XCP over Ethernet

Master's thesis in Embedded Electronic System Design

Ekin Ada Ustundag
Honglu Bian

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

MASTER'S THESIS 2020

XCP over Ethernet

Ada Ustundag Honglu Bian

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

XCP over Ethernet
Ada Ustundag Honglu Bian

Supervisor: Tomas Olovsson, Networks and Systems
Company Supervisors: Sumeet Thombre and Sara Gothäll, Volvo Group
Examiner: Per-Larsson Edefors, Embedded Electronic System Design

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

XCP over Ethernet
Ada Ustundag and Honglu Bian
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modern vehicles are increasingly becoming dependent on computers to control the equipment, such as cruise control, engine monitoring and so on. The computers and hence the ways to deal with the data transmissions are becoming more and more advanced. This has meant that the tools used for development, testing and debugging are no longer sufficient to be able to transfer the large amounts of data traffic that are created. To be able to handle these new data volumes, Volvo Group would like to adapt the newer standards i.e. the implementation of XCP(Universal Measurement and Calibration Protocol) over Ethernet. By the case of ECUs, it is important to have an optimal iterative process of measurement and calibration at the run-time. This is to fetch or tune the parameters that reside in the memory of ECU. During updating the firmware, flashing is used to update the application and to adapt the parameters in the ECU. Before, Volvo Powertrain used the tool ATI Vision which uses CCP (CAN calibration protocol) to achieve flashing and monitoring of the ECU over CAN. Many CCP implementations are migrating to XCP for calibration, data acquisition, and ECU flashing due to its independence from the transport layer and higher throughput.

Keywords: XCP, CCP, ECU, Measurement, Calibration, Ethernet, CAN

Acknowledgements

Hereby, we would like to thank the following people for providing us constant support throughout our thesis project. Without them this would not come true:
Sumeet Thombre, our supervisor at Volvo, who was with us from the start till the end, always bringing the brightest idea with a smile,
Sara Gothäll for her tutoring and monitoring,
Tomas Olovsson, our supervisor at Chalmers, who worked hard to guide us in the correct direction for a successful thesis,
Per-Larsson Edefors for helping us setting up the first steps of the project,
Magnus Stålesjö for taking our applications, although it was late,
Andrei Kovacz for all the instructions and laughter and,
all the amazing people at Volvo Powertrain for making us feel at home from day one.
Last but not least, we would like to thank our families for the constant support!

Ada Ustundag, Honglu Bian, Gothenburg, 2020

Contents

List of Figures

List of Tables

1
Introduction

Automotive companies and manufacturers of electronic control units (ECU) try to reduce the development costs and shrink the time to market. Concurrently, more challenging requirements on exhaust gas emissions, drivability, on-board electronics and obviously fuel/energy consumption is leading the industry into a complex paradigm. Architecturally, this equates to more control parameters and look-up tables (LUTs) in the ECU.

For the sake of these requirements, the calibration and measurement has become time-consuming and expensive task to tackle within the ECU (Electronic Control Unit) development systems. The powertrain teams constantly need ways to mitigate the over-complicated systems in order to efficiently conduct calibration and measurement [1].

There are three functions that are of significant importance to ECU application processes. They are measuring, calibrating and flash programming [2]. Measurement is the process of reading signals and parameters from the ECU. It aids to monitor key parameters of the ECU. Calibration is the iterative mechanism of tuning the ECU parameters. Flash programming means overwriting the EEPROM of the ECU is utilized to update the application already embedded within the ECU.

Today, the automotive industry introduces more complex systems and tend to have more strict timing constraints such as measurement/calibration delays. Every initiative such as Automotive Open Software Architecture (AUTOSAR) and third-party solutions are built to simplify the integration of these difficult applications. Therefore, under these circumstances, the tasks pertaining to the ECUs such as measurement, calibration and flashing become more and more challenging to achieve. Additionally, in vast environments such as Volvo Group interoperability and portability also gain higher priorities.

In the light of these premises, the thesis is structured in a way to overcome the weaknesses which were posed by CCP (Can Calibration Protocol). CCP is delimiting in the sense of allowing the usage of only one transport medium i.e. CAN bus. However, via migrating to XCP (Universal Measurement and Calibration Protocol), not only cumbersome waiting times are eliminated, but also interdependence to the transport layer is attained.

In this thesis, the goal is to implement XCP over Ethernet. In order to be compatible with the latest technology and know-how and the implementation is done conforming the requirements specified by Volvo Group. Afterwards, we have done some benchmarking where the throughput and data rate of the new implementation were compared to the default CCP implementation.

1.1 Aim

The purpose of this work was to implement and integrate XCP over Ethernet to facilitate testing and troubleshooting. After the implementation is done, the paper will look forward to juxtapose XCP and CCP to examine the CPU load of ECU. Moreover, there will be some benchmarking done and compare the throughput of new implementation to the default, existing CCP implementation. The goal is to use and adapt the next generation engine controller ECM4 with a supported Ethernet interface, which is a Vector delivered SIP (Software in Package) tailored for Ethernet communication. The tool CANape with support for XCP devices will be used to measure and calibrate. Additionally, a Vector CAN gateway VN5610A will be utilized to connect to the ECU.

1.1.1 Research Questions

In the light of the aforementioned points, we propose the following research questions:

1. What are the metrics of performance in computer networks that can be applied to measurement and calibration?
2. How does XCP over Ethernet perform vs CCP over CAN on the metrics of performance?

1.2 Motivation

Apart from the practical reason that the implementation is going to be used in the in house applications at Volvo Powertrain, the thesis also investigates how performance is defined within the network domain and how that translates into the benchmarking. Another upside would be the elimination of serial interface module's of ATI A7s and A8s. According to Volvo, they are expensive ECU add-on technologies that would be good to minimize the need of.

1.3 Thesis Outline

The thesis will follow this format:

- **Chapter 2: Technical Background** expresses the communication protocols and constitutes a background for the thesis.
- **Chapter 3: Problem Description** is an outline of the problem that is intended to be solved.
- **Chapter 4: Tools** are the collection of all the software and hardware tools that are used throughout the thesis.
- **Chapter 5: Results** present the outcome, the improvements over CCP and the benchmarking between XCP and CCP.
- **Chapter 6: Conclusion** indicates the summary of the whole work and also adds what could be done, as for the future work.

2 Technical Background

The technical background chapter's aim is to explain the concepts and the methodology that was followed throughout the thesis. First, network theory and modelling is to be examined. Afterwards, the metrics of performance within the networking applications are identified. To list, these are: end-to-end throughput (E2E throughput), Transmission Rate in Frames per Second (FPS) and CPU load [3], [4], [5].

Within the computer sciences domain, performance is a term that holds many definitions. Hence, more often than not, it requires a context-specific clarification. When the field of computer networks are considered, once again many definitions appear. Performance of a node can be measured in terms of queuing delay, probability of packet loss, probability of link down, end-to-end throughput and so on [5].

To counteract the confusion that might be aroused due to the indistinct definitions of performance, a research has been conducted to come up with suitable, well-fitting criteria. End-to-end throughput is a recurring performance metric occurring in the context of computer networks such as in [6], [7] and [8]. All these sources evaluate performance in networks via measuring the throughput. Additionally, bandwidth is another performance criteria that is inspected in [5] and [9]. Hence, addressing the first research question, the project takes the following metrics: bandwidth and end-to-end throughput.

XCP and CCP are two network protocols developed by ASAM (Association for Standardization of Automation and Measuring Systems). As an introduction to these protocols, they are both implemented to specifically target the ECU communication field. It must be mentioned that CCP was the predecessor of XCP and CCP was only bound to CAN bus. XCP, on the other hand, is more portable across the transportation layers, which gives itself an edge against its older brother in terms of performance [10]. Further in this section , there will be distinctive subsections (2.5 for CCP and 2.6 XCP) to thoroughly explain them. End-to-end (E2E) throughput is the amount of data per second that can be transferred. E2E throughput is an interesting metric to look at in XCP over Ethernet, while using DAQ. It opens up good contrast points between XCP and CCP. It is also highlighted that two critical aspects of performance in computer networks are E2E throughput and bandwidth [5]. Bandwidth management is the process of monitoring and controlling the traffic over a communication line. A particular transport layer's bandwidth is not much of a significant metric in the given project, due to the scarce amount of transmitting objects. There are no concerns such as fairness or bandwidth throttling, simply because there is only one hub (i.e. one master) that pushes data on the link. Hence, the link speed and bandwidth management are not thoroughly examined for the purpose of this project.

2.1 Network Topologies

A network topology is used in many areas to provide a medium to connect and interact with the other nodes in the network remotely. From the local area networks to the wide area networks, there are many types of networks tailored for use-cases. However, the simplest cases is the distributed peer-to-peer networks, where there is one client and one server. The other scenario is that there exists a master and a slave. These paradigms are quite close, yet they have their differences [11].

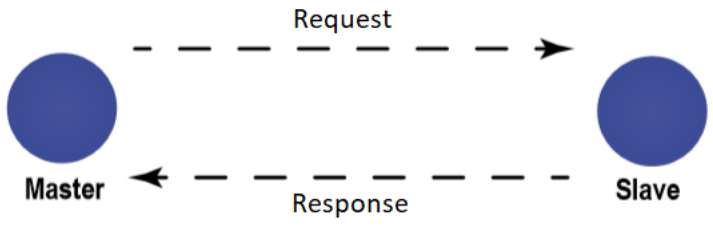


Figure 2.1: Master-slave network model

As observed in Fig. 2.1, the master is the requesting or "demanding" node which establishes the communication and requests data from the slave. In a master/ slave network, the slave cannot request nor initiate communications with the other devices. The slave node's responsibility is to respond the corresponding request and wait for the next request [12]. The master can change the memory of the slave device. This is useful within the context of the thesis, due to the calibration demands.

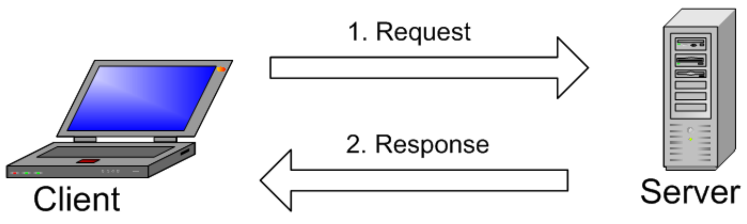


Figure 2.2: Client-server network model [13]

On the other hand, Fig. 2.2 describes how the client/server model is established between two computers. The client is the initiator of the communication and the server is expected to serve the query. The service request is done by the client and it is fulfilled by the server [14]. Quoting Wilfried Voss, "A Client/Server configuration is used for accessing but not modifying data in the Server since there could be unpredictable results if multiple Clients modified the data in an unsynchronized way" [14], master/slave networking is used in the applications where data manipulation and unidirectional transmission is desired. Therefore, master/slave archetype was better suited for what Volvo aimed to achieve.

Next up, we have investigated the metrics of performance within the computer networks, especially in the standards that are used in the automotive industry.

2.2 E2E Throughput and Frame Rates Per Second

Calculating FPS is a reliable way to rate the performance metric of a network. It is mostly used for the so-called monitoring and control systems. By definition, FPS yields the amount of frames that are transferred over in a given second [15]. Before looking at the FPS of each network protocol, it is beneficial to examine the rates of transmission these protocols offer. The distinctive rates of E2E throughput clearly play an important role to determine the FPS. The Ethernet protocol that was used granted 1000 Kbps. On the other hand, CAN yielded 500 Kbps. Therefore, without even considering the packaging and PDUs, Ethernet has the advantage here.

Hereby, the FPS investigation regarding CAN vs Ethernet is done. However, before that it is critical to note several premises. Actually, Ethernet alone cannot be compared to CAN in terms of FPS. The reason is raw Ethernet does not provide a re-transmission in case of a discarded/lost frame. Ethernet requires another layer to notify the transmitter. Therefore, higher level protocol layer, in this case TCP/IP, is needed, as an additional layer to get a confirmation message delivered to the transmitter. With this setup, the not acknowledgement is delivered to the transmitter, upon the frame's loss. The TCP/IP is taken into consideration, and hence the overhead [15].

An Ethernet package can contain 333 bytes with the following: MAC Preamble (7 bytes), Start Frame Delimiter (1 Byte), Destination and Source MAC Addresses (6 bytes each, 12 bytes in total), IP-Header (20 Bytes), TCP - Header (20 Bytes), Padding Bytes (6 Bytes), Frame Check Sequence (4 Bytes), Interframe Gap (12 Bytes), Payload (255 Bytes). With 1000 Kbps and 8 bytes per frame, it yields

$$\frac{1000000 \text{ bits/s}}{333 \times 8 \; bits/frame} = 3003 \text{ frames/s}$$

On the other hand, CAN packet is formed as below: SOF (1 Bit), Identifier + RTR (12 Bit in total), Data Length Code (6 Bits), CRC (16 Bits), ACK-Field (2 Bits), additional data (7 Bits), Interframe Space (3 Bits), Stuff Bits (3 Bits), so in total 114 Bits for 8 Bytes of Payload.

For a CAN network with 500 Kbps throughput,

$$\frac{500000 \text{ bits/s}}{112 \; bits/frame} = 4385 \text{ frames/s}$$

Just by solely looking at the FPS performance metric, CAN is the winner between the two protocols. It must be noted, though, that the net payload that is transferred by each frame also holds a significance. If the net payload is defined as,

$$(frame/s) \cdot (payload/frame)$$

then for CAN, it would yield:

$$4385 \text{ frames/s} \times 8 \text{ bits/frame} = 35080 \text{ bits/s}$$

For XCP,

$$3003 \text{ frames/s} \times 255 \; bits/frame = 765765 \text{ bits/s}$$

It actually implies that Ethernet transmits 21.8 times more net payload than CAN in a second. While measuring or calibrating, the net data that is transferred over the data bus is the most important parameter to monitor. Therefore, FPS metric alone, may not be enough to judge the performance of a network.

2.3 CPU Load

There are many relevant theories and field-specific research that presents the CPU utilization and load, as a performance metrics [3], [11], [5]. In our project, the CPU utilization was not measured, however the load was examined and benchmarked. The CPU load is a measurement signal that is fetched from the ECM (Engine Control Module). It is a significant metric to monitor due to how much of a trade-off Volvo ends up paying for opting Ethernet. In the previous section it is described that the net payload transfer of Ethernet is significantly higher than that of CAN, therefore, there will be a price to pay in the means of CPU load. In the Results section, this will be investigated and how much of a pay-off there is for opting Ethernet will be laid out.

Consequently, although CAN outperforms Ethernet in terms of raw efficiency, the standalone FPS would not be a meaningful performance metric to assess whether a network is inferior to the other [15], [4]. When the bandwidth and the payload is taken into account, it is quite clear that the winner is the Ethernet. Ethernet not only has higher E2E throughput, but also has greater payload, resulting into its superiority over CAN. As also another automotive research lays out, Ethernet is marginally the better interface, when the bandwidth requirements get higher. CAN, on the other hand, seems to be the go-to solution for the industry professionals, when the cost is the driving decision mechanism [4]. This is, due to the fact of the low efficiency nature of the Ethernet. Since Ethernet can transfer more data per packet, the packets contain more payload and also the transmission rate is higher, the decision was to opt for that instead of implementing XCP over CAN. For the sake of CPU load, there is a need to empirically test and observe, which implementation performs better, hence the CPU load will be scrutinized in the Results section.

2.4 Background on Volvo's Measurement and Calibration

At the start of the thesis, Volvo Powertrain used the tool called ATI Vision. This is an application tool using CCP to conduct flashing and measurement/calibration via CAN bus communication. Based on the Volvo implementation, it takes around

20 minutes to flash the memory of the ECU which became cumbersome and asked to be improved by the measurement engineers.

For the sake of faster data transmission, the newer technology is rebranded as XCP (Universal Measurement and Calibration). Compare to CCP which can only be used on CAN bus, XCP is able to work on different transport layers (CAN bus, Ethernet,USB,etc) which means more flexible load and better compatibility.

Polling and DAQ (Data AcQuision) are two ways of measuring data based on XCP or CCP. These two methods for data transmission are used in this project and the speed of data delivery and data amount are compared in the result. It is worth mentioning here that CCP is only available for polling, so XCP is already better for more approaches of data measurement than CCP.

Besides the explanation of protocol used in the project, the software components platform is mentioned in Section 2.7. AUTOSAR (Automotive Open System Architecture) system combined with more third party components is the basic structure Volvo Powertrain uses in Bus industry nowadays. The whole platform is composed by multiple functioning layers from the bottom hardware to the highest level application.

Based on the knowledge of platform in Section 2.7.1, to build XCP over Ethernet in the AUTOSAR, drivers and interfaces between different layers and components should be activated. Section 3.4 contains the stack of the XCP over Ethernet where seven software modules functioning XCP over Ethernet are introduced.

The scientific literature was also researched which we conclude after reading some other projects, papers, studies and other types of research. They are very important experiences from others which are quite helpful for this thesis and even further implementation.

2.5 CCP (CAN Calibration Protocol)

Back in the 1990s, CAN Calibration Protocol was developed for the purposes such as supporting the development of ECU, conducting functional and environmental tests of an ECU and for on-board testing and measurement [16]. The motivation was to move from proprietary, company-specific calibration schemes to a standardized, centralized protocol for calibration and data acquisition tools.

The CAN Calibration Protocol is a CAN-based master-slave protocol for calibration and data acquisition. Two types of CAN messages is needed, CRO and DTO as shown in Fig. 2.3. The CRO (Command Receive Object) messages are sent from master to slave and contain control commands. DTO (Data Transmission Object) messages are sent from slave to master. When a slave has received a CRO message it performs the given instructions and then answers with a DTO message containing a CRM (Command Return Message). CRM tells the master if the corresponding control command has been performed as planned or not.

The master device (host) is a calibration tool or a monitoring/diagnostics tool or a measurement system which start the data transfer on the CAN bus by sending packets to the slave devices. The CCP implementation both supports the commands for trivial memory transfers and for large torrent transfer of data acquisition [16].

