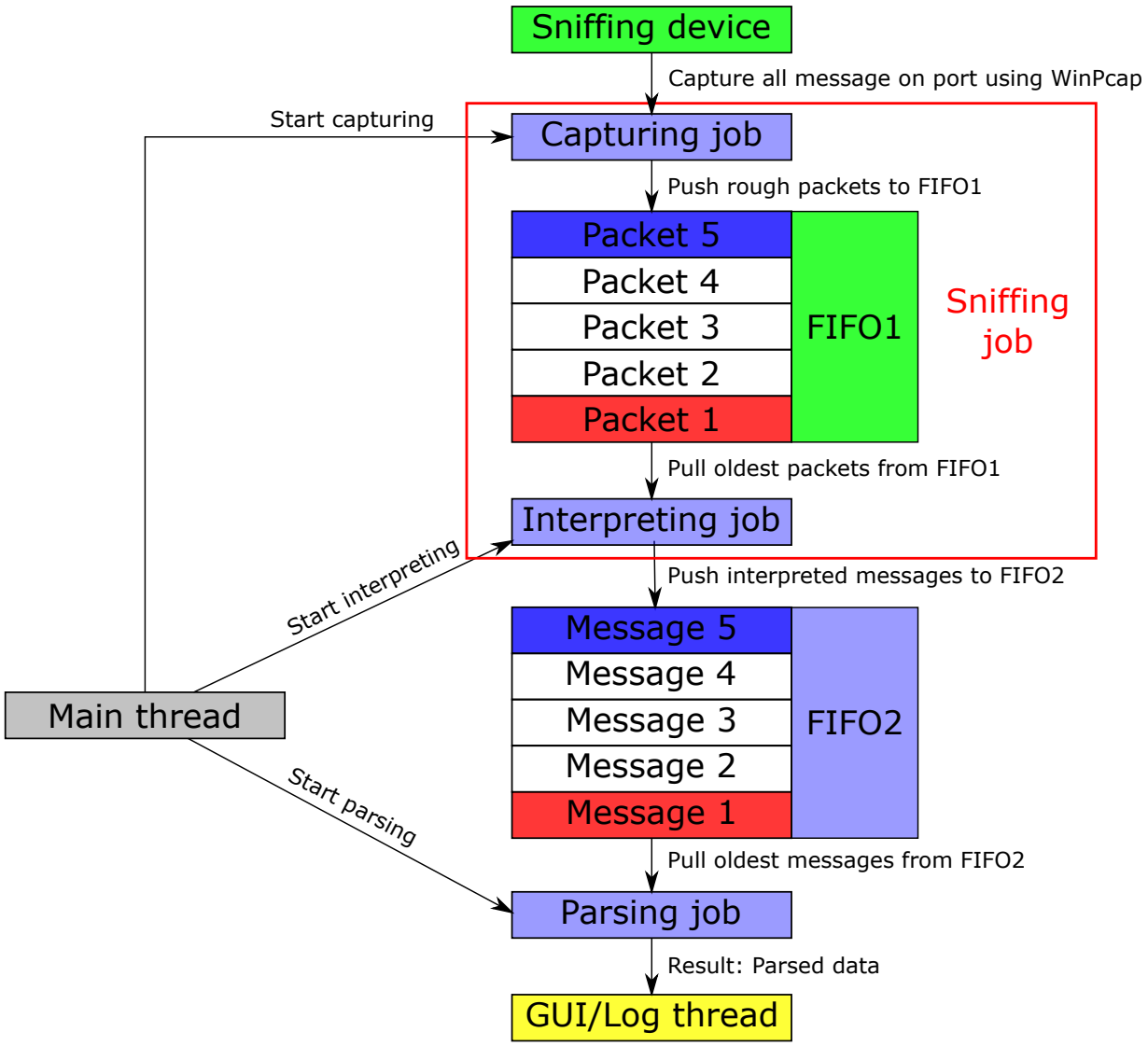


Figure 20: Internal principle of the system.

a speed of the slowest subroutine, FIFO buffers with dynamic length are implemented to store input/result of each job (see Figure 20).

When the user pushes a "start" button or types an appropriate command, threads for parallel jobs are activated. The first thread is assigned to the Capturing job, which handles incoming packets using the WinPcap library loop routine. It listens on a port, where media convertor (RAD-Moon) resends all captured packets. When any packet is received, it asserts an interrupt and calls a callback, which pushes packets into FIFO1. This very simple mechanism guarantees that all packets will be processed in the future without any packet loss.

The next thread assigned to Interpreting job retrieves packets from FIFO1. This thread decodes rough data into communication protocol headers and useful data. This thread also fills all fields in headers. Due to it is possible to filter out all non-compliant packets that are not UDP/TCP IPv6, these packets do not carry any appropriate information for interpreting from the ARXML file and are forgotten (see Chapter 5). Remaining packets are pushed into the FIFO2 queue. The two above threads are parts of Sniffing job routine.

The next subroutine Parsing job is using the ARXML parser library functions and it is a bottleneck of the designed system due to the computation demands of searching too much data in an ARXML format. For this reason, this job is attached to multiple threads. The FIFO2 buffer length is periodically checked by parsing threads. If the length of the buffer is higher than 0, the oldest message is retrieved, and the parsing job starts. It should be noted that all threads started Parsing job routine must finish in the same order to keep the right order of incoming data. The result of the Parsing job is a structure where the packet is represented by human-readable meanings with assigned data (see Chapter 4). The data is then pushed directly to GUI or to buffer which is periodically read by Log thread in the case of command-line interfaces (see Chapter 6).

4 Implementation of ARXML parser

A lot of information is received from captured data (see Chapter 2.3) but how to interpret data using ARXML? There are many ways to get a message description from ARXML. In this thesis, the Protocol Data Unit (PDU) ID is mainly used as a key to finding a message description in ARXML. Note that PDU ID is unique key same as port (see Chapter 2.2).

Every Automotive Ethernet message may contain one or more PDUs. Each PDU is composed from PDU ID, PDU DLC and data (see Figure 21). The PDU ID and the PDU DLC are fields with a fixed size. As mentioned before, the PDU ID is used as a keyword for searching in the ARXML file. The PDU DLC determines the size of the data field. This way all the captured data are parsed into PDUs.

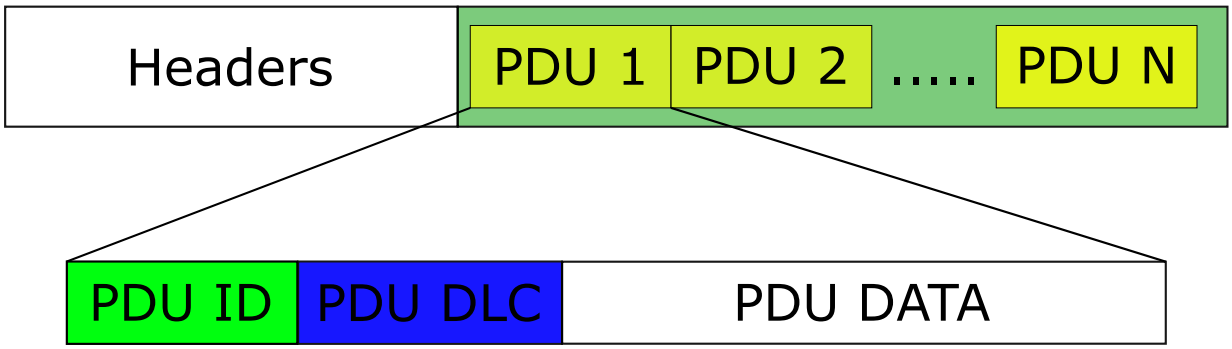


Figure 21: Internal structure of Protocol Data Unit.

