Joan Fonts Gómez

Automotive Ethernet. High-end Ethernet Switches within the Automotive framework.

Final Degree Project (TFG)

Directed by Dr. Antoni Martínez Ballesté

Degree in Industrial Electronics and Automation Engineering

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Index

1. METHODOLOGY AND OBJECTIVES

1.1. About this project

The main purpose of this research is to implement Automotive Ethernet in modern cars as well as discovering possible advantages and disadvantages while researching it. To make Automotive Ethernet possible, an Ethernet Switch must be programmed and configured so as to enable audio and video communications (AVB) within the automobile, alongside other types of data communications. The aim of the project is also to encrypt those messages at layer 2 (MACsec) so as to make the communications more secure. Furthermore, those messages need to be perfectly synchronized, which leads to Time-Synchronization protocols such as gPTP (Time Sensitive Networking). The research also involves the Ethernet switch selection and the development of different testbenches, which will be necessary to start with the configuration of the Ethernet Switch.

1.2. Objectives

To begin with the project, we firstly need to establish some objectives, which will serve as a guide of the whole research. These objectives are the following:

Firstly, study and learn many concepts of networking protocols of the TCP/IP model, as well as the study of the Time-Synchronization and Audio Video Bridging protocols and the layer 2 MACsec encryption. Also create different testbenches that will be furtherly used to test the functionality of the switch and to consolidate the knowledge learnt during the previous process.

Secondly, study the features of a wide range of Ethernet Switches from different brands. After that, make a decision about which device suits best for project. This decision must include both a hardware and a software evaluation in order to choose a proper switch.

Thirdly, successfully configure and program the previously selected Ethernet Switch and implement the basic features of communication. Those basic features include the Virtual LANs, Audio Video Bridging communication, MACsec encryption and Time Sensitive Networking. Although all of those features are englobed in a single objective, they will be treated and developed individually.

1.3. Methodology and Project Plan

After explaining the different objectives, I will outline the main steps that need to be followed in order to achieve the results of research project. This part does not include the specific details of each step, it is only a brief explanation to introduce the reader into the process of doing the research.

In the following image, there is a Gantt diagram of the planification of the project. It is divided into two main parts: the base steps, during which all of the research will be done; and the project development, when all the different Switch functionalities will be developed.

In the first part of the project, deep research needs to be done about the TCP/IP Internet model (including link, network and transport layers). The purpose of this research is to get the grips with Internet concepts so as it will be easier to understand further concepts, such as Audio Video Bridging protocol (AVB) and MACsec, which are essential to Automotive Ethernet. Although the main goal of the project is to research about AVB, we firstly need to research the base onto which AVB is implemented, which is TCP/IP model.

Alongside this research, a testbench will be developed. This testbench will include two different approaches, which are separated with two different arrows in the previous image. The first approach will be done with a virtual environment (virtual machines and a virtual switch). The second approach will be implemented with an actual physical network, with a real switch and Raspberry Pi's as hardware endpoints.

In parallel of those previous stages, the hardware for the project must be selected. To begin with the selection, some hardware features need to be analyzed, such as 1000BASE-T1, MACsec, and other capabilities. Many of the switches that do not meet those requirements will be directly discarded and. After this, a software evaluation needs to be done in order to choose which switch is more suitable. At this point, and when the switch is selected, the base steps will be completed.

The first step to start with the project development is to get to know how the Evaluation Board (EVB) works. After that, the last but longest step of the project will start. In this last phase, the different functionalities of the switch will be implemented, including VLAN, Time Sensitive Networking, Audio Video Bridging and MACsec.