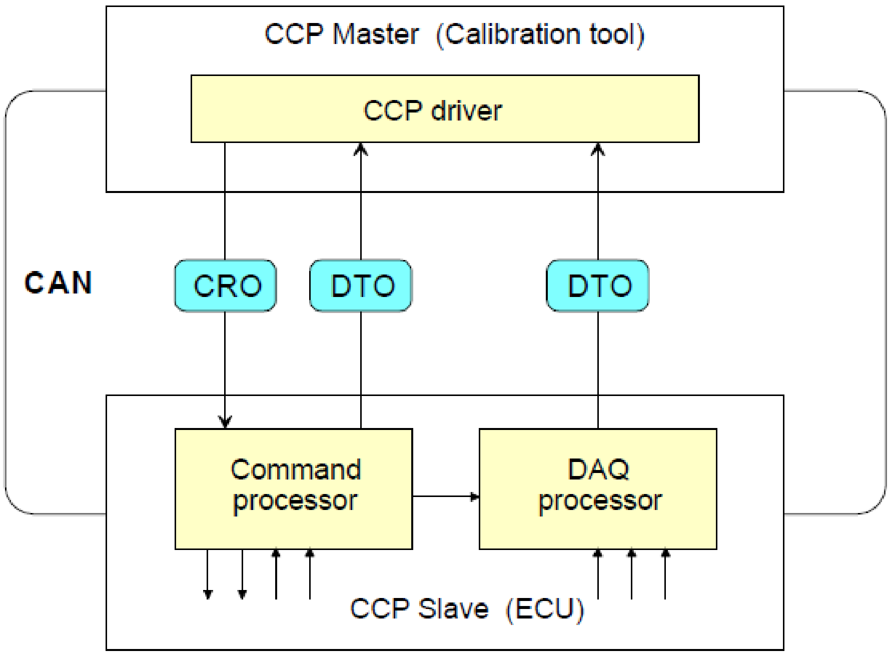


Figure 2.3: CCP Communication flow between the master and slave [16]

Despite the fact that CCP is still used within the industry, it is considered to be obsolete and no longer updated by the ASAM (Association for Standardisation of Automation and Measuring Systems)[10]. Even though the technology of CCP is quite mature in the Automotive industry, since the limit of CCP that is only available for CAN bus but not working on Ethernet and other transport layers, XCP over Ethernet is replacing it nowadays.

2.5.1 CAN (Controller Area Network) Bus

Controller Area Network is a communications protocol which is developed and maintained by Robert Bosch GmbH. The protocol is designed to manage multiplexed communications between multiple CPUs. It is control information oriented, uses non-destructive bit-wise arbitration to decide, which node "owns" the bus, and has a message priority scheme based on the value of the message identifier transmitted with each message.

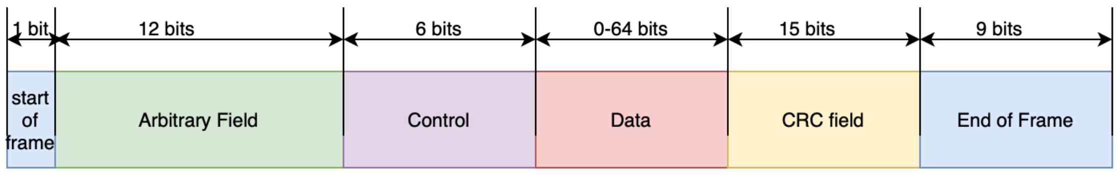


Figure 2.4: CAN bus Frame Format

CAN bus is a robust vehicle bus responsible for communication between Electronic control units (ECUs). CAN has four types for frames: data frame, error frame, remote frame, overload frame. Data frame is the only frame for actual data transmission among them. Observed from the base frame format shown in Fig. 2.4, we can see six parts compose a complete CAN frame. They are start of frame, arbitration field, Control, Data, CRC field and End of frame. However, the Data field which contains the data to be transmitted is at most 8 bytes which is quite limited. This means, if we have large amount of data, we need to send a great number of CAN frames and it takes more time than using Ethernet frame which we talk about in the next section.

2.5.2 Ethernet

Ethernet, a data link layer protocol which is most widely used in LAN technology in homes, offices, and factories, is increasingly utilized in the automotive world in the form of Automotive Ethernet. Vehicles now routinely accommodate multiple cameras, on-board diagnostics, advanced driver assistance systems (ADAS), infotainment systems, and in-dash displays. With all the added hardware and software comes a massive demand for bandwidth.

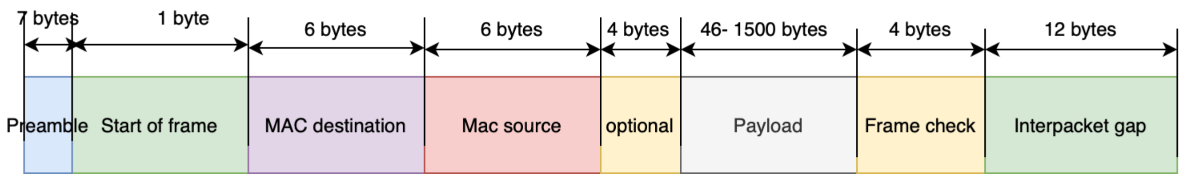


Figure 2.5: Ethernet Frame Format

Ethernet frame contains Preamble, Start of frame (SFD), Destination MAC address, Source MAC address, EtherType, Payload, Frame checks, etc as shown in Fig. 2.5. Payload meaning field which contains data to be transmit is able to accommodate up to 1500 bytes which is much higher than CAN bus. Meanwhile, the speed of the Ethernet frame reaches to 100Mbits/s in industry which means sending same amount of data, using Ethernet frame with higher capacity and speed than CAN bus is much faster.

2.6 XCP (Universal Measurement and Calibration Protocol)

XCP implies 'Universal Measurement and Calibration Protocol' which is an ASAM standard for calibrating parameters and measuring signals between ECUs and the development tool. The idea is to interconnect calibration systems with the ECUs [17]. It is mainly used for development and testing, by being able to retrieve and send information, and thus examine how an ECU behaves in different situations. As a replacement of CCP, XCP is developed to implement read and write access to internal ECUs data via variable and changeable transport layers.

AUTOSAR's definition of XCP is as follows: "XCP was designed according to the following principles: minimal slave resource consumption (RAM, ROM, runtime), efficient communication, simple slave implementation" [18]. The ECU is the slave and the measurement and calibration tool is the master. Based on the physical connection in between, master and slave communicate in the XCP standard and one master can contact many slaves concurrently as shown in Fig. 2.6.

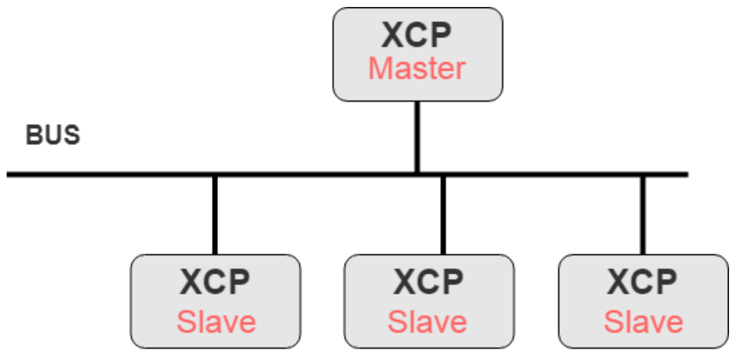


Figure 2.6: An XCP Master communicate with multiple slaves concurrently

A significant functionality of XCP is that XCP implements read and write access to the slave memory. Read access allows measurement of parameters from RAM, and write access enables calibration of parameters in RAM. Furthermore, acquiring the measured values from RAM is synchronous to the events in the ECU [10].

XCP is subdivided into two layers as mentioned before:a transport layer and a protocol layer. XCP data in the protocol layer package is embedded in the frame of the transport layer. XCP can both work over CAN and Ethernet.

XCP came as the successor of CCP, because the ECUs started to support more networks and hence the interface was expected to have more compatibility to different transport methods. Furthermore, the higher demands of throughput was also desired which was greater than CCP could provide [19].

2.6.1 XCP Protocol Layers

XCP data is generated in the protocol layer transmitted between master and slave. The structure of XCP message contains three parts: the XCP header, XCP packet and the XCP Tail as shown in Fig. 2.7. The contents of the XCP header and the XCP tail are based on type of transport layer used.

Formed independent of the transport layer, XCP packet consists of identification field, timestamp field and current data field. Identification field is used to identify the packet which decides the function of the packet. Timestamp field supplies the time information of measured data. Data field stores the data sent between slave and master.

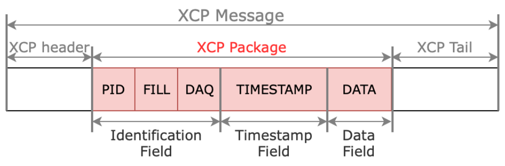


Figure 2.7: XCP message structure

Compared to CCP communication between the master and slave, XCP communication is also divided into two ways: one way is via CTOs (Command Transfer Objects) and one way is via DTOs (Data Transfer Objects) as shown in Fig. 2.8.

Within CTOs, command is sent from master to slave. When slave receives a command from the master, slave should react with RES (response) or ERR (error) as a response to the master. Contact between the master and the slave is initiated in this way. As to other messages in CTOs, EV (Event package) and SERV (Service Request Packet) are sent [20].

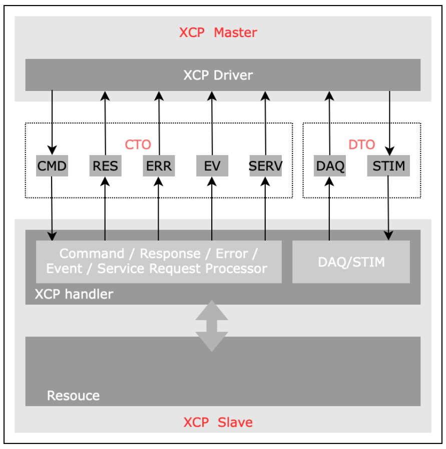


Figure 2.8: XCP Communication

For exchanging synchronous measurement and calibration data, DTOs consist of DAQ (Data AcQuisiton) and STIM (STIMulation). DAQ is the path for measured data sent from slave to master which is synchronous to internal events. Triggering of measurement data acquisition and transmission is controlled by events in the ECU. STIM is used for sending data from master to slave in the opposite direction [20].

2.6.2 XCP Transport Layer

XCP is able to support multiple transport layers: CAN, FlexRay, Ethernet and USB as shown in Fig. 2.9. XCP over different transportation layers are used for various requirements of speed, load, safety and the hardware restriction. For example, CAN network has a upper bound of 1 Mbit transfer rate which is merely possible with no other bus load. Data transmission above CAN bus is limited to 8 useful bytes.[16].

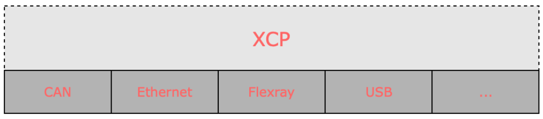


Figure 2.9: XCP layers

CAN bus is a serial communication bus used for communications between micro-controllers. In the case of XCP over CAN, Communication initiation between master and slave need command and response sent thus they take up the first useful byte of CAN. This means that only seven bytes are available per CAN message for each data transmission,but it allows for transfer of data without identifier by using the CAN packet identifier. XCP message over CAN is shown as Fig. 2.10.

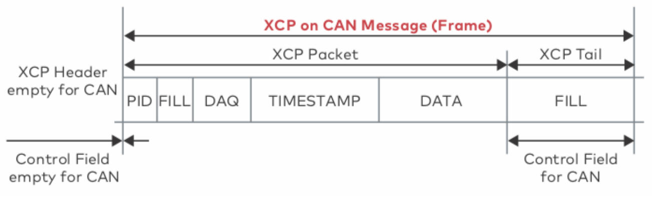


Figure 2.10: XCP message on CAN bus

XCP over Ethernet can be used for both TCP/IP or UDP/IP. The difference between TCP and UDP is that TCP can detect the loss of a packet with handshake and is responsible to resend missing packets. UDP ignores the package loss.

The structure of XCP message over Ethernet shown in Fig. 2.11 is different from the structure of XCP over CAN. The header consists of a control field with LEN (length of the XCP packet) and the CTR (counter for detecting packet loss only used in TCP). As a single Ethernet packet is able to contain multiple XCP packets but an XCP packet may not take up all the UDP/IP packet, thus the XCP tail over the Ethernet is empty.

Within the initiation of slave-master communication by XCP over the Ethernet, the response of slave contains: IP address, port number, TCP, UDP or both, information on the status (connected, unconnected).

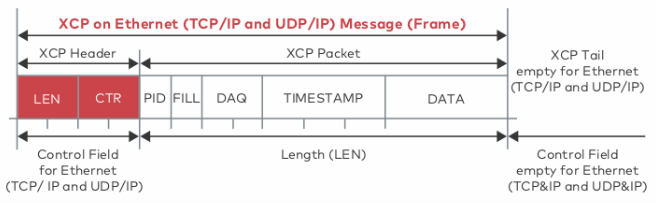

XCP on Ethernet (TCP/IP and UDP/IP) Message (Frame)
XCP Header
XCP Packet
XCP Tail empty for Ethernet (TCP/IP and UDP/IP)
LEN
CTR
PID
FILL
DAQ
TIMESTAMP
DATA
Control Field for Ethernet (TCP/ IP and UDP/IP)
Length (LEN)
Control Field empty for Ethernet (TCP&IP and UDP&IP)

Figure 2.11: XCP message over Ethernet

2.6.3 Polling vs DAQ (Data AcQuisition)

The most intuitive way to read memory contents of ECU is via polling which is trivial and straight-forward to implement. Basically, the idea is to ask for the measurement in a periodic manner initiated by the master. However, for every cyclic request, we need to issue a command to measure from the master and a response from the slave. That means for every measured value, the traffic in the stack is two messages. The extra generated overhead clogs the network twice for each measured variable. Bandwidth optimization, therefore is a problem [10]. The other drawback is that the acquisition time is not synchronized with the ECU tasks [21]. Due to the aforementioned reasons, we have chosen to implement synchronous DAQ in our project.

XCP enables to use and configure DAQ lists. They include tables that are called ODT (Object Descriptor Table). These offer to transmit vast amounts of data in a continuous manner with the mentioned advantages [17]. A DAQ list is a set of ODTs. Each entry in a DAQ list points to a memory element with specified address and length. DAQ lists are distinguished by different PIDs appended to the packets. Different DAQ lists can be configured in order to cover different events [10]. The address and the object length are the indicators to identify a measured object. The message is transmitted as a DTO.

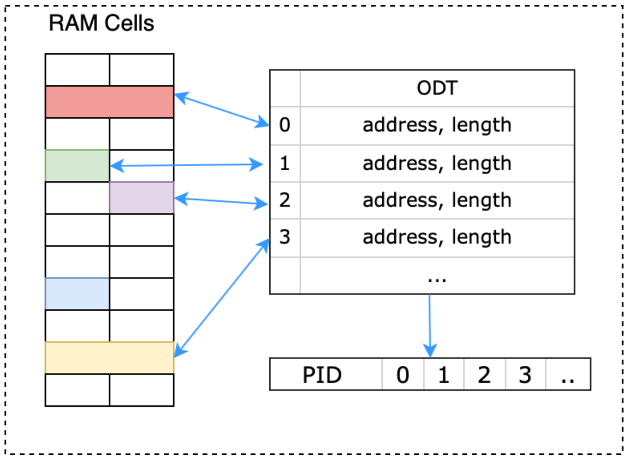


Figure 2.12: ODT to RAM mapping

To clarify how DAQ tackles the problems listed above, the features of DAQ should be investigated. The correlation of the measured values are achieved through binding the acquisition of the measurements to the events triggered in the ECU. The measured values are stalled and not transferred till all the computations have been finished. For reducing the load on the network, there are two phases called configuration and the transfer phase. In the configuration stage, Master states the values to be measured and the transfer stage only involves Slave dispatching the requested addresses [10].

XCP Master can list the desired parameters in the lists called DAQ lists. The events correspond to the measured variables. After the initial configuration is delivered to the Slave, the Slave fetches the addresses and the measured values are buffered.

It sorts out both problems that occur in polling: bandwidth is used efficiently, because the Master no longer needs to poll each value during the measurement and the values are synchronized with the Slave.

This methodology is specifically tailored for sending periodic events/measured values.

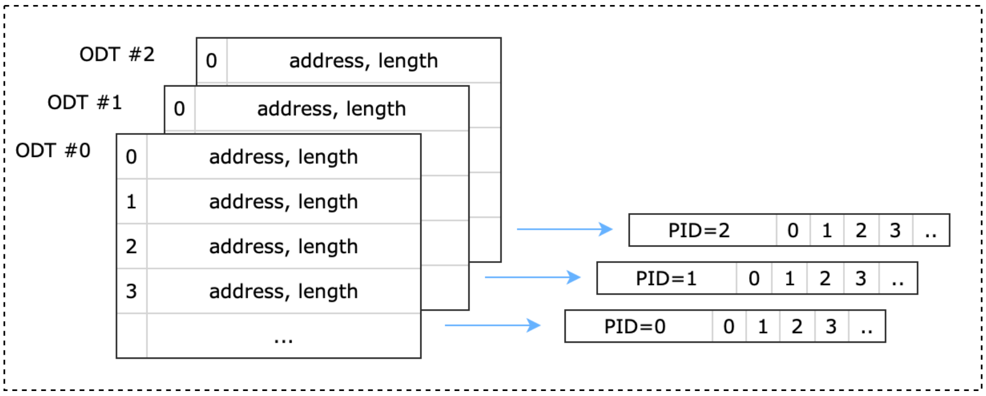


Figure 2.13: DAQ list and ODT configurations

2.7 Software Components Platform

2.7.1 AUTOSAR

Electrics and electronics (EE) is a pivotal part of the current automotive industry. The approach to conduct the development within the sector has been transformed over the years [18].

Contemporary development process within the automotive industry is replaced by function-driven process. According to AUTOSAR, "Engineering does not only aim at optimizing single components but optimizing on system level. This requires an open architecture as well as scalable and exchangeable software modules" [22]. Hence, it aims to create a joint environment of software development with OEMs, individual companies so as not to clog the industry with proprietary solutions. The main idea is also described as the re-using the software components so that the increased complexity can be governed in the future [22].