Note that, this way error detection of incoming frames is easily done because for each received packet must pay an equation

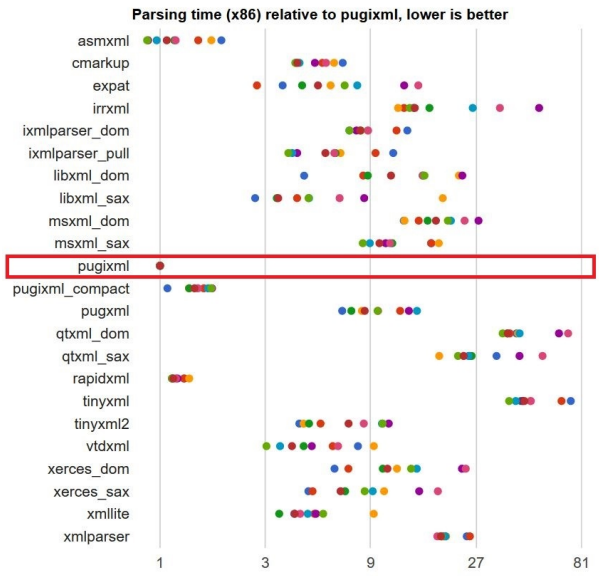
$$captured\_data\_len = \sum_{i=0}^{n} (pdu(i).id\_size() + pdu(i).dlc\_size() + pdu(i).dlc()). \quad (2)$$

The searching mechanism is explained in algorithm 1. Firstly the captured data is parsed into PDUs, then the PDUs are error checked using equation 2. After that, for every single PDU is found XML node using its ID in ARXML. Finally, the remaining information is retrieved using this node. This method looks for specific references, that should lead to specific data. This approach relies on the assumption that the ARXML hierarchy is unchangeable.

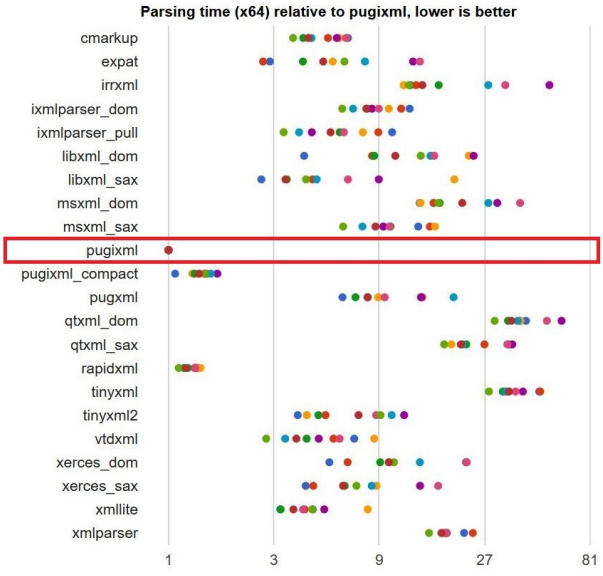
Algorithm 1: ARXML searching mechanism

```
input  : string input - data of any captured Ethernet message
         string path - path to ARXML
output: EthernetMessages output - data parsed into structure
begin
    // This opens arxml file (pugixml mechanism)
    xmlDoc arxml = openArxml(path);
    // This parses captured input data into pdus.
    pduVector pdus = parseInputDataIntoPDUs(input);
    // It is possible to parse data?
    if pdus == NULL then
        throw("Input data cannot be parsed.");
        return NULL;
    end
    // This finds information for all PDUs in ARXML.
    for i = 0 to pdus.size() do
        // This finds node due to text.
        xmlNode n = findNodeDueToID(pdus.at(i).ID, arxml);
        // This fills ethernet structure due to predefined references.
        output.Messages.pushBack(fillEthernetMessages(n, arxml));
    end
    return output;
end
```

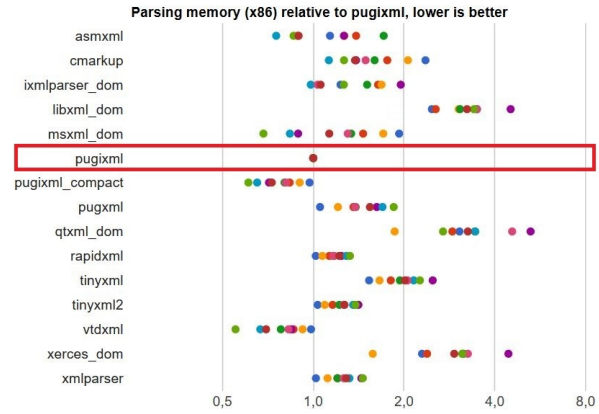
As already mentioned, searching information using ARXML is the bottleneck of the designed system. Therefore, the greatest emphasis is placed on the efficiency and speed of XML parser and ARXML file search. As mentioned in Chapter 3, for searching in ARXML the pugixml library is used. This library provides fast elementary functions for manipulation with large XML documents. See Figure 22 where the comparison of other XML parsers relatively to pugximl is shown. There are several dots in the graph, each representing a different XML document. In Figure 22a and 22b the parsing time of each XML parsers relatively to pugixml parser for 32-bit (x64) or 64-bit (x86) application is shown. In Figure 22c and 22d the parsing memory of each XML parsers relatively to pugixml parser for 32-bit (x64) or 64-bit (x86) application is shown [10].



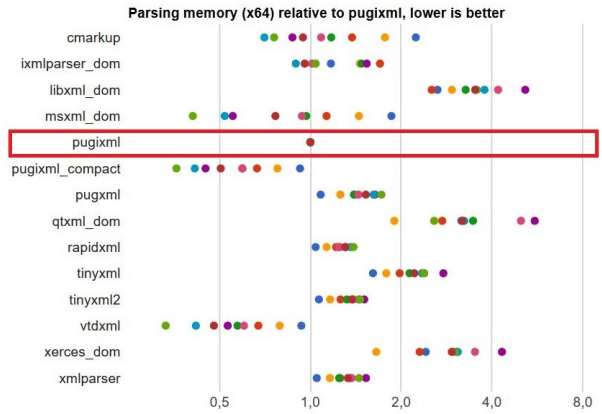
(a) Comparison of parsing time (x86).



(b) Comparison of parsing time (x64).



(c) Comparison of parsing memory (x86).



(d) Comparison of parsing memory (x64).

Figure 22: Comparison of XML parsers.[9].

[9]Source: https://pugixml.org/benchmark.html

The same important as choosing fast XML parser is to implement effective functions for finding information in ARXML. For better performance the ARXML parser is written in C++ language using only basic structures, classes, and pointers. A fact that the system of XML branching of the ARXML is always the same is also used to increase search speed. Under this assumption, it is possible to create a map of information and paths to access it (see Figure 23). It is also possible to navigate starting part of the route statically. Taking advantage of this fact, the considerably speed up the search is possible, as it significantly reduces the number of branches to be searched.

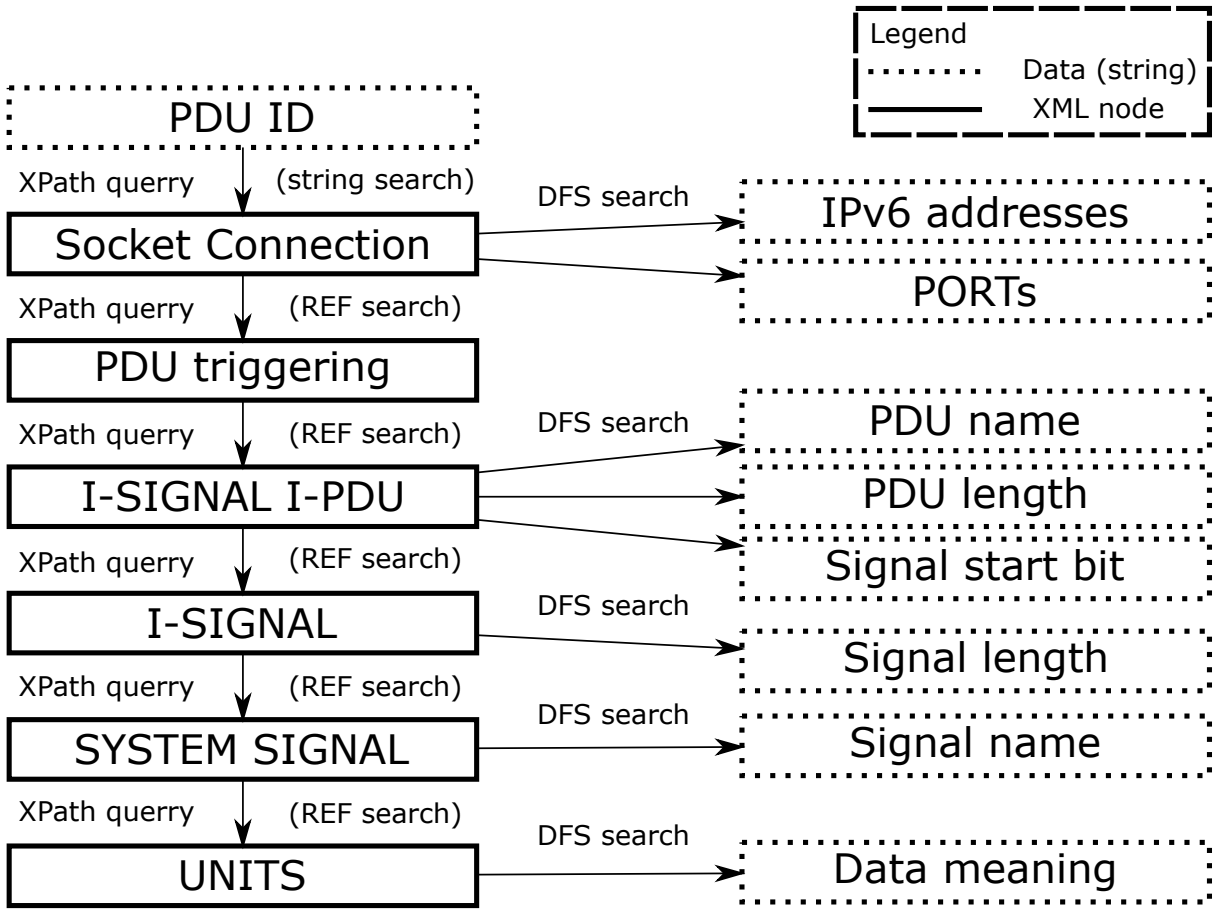


Figure 23: ARXML map of information.

As shown in Figure 23, different searching methods for different cases are used. First PDU ID is obtained from the message, this ID is used for searching in ARXML using XML Path Language (XPath) query, which is used for selecting a subset of nodes that match specified criteria [10]. In this case, the XML node is searched with determining value and name. This is a very general search and therefore very slow, it takes hundreds of milliseconds to find this XML node. After this is used XPath query for moving between XML nodes, this search is less computationally demanding because the relative path (obtained from

REF) to the target XML node and also part of the start path is known (see Chapter 2.2.2). Finally, the required information is searched using the Depth-First Search (DFS) method.

5 Capturing packets

The Network capture manager library is responsible for capturing packets and it also encapsulates all libraries in the system. The system architecture is described in more detail in Chapter 3.

This library provides functions for printing all available sniffing devices, opening/closing sniffing devices, getting one packet from a device, exporting packets into a pcap file. These functions are used by interfaces described in Chapter 6.

When any suitable UDP or TCP IPv6 packet is received (see Figure 24), the information is parsed into fields with knowledge of packets/frames structure described in previous chapters. Then the parsed information is returned to the caller.

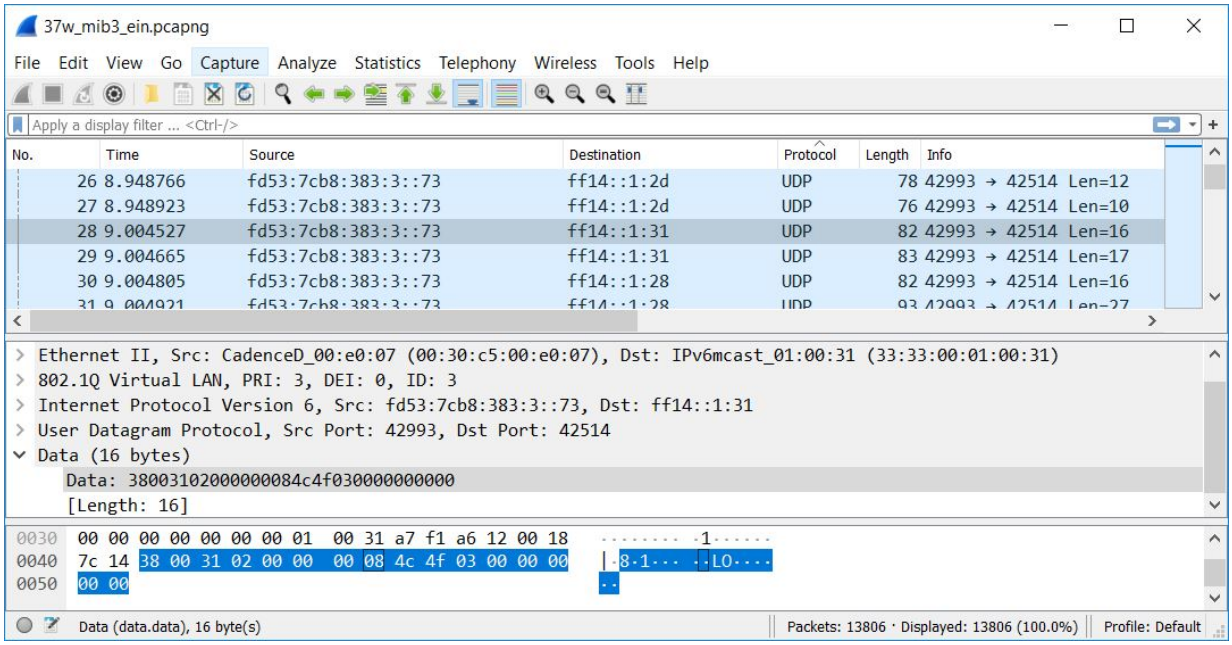


Figure 24: Example of rough data.[10].

For communication between the library and the interfaces, the returned data is encapsulated into a binary string in XML format. The main advantage is that the format is human readable and it is operable, possible to use in both C# and C++ applications.

[10]Source: Wireshark

6 Implementation of interfaces

The most important part of the designed system is an interface that interacts with the user. As mentioned in Chapter 3 and shown in Figure 19, three interfaces have been designed. First is C# Graphical User Interface (GUI), which is ideal for interacting with the user. This interface is designed as a tool for analysing an Automotive Ethernet network using a personal computer. Second interface is the Tcl shell interface, designed for use as an embedded system commanded in Tcl language. Finally, an interface for commanding from Windows using Command Prompt has been designed. The last two interfaces are controlled from the command line, they are based on the same C++ library, which provides implemented all functions for any command line like interface. This gives the possibility to create an unlimited amount of command-line interfaces with the same functionalities.

6.1 GUI interface

The GUI Interface is implemented in C# language. This language has been chosen because it is an effective tool for creating a nice-looking graphical interface. Due to the designed system architecture described in Chapter 3 there are no high-performance requirements because all computationally demanding operations are done inside of C++ libraries.

This GUI provides a user-friendly tool for analysing the Automotive Ethernet network. The GUI contains only the necessary number of buttons, which makes this interface very intuitive and easy to use.

One of the most important things that are needed to be set before any use of the GUI is a path to the ARXML file, which is used for interpreting packets. If the user chooses an option that requires ARXML, the GUI first searches for file "Default.arxml" in the same folder level where the binary files of the application are located and if the file is not found the user is asked to specify the ARXML file (see Figure 25).

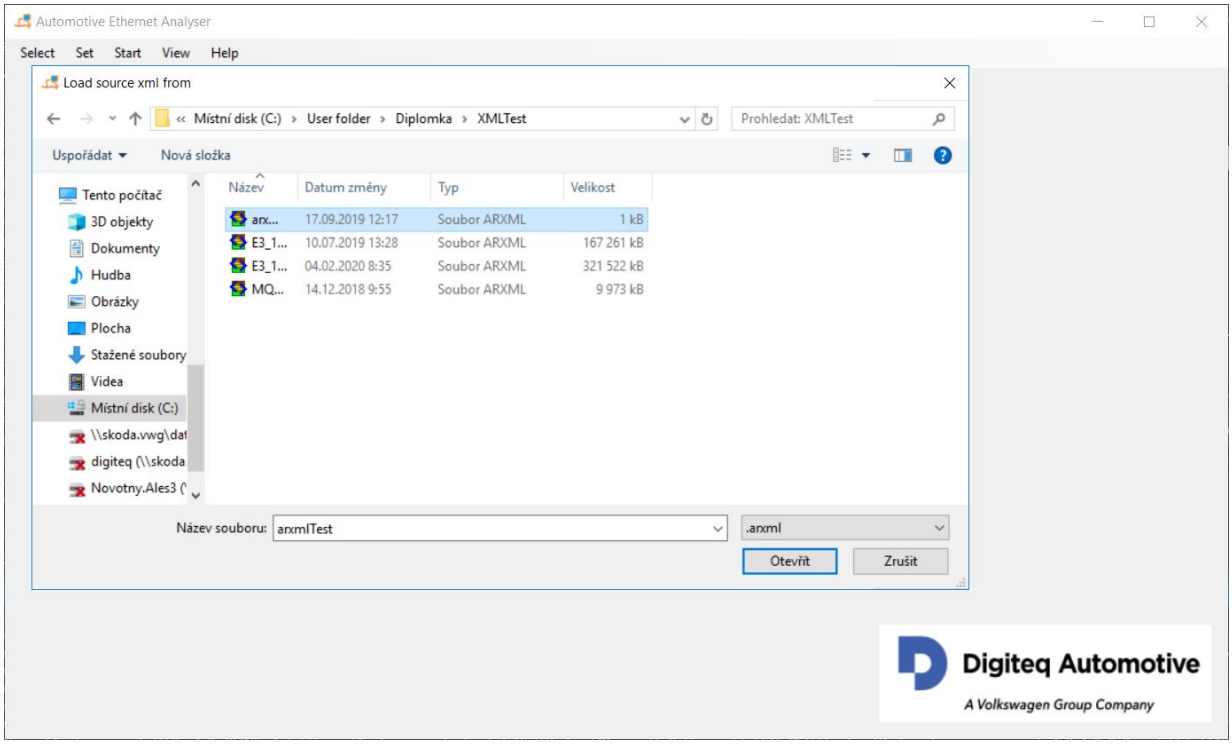


Figure 25: Loading ARXML file in GUI interface.