Habitually, the way to develop EE in vehicles is to have one unit for every service. Building XCP over Ethernet using AUTOSAR is not only because the current ECUs in Volvo Powertrain is settled on AUTOSAR, but also AUTOSAR has very clear hierarchy. Modules in side each hierarchy are having independent functionalities. Building XCP over Ethernet in AUTOSAR just needs to set functioning modules and activate corresponding interfaces and drivers making sure not affecting the orginal process of ECUs.

2.7.2 AUTOSAR Layer Infrastructure

The AUTOSAR platform has three main layers, with each layer having different functionalities. The bottom layer of the AUTOSAR platform is Basic Software Layer (BSW) which is connected to the Microcontroller Layer directly and enables the communication between MCU with the Runtime Environment (RTE). As shown of the Fig. 2.14, the platform consists of a number of modules which are used by application layer. The second layer is the Runtime Environment which is responsible for mapping the components in the application layer with the modules in basic software layer. The third layer is application layer which includes software components

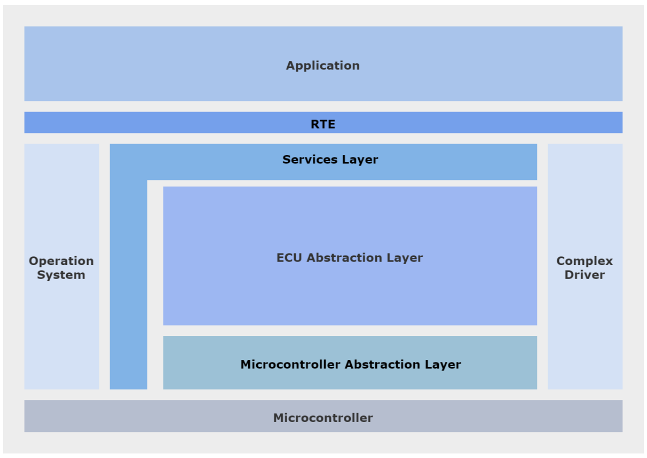


Figure 2.14: AUTOSAR Architecture

providing functionalities to the vehicle system.

The basic software layer which is the only layer having access to the hardware consists of a number of modules. Each module satisfies different requirements of the AUTOSAR system and divided into 3 sub-layers: Micro Controller Abstraction Layer (MCAL), ECU Abstraction Layer, Services Layer.

Looking at the Fig. 2.15, the stacks that have been implemented are placed onto the AUTOSAR standard.

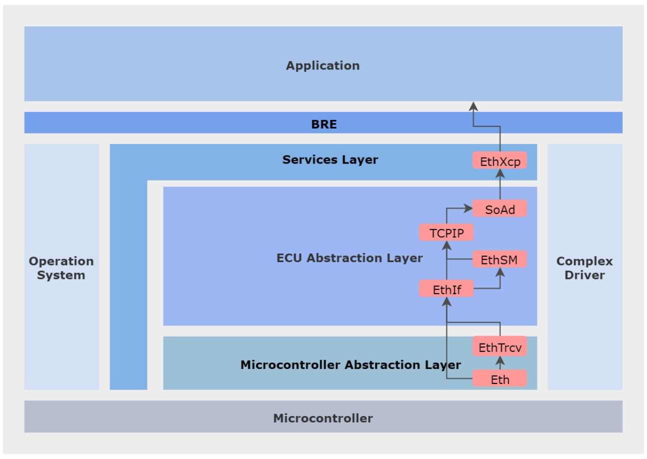


Figure 2.15: Component Stack for XCP over Ethernet

2.8 Component stack for XCP over Ethernet

Within the basic software layer of XCP over Ethernet, 7 blocks are used for the component stack. They are Ethernet Driver, Ethernet Transceiver, Ethernet Interface, Ethernet State Manager, TCPIP, Socket adapter and XCP as shown in Fig. 2.16.

The ETH (Ethernet Driver) provides hardware independent access to configure and control Ethernet Controllers integrated in the microcontroller or connected to

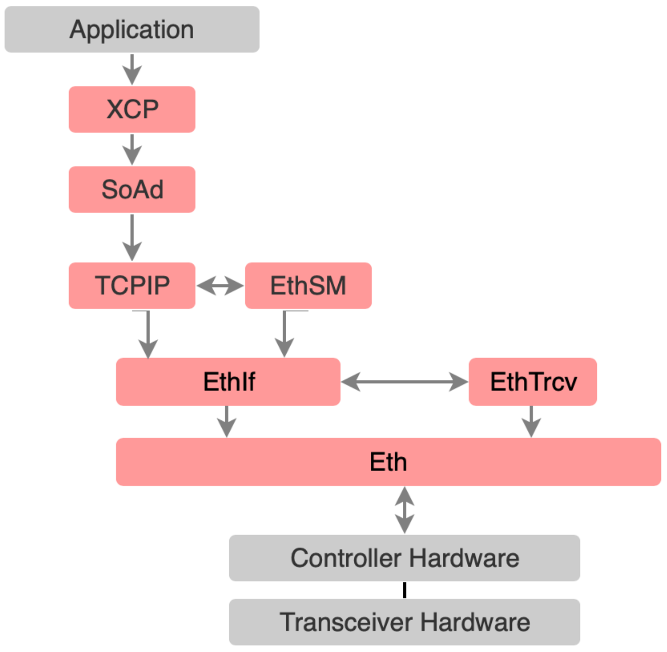


Figure 2.16: XCP over Ethernet Stack

it via an external interface like for example SPI. In this module, we have to ensure the Ethernet drivers are working and pins are set correctly.

The EthTrcv (Ethernet Transceiver Driver) provides hardware independent access to control connected transceivers in a generic way. It offers the functionality to control the mode of operation of connected transceivers as well as to determine their current state. The transceiver itself is a hardware device, which mainly transforms the logical I/0 signals of the Ethernet Controller to the bus compliant electrical levels, currents and timings.

The Ethernet Interface (EthIf) provides hardware independent access to control connected Ethernet Controller Drivers and Ethernet Transceiver Drivers in a generic way. Ethernet frames are sent and received through EthIf as well as it is an interface for MCAL to connect to higher layers.

The EthSM realizes a software layer between Ethernet Interface (EthIf) and Communication Manager (ComM). It handles the start-up and shutdown of the communication of an Ethernet network. It works as a state machine for Ethernet Communication.

The TCPIP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. Here is a block which provides access to a socket interface based on the Internet protocol (IPv4 and IPv6) over Ethernet.

The Socket Adaptor provides communication between PDU based communication and socket based communication via TcpIp. It supports the TCP and UDP sockets over lower module TCPIP.

XCP is a higher level protocol used for communication between a measurement and calibration system and an electronic control unit. It can be connected to application directly. XCP layer here supports the ASAM XCP 1.1 Specification.

3
Problem Description

The goal of this thesis work is to implement a new protocol according to the thesis' specification i.e. to provide at least the features Volvo had in the CCP via the XCP implementation. We then do some benchmarking and compare the throughput and data rate of the new implementation to the default, existing CCP implementation. We will also investigate how performance is investigated and defined in the sentiments of the network research.

The described problem can be depicted as below in Fig. 3.1:

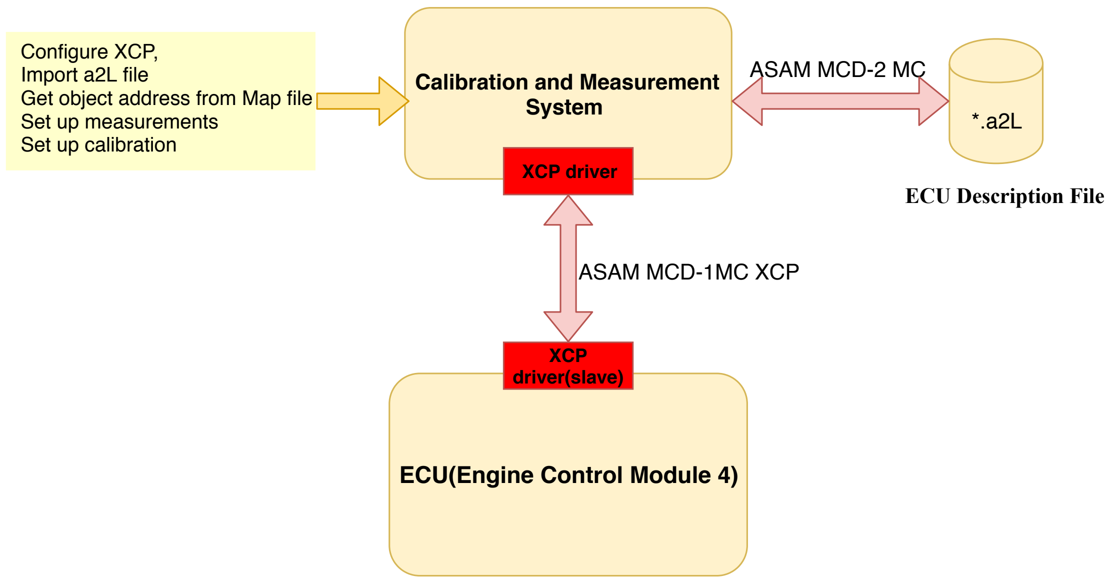


Figure 3.1: Overiew of the XCP over Ethernet system

XCP as a whole is quite extensive to implement and hence the following features have been selected to be implemented: calibration and measurement of ECU parameters over XCP. Following that, metrics of performance evaluation are efficiency and bandwidth are also studied.

4
Tools

4.1 Software Tools

As it is been laid out, tremendous help was granted from other software vendors who provide key solutions in the use case of the XCP project. One of such companies that was used throughout the course of the thesis was Vector Group. Vector serve many tools which address the common problems within the automotive industry. In the course of the thesis numerous Vector tools were used. They are DaVinci Configurator, CANape, CANalyzer and lastly Ethernet/XCP adapter VN 5610A.

4.2 DaVinci Configurator

DaVinci Configurator was a pivotal part of the project and as it will be discussed later, one of the significant contributors to the handful issues the thesis workers had. A snapshot of the tool can be observed in Fig. 4.1. According to Vector, DaVinci is: "... the central tool for configuring, validating and generating the basic software (BSW) and the runtime environment (RTE) of an AUTOSAR ECU" [23]. It is used for a similar purpose in Volvo Group: to generate the BSW and to interconnect the network stack, which is thoroughly explained in the Background section.

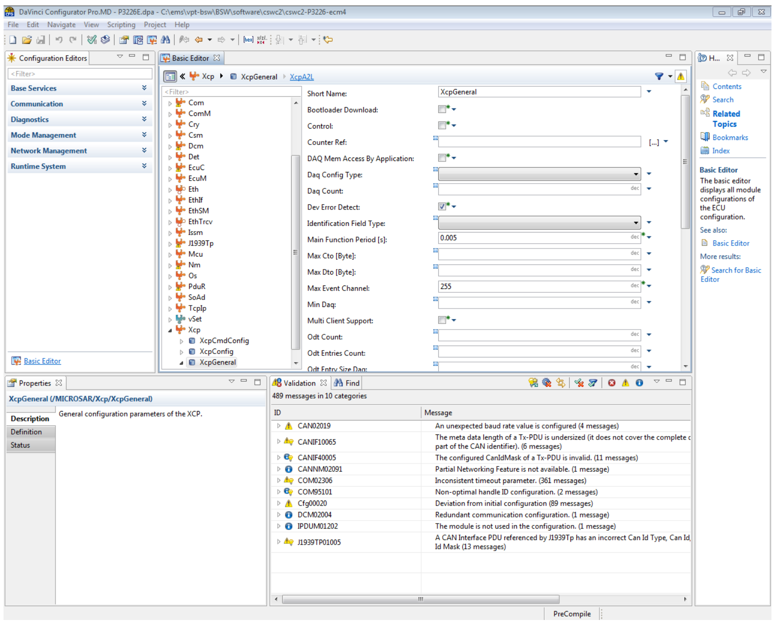
According to the component stack shown in Fig. 2.16, 7 blocks (ETH, EthTrcv, EthIf, EthSM, TCPIP, SoAd and XCP) are added as mentioned before.

The Ethernet Driver is connected to the hardware layer directly. In the whole system, it functions for the initialization, activation and deactivation of the Ethernet controller. In addition, it is used for transmission and reception of Ethernet frames. MAC address and the size of the Ethernet frame should be set correctly in this module as shown in figure.

The Ethernet Transceiver module is connected to the hardware through Ethernet Driver layer. It masters the initialization of the Ethernet transceiver and setting and getting the Transceiver mode.

The Ethernet Interface module mainly manage the mode of Ethernet driver and Ethernet Transceiver Driver. It is also responsible for transmission and reception of Ethernet frames.

The Ethernet State Manager is a state machine for transmitting Ethernet Frame. It handles the start-up and shutdown of the communication of an Ethernet network.



Basic Editor

The basic editor displays all module configurations of the ECU configuration.

Short Name:	XcpGeneral
Bootloader Download:	☐
Control:	☐
Counter Ref:	
DAQ Mem Access By Application:	☐
Daq Config Type:	
Daq Count:	
Dev Error Detect:	✓
Identification Field Type:	
Main Function Period [s]:	0.005
Max Cto [Byte]:	
Max Dto [Byte]:	
Max Event Channel:	255
Min Daq:	
Multi Client Support:	☐
Odt Count:	
Odt Entries Count:	
Odt Entry Size Daq:	

XcpGeneral (/MICROSAR/Xcp/XcpGeneral)

General configuration parameters of the XCP.

489 messages in 10 categories

ID	Message
CAN02019	An unexpected baud rate value is configured (4 messages)
CANIF10065	The meta data length of a Tx-PDU is undersized (it does not cover the complete c part of the CAN identifier). (6 messages)
CANIF40005	The configured CanIdMask of a Tx-PDU is invalid. (11 messages)
CANNM02091	Partial Networking Feature is not available. (1 message)
COM02306	Inconsistent timeout parameter. (361 messages)
COM95101	Non-optimal handle ID configuration. (2 messages)
Cfg00020	Deviation from initial configuration (89 messages)
DCM02004	Redundant communication configuration. (1 message)
IPDUM01202	The module is not used in the configuration. (1 message)
J1939TP01005	A CAN Interface PDU referenced by J1939Tp has an incorrect Can Id Type, Can Id, Id Mask (13 messages)

Figure 4.1: DaVinci configuration of the network stack

4.3 CANape

CANape is a calibration tool for ECUs. Fig. 4.2 depicts the program window. The communication between CANape and ECU was done with XCP and interface with VN5610. The user can identify an XCP device with Ethernet and configure A2L to track/change parameters via plots and tables on CANape [24].

The internal data measured by CANape can be accessed via ASAM standardized protocols such as CCP or XCP.

There are three main windows that are addressed. It is fruitful to identify them in the CANape tool. They are the Trace Window, Data Window and lastly the Write Window. The Trace Window is much like an output window, where the verbose output commands and other details are listed. The Data Window displays the data which are added to the watch window. The engineer, then, can proceed to query and fetch those signals. Finally, the Write Window is the window that shows the time stamps of Tool state changes, error and warning logs.

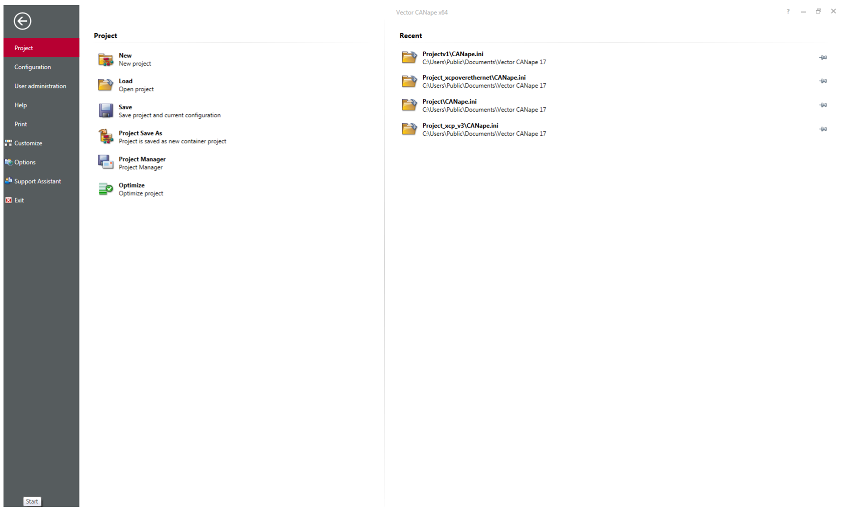
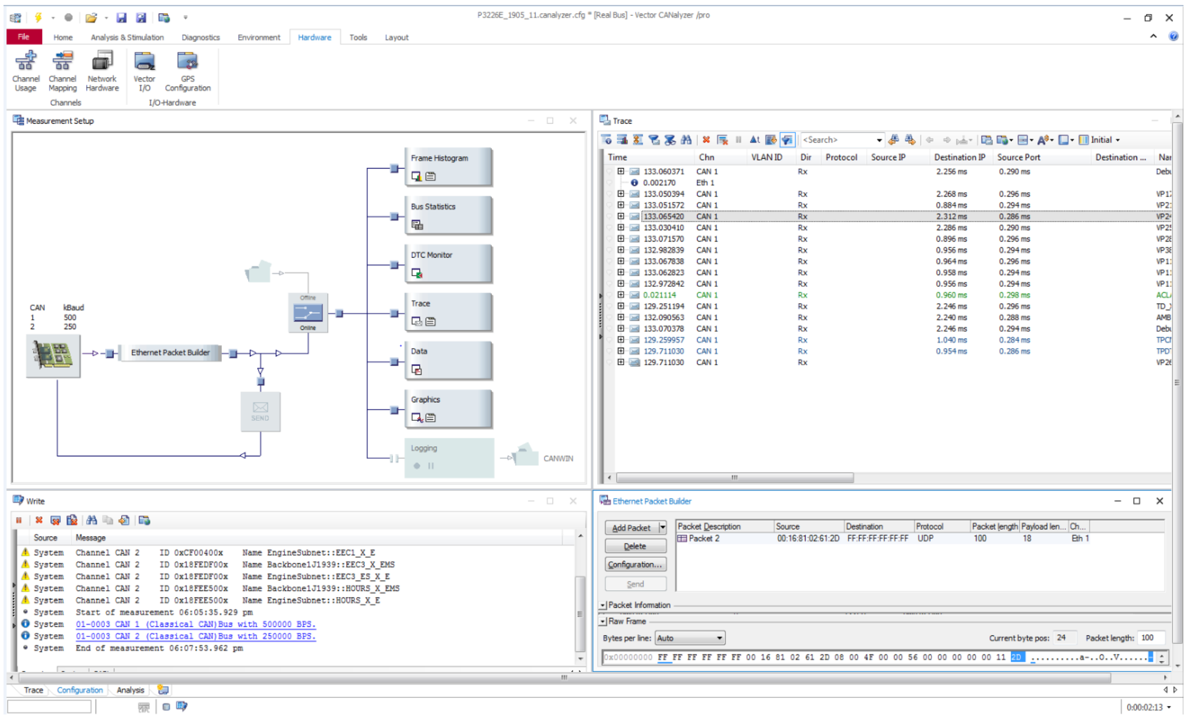

Vector CANape x64
Project
Configuration
User administration
Help
Print
Customize
Options
Support Assistant
Exit
Start
Project
New
New project
Load
Open project
Save
Save project and current configuration
Project Save As
Project is saved as new container project
Project Manager
Project Manager
Optimize
Optimize project
Recent
Projectv1\CANape.ini
C:\Users\Public\Documents\Vector CANape 17
Project_xcpoverethernet\CANape.ini
C:\Users\Public\Documents\Vector CANape 17
Project\CANape.ini
C:\Users\Public\Documents\Vector CANape 17
Project_xcp_v3\CANape.ini
C:\Users\Public\Documents\Vector CANape 17

Figure 4.2: CANape project creation

4.4 CANalyzer

CANalyzer was mainly used to send ETH packets with the Ethernet Packet Builder functionality. The packet creation process can be observed in Fig. 4.3. The Packet Builder enables user to form an ETH packet with a source/destination address and an upper layer of choice. Hence, it is possible to send ETH frames to a specified destination. The test was to observe whether the ECU responded to the ETH frames. It is also possible to analyze and stimulate the network communication [25].