Setting a sniffing device that is connected to Automotive Ethernet by the media converter is also important. This is needed for any mode when the live network is sniffed. The chosen option is then remembered until the application ends, so the user does not need to specify it again. If the user wants to use another device, the connection must be set manually. The Figure 26 shows example of available connections. If the user connects a new device after this window is shown, the "Reload connection devices" button should be pressed and the new device appears in the list.

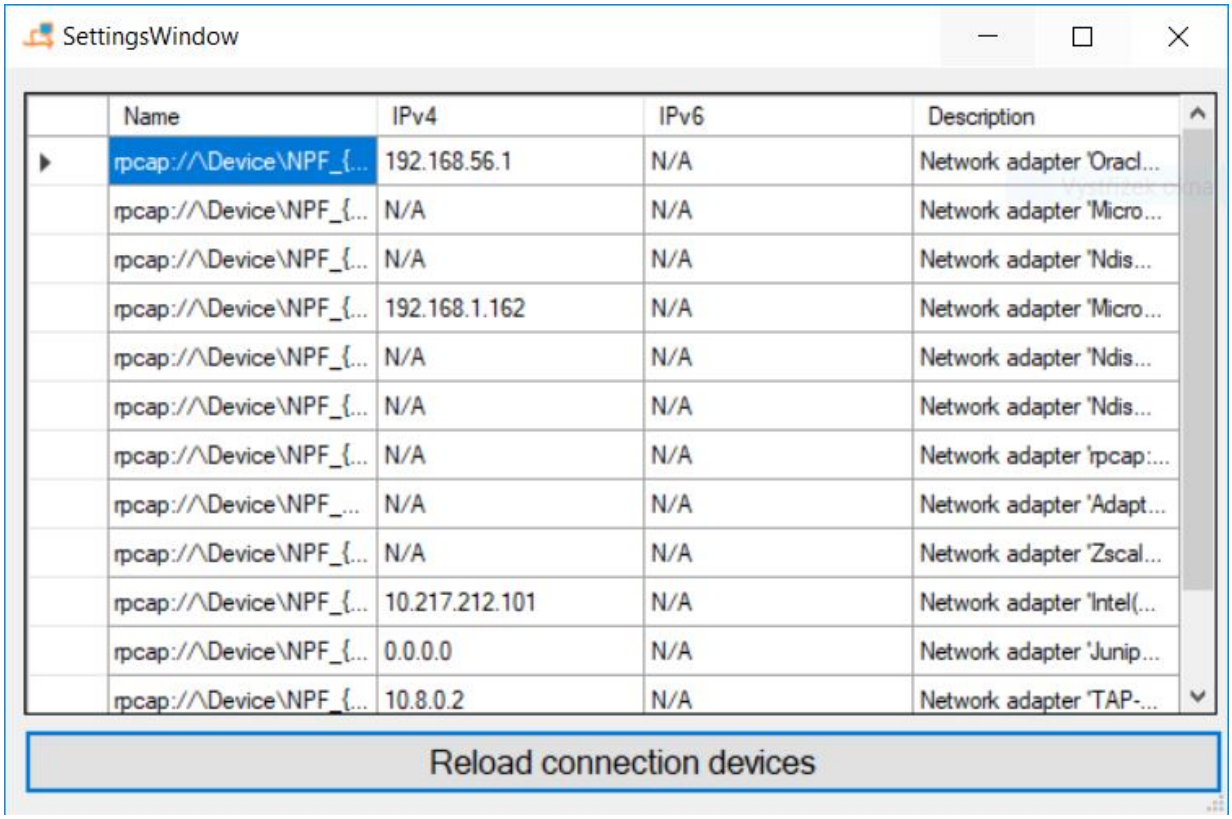


Figure 26: List of available sniffing devices.

The path to ARXML and connection to a sniffing device can be remembered for the next run of the GUI by clicking to "Save fast config". This option save file in location, where the quick config file is saved in the same folder level where the binary files of the application are located or by clicking to "Save config as" can the user specify where the config file should be saved (see Figure 27). Similarly, the config should be loaded during next time when GUI is running by clicking options "Load fast config" or "Load config from".

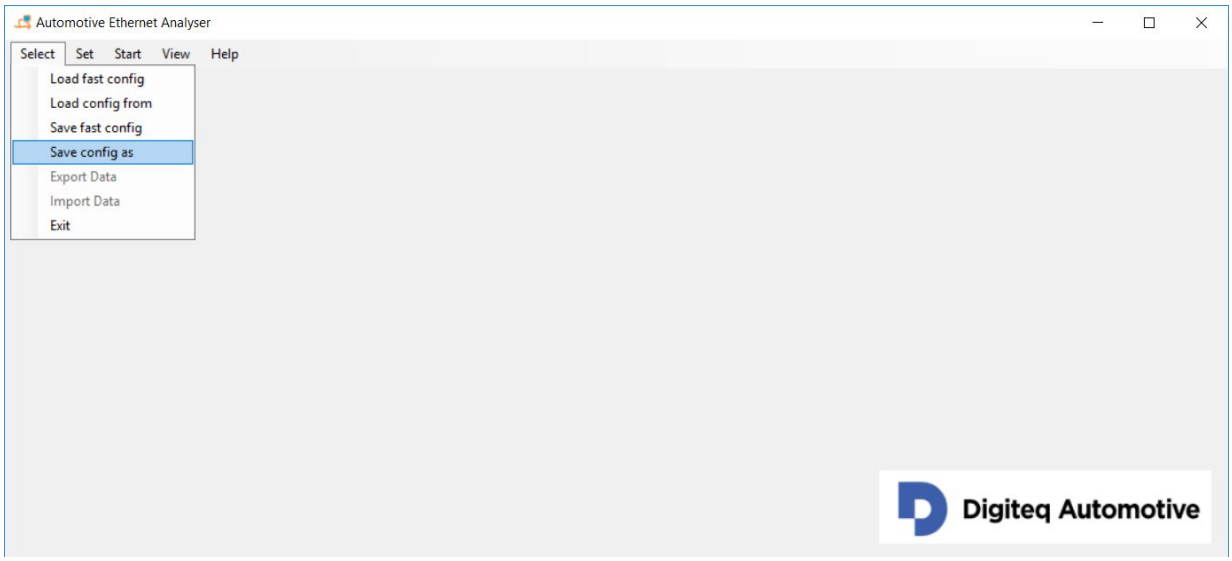


Figure 27: Saving configure file.

After the user selects the sniffing option and the GUI is configured, the filtering window is shown (see Figure 28). This allows the user to choose one of four filtering methods. When the "Filter off" option is selected, all suitable UDP or TCP packets are interpreted. The option "Filter due to ID" means that only desired PDU ID appears in results. The next option is "Filter due to IPv6", this means that only packets sent from/to this node address are accepted. And finally "Filter due to signal" means that only signal with the specified name is shown in the result table.