Time	Chn	VLAN ID	Dir	Protocol	Source IP	Destination IP	Source Port	Destination ...	Na
133.060371	CAN 1		Rx			2.256 ms	0.290 ms		Debı
0.002170	Eth 1								
133.050394	CAN 1		Rx			2.268 ms	0.296 ms		VP1
133.051572	CAN 1		Rx			0.884 ms	0.294 ms		VP2
133.065420	CAN 1		Rx			2.312 ms	0.286 ms		VP2
133.030410	CAN 1		Rx			2.286 ms	0.290 ms		VP2
133.071570	CAN 1		Rx			0.896 ms	0.296 ms		VP2
132.982839	CAN 1		Rx			0.956 ms	0.294 ms		VP3
133.067838	CAN 1		Rx			0.964 ms	0.296 ms		VP1
133.062823	CAN 1		Rx			0.958 ms	0.294 ms		VP1
132.972842	CAN 1		Rx			0.956 ms	0.294 ms		VP1
0.021114	CAN 1		Rx			0.960 ms	0.298 ms		ACL
129.251194	CAN 1		Rx			2.246 ms	0.296 ms		TD_
132.090563	CAN 1		Rx			2.240 ms	0.288 ms		AMB
133.070378	CAN 1		Rx			2.246 ms	0.294 ms		Debı
129.259957	CAN 1		Rx			1.040 ms	0.284 ms		TPC
129.711030	CAN 1		Rx			0.954 ms	0.286 ms		TPD
129.711030	CAN 1		Rx						VP2

```
Source   Message
System   Channel CAN 2    ID 0xCF00400x    Name EngineSubnet::EEC1_X_E
System   Channel CAN 2    ID 0x18FEDF00x   Name Backbone1J1939::EEC3_X_EMS
System   Channel CAN 2    ID 0x18FEDF00x   Name EngineSubnet::EEC3_ES_X_E
System   Channel CAN 2    ID 0x18FEE500x   Name Backbone1J1939::HOURS_X_EMS
System   Channel CAN 2    ID 0x18FEE500x   Name EngineSubnet::HOURS_X_E
System   Start of measurement 06:05:35.929 pm
System   01-0003 CAN 1 (Classical CAN)Bus with 500000 BPS.
System   01-0003 CAN 2 (Classical CAN)Bus with 250000 BPS.
System   End of measurement 06:07:53.962 pm
```

Packet Description	Source	Destination	Protocol	Packet length	Payload len...	Ch...
Packet 2	00:16:81:02:61:2D	FF:FF:FF:FF:FF:FF	UDP	100	18	Eth 1

Bytes per line: Auto Current byte pos: 24 Packet length: 100

```
0x00000000 FF FF FF FF FF FF 00 16 81 02 61 2D 08 00 4F 00 00 56 00 00 00 00 00 11 2D ..........a-..O..V......-
```

Figure 4.3: CANalyzer packet creation

4.5 Trace32: T32

TRACE32 is a multipurpose debugger tool from Lauterbach GmbH. According to the product specification, T32 is a high performance MCU debug, trace and logic analyzer tool which support over 60 processor architectures. The debugger will supervise and control the CPU execution such as writing in the memory on the fly [26]. Real time monitoring of the target variables are also supported, as the CPU load was depicted by T32 in the Result section of the report.

The tool helps with debugging and monitoring of multicore systems. Furthermore, it allows the embedded engineer to step into the APIs line-by-line and setting up breakpoints within the source code. The embedded software specialist can easily flash the MCU, monitor the flow of the flashed progrom and the memory locations of the MCU. Since the flashing was omitted for the sake of timely completion of the thesis, software download was sustained through T32 [27].

A PowerDebug JTAG Debugger probe is needed to access the ECU. This tool lets the T32 access the MCU within the ECU. It is convenient to start/stop the execution of the program and step in the function calls or skip through any API through the T32 functionality [26].

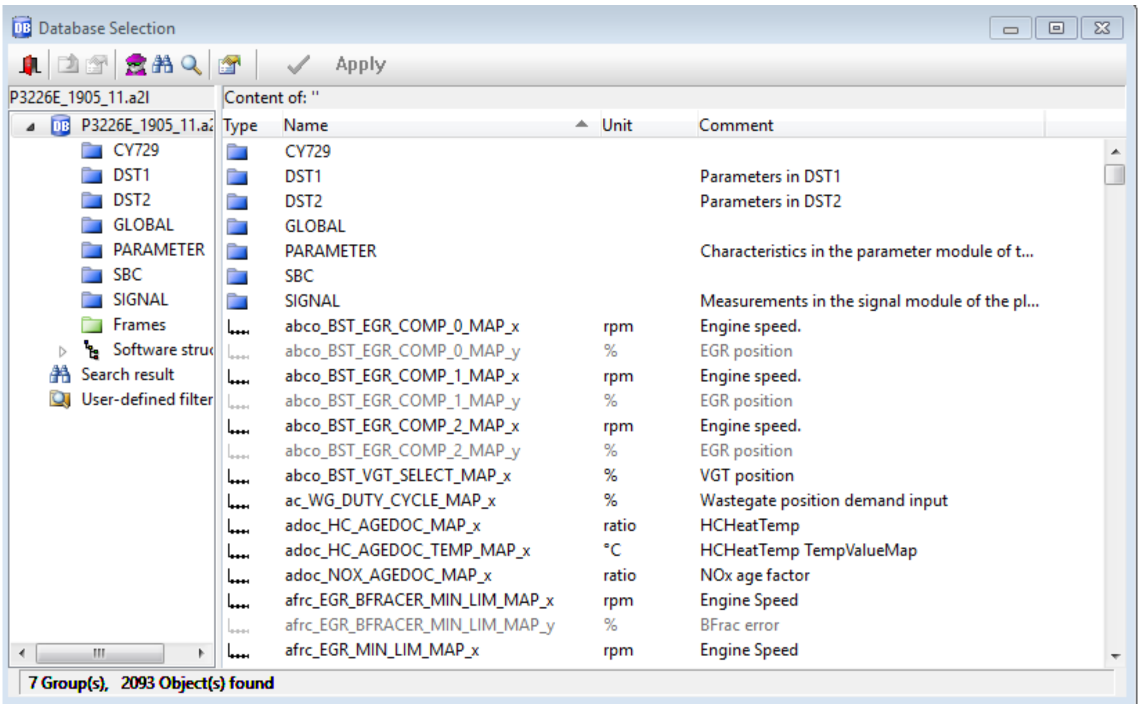
4.6 VISION Software

Although only used for benchmarking purposes, Vision is a data acquisition and calibration tool. Unfortunately, through the course of the thesis Vision had not supported Vector's Ethernet/CAN Interface, VN5610A, hence additional solutions had to be taken. Vision was only used for testing CCP and software download in CCP.

4.7 A2l file

The A2l file contains the integral part of the work that has been done. In the case of measurement and calibration, the user may try to access the addresses in ECU. An A2l file basically is a gigantic mapping file which contains symbolic names that correspond to the addresses. The file provides better accessibility and ease-of-use, hence the measurement/calibration engineer does not have to deal with the direct hex addresses of flash [10]. It creates a more manageable and familiar environment to work with.

At this point, CANape was used to parse the A2l file and get the correct logical namings of the addresses. Afterwards, the engineer just has to highlight the object by the name and the XCP Master would know the corresponding address, data type and the data size. Many of the parameters that are of interest to the calibration have also a minimum and maximum value which can be predefined in the A2l file. This especially is a very helpful safety-lock in the cases where hazardous results may occur, such as tuning the power output or engine torque. Of course for some other parameters, outlying ranges can be more forgiving, but in the case of engine speed, it is no longer a redundant field. Lastly, there is a set called parameter set within the A2l that defines the communication between the ECU and the measurement/calibration tool. All in all, the A2l consists of everything that the tool needs in order to connect and communicate with the ECU [10].



Type	Name	Unit	Comment
	CY729		
	DST1		Parameters in DST1
	DST2		Parameters in DST2
	GLOBAL		
	PARAMETER		Characteristics in the parameter module of t...
	SBC		
	SIGNAL		Measurements in the signal module of the pl...
	abco_BST_EGR_COMP_0_MAP_x	rpm	Engine speed.
	abco_BST_EGR_COMP_0_MAP_y	%	EGR position
	abco_BST_EGR_COMP_1_MAP_x	rpm	Engine speed.
	abco_BST_EGR_COMP_1_MAP_y	%	EGR position
	abco_BST_EGR_COMP_2_MAP_x	rpm	Engine speed.
	abco_BST_EGR_COMP_2_MAP_y	%	EGR position
	abco_BST_VGT_SELECT_MAP_x	%	VGT position
	ac_WG_DUTY_CYCLE_MAP_x	%	Wastegate position demand input
	adoc_HC_AGEDOC_MAP_x	ratio	HCHeatTemp
	adoc_HC_AGEDOC_TEMP_MAP_x	°C	HCHeatTemp TempValueMap
	adoc_NOX_AGEDOC_MAP_x	ratio	NOx age factor
	afrc_EGR_BFRACER_MIN_LIM_MAP_x	rpm	Engine Speed
	afrc_EGR_BFRACER_MIN_LIM_MAP_y	%	BFrac error
	afrc_EGR_MIN_LIM_MAP_x	rpm	Engine Speed

7 Group(s), 2093 Object(s) found

Figure 4.4: A2l parsing via CANape

Fig. 4.4 shows how the A2l file looks like on CANape. It is a parsed version where it is possible to see all the signals in the measurement and calibration tool. By choosing one of the measurement variables on the window, it is possible to read and write to the memory without having to decode the memory address mapping.

4.8 Hardware Tools

4.8.1 VN5610A - Ethernet/CAN Interface

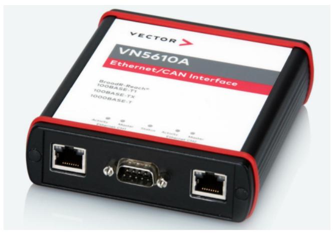


Figure 4.5: VN5610A Ethernet/CAN Interface

As can be seen in Fig. 4.5, VN5610A is a Vector hardware that works as a gateway between the ECU and the PC. It supports 10BASE-T and 100BASE-T1.

The device enables the monitoring and saving Ethernet data flows with precise time stamps [28]. In the thesis' use case, time stamping was not utilized.

The device can be tailored to fit many use cases such as bus analysis, diagnostics and calibration purposes with CANape.

5
Results

The realization of the setup of the problem description was done as shown in Fig. 5.1

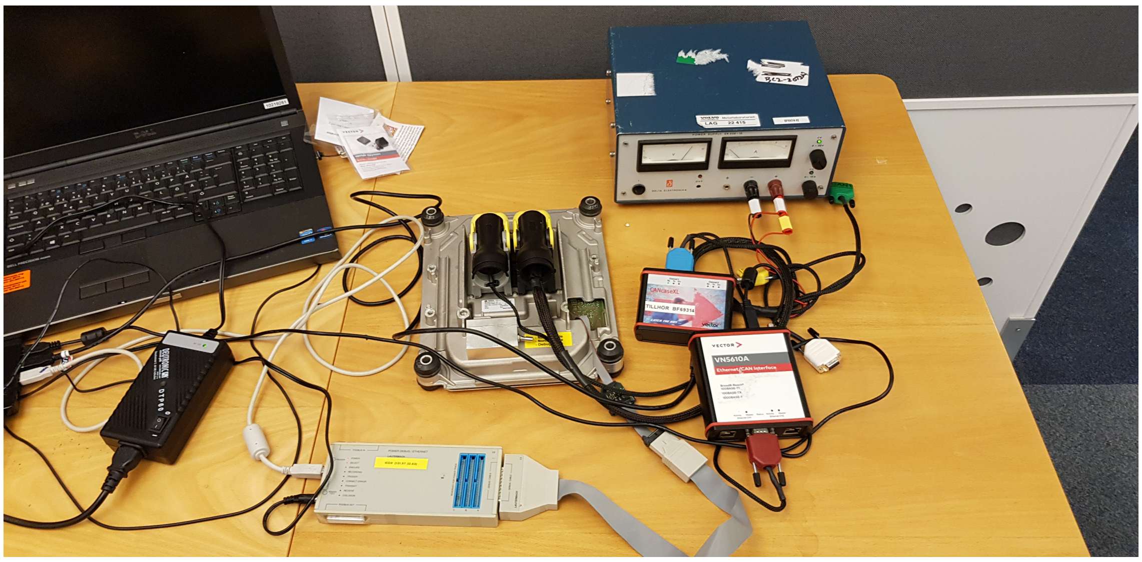
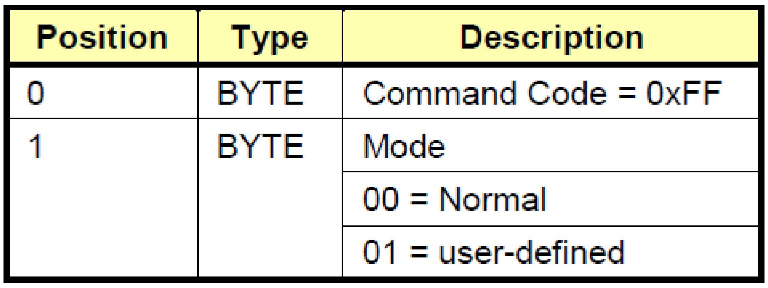


Figure 5.1: Real Implementation Connection

Establishing an XCP connection over Ethernet was the first achievement in the thesis work. If the ASAM's document on XCP is checked, the CONNECT command would yield as shown in Fig. 5.2.



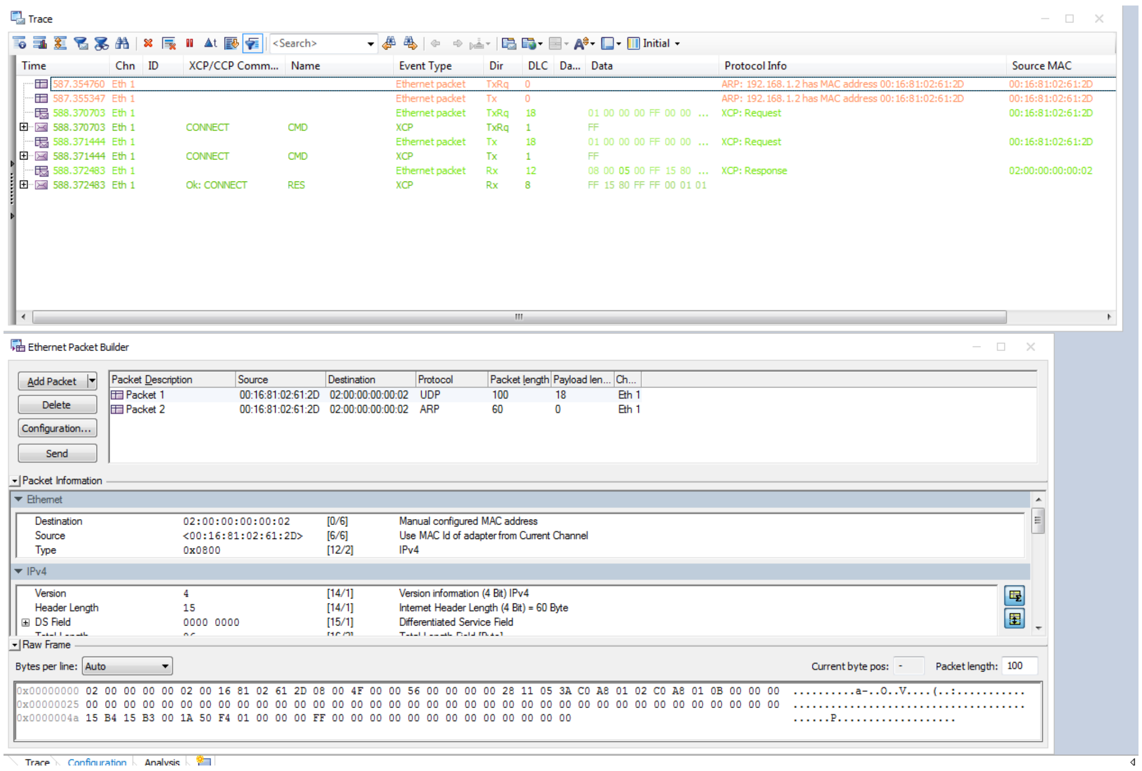
Position	Type	Description
0	BYTE	Command Code = 0xFF
1	BYTE	Mode
		00 = Normal
		01 = user-defined

Figure 5.2: Connect command in XCP [29]

In order to do that, the master needs to issue an XCP packet containing the payload FF.