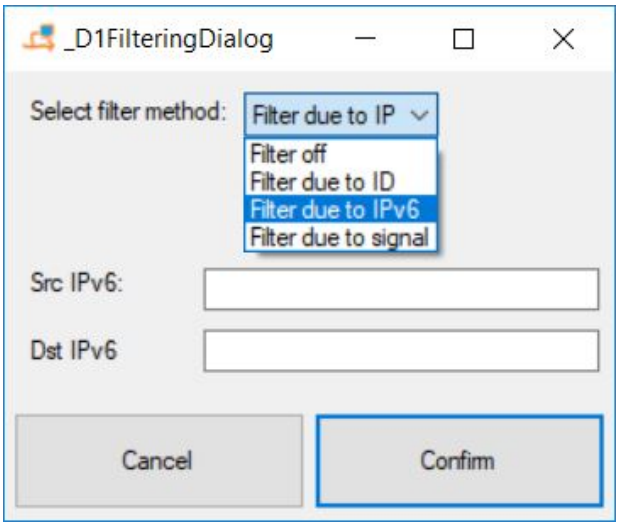
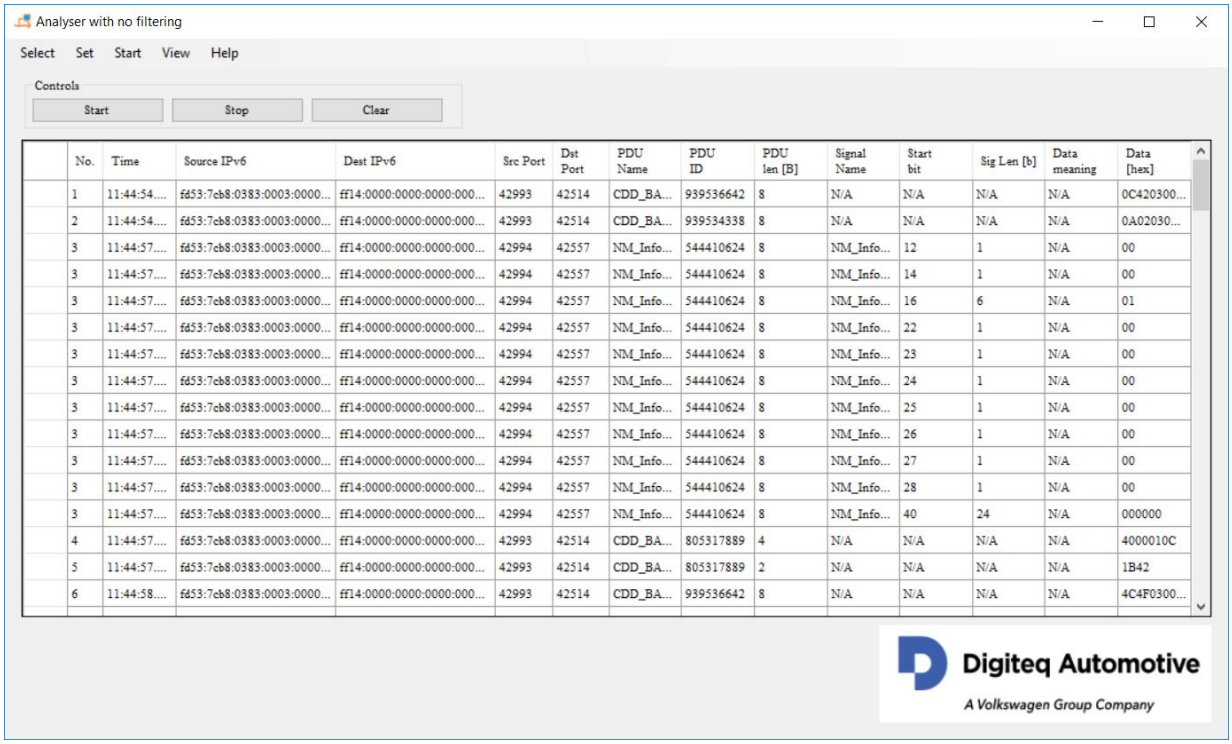


Figure 28: 100BASE T1 physical layers and its sublayers.

Once the sniffing mode (filters) is selected, the threads are started. When an appropriate packet is received, it is interpreted and the results are stored in a table (see Figure 29). The user can control sniffing by control buttons. Pressing the "Start" button cause that all sniffing threads starts, pressing the "Stop" button are all threads stopped, and after pressing the "Clear" button, table is cleared.



No.	Time	Source IPv6	Dest IPv6	Src Port	Dst Port	PDU Name	PDU ID	PDU len [B]	Signal Name	Start bit	Sig Len [b]	Data meaning	Data [hex]
1	11:44:54....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42993	42514	CDD_BA...	939536642	8	N/A	N/A	N/A	N/A	0C420300...
2	11:44:54....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42993	42514	CDD_BA...	939534338	8	N/A	N/A	N/A	N/A	0A02030...
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	12	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	14	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	16	6	N/A	01
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	22	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	23	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	24	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	25	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	26	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	27	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	28	1	N/A	00
3	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42994	42557	NM_Info...	544410624	8	NM_Info...	40	24	N/A	000000
4	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42993	42514	CDD_BA...	805317889	4	N/A	N/A	N/A	N/A	4000010C
5	11:44:57....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42993	42514	CDD_BA...	805317889	2	N/A	N/A	N/A	N/A	1B42
6	11:44:58....	fd53:7cb8:0383:0003:0000...	ff14:0000:0000:0000:000...	42993	42514	CDD_BA...	939536642	8	N/A	N/A	N/A	N/A	4C4F0300...

Figure 29: Example of sniffing without filters.

The results can be then directly exported into file by selecting "Export Data" option (see Figure 30). This allows user to export data from table into various formats for opening data in most common application. The first is format "blf", which is one of the most widespread logging formats in the automotive industry, this format is used by programs like CANoe developed by Vector. Next format is "pcap", this format is widely used among network analysing applications based on WinPcap library like Wireshark. Follow format is "csv", which allows to export data into table which can be then opened by standard office applications like Microsoft Excel or Open Office. The last possible option is saving into "txt" text file which can be opened on any PC with almost every application.

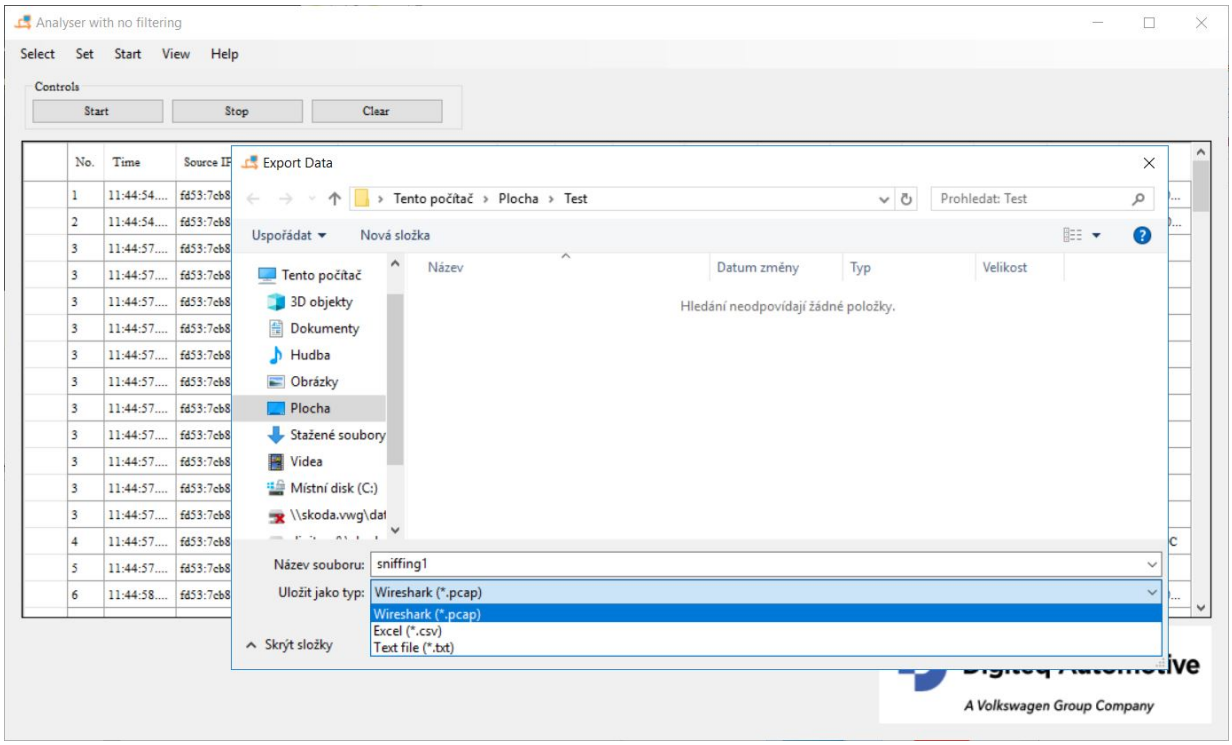


Figure 30: Example of exporting captured and parsed data.

The GUI also provides the possibility of displaying signal value changes over time into a chart (see Figure 31). This is an extension of sniffing "Filter due to signal" filter option (see Figure 28). The user can control the application by control buttons similarly to previously described sniffing control buttons. The "Start" button starts sniffing threads, "Stop" button stops threads, and "Clear" button clears the chart. The user has to specify the signal name, which is tracked and voluntary history of the chart, which allows setting how many values should be viewed. Also when the user points on some point with a mouse, the value is shown (see Figure 31). This allows better readability of results than in the table.

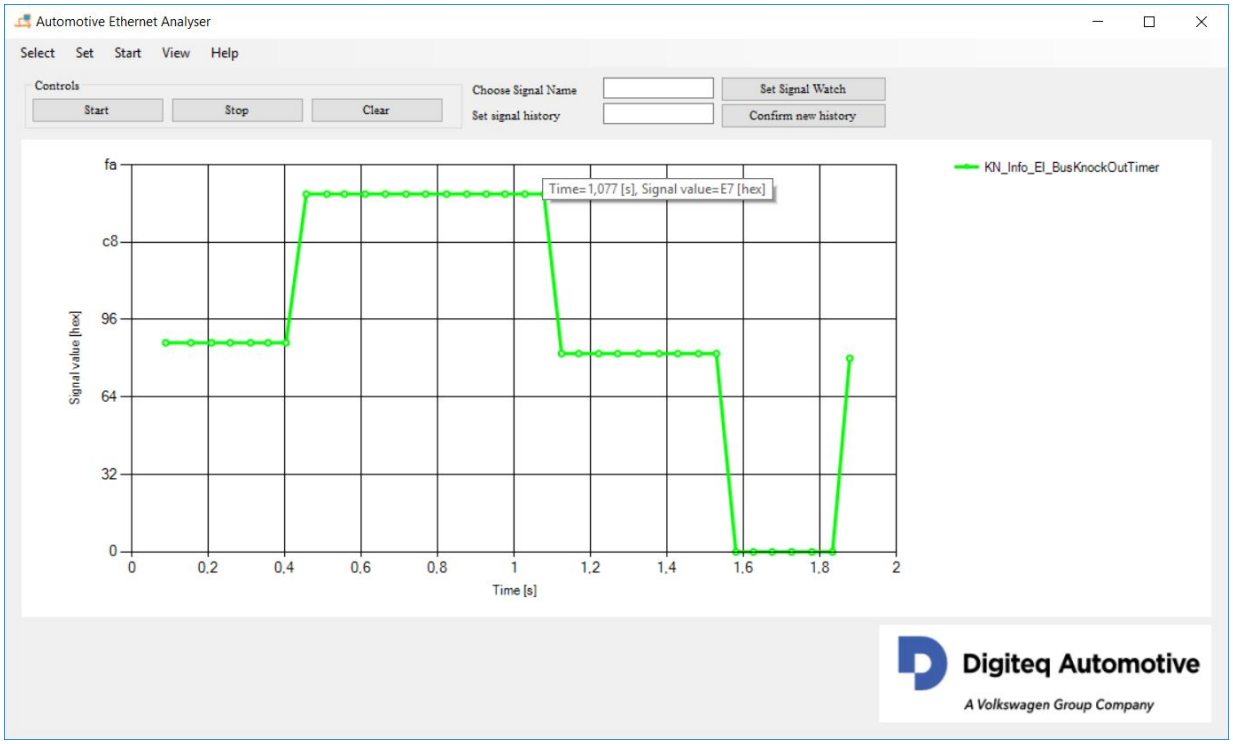
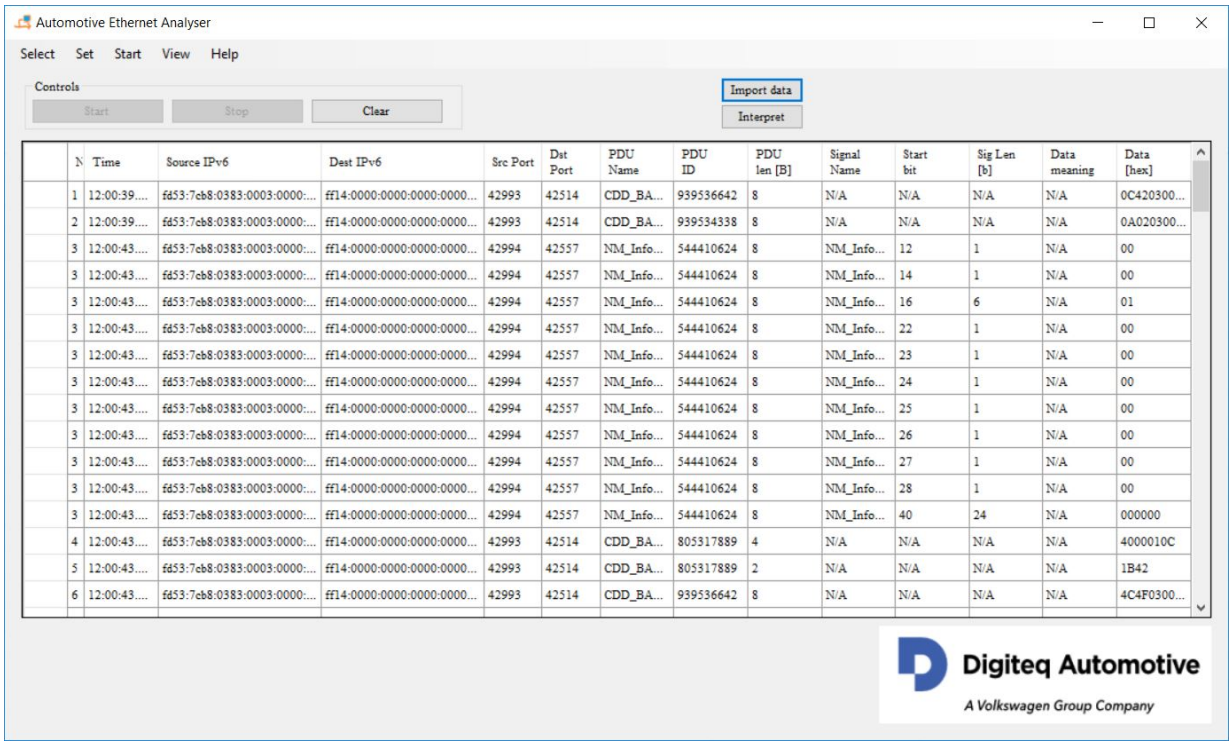


Figure 31: Signal tracker example.

The last mode of the program is offline log interpreter, which allows importing previously saved data from this GUI or different application as Wireshark by the "Import data" button and then the "Interpret" button interprets imported data by due to selected ARXML file (see Figure 32).



N	Time	Source IPv6	Dest IPv6	Src Port	Dst Port	PDU Name	PDU ID	PDU len [B]	Signal Name	Start bit	Sig Len [b]	Data meaning	Data [hex]
1	12:00:39....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42993	42514	CDD_BA...	939536642	8	N/A	N/A	N/A	N/A	0C420300...
2	12:00:39....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42993	42514	CDD_BA...	939534338	8	N/A	N/A	N/A	N/A	0A020300...
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	12	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	14	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	16	6	N/A	01
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	22	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	23	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	24	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	25	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	26	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	27	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	28	1	N/A	00
3	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42994	42557	NM_Info...	544410624	8	NM_Info...	40	24	N/A	000000
4	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42993	42514	CDD_BA...	805317889	4	N/A	N/A	N/A	N/A	4000010C
5	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42993	42514	CDD_BA...	805317889	2	N/A	N/A	N/A	N/A	1B42
6	12:00:43....	fd53:7cb8:0383:0003:0000:...	ff14:0000:0000:0000:0000...	42993	42514	CDD_BA...	939536642	8	N/A	N/A	N/A	N/A	4C4F0300...

Figure 32: Log interpreter example.

6.2 Tool Command Language interface

Tool Command Language (Tcl) is an easy-to-learn open-source script language, suitable for a variety of applications such as testing, administration, networking, etc. [33].

This interface can be loaded and controlled in any Tcl console. Because it is a command line-like interface, many functions have been reduced. Basically, it is an interface that loads an ARXML file for interpreting packets (filtering) and stores results in a log file.

6.3 Windows Command Prompt interface

The last interface is implemented for use with PC with installed Windows operating system without the need to install additional programs and can be easily integrated into a larger system. It is a console application that is used for the same as the Tcl interface in Chapter 6.2, for interpreting packets (filtering) using ARXML and logging into a file.