For the first initial result, the XCP communication was initiated via the tool Vector CANalyzer. The figure below depicts that the first messages that were transferred over the XCP network.

For the sake of testing, the master forms and sends two XCP packets. Due to the requirement of the protocol, the first step that the master is obliged to send a CONNECT command. Hence, these two packets are shown in Fig. 5.3.



Time	Chn	ID	XCP/CCP Comm...	Name	Event Type	Dir	DLC	Da...	Data	Protocol Info	Source MAC
587.354760	Eth 1				Ethernet packet	TxRq	0			ARP: 192.168.1.2 has MAC address 00:16:81:02:61:2D	00:16:81:02:61:2D
587.355347	Eth 1				Ethernet packet	Tx	0			ARP: 192.168.1.2 has MAC address 00:16:81:02:61:2D	00:16:81:02:61:2D
588.370703	Eth 1				Ethernet packet	TxRq	18		01 00 00 00 FF 00 00 ...	XCP: Request	00:16:81:02:61:2D
588.370703	Eth 1		CONNECT	CMD	XCP	TxRq	1		FF		
588.371444	Eth 1				Ethernet packet	Tx	18		01 00 00 00 FF 00 00 ...	XCP: Request	00:16:81:02:61:2D
588.371444	Eth 1		CONNECT	CMD	XCP	Tx	1		FF		
588.372483	Eth 1				Ethernet packet	Rx	12		08 00 05 00 FF 15 80 ...	XCP: Response	02:00:00:00:00:02
588.372483	Eth 1		Ok: CONNECT	RES	XCP	Rx	8		FF 15 80 FF FF 00 01 01		

Packet Description	Source	Destination	Protocol	Packet length	Payload len...	Ch...
Packet 1	00:16:81:02:61:2D	02:00:00:00:00:02	UDP	100	18	Eth 1
Packet 2	00:16:81:02:61:2D	02:00:00:00:00:02	ARP	60	0	Eth 1

Ethernet

Destination	02:00:00:00:00:02	[0/6]	Manual configured MAC address
Source	<00:16:81:02:61:2D>	[6/6]	Use MAC Id of adapter from Current Channel
Type	0x0800	[12/2]	IPv4

IPv4

Version	4	[14/1]	Version information (4 Bit) IPv4
Header Length	15	[14/1]	Internet Header Length (4 Bit) = 60 Byte
DS Field	0000 0000	[15/1]	Differentiated Service Field

Bytes per line: Auto — Current byte pos: - — Packet length: 100

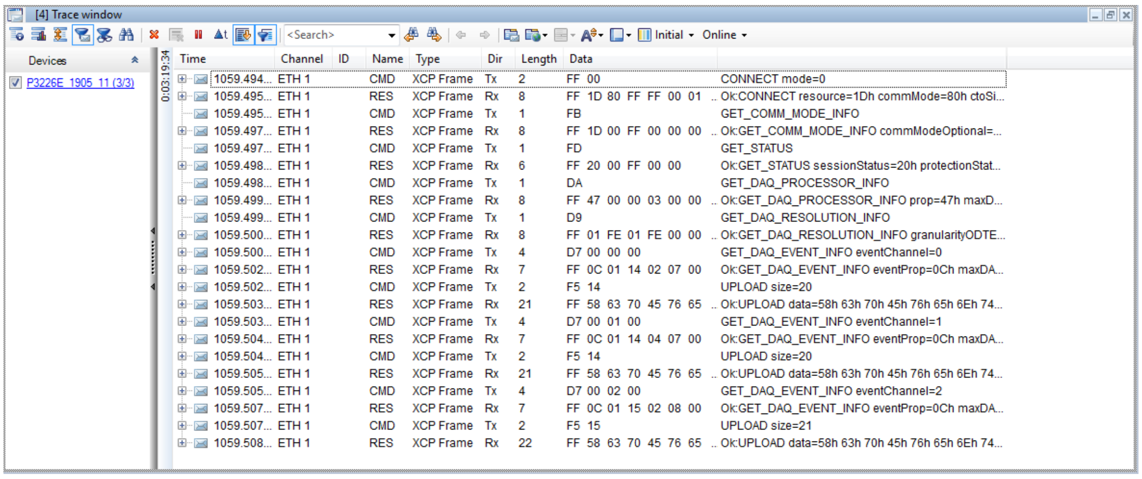
```
0x00000000 02 00 00 00 00 02 00 16 81 02 61 2D 08 00 4F 00 00 56 00 00 00 00 28 11 05 3A C0 A8 01 02 C0 A8 01 0B 00 00 00  ..........a-..O..V....(..:...........
0x00000025 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................................
0x0000004a 15 B4 15 B3 00 1A 50 F4 01 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00                                      ......P...................
```

Figure 5.3: Communication through XCP over Ethernet

Clearly, the slave (ECU) responds with an affirmative reply by returning OK to the connect command to the master. The destination MAC address is 02:00:00:00:00:02, while using the IPv4. Source port is 5556 and destination port is 5555. Once the slave responds with the reply OK, the XCP connection has been established until the Master sends a DISCONNECT command.

Despite of the fact that XCP communication between the ECU and the PC was established via CANalyzer, it was problematic to work with. The reasons for that were stemming from CANalyzer inherit properties. CANalyzer is originally an ECU network analyzer. It is not designed to be a measurement and calibration tool. At this point, Volvo Group acquired a license of a professional calibration/measurement tool called CANape.

After tweaking the settings according to the requirements of the project, the XCP communication was sustained through CANape shown in Fig. 5.4.



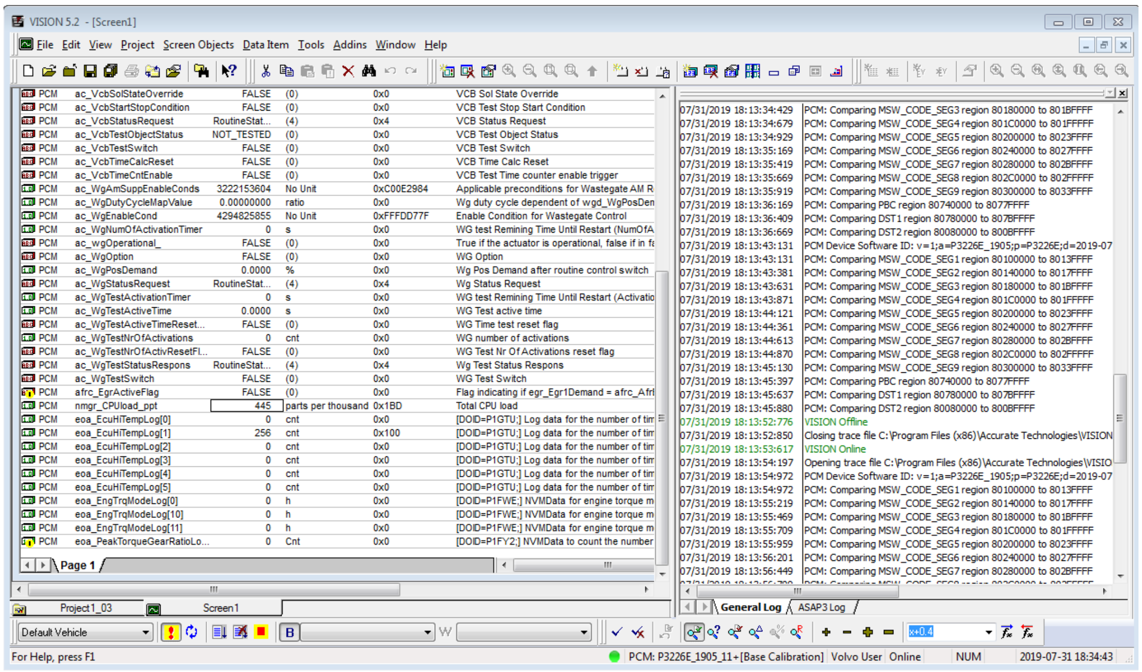
```
[4] Trace window
Devices: P3226E_1905_11 (3/3)

Time        Channel  ID  Name  Type       Dir  Length  Data
1059.494... ETH 1        CMD   XCP Frame  Tx   2       FF 00                 CONNECT mode=0
1059.495... ETH 1        RES   XCP Frame  Rx   8       FF 1D 80 FF FF 00 01 .. Ok:CONNECT resource=1Dh commMode=80h ctoSi...
1059.495... ETH 1        CMD   XCP Frame  Tx   1       FB                    GET_COMM_MODE_INFO
1059.497... ETH 1        RES   XCP Frame  Rx   8       FF 1D 00 FF 00 00 00 .. Ok:GET_COMM_MODE_INFO commModeOptional=...
1059.497... ETH 1        CMD   XCP Frame  Tx   1       FD                    GET_STATUS
1059.498... ETH 1        RES   XCP Frame  Rx   6       FF 20 00 FF 00 00     Ok:GET_STATUS sessionStatus=20h protectionStat...
1059.498... ETH 1        CMD   XCP Frame  Tx   1       DA                    GET_DAQ_PROCESSOR_INFO
1059.499... ETH 1        RES   XCP Frame  Rx   8       FF 47 00 00 03 00 00 .. Ok:GET_DAQ_PROCESSOR_INFO prop=47h maxD...
1059.499... ETH 1        CMD   XCP Frame  Tx   1       D9                    GET_DAQ_RESOLUTION_INFO
1059.500... ETH 1        RES   XCP Frame  Rx   8       FF 01 FE 01 FE 00 00 .. Ok:GET_DAQ_RESOLUTION_INFO granularityODTE...
1059.500... ETH 1        CMD   XCP Frame  Tx   4       D7 00 00 00           GET_DAQ_EVENT_INFO eventChannel=0
1059.502... ETH 1        RES   XCP Frame  Rx   7       FF 0C 01 14 02 07 00  Ok:GET_DAQ_EVENT_INFO eventProp=0Ch maxDA...
1059.502... ETH 1        CMD   XCP Frame  Tx   2       F5 14                 UPLOAD size=20
1059.503... ETH 1        RES   XCP Frame  Rx   21      FF 58 63 70 45 76 65 .. Ok:UPLOAD data=58h 63h 70h 45h 76h 65h 6Eh 74...
1059.503... ETH 1        CMD   XCP Frame  Tx   4       D7 00 01 00           GET_DAQ_EVENT_INFO eventChannel=1
1059.504... ETH 1        RES   XCP Frame  Rx   7       FF 0C 01 14 04 07 00  Ok:GET_DAQ_EVENT_INFO eventProp=0Ch maxDA...
1059.504... ETH 1        CMD   XCP Frame  Tx   2       F5 14                 UPLOAD size=20
1059.505... ETH 1        RES   XCP Frame  Rx   21      FF 58 63 70 45 76 65 .. Ok:UPLOAD data=58h 63h 70h 45h 76h 65h 6Eh 74...
1059.505... ETH 1        CMD   XCP Frame  Tx   4       D7 00 02 00           GET_DAQ_EVENT_INFO eventChannel=2
1059.507... ETH 1        RES   XCP Frame  Rx   7       FF 0C 01 15 02 08 00  Ok:GET_DAQ_EVENT_INFO eventProp=0Ch maxDA...
1059.507... ETH 1        CMD   XCP Frame  Tx   2       F5 15                 UPLOAD size=21
1059.508... ETH 1        RES   XCP Frame  Rx   22      FF 58 63 70 45 76 65 .. Ok:UPLOAD data=58h 63h 70h 45h 76h 65h 6Eh 74...
```

Figure 5.4: Calibration achieved through CANape

5.1 Asynchronous Measurement: Polling

One of the first implementations done over the XCP/Ethernet stack was to conduct polling. When the system is in the polling mode, once the XCP master sends the request command which contains the datum address to slave side, the slave responds one data information immediately. The advantage of this mode is that the master could get a fast reply from slave. Both VISION and CANape were used for polling.



```
VISION 5.2 - [Screen1]
File  Edit  View  Project  Screen Objects  Data Item  Tools  Addins  Window  Help

PCM  ac_VcbSolStateOverride          FALSE       (0)               0x0         VCB Sol State Override
PCM  ac_VcbStartStopCondition        FALSE       (0)               0x0         VCB Test Stop Start Condition
PCM  ac_VcbStatusRequest             RoutineStat... (4)            0x4         VCB Status Request
PCM  ac_VcbTestObjectStatus          NOT_TESTED  (0)               0x0         VCB Test Object Status
PCM  ac_VcbTestSwitch                FALSE       (0)               0x0         VCB Test Switch
PCM  ac_VcbTimeCalcReset             FALSE       (0)               0x0         VCB Time Calc Reset
PCM  ac_VcbTimeCntEnable             FALSE       (0)               0x0         VCB Test Time counter enable trigger
PCM  ac_WgAmSuppEnableConds          3222153604  No Unit           0xC00E2984  Applicable preconditions for Wastegate AM R
PCM  ac_WgDutyCycleMapValue          0.00000000  ratio             0x0         Wg duty cycle dependent of wgd_WgPosDen
PCM  ac_WgEnableCond                 4294825855  No Unit           0xFFFDD77F  Enable Condition for Wastegate Control
PCM  ac_WgNumOfActivationTimer       0           s                 0x0         WG test Remining Time Until Restart (NumOfA
PCM  ac_wgOperational_               FALSE       (0)               0x0         True if the actuator is operational, false if in fa
PCM  ac_WgOption                     FALSE       (0)               0x0         WG Option
PCM  ac_WgPosDemand                  0.0000      %                 0x0         Wg Pos Demand after routine control switch
PCM  ac_WgStatusRequest              RoutineStat... (4)            0x4         Wg Status Request
PCM  ac_WgTestActivationTimer        0           s                 0x0         WG test Remining Time Until Restart (Activatio
PCM  ac_WgTestActiveTime             0.0000      s                 0x0         WG Test active time
PCM  ac_WgTestActiveTimeReset...     FALSE       (0)               0x0         WG Time test reset flag
PCM  ac_WgTestNrOfActivations        0           cnt               0x0         WG number of activations
PCM  ac_WgTestNrOfActivResetFl...    FALSE       (0)               0x0         WG Test Nr Of Activations reset flag
PCM  ac_WgTestStatusRespons          RoutineStat... (4)            0x4         Wg Test Status Respons
PCM  ac_WgTestSwitch                 FALSE       (0)               0x0         WG Test Switch
PCM  afrc_EgrActiveFlag              FALSE       (0)               0x0         Flag indicating if egr_Egr1Demand = afrc_Afrl
PCM  nmgr_CPUload_ppt                445         parts per thousand 0x1BD      Total CPU load
PCM  eoa_EcuHiTempLog[0]             0           cnt               0x0         [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EcuHiTempLog[1]             256         cnt               0x100       [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EcuHiTempLog[2]             0           cnt               0x0         [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EcuHiTempLog[3]             0           cnt               0x0         [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EcuHiTempLog[4]             0           cnt               0x0         [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EcuHiTempLog[5]             0           cnt               0x0         [DOID=P1GTU;] Log data for the number of tim
PCM  eoa_EngTrqModeLog[0]            0           h                 0x0         [DOID=P1FWE;] NVMData for engine torque m
PCM  eoa_EngTrqModeLog[10]           0           h                 0x0         [DOID=P1FWE;] NVMData for engine torque m
PCM  eoa_EngTrqModeLog[11]           0           h                 0x0         [DOID=P1FWE;] NVMData for engine torque m
PCM  eoa_PeakTorqueGearRatioLo...    0           Cnt               0x0         [DOID=P1FY2;] NVMData to count the number

Page 1
Project1_03    Screen1

07/31/2019 18:13:34:429  PCM: Comparing MSW_CODE_SEG3 region 80180000 to 801BFFFF
07/31/2019 18:13:34:679  PCM: Comparing MSW_CODE_SEG4 region 801C0000 to 801FFFFF
07/31/2019 18:13:34:929  PCM: Comparing MSW_CODE_SEG5 region 80200000 to 8023FFFF
07/31/2019 18:13:35:169  PCM: Comparing MSW_CODE_SEG6 region 80240000 to 8027FFFF
07/31/2019 18:13:35:419  PCM: Comparing MSW_CODE_SEG7 region 80280000 to 802BFFFF
07/31/2019 18:13:35:669  PCM: Comparing MSW_CODE_SEG8 region 802C0000 to 802FFFFF
07/31/2019 18:13:35:919  PCM: Comparing MSW_CODE_SEG9 region 80300000 to 8033FFFF
07/31/2019 18:13:36:169  PCM: Comparing PBC region 80740000 to 8077FFFF
07/31/2019 18:13:36:409  PCM: Comparing DST1 region 80780000 to 807BFFFF
07/31/2019 18:13:36:669  PCM: Comparing DST2 region 80080000 to 800BFFFF
07/31/2019 18:13:43:131  PCM Device Software ID: v=1;a=P3226E_1905;p=P3226E;d=2019-07
07/31/2019 18:13:43:131  PCM: Comparing MSW_CODE_SEG1 region 80100000 to 8013FFFF
07/31/2019 18:13:43:381  PCM: Comparing MSW_CODE_SEG2 region 80140000 to 8017FFFF
07/31/2019 18:13:43:631  PCM: Comparing MSW_CODE_SEG3 region 80180000 to 801BFFFF
07/31/2019 18:13:43:871  PCM: Comparing MSW_CODE_SEG4 region 801C0000 to 801FFFFF
07/31/2019 18:13:44:121  PCM: Comparing MSW_CODE_SEG5 region 80200000 to 8023FFFF
07/31/2019 18:13:44:361  PCM: Comparing MSW_CODE_SEG6 region 80240000 to 8027FFFF
07/31/2019 18:13:44:613  PCM: Comparing MSW_CODE_SEG7 region 80280000 to 802BFFFF
07/31/2019 18:13:44:870  PCM: Comparing MSW_CODE_SEG8 region 802C0000 to 802FFFFF
07/31/2019 18:13:45:130  PCM: Comparing MSW_CODE_SEG9 region 80300000 to 8033FFFF
07/31/2019 18:13:45:397  PCM: Comparing PBC region 80740000 to 8077FFFF
07/31/2019 18:13:45:637  PCM: Comparing DST1 region 80780000 to 807BFFFF
07/31/2019 18:13:45:880  PCM: Comparing DST2 region 80080000 to 800BFFFF
07/31/2019 18:13:52:776  VISION Offline
07/31/2019 18:13:52:850  Closing trace file C:\Program Files (x86)\Accurate Technologies\VISION
07/31/2019 18:13:53:617  VISION Online
07/31/2019 18:13:54:197  Opening trace file C:\Program Files (x86)\Accurate Technologies\VISIO
07/31/2019 18:13:54:972  PCM Device Software ID: v=1;a=P3226E_1905;p=P3226E;d=2019-07
07/31/2019 18:13:54:972  PCM: Comparing MSW_CODE_SEG1 region 80100000 to 8013FFFF
07/31/2019 18:13:55:219  PCM: Comparing MSW_CODE_SEG2 region 80140000 to 8017FFFF
07/31/2019 18:13:55:469  PCM: Comparing MSW_CODE_SEG3 region 80180000 to 801BFFFF
07/31/2019 18:13:55:709  PCM: Comparing MSW_CODE_SEG4 region 801C0000 to 801FFFFF
07/31/2019 18:13:55:959  PCM: Comparing MSW_CODE_SEG5 region 80200000 to 8023FFFF
07/31/2019 18:13:56:201  PCM: Comparing MSW_CODE_SEG6 region 80240000 to 8027FFFF
07/31/2019 18:13:56:449  PCM: Comparing MSW_CODE_SEG7 region 80280000 to 802BFFFF

General Log   ASAP3 Log
Default Vehicle   x+0.4
For Help, press F1   PCM: P3226E_1905_11+[Base Calibration]  Volvo User  Online  NUM  2019-07-31 18:34:43
```

Figure 5.5: Polling 80 parameters via CCP depicted on VISION

Using VISION, only 80 to 82 parameters can be configured to be asynchronously measured. The size of the elements are problematic in CAN due to the inherited 8-byte payload restriction. In Fig. 5.5, the yellow warning signs indicate that the signal could not be read from the ECU. In Ethernet, however the maximum payload is 1500 bytes [30].

The issue is exacerbated, when adding more than 80 parameters to poll via CCP. All the new added signals get locked due to congestion in the communication bus. We are unsure, whether this is an implementation error or a protocol-induced threshold.

There was not such a locking limit in the case of XCP. However, of course the delay between fetched parameters were increasing, as more parameters have been added to the queue. As shown in Table 5.1, the command which requests for the data of CPU load is sent every 10ms to ECU. As the increment number of the parameters, there is heavier Ethernet Bus load and time to receive the CPU load is longer. However the data can be still obtained even the number of parameters is raised to 2000. In addition, when the time interval of the command is decreased to 1.25ms, the CPU load can be fetched very quickly (2.5s) through Ethernet as shown in Table 5.2. It is much more efficient than using CCP through CAN which is observed by the VISION figure above. CCP could barely poll at a maximum efficiency of 80 signals. On the other hand, the performance gains of XCP over Ethernet is basically uncapped and limitless, since it is possible to poll incomparable figures.

# of Parameters	Parameter size (bytes)	ETH Bus Load %	Time to Respond for CPU load (s)	CPU Load %	Measured Period
1	4	0	0.01	43.6	0.1
75	191	3	0.74	44.2	0.75
100	217	4	0.99	44.3	1
150	376	6	1.49	44.1	1.5
200	467	9	1.98	44.2	1.97
250	518	10	2.5	44.2	2.5
500	900	21	5.00	44.2	4.99
1000	2136	41	10.15	44.1	10
2000	4993	83	19.79	44.2	19.79

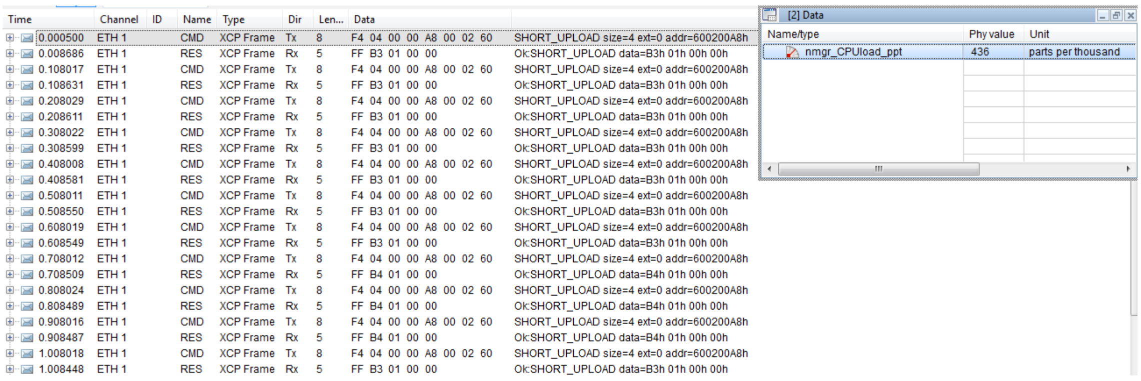
Table 5.1: XCP polling with time interval 10ms

# of Parameters	Parameter size (bytes)	ETH Bus Load %	Time to Respond for CPU load (s)	CPU Load %	Measured Period
1	4	0	0	46.8	0.1
75	134	3	0.083	51.6	0.101
100	189	4	0.114	52.0	0.123
150	356	6	0.177	52.1	0.186
200	535	8	0.24	52.1	0.24
250	710	10	0.30	52.0	0.31
500	1347	21	0.612	52.0	0.63
1000	2936	41	1.24	52.1	1.25
2000	5942	83	2.50	52.1	2.50

Table 5.2: XCP polling with time interval 1.25ms

There is also another bottleneck that needs to be addressed. This was caused by the limitation of CTO/DTO sizes in the Vector tool: DaVinci Configuration, where the maximum packet size was set to 255 bytes. Further experimentation was done for the 255 bytes cap later on.

Another notion to emphasize for the sake of benchmarking is CPU load. CPU is a very critical resource for the sake of ECU. Hence, the excessive borrowing from the lower priority tasks must be prohibited. By all means, Volvo's point of view is that the ECU Ethernet communication it is the most significant measurement variable to track. For the higher priority tasks, it holds pivotal importance. This is more investigated under the section 5.3 Benchmarking.



Time	Channel	ID	Name	Type	Dir	Len...	Data	
0.000500	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.008686	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.108017	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.108631	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.208029	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.208611	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.308022	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.308599	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.408008	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.408581	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.508011	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.508550	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.608019	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.608549	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h
0.708012	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.708509	ETH 1		RES	XCP Frame	Rx	5	FF B4 01 00 00	Ok:SHORT_UPLOAD data=B4h 01h 00h 00h
0.808024	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.808489	ETH 1		RES	XCP Frame	Rx	5	FF B4 01 00 00	Ok:SHORT_UPLOAD data=B4h 01h 00h 00h
0.908016	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
0.908487	ETH 1		RES	XCP Frame	Rx	5	FF B4 01 00 00	Ok:SHORT_UPLOAD data=B4h 01h 00h 00h
1.008018	ETH 1		CMD	XCP Frame	Tx	8	F4 04 00 00 A8 00 02 60	SHORT_UPLOAD size=4 ext=0 addr=600200A8h
1.008448	ETH 1		RES	XCP Frame	Rx	5	FF B3 01 00 00	Ok:SHORT_UPLOAD data=B3h 01h 00h 00h

[2] Data

Name/type	Phy value	Unit
nmgr_CPUload_ppt	436	parts per thousand

Figure 5.6: XCP polling only the CPU l7oad in CANape

The CPU load in this case is measured as 43.6 % as shown in Fig. 5.6.

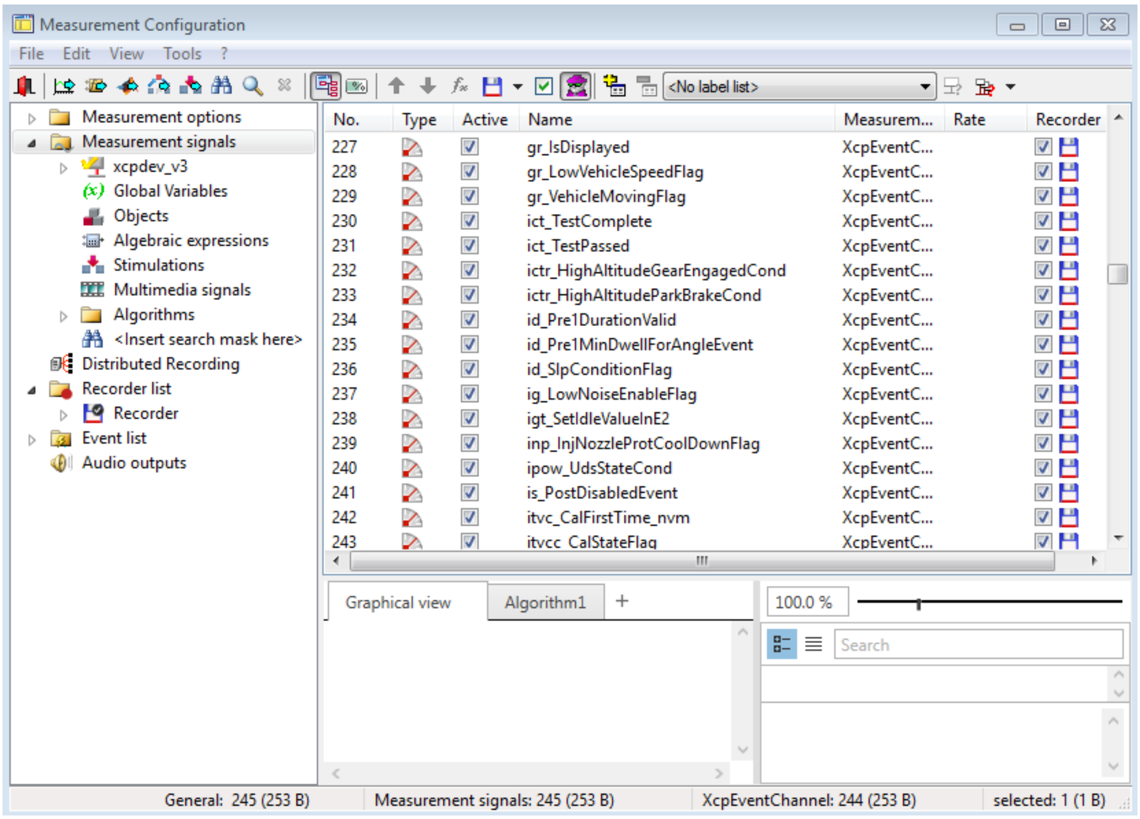
5.2 Synchronous Measurement: DAQ

In synchronous measurement, first the Master communicates to the Slave which data should be sent for different events. And then the Slave replies the data to Master until the Master says 'Measurement Stop'. Within the implementation for the high level, XCP events should be pre-defined through DaVinci before writting them to ECU memory. Hence, once they are called, the slave can understand of which event the master is referring to. The XCP events are embedded into a2l file as well so that CANape can send DAQ related commands from Master side automatically.

5.2.1 DAQ with 1 XCP event

During the implementation, the first hardship was to put parameters into one event and conduct measurement. After succeeding measurement with one XCP event, the goal was to set multiple events. The performance tests, therefore, were done via testing with one, two and three XCP events. Pushing the ECU and the underlying XCP network was one of the tests that was conducted. Moreover, the CPU load increase was another topic, attracting the attention of the performance investigation, as laid out in the research question.

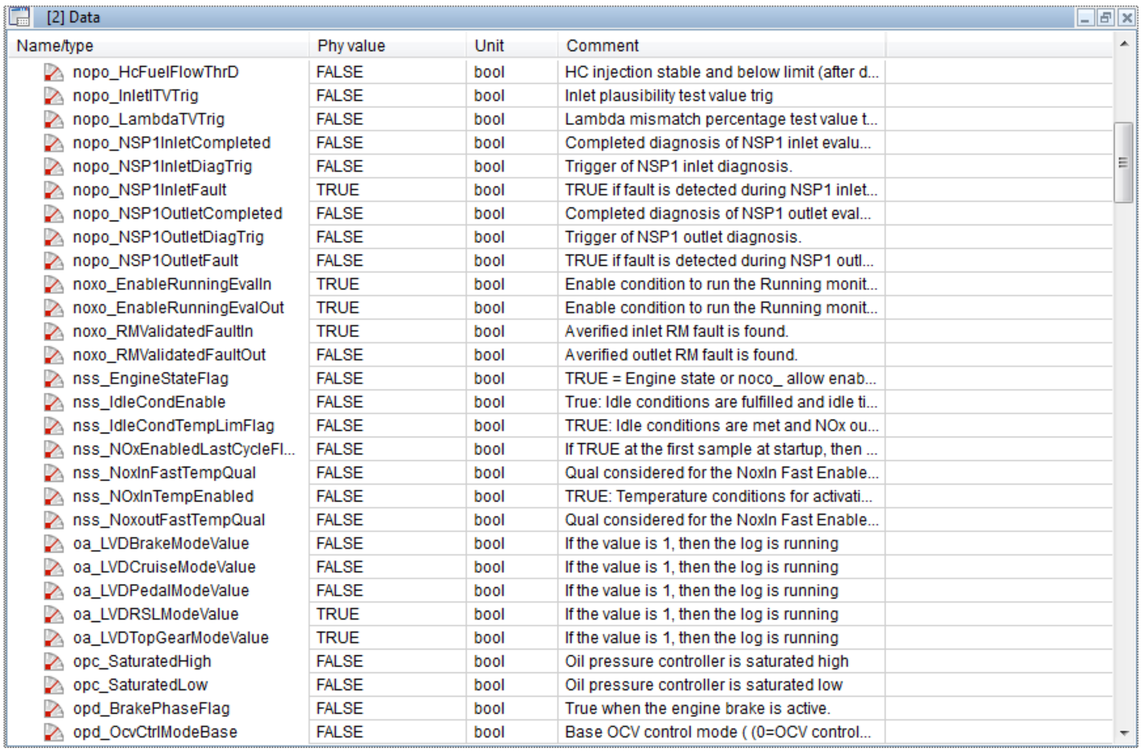
As shown in Fig. 5.7, a full packed XCP DAQ request can be found. As aforementioned, the design was restricted by the 255 bytes bottleneck and could not utilize the max Ethernet MTU: 1500 bytes. It can be observed that there are 245 total measurement signals in one XCP event with a total size of 253 bytes and the first 2 bytes are Ethernet Identification as defined in XCP.



No.	Type	Active	Name	Measurem...	Rate	Recorder
227		✓	gr_IsDisplayed	XcpEventC...		✓
228		✓	gr_LowVehicleSpeedFlag	XcpEventC...		✓
229		✓	gr_VehicleMovingFlag	XcpEventC...		✓
230		✓	ict_TestComplete	XcpEventC...		✓
231		✓	ict_TestPassed	XcpEventC...		✓
232		✓	ictr_HighAltitudeGearEngagedCond	XcpEventC...		✓
233		✓	ictr_HighAltitudeParkBrakeCond	XcpEventC...		✓
234		✓	id_Pre1DurationValid	XcpEventC...		✓
235		✓	id_Pre1MinDwellForAngleEvent	XcpEventC...		✓
236		✓	id_SlpConditionFlag	XcpEventC...		✓
237		✓	ig_LowNoiseEnableFlag	XcpEventC...		✓
238		✓	igt_SetIdleValueInE2	XcpEventC...		✓
239		✓	inp_InjNozzleProtCoolDownFlag	XcpEventC...		✓
240		✓	ipow_UdsStateCond	XcpEventC...		✓
241		✓	is_PostDisabledEvent	XcpEventC...		✓
242		✓	itvc_CalFirstTime_nvm	XcpEventC...		✓
243		✓	itvcc_CalStateFlag	XcpEventC...		✓

General: 245 (253 B) Measurement signals: 245 (253 B) XcpEventChannel: 244 (253 B) selected: 1 (1 B)

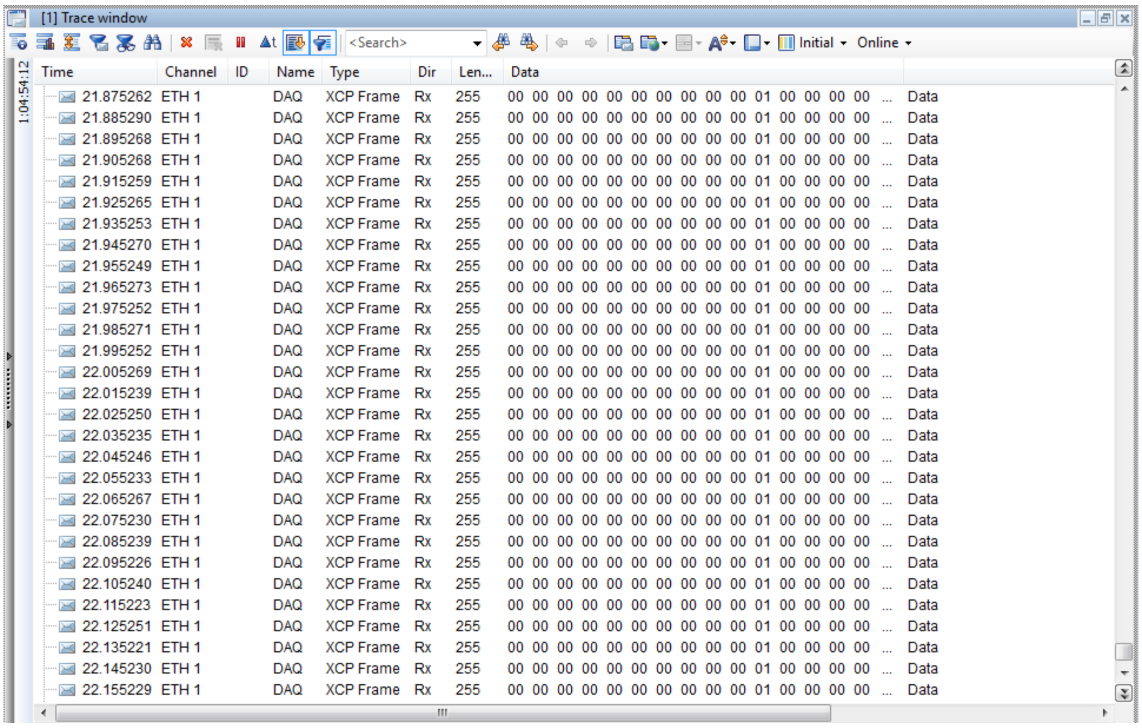
Figure 5.7: Measurement configuration of the XCP event