7 Functionality verification by a simulation

7.1 Simulation environment

Simulations are extremely important in the automotive industry. In our case, a console application is used for simulating the functionality of the system. This application can simulate connected sniffing devices by loading packets from a log and forwarding them to the required Ethernet port.

For simulating the functionality of the system, real communication between ECUs was sniffed by Wireshark. This data is then loaded by the application using the Winpcap library and sending in the same relative times to start as received. After all packets from .pcap log are sent, the application starts sending from the beginning of the log doing this until the application is not quit. This provides good training data for interpreting packets and measuring and improving the performance of the system functions (see Figure 33).

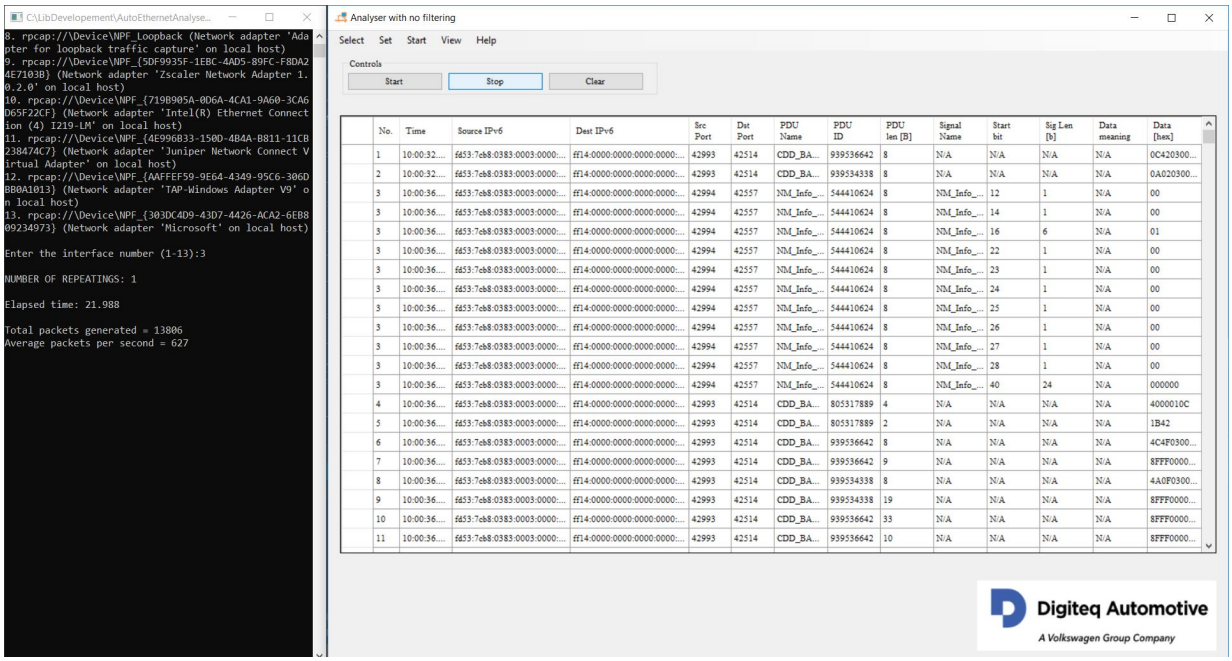


Figure 33: Example of simulation on one PC.

7.2 Verification by a simulation

Only two PCs connected by an ethernet cable are needed for the simulation. One PC simulates an ECU with the use of the application mentioned in Chapter 7.1, on the second PC, my analyser is running.

The system was running on a laptop with an Intel Core processor with 4 cores and 8 threads and with 16 GB installed RAM. The datasets used for the simulation are described in a Table 2. The "Time of record" field means how long takes to run each record.

Name	**Number of packets**	**Time of record** [s]	**Packet rate** [packets/s]
File1	13806	21.988	627
File2	1000	1.83	546

Table 2: Overview of recorded sniff used for simulation.

Two interfaces were verified by the simulation, C# GUI and C++ Windows Command Prompt interfaces. Tcl interface was not verified by these tests because it uses the same functions as Windows Command Prompt, this is described in Chapter 6.3.

7.2.1 Test of real rough data

In the following simulation, packets from File1 (see Table 2) were sent, both interfaces were tested for capturing and parsing packets, and processing time was measured.

Name of test	**Capturing and parsing time** [s]
GUI: without filter	63
GUI: PDU ID filter	30
GUI: IPv6 filter	23
GUI: Signal name filter	31
GUI: Signal tracker	25
Command Prompt: without filter	26
Command Prompt: PDU ID filter	26
Command Prompt: IPv6 filter	26
Command Prompt: Signal name filter	26

Table 3: Results of the first part of the simulation (running record from File1).

The results of the simulation are shown in Table 3. The command-line interface is much faster than GUI, this is caused by interaction with GUI elements. The command-line interface was able to process almost 22 s record with only little delay.

7.2.2 Test of capture data loss

In this simulation, the reliability of the program was tested. 1000 same packets from File2 (see Table 2) were sent in a very short time. They were captured and parsed with two mentioned interfaces. The time of capturing and parsing was measured, and the total number of packets was counted.

Name of test	**Capturing and parsing time** [s]	**Total number of packets** [-]
GUI: without filter	19	1000
GUI: PDU ID filter	15	1000
GUI: IPv6 filter	18	1000
GUI: Signal name filter	5	1000
GUI: Signal tracker	106	1000
Command Prompt: without filter	3	1000
Command Prompt: PDU ID filter	3	1000
Command Prompt: IPv6 filter	3	1000
Command Prompt: Signal name filter	3	1000

Table 4: Results of the second part of the simulation (running record from File2).

The results are shown in Table 4, it confirms that all packets were reliably captured by all described interfaces. The fact, that the GUI is now a bottleneck of the designed system was also confirmed (see Chapter 7.2.1). The GUI algorithm is a little bit smarter than the algorithm used for the Command Prompt, in the case of the PDU ID filter and Signal name filter, information for packets was searched in the ARXML only once, in this case, but still, the GUI was much slower than Command Prompt.

7.2.3 Test of logger functionality

In the last simulation, logging into a file was tested using the same record file as in the previous simulation (see Chapter 7.2.2). In the GUI case, the packets were first captured and parsed, then logged to various formats. The logging time was measured and logged packets were counted. In the Command Prompt case, the packets were captured, parsed, and logged simultaneously, therefore it was measured time of all process parts, not only logging into a file.

Name of test	**Logging time [s]**	**Total number of packets [-]**
GUI: .blf	0.7	1000
GUI: .pcap	1.1	1000
GUI: .csv	0.6	1000
GUI: .txt	5.6	1000
Name of test	**Capturing, parsing and logging time [s]**	**Total number of packets [-]**
Command Prompt: .blf	3.6	1000
Command Prompt: .pcap	3.7	1000
Command Prompt: .csv	3.6	1000
Command Prompt: .txt	3.6	1000

Table 5: Results of the third part of the simulation (running record from File2).

The results presented in Table 5 confirmed that this application is able to create a fast log file without data loss. Logging into "blf" and "pcap" takes a similar time because the logging principle is similar. Logging into "csv" and "txt" is more complex and creates bigger files because it holds all information shown in the table.

8 Conclusion

This thesis deals with Automotive Ethernet communication analysis for the automotive industry using a common PC equipped with a simple media convertor from Automotive Ethernet to Ethernet.

The first subtask of this thesis was to implement methods allowing capturing frames at Ethernet interface using common personal computer. The WinPcap library was used for mechanical capturing packets, which is described in Chapter 3. After this, useful information is gathered with knowledge of ISO/OSI model, which is described in Chapter 2.3. Then the principle of capturing is described in Chapter 5.

The next subtask was to filter captured data by address at least. This has been implemented at the level of interfaces and described in Chapter 6. Not only address filtering has been implemented, but also PDU ID and signal name filtering.

The next point was to log data in a format suitable for further processing. This has been also implemented at the interface level and it is described in Chapter 6. It is possible to save data into "blf" format, which is one of the most widespread logging formats in the automotive industry, "pcap" format, which is widely used among network analysing applications based on WinPcap library like Wireshark. Next is "csv" format, which allows exporting data into a table which can be then opened by standard office application like Microsoft Excel or Open Office. Finally, there is an option to save data into "txt" format, which can be opened on any PC with almost every application.

The next subtask was to import descriptions from AUTOSAR XML (ARXML). For mechanical loading ARXML file into memory and parsing to DOM pugixml library was used, this is described in Chapter 3. The performance of the pugixml library is shown in Chapter 4. The AUTOSAR standard, ARXML format, and the searching methods are described in Chapter 2.2.