Name/type	Phy value	Unit	Comment
nopo_HcFuelFlowThrD	FALSE	bool	HC injection stable and below limit (after d...
nopo_InletITVTrig	FALSE	bool	Inlet plausibility test value trig
nopo_LambdaTVTrig	FALSE	bool	Lambda mismatch percentage test value t...
nopo_NSP1InletCompleted	FALSE	bool	Completed diagnosis of NSP1 inlet evalu...
nopo_NSP1InletDiagTrig	FALSE	bool	Trigger of NSP1 inlet diagnosis.
nopo_NSP1InletFault	TRUE	bool	TRUE if fault is detected during NSP1 inlet...
nopo_NSP1OutletCompleted	FALSE	bool	Completed diagnosis of NSP1 outlet eval...
nopo_NSP1OutletDiagTrig	FALSE	bool	Trigger of NSP1 outlet diagnosis.
nopo_NSP1OutletFault	FALSE	bool	TRUE if fault is detected during NSP1 outl...
noxo_EnableRunningEvalIn	TRUE	bool	Enable condition to run the Running monit...
noxo_EnableRunningEvalOut	TRUE	bool	Enable condition to run the Running monit...
noxo_RMValidatedFaultIn	TRUE	bool	A verified inlet RM fault is found.
noxo_RMValidatedFaultOut	FALSE	bool	A verified outlet RM fault is found.
nss_EngineStateFlag	FALSE	bool	TRUE = Engine state or noco_ allow enab...
nss_IdleCondEnable	FALSE	bool	True: Idle conditions are fulfilled and idle ti...
nss_IdleCondTempLimFlag	FALSE	bool	TRUE: Idle conditions are met and NOx ou...
nss_NOxEnabledLastCycleFl...	FALSE	bool	If TRUE at the first sample at startup, then ...
nss_NoxInFastTempQual	FALSE	bool	Qual considered for the NoxIn Fast Enable...
nss_NOxInTempEnabled	FALSE	bool	TRUE: Temperature conditions for activati...
nss_NoxoutFastTempQual	FALSE	bool	Qual considered for the NoxIn Fast Enable...
oa_LVDBrakeModeValue	FALSE	bool	If the value is 1, then the log is running
oa_LVDCruiseModeValue	FALSE	bool	If the value is 1, then the log is running
oa_LVDPedalModeValue	FALSE	bool	If the value is 1, then the log is running
oa_LVDRSLModeValue	TRUE	bool	If the value is 1, then the log is running
oa_LVDTopGearModeValue	TRUE	bool	If the value is 1, then the log is running
opc_SaturatedHigh	FALSE	bool	Oil pressure controller is saturated high
opc_SaturatedLow	FALSE	bool	Oil pressure controller is saturated low
opd_BrakePhaseFlag	FALSE	bool	True when the engine brake is active.
opd_OcvCtrlModeBase	FALSE	bool	Base OCV control mode ((0=OCV control...

Figure 5.8: Signals chosen for measurement

Measured signals are shown in Fig. 5.8. Overtly, the boolean signals are measured and has a physical value according to their assignment in their respective ECU addresses. Fig. 5.9 is the Trace Window of CANape depicting the overall frames being sent with their timestamps. The log depicts that the whole frame (255 bytes) of XCP was used up. This is due to the 2 bytes overhead that the XCP appends to the payload.



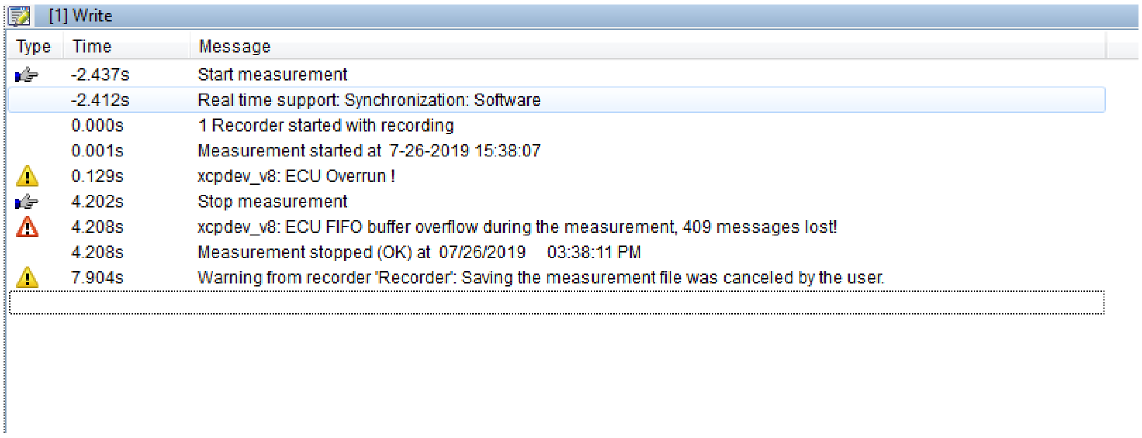
```
Time        Channel  ID  Name  Type       Dir  Len...  Data
21.875262   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.885290   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.895268   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.905268   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.915259   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.925265   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.935253   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.945270   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.955249   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.965273   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.975252   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.985271   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
21.995252   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.005269   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.015239   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.025250   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.035235   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.045246   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.055233   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.065267   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.075230   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.085239   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.095226   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.105240   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.115223   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.125251   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.135221   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.145230   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
22.155229   ETH 1        DAQ   XCP Frame  Rx   255     00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 ...  Data
```

Figure 5.9: Measurement log of the XCP event

As pointed out before the bottleneck in the design, where the maximum CTO/DTO size was 255 bytes, was also experienced empirically. We tried to send more than 255 bytes through one event. As expected, there were dropped frames. The result is shown in Fig. 5.10 displaying the problem encountered. A rather nonverbose prompt is on the Write Window of CANape stating the ECU overrun and the amount of messages that were lost during the course of the measurement.

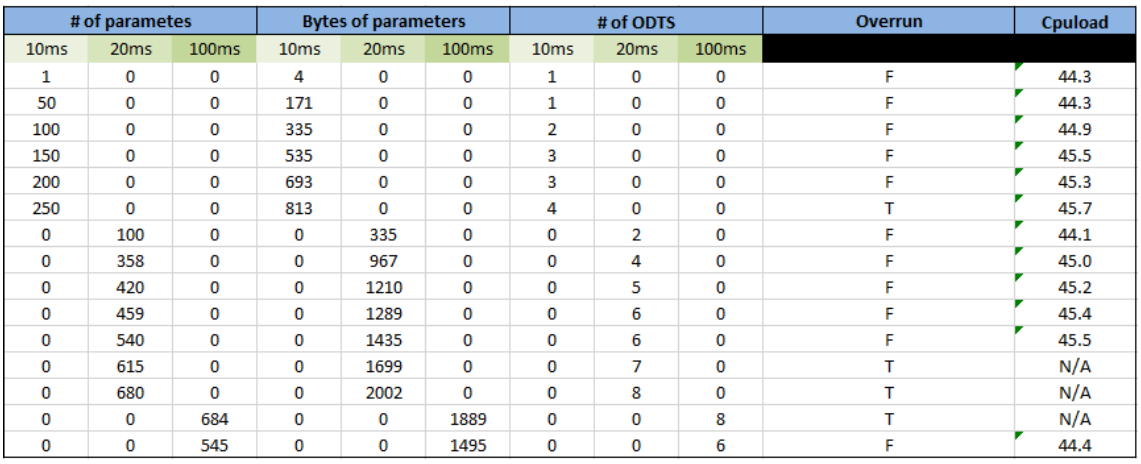
To clarify, the timings that were selected as $10ms$, $20ms$ and $100ms$ were the windows that were previously used in the CCP implementation. So as not to disturb the coherency and consistency, the same cyclic periods were used. After stopping the measurement process, another prompt called FIFO overflow arrives. There is data loss experienced and some signals are omitted during the measurement.

The performance tests were conducted as well. As shown in Fig. 5.11, for event $10ms$, the largest ODT number is four which means at most send 1012 bytes parameters within $10ms$. But for event $20ms$ and event $100ms$, 8 ODTs which means 1518 bytes parameters are adaptable. Compared to CCP over CAN bus, more data can be delivered efficiently. In addition, the CPU load went up till 45.7% less than 50% which means the requirement and consuming of the processor is good.



```
[1] Write
Type  Time      Message
      -2.437s   Start measurement
      -2.412s   Real time support: Synchronization: Software
      0.000s    1 Recorder started with recording
      0.001s    Measurement started at  7-26-2019 15:38:07
      0.129s    xcpdev_v8: ECU Overrun !
      4.202s    Stop measurement
      4.208s    xcpdev_v8: ECU FIFO buffer overflow during the measurement, 409 messages lost!
      4.208s    Measurement stopped (OK) at  07/26/2019     03:38:11 PM
      7.904s    Warning from recorder 'Recorder': Saving the measurement file was canceled by the user.
```

Figure 5.10: Measurement log of the XCP event

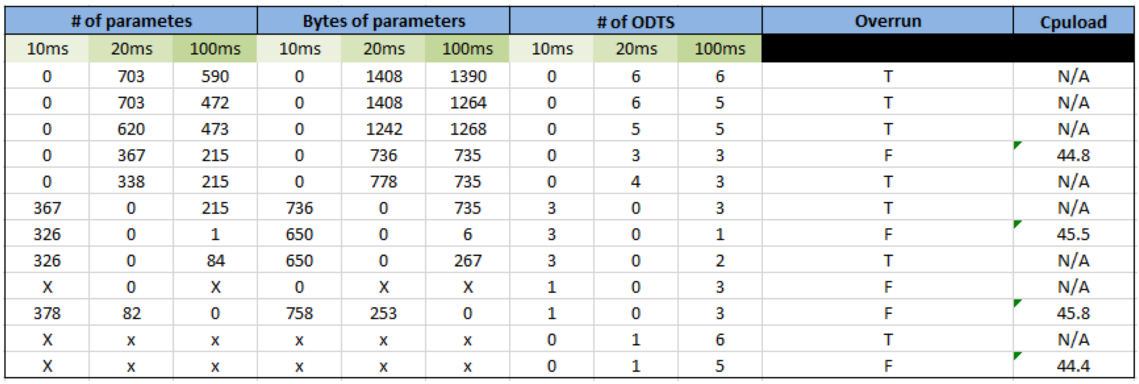


# of parametes			Bytes of parameters			# of ODTS			Overrun	Cpuload
10ms	20ms	100ms	10ms	20ms	100ms	10ms	20ms	100ms		
1	0	0	4	0	0	1	0	0	F	44.3
50	0	0	171	0	0	1	0	0	F	44.3
100	0	0	335	0	0	2	0	0	F	44.9
150	0	0	535	0	0	3	0	0	F	45.5
200	0	0	693	0	0	3	0	0	F	45.3
250	0	0	813	0	0	4	0	0	T	45.7
0	100	0	0	335	0	0	2	0	F	44.1
0	358	0	0	967	0	0	4	0	F	45.0
0	420	0	0	1210	0	0	5	0	F	45.2
0	459	0	0	1289	0	0	6	0	F	45.4
0	540	0	0	1435	0	0	6	0	F	45.5
0	615	0	0	1699	0	0	7	0	T	N/A
0	680	0	0	2002	0	0	8	0	T	N/A
0	0	684	0	0	1889	0	0	8	T	N/A
0	0	545	0	0	1495	0	0	6	F	44.4

Figure 5.11: DAQ with 1 event

5.2.2 DAQ with two XCP events

In this scenario, two XCP events are configured. All the three configurations are exploited 10ms with 20ms, 10ms with 100ms and 20ms with 100ms. The results are displayed as Fig. 5.11.



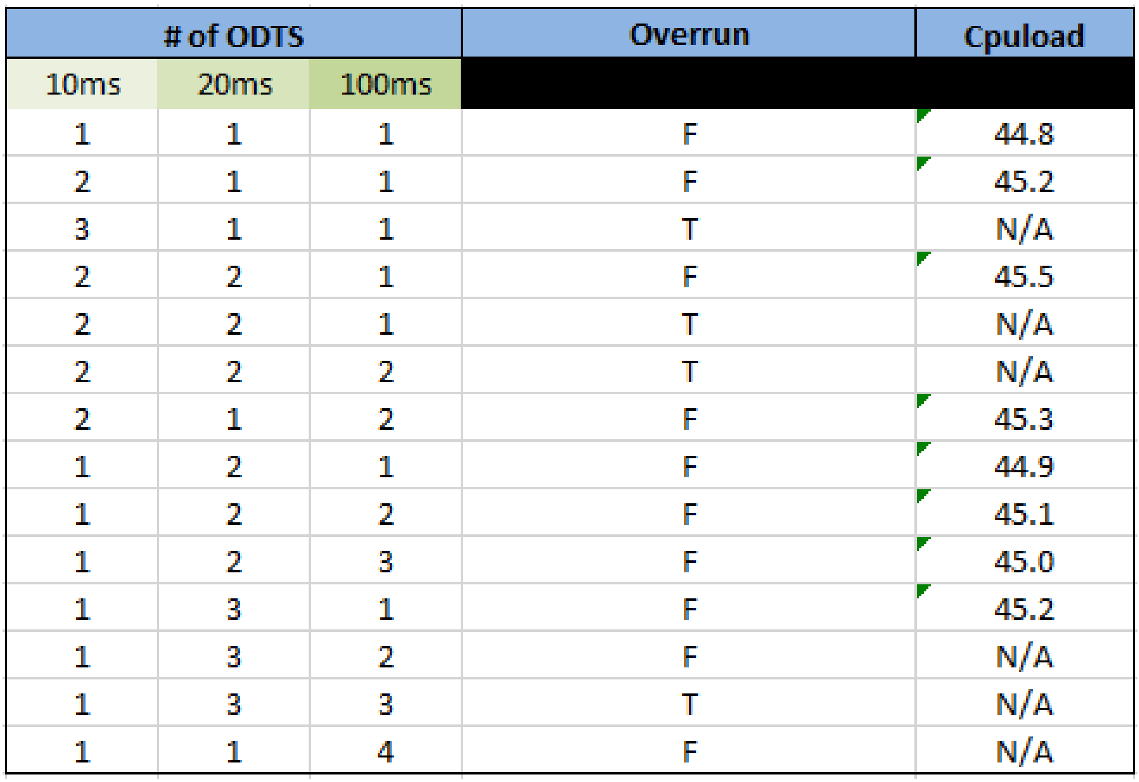
# of parametes			Bytes of parameters			# of ODTS			Overrun	Cpuload
10ms	20ms	100ms	10ms	20ms	100ms	10ms	20ms	100ms		
0	703	590	0	1408	1390	0	6	6	T	N/A
0	703	472	0	1408	1264	0	6	5	T	N/A
0	620	473	0	1242	1268	0	5	5	T	N/A
0	367	215	0	736	735	0	3	3	F	44.8
0	338	215	0	778	735	0	4	3	T	N/A
367	0	215	736	0	735	3	0	3	T	N/A
326	0	1	650	0	6	3	0	1	F	45.5
326	0	84	650	0	267	3	0	2	T	N/A
X	0	X	0	X	X	1	0	3	F	N/A
378	82	0	758	253	0	1	0	3	F	45.8
X	x	x	x	x	x	0	1	6	T	N/A
X	x	x	x	x	x	0	1	5	F	44.4

Figure 5.12: DAQ with 2 events

Three ODTs from 10ms and one ODT from 100ms worked without ECU being overrun in this configuration. Furthermore, up to six ODTs via 20ms and 100ms could be arranged.

5.2.3 DAQ with three XCP events

The events were already configured to be 10ms, 20ms and 200ms. Therefore, there are three identical XCP events created. The results are produced, accordingly:



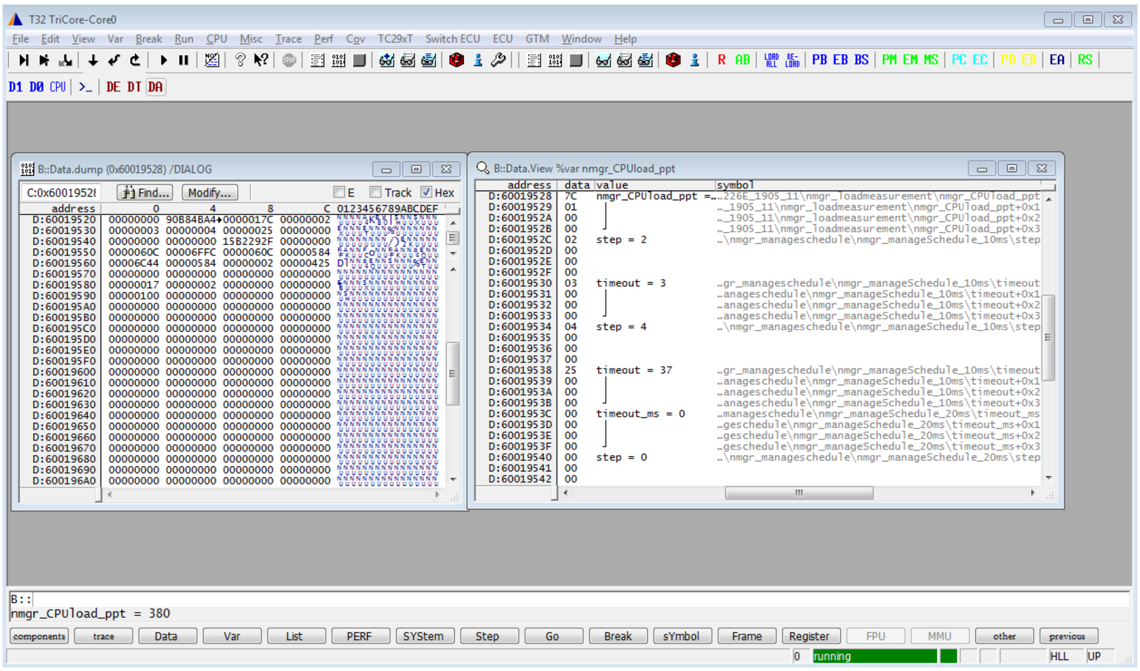
# of ODTS			Overrun	Cpuload
10ms	20ms	100ms		
1	1	1	F	44.8
2	1	1	F	45.2
3	1	1	T	N/A
2	2	1	F	45.5
2	2	1	T	N/A
2	2	2	T	N/A
2	1	2	F	45.3
1	2	1	F	44.9
1	2	2	F	45.1
1	2	3	F	45.0
1	3	1	F	45.2
1	3	2	F	N/A
1	3	3	T	N/A
1	1	4	F	N/A

Figure 5.13: DAQ with 3 events

Similar results have been obtained. Where a configuration leading to six ODTs would not cause an ECU overrun. However, 2-2-2 configuration would yield an ECU overrun.

5.3 Benchmarking

There was an interest to observe the CPU load without any Ethernet block and XCP implementation. Hence, the remaining software blocks were the base software. Originally, before this implementation there was not a similar network stack and processing done at the ECU, hence it would be logical to expect the CPU activity being increased on the ECU side. To address the question quantitatively, any additional modules were removed from the ECM4 structure and the BSW was compiled without any XCP or Ethernet module. Only retaining the CCP and CAN modules at the CSWC, the load was measured to be 38%. Clearly, CCP is less taxing on the CPU load than the Ethernet implementation.


T32 TriCore-Core0
File Edit View Var Break Run CPU Misc Trace Perf Cov TC29xT Switch ECU ECU GTM Window Help
B::Data.dump (0x60019528) /DIALOG
C:0x6001952{
Find...
Modify...
E
Track
Hex
B::Data.View %var nmgr_CPUload_ppt
components trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register FPU MMU other previous
running
HLL
UP

```
   address         0        4        8        C 0123456789ABCDEF
D:60019520  00000000 90B84BA4 0000017C 00000002
D:60019530  00000003 00000004 00000025 00000000
D:60019540  00000000 00000000 15B2292F 00000000
D:60019550  0000060C 00006FFC 0000060C 00000584
D:60019560  00006C44 00000584 00000002 00000425
D:60019570  00000000 00000000 00000000 00000000
D:60019580  00000017 00000002 00000000 00000000
D:60019590  00000100 00000000 00000000 00000000
D:600195A0  00000000 00000000 00000000 00000000
D:600195B0  00000000 00000000 00000000 00000000
D:600195C0  00000000 00000000 00000000 00000000
D:600195D0  00000000 00000000 00000000 00000000
D:600195E0  00000000 00000000 00000000 00000000
D:600195F0  00000000 00000000 00000000 00000000
D:60019600  00000000 00000000 00000000 00000000
D:60019610  00000000 00000000 00000000 00000000
D:60019620  00000000 00000000 00000000 00000000
D:60019630  00000000 00000000 00000000 00000000
D:60019640  00000000 00000000 00000000 00000000
D:60019650  00000000 00000000 00000000 00000000
D:60019660  00000000 00000000 00000000 00000000
D:60019670  00000000 00000000 00000000 00000000
D:60019680  00000000 00000000 00000000 00000000
D:60019690  00000000 00000000 00000000 00000000
D:600196A0  00000000 00000000 00000000 00000000
```

```
   address  data value                symbol
D:60019528  7C   nmgr_CPUload_ppt =.. ..226E_1905_11\nmgr_loadmeasurement\nmgr_CPUload_ppt
D:60019529  01                        .._1905_11\nmgr_loadmeasurement\nmgr_CPUload_ppt+0x1
D:6001952A  00                        .._1905_11\nmgr_loadmeasurement\nmgr_CPUload_ppt+0x2
D:6001952B  00                        .._1905_11\nmgr_loadmeasurement\nmgr_CPUload_ppt+0x3
D:6001952C  02   step = 2             ..\nmgr_manageschedule\nmgr_manageSchedule_10ms\step
D:6001952D  00
D:6001952E  00
D:6001952F  00
D:60019530  03   timeout = 3          ..gr_manageschedule\nmgr_manageSchedule_10ms\timeout
D:60019531  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x1
D:60019532  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x2
D:60019533  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x3
D:60019534  04   step = 4             ..\nmgr_manageschedule\nmgr_manageSchedule_10ms\step
D:60019535  00
D:60019536  00
D:60019537  00
D:60019538  25   timeout = 37         ..gr_manageschedule\nmgr_manageSchedule_10ms\timeout
D:60019539  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x1
D:6001953A  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x2
D:6001953B  00                        ..anageschedule\nmgr_manageSchedule_10ms\timeout+0x3
D:6001953C  00   timeout_ms = 0       ..manageschedule\nmgr_manageSchedule_20ms\timeout_ms
D:6001953D  00                        ..geschedule\nmgr_manageSchedule_20ms\timeout_ms+0x1
D:6001953E  00                        ..geschedule\nmgr_manageSchedule_20ms\timeout_ms+0x2
D:6001953F  00                        ..geschedule\nmgr_manageSchedule_20ms\timeout_ms+0x3
D:60019540  00   step = 0             ..\nmgr_manageschedule\nmgr_manageSchedule_20ms\step
D:60019541  00
D:60019542  00
```

```
B::
nmgr_CPUload_ppt = 380
```

Figure 5.14: CPU load measured through T32

CPU load via Polling was increased by 18%, when the interrupt was scheduled at 1.25ms instead of 10ms, from average 44.2% to 52.1%. This is considered to be a good pay-off regarding Volvo's perspective, even though the performance is increased by 8 times, however the load has increased more than 12.5%.

On average, having additional DAQ setups do not hinder the CPU load. The average of one, two or three XCP events appear to be around 45.5%. Nevertheless, the CPU load has marginally decreased, while switching the measurement method from Polling to DAQ. The decrease is 12.6%.

It holds a paramount importance to highlight the CCP's shortcoming where the amount of parameters that can be measured via DAQ are given in the Table 5.1.

These measurements are CCP measurements without using on-board A7/A8 units. Volvo Group has stated in multiple points that it would be good to minimize the need of those, due to cost of the units. The further findings suggest that the XCP performance even surpasses the maximized throughput, which CCP can output.

Period	Number of measured parameters
10 ms	5
20 ms	15
100 ms	38

Table 5.3: CCP with maximum amount of variables that can be measured

Period	CCP with A7/A8	XCP
10 ms	75	200
100 ms	400	545
20 ms	N/A	540

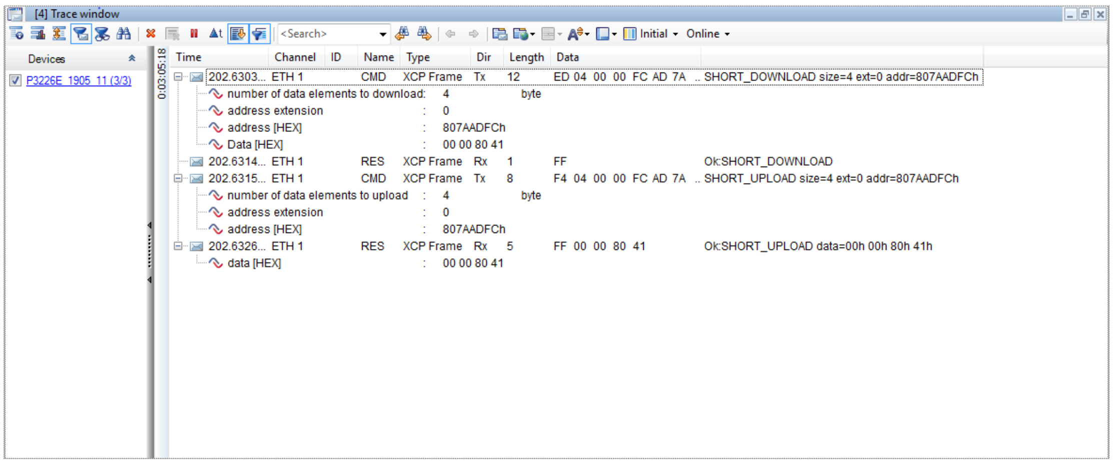
Table 5.4: CCP via A7/A8 units vs XCP

The result of the juxtaposition of XCP over Ethernet and CCP heavily lays in favor of XCP over Ethernet. As depicted by the Table 5.4, the amount of variables that can be measured by the 10ms event achieved 167% increase by XCP over CCP. The 100ms event, however, has gained a 36% boost. The differences are greater, when comparing the true CCP i.e. without the A7/A8 devices with XCP, according to Table 5.3 and 5.4.

5.4 Calibration

As mentioned before, calibration is the process of optimally tuning the ECU parameters [10]. In Volvo Group's use case, that would be adding a signal parameter from CANape to the list of calibration elements and adjusting that parameter.

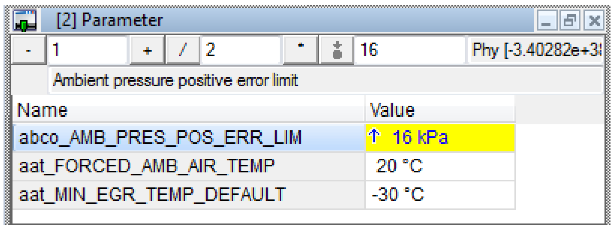
The Fig. 5.15 was created via the calibration functionality of the tool CANape. A random parameter was selected from the memory and the value was changed to 8041 in hex. The trace window depicts the breakdown of the command chain.



```
Time          Channel  ID  Name  Type       Dir  Length  Data
202.6303...   ETH 1        CMD   XCP Frame  Tx   12      ED 04 00 00 FC AD 7A .. SHORT_DOWNLOAD size=4 ext=0 addr=807AADFCh
    number of data elements to download:   4          byte
    address extension                  :   0
    address [HEX]                      :   807AADFCh
    Data [HEX]                         :   00 00 80 41
202.6314...   ETH 1        RES   XCP Frame  Rx   1       FF                      Ok:SHORT_DOWNLOAD
202.6315...   ETH 1        CMD   XCP Frame  Tx   8       F4 04 00 00 FC AD 7A .. SHORT_UPLOAD size=4 ext=0 addr=807AADFCh
    number of data elements to upload  :   4          byte
    address extension                  :   0
    address [HEX]                      :   807AADFCh
202.6326...   ETH 1        RES   XCP Frame  Rx   5       FF 00 00 80 41          Ok:SHORT_UPLOAD data=00h 00h 80h 41h
    data [HEX]                         :   00 00 80 41
```

Figure 5.15: Calibration achieved through CANape

To explain the command chain, first "Short-Download" command is issued by the master denoting four points within the XCP frame. The first is the number of data elements to download. The second is address extension which is zero in this case. The third is address in hexadecimal value. The last one is the actual data to be written to the address given in hex. Overtly, the first command underscores the parameter residing in 807AADFCh and the value to be changed to 8041h. After the command is executed, the slave responds as OK. Thereafter, in order to verify the parameter change, the master issues the "Short-Upload" command. Basically, the master would like to fetch the same address that the change was incurred. Slave's response is OK with the value of that address: 8041h. It indicates that the selected parameter is altered successfully.



Name	Value
abco_AMB_PRES_POS_ERR_LIM	↑ 16 kPa
aat_FORCED_AMB_AIR_TEMP	20 °C
aat_MIN_EGR_TEMP_DEFAULT	-30 °C

Figure 5.16: Calibration achieved through CANape

The above Fig. 5.16 clarifies that the signal called abco_AMB_PRES_POS_ERR_LIM is incremented to 16 kPa. The highlight on the signal identifies that the parameter has been successfully calibrated.

6
Conclusion

The XCP over Ethernet was realized within the use cases of Volvo Group. A next generation ECU, ECM4, was powered via XCP over Ethernet. The ECU parameters were measured and calibrated successfully via XCP packets that are sent along the Ethernet link. The work was found to be promising and received positive responses mostly due to faster measurement and calibration tasks and higher throughput i.e. higher amount of parameters that can be fetched at the same time, both via synchronous and asynchronous data acquisition methodologies.

To summarize, with $10ms$ periodicity, XCP achieves 2.67 times as many measurement values as CCP achieves. With $100ms$ periodicity, XCP could fetch 1.37 times as many measurement signals as CCP.

The conclusion is dire, while looking at the CCP performance measurements done without an A7/A8 unit. XCP accomplishes 39 fold increase than CCP for the $10ms$ periodic measurement requests. 13 fold increase is calculated for the case of $100ms$ periodicity between XCP and CCP.

87% of increase was reached for the sake of synchronous measurement with the period of $10ms$.

The greatest difficulty regarding the completion of the thesis, which can also be used to bolster the argument of the three months delay to the delivery, was; Familiarizing with the OEM solutions and integrating them into the Volvo Group's BSW, configurating the network stack via the Vector DaVinci Configurator and enabling the Ethernet communication enabler, pin settings and speed grade settings within the new generation ECM4.

6.1 Future Work

The following checkpoints can be achieved in order to take this project further. They are omitted either for time restrictions or for environmental dependencies:

- Flashing or software download needs to be achieved via XCP over Ethernet. This was omitted due to the time restrictions of the thesis project.

- Increasing the CTO/DTO packet sizes further from 255.
  There were discussions with Vector's German office that this is actually atteinable, however in the current DaVinci Configurator settings, the maximum CTO and DTO packet sizes were set to 255. Given the Maximum Transmission Unit is 1500 [30], there is a potential resource to be used. The current

implementation only uses 0.17 of the total resource available by the MTU.

- CANape dependecy.
  Volvo Group's interests and the know-how at the time of the thesis did not align with CANape. The powertrain teams were experienced to use the tool VISION, hence the teams are not familiar with CANape. Furthermore, VISION license is still ongoing for Volvo Group. The drawback is that VISION did not support VN5610A during the time of the thesis, hence CANape had to be used to attain measurement and calibration via XCP over Ethernet.

- Implementing seed/key exchange for security.
  Seed/key exchange algorithm is of course supported by XCP. This was neglected, due to the time constraints. Nevertheless, seed/key is a very significant security functionality in ECU communication protocols that needs to be implemented in Volvo Group's point of view.

- Checksum support.
  Checksum is useful to determine the changes in the memory. Otherweise, the measurement and calibration tool has to check all the memory addresses one-by-one in order to determine whether they have been changed or not.

- Timestamping coming from the ECU.
  Timestamping is provided via the CANape. This is prone to errors due to the added propagation delay. The change should be made so that the timestamping would be generated from the ECU, hence no offset between the time that the data is generated and reaching to the master.

Bibliography

[1] Andre Rolfsmeier et al. “A New Calibration System for ECU Development”. In: (2003).

[2] Andreas Patzer. “Optimize ECU parameters with XCP”. In: (2009).

[3] Pucha et. al. “Understanding Network Delay Changes Caused by Routing Events”. In: (2007).

[4] Mark Sauerwald. *CAN bus, Ethernet, or FPD-Link: Which is best for automotive communications?* URL: `http://www.ti.com/lit/an/slyt560/slyt560.pdf`.

[5] Kurose and Ross. “Computer Networking - A Top Down Approach”. In: (2000).

[6] Kim et al. “Performance Analysis of the TCP/IP Protocol Under UNIX Operating Systems for High Performance Computing and Communications”. In: (1997).

[7] Maping Li. “Performance Analysis of Wireless Network Maximum Throughput Based on Network Coding”. In: (2017).

[8] Dridi et al. “Coupling Latency Time to the Throughput Performance Analysis on Wireless CAN Networks”. In: (2006).

[9] Nguyen et al. “A cross-layer approach for improving WiFi performance”. In: (2014).

[10] Andreas Patzer and Rainer Zaiser. *XCP - The Standard Protocol for ECU Development.* 2016. URL: `https://assets.vector.com/cms/content/application-areas/ecu-calibration/xcp/XCP_ReferenceBook_V3.0_EN.pdf`.

[11] Vasudeva Varma. “Software Architecture: A Case Based Approach”. In: (2009).

[12] *Theory and Terminology - Master/Slave vs Peer-to-Peer.* URL: `http://info.bannerengineering.com/cs/groups/public/documents/literature/tt_masterslave.pdf`.

[13] URL: `https://stackoverflow.com/questions/13121531/multi-client-server-common-way-for-2-way-connection-in`.

[14] Wilfried Voss. *Industrial Ethernet Guide - Client/Server vs Master/Slave.* URL: `https://copperhilltech.com/blog/industrial-ethernet-guide-clientserver-vs-masterslave/`.

[15] Konrad Etschberger. “Comparing CAN- and Ethernet-based Communication”. In: ().

[16] H. Kleinknecht. *CAN Calibration Protocol.* 1999. URL: `https://automotivetechis.files.wordpress.com/2012/06/ccp211.pdf`.

[17] Joakim Plate and Peter Fridlund. *XCP OVER CAN AND ETHERNET ON AUTOSAR*. 2011. URL: http://publications.lib.chalmers.se/records/fulltext/140407.pdf.

[18] AUTOSAR Release Management. *Specification of Module XCP*. 2013.

[19] ATI Support. *The Move from ASAM CCP to XCP Communication Protocol*. 2016. URL: https://www.kvaser.com/wp-content/uploads/2016/05/ccp-and-xcp-papermay2016.pdf.

[20] Kim Lemon. *Introduction to the Universal Measurement and Calibration Protocol XCP*. 2003. URL: https://saemobilus.sae.org/content/2003-01-1205/#abstract.

[21] *ASAM MCD-1 XCP*. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[22] AUTOSAR Release Management. *Layered Software Architecture*. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[23] *DaVinci Configurator Pro*. URL: https://www.vector.com/int/en/products/products-a-z/software/davinci-configurator-pro/#c6343.

[24] *ECU Calibration with CANape*. URL: https://www.vector.com/int/en/products/products-a-z/software/canape/.

[25] *Analyzing ECUs and Networks with CANalyzer*. URL: https://www.vector.com/int/en/products/products-a-z/software/canalyzer/.

[26] *Lauterbach TRACE32 Target Interface*. URL: https://www.nxp.com/docs/en/user-guide/LAUTERBACHTRACE32UG.pdf.

[27] *Product Overview*. URL: https://www.lauterbach.com/product-overview_flyer_web.pdf.

[28] *VN5610/VN5610A Ethernet/CAN Interface Manual*. URL: https://assets.vector.com/cms/content/products/VN56xx/docs/VN5610_Manual_EN.pdf.

[29] ASAM. “ASAM MCD-1 (XCP) Universal Measurement and Calibration Protocol”. In: (2017).

[30] *Ethernet - Maximum transmission unit (MTU)*. URL: https://gerardnico.com/network/mtu.