The last but one point was presentation of the captured data by graph or table. Both of them have been implemented in C# GUI interface described in 6.1.

The last subtask was the implementation of API for external program control. Two APIs were implemented, one can be turned on and commanded by Tool Command Language console (see Chapter 6.2) and second by Windows Command Prompt (see Chapter 6.3). These APIs are based on universal C++ library with implemented functionality. The library can be a basis for other command-line interfaces with the same functionality.

The reliability and performance of the designed system were tested in several simulations (see Chapter 7). Results proved that the system was able to capture, parse, and log packets without data loss, and in a relatively short time. The results showed that the GUI was slower than command-line interfaces because of interaction with GUI elements.

References

[1] W. Pananurak, S. Thanok, and M. Parnichkun, "Adaptive cruise control for an intelligent vehicle," in *2008 IEEE International Conference on Robotics and Biomimetics*, Feb 2009, pp. 1794–1799.

[2] Y. Ying and O. Odunayo Solomon, "Research on adaptive cruise control systems and performance analysis using matlab and carsim," in *2017 5th International Conference on Mechanical, Automotive and Materials Engineering (CMAME)*, Aug 2017, pp. 249–253.

[3] J. Li and X. Sun, "A parking algorithm for parking assist system," in *2011 International Conference on Electric Information and Control Engineering*, April 2011, pp. 86–90.

[4] Joshi and Svb, "Can, flexray, most versus ethernet for vehicular networks," 04 2018.

[5] C. M. Kozierok, C. Correa, R. B. Boatright, and J. Quesnelle, *Automotive Ethernet: The Definitive Guide*. Intrepid Control Systems, 2014, ISBN-10: 0-9905388-0-X, ISBN-13: 978-0-9905388-0-6.

[6] D. Bučkovský, "Ethernet application in passenger cars," Bachelor's thesis, Czech technical university in Prague, 5 2015.

[7] A. Sumorek and M. Buczaj, "The evolution of "media oriented systems transport" protocol," *Teka Komisji Motoryzacji i Energetyki Rolnictwa PAN*, vol. 14, pp. 115 – 120, 08 2014.

[8] S. B. Carlson, J. Farkas, N. Finn, D. Pannell, M. Potts, M. Seaman, N. Wienckowski, and G. Zimmerman, "Ieee 802 ethernet networks for automotive," 07 2017. [Online]. Available: http://ieee802.org/802_tutorials/2017-07/tutorial-Automotive-Ethernet-0717-v02.pdf

[9] Intrepid Control Systems, Inc., *RAD-Moon*, 05 2016, (accessed: 20.03.2020). [Online]. Available: https://cdn.intrepidcs.net/guides/radmoon/rad-moon_ug.pdf

[10] A. Kapoulkine. (2018) pugixml. (accessed: 20.03.2020). [Online]. Available: https://pugixml.org/

[11] (2018) Winpcap. (accessed: 20.03.2020). [Online]. Available: https://www.winpcap.org/

[12] J. Gosda, *AUTOSAR communication stack*, 03 2009. [Online]. Available: https://hpi.de/fileadmin/user_upload/fachgebiete/giese/Ausarbeitungen_AUTOSAR0809/CommunicationStack_gosda.pdf

[13] J. Alexandersson and O. Nordin, "Implementation of can communication stack in autosar," Degree project, Linköping University, 06 2015. [Online]. Available: http://liu.diva-portal.org/smash/get/diva2:822343/FULLTEXT01.pdf

[14] AUTOSAR. (2020) Classic platform. (accessed: 14.05.2020). [Online]. Available: https://www.autosar.org/standards/classic-platform/

[15] A. Nyßen and P. Könemann, "Model-based automotive software development using autosar, uml, and domain-specific languages," 02 2013.

[16] AUTOSAR, *Virtual Functional Bus*, 12 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_VFB.pdf

[17] AUTOSAR, *ARXML Serialization Rules*, 11 2019. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_TPS_ARXMLSerializationRules.pdf

[18] AUTOSAR, *Application Interfaces User Guide*, 11 2019. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_EXP_AIUserGuide.pdf

[19] Vector Informatik GmbH. (2018, 12) Ieee 100base-t1 (formerly oabr). (accessed: 30.03.2020). [Online]. Available: https://elearning.vector.cm/mod/page/view.php?id=152

[20] OPEN Alliance SIG. (2017) About open alliance. (accessed: 30.03.2020). [Online]. Available: https://www.opensig.org/about/about-open/

[21] Broadcom Corporation, *BroadR-Reach Physical Layer Transceiver Specification For Automotive Applications*, 5 2014. [Online]. Available: http://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.0.pdf

[22] IEEE Computer Society, "Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)," in *IEEE Standard for Ethernet*, 3 2016, ISBN: 9781504401371. [Online]. Available: https://standards.ieee.org/standard/802_3bw-2015.html

[23] IEEE Computer Society, "Amendment 4: Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over a Single Twisted-Pair Copper Cable," in *IEEE Standard for Ethernet*, 9 2016, ISBN: 9781504422888. [Online]. Available: https://standards.ieee.org/standard/802_3bp-2016.html

[24] Wikipedia. (2020, 4) Data link layer. (accessed: 21.04.2020). [Online]. Available: https://en.wikipedia.org/wiki/Data_link_layer

[25] D. Porter, *100BASE-T1 Ethernet: the evolution of automotive networking*, Texas Instruments, 2018. [Online]. Available: http://www.ti.com/lit/wp/szzy009/szzy009.pdf

[26] Wikipedia. (2020, 5) Ethernet frame. (accessed: 14.05.2020). [Online]. Available: https://en.wikipedia.org/wiki/Ethernet_frame

[27] Wikipedia. (2020, 4) IEEE 802.1Q. (accessed: 14.05.2020). [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.1Q

[28] Wikipedia. (2019, 9) Network layer. (accessed: 21.04.2020). [Online]. Available: https://en.wikipedia.org/wiki/Network_layer

[29] Wikipedia. (2019, 10) Ipv6 packet. (accessed: 21.04.2020). [Online]. Available: https://en.wikipedia.org/wiki/IPv6_packet

[30] Wikipedia. (2020, 5) Transport layer. (accessed: 14.05.2020). [Online]. Available: https://en.wikipedia.org/wiki/Transport_layer

[31] Wikipedia. (2020, 4) User datagram protocol. (accessed: 14.05.2020). [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol

[32] Wikipedia. (2020, 5) Transmission control protocol. (accessed: 14.05.2020). [Online]. Available: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[33] Welcome to the tcl developer xchange! (accessed: 19.04.2020). [Online]. Available: https://www.tcl.tk/

Appendix A DVD Content

In Table 6 are listed names of all root directories on DVD.

Directory name	Description
code	Implemented Automotive Ethernet analyser
doc	This thesis in pdf

Table 6: DVD Content

Appendix B List of abbreviations

In Table 7 are listed abbreviations used in this thesis.

Abbreviation	Meaning
ACC	Adaptive Cruise Control
PLA	Parking Assist
OPS	Optical Parking System
DTR	Data Transfer Rate
ECU	Electronic Control Unit
CAN	Controller Area Network
AUTOSAR	AUTomotive Open System ARchitecture
XML	eXtensible Markup Language
ARXML	AUTOSAR XML
PDU	Protocol Data Unit
ISO	International Organization for Standardization
OSI	Open Systems Interconnection
DOM	Document Object Model
BSW	Basic Software
RTE	Runtime Enviroment
MCAL	MicroController Abstraction Layer
VFB	Virtual Functional Bus
XPath	XML Path Language
DFS	Depth-First Search
OABR	Open Alliance BroadR-Reach
PHY	PHYsical layer
MAC	Medium Access Control
PCS	Physical Coding Sublayer
MCU	MicroController Unit
PMA	Physical Medium Attachment
MII	Media-Independent Interface
RMS	Root Mean Square
PAM	Pulse-Amplitude Modulation
MDI	Medium-Dependent Interface
MDIO	Media Data Input/Output
SFD	Start Frame Delimiter
VLAN	Virtual LAN
TPID	Tag Protocol IDentifier
PCP	Priority Code Point
DEI	Drop Eligible Indicator
VID	VLAN IDentifier

IPG	InterPacket Gap
CRC	Cyclic Redundancy Check
DS	Differentiated Services
ECN	Explicit Congestion Notification
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
ACK	ACKnowledgement
PSH	PuSH
RST	ReSeT
SYN	SYNchronize
FIN	FINish
GUI	Graphical User Interface
Tcl	Tool Command Language

Table 7: Lists of abbreviations