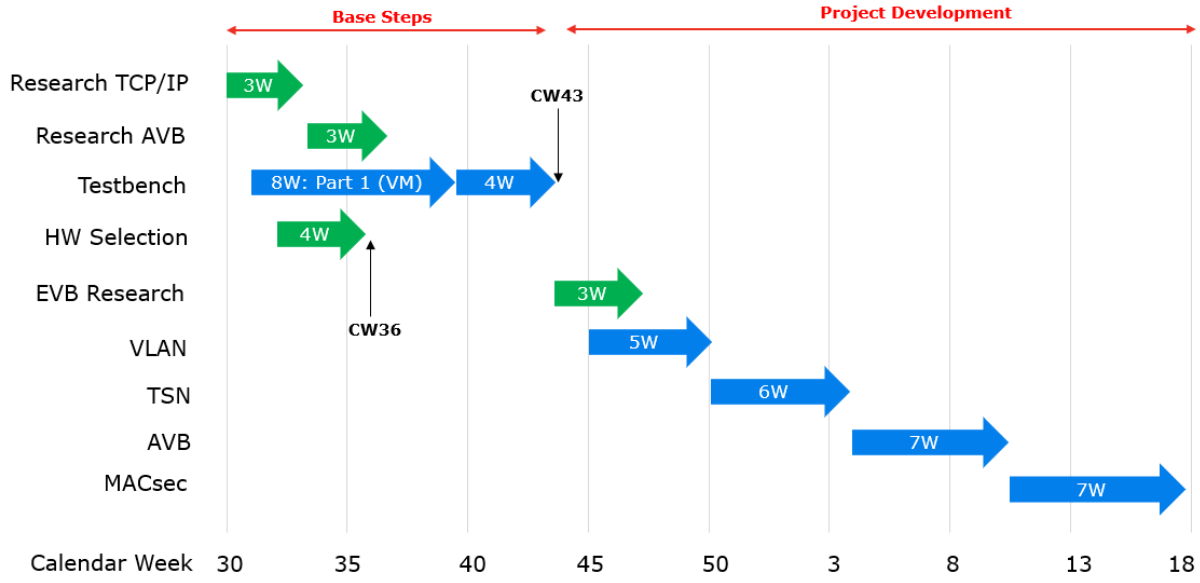


Figure 1. Gantt diagram of the Planification of the project.

1.4. Acknowledgements

To end with this section, I would like to thank my Lear tutor, Ricardo Martinez, for all the technical support, help and motivation given during the development of the project; to Javi Cuenca, for all the work done externally with the suppliers and all the help provided during the Trainee Program; to my university tutor, Antoni Martínez, for the help and communication provided during the development of the project; and, finally, to my parents and my sister, for all the unconditional support and love that I was given during this long period of time.

2. AUTOMOTIVE ETHERNET

In the first part of the introduction, there will be a brief explanation about the context and the motivations of implementing Ethernet into Automobiles. Following this part, we'll revise thoroughly the vehicle architecture and see how the data is distributed across the vehicle. Continuing with the third part, we'll look through the Ethernet Layer (also referred as layer 2 of the OSI model), which is essential to Automotive Ethernet since AVB and other protocols are built on top of this layer. To end with the introduction, there will be a deep explanation about Precision Time Protocol, MACsec and Audio Video Bridging with the main protocols it includes.

2.1. Context and motivation for automotive ethernet

Before hitting 21st century, cars weren't as complex as they have been in the last decade. Each vehicle contained just a few components and cables, used for power distribution and pretty simple options, such as turning on the lights or the radio. However, the more the technology has advanced, the more complex the cars have become, and we've reached a point in which cars could be defined as "computers with wheels".

Actual cars have a huge processing power, and this due to the amount of electronics involved in every single car. These vehicles are design according to a *zonal architecture*, with which different *Electronic Control Units* (also known as ECUs) are distributed alongside the car and each one is designed to do a specific function. Each of these ECUs needs to be communicated with a central processing unit, and this process involves cables in some way.

Generally, in cars, the way with which different modules communicate with each other are the *Control Area Network* buses (or CAN). It is a "reliable" and affordable way to transfer data between electronic modules. However, as soon as we add more ECUs in the vehicle, the number of cables needed increases considerably and, therefore, its weight increases as well.

Cable weight is directly responsible for vehicle's consumption. The three main elements that are responsible for almost all the weight of the vehicle are the motor and mechanics, the external carrosserie and the cable distribution. Consequently, if we reduced the amount of cable weight, we would slightly reduce the car pollution.

Regarding the CAN bus, apart from the cable weight, this bus also has a great disadvantage in terms of speed. Although it can carry data up to 1 MB/s, it's not as fast as it would require in modern cars functions, such as video streaming. Therefore, this is when Automotive Ethernet comes into place.

The concept of Automotive Ethernet looks simply, and it is. We just need to replace some of our previous CAN communication buses and use Ethernet cables instead, so as to increase the speed of the transfer of data. Apart from increasing speed, there is another great advantage, which is cable weight reduction, less fuel consumption and better reliability.

2.2. Architecture

In order to use Ethernet for transferring data, we need some sort of "order" inside the vehicle. Modern cars follow a type of structure know as *zonal architecture*. The image below shows an example of a possible zonal architecture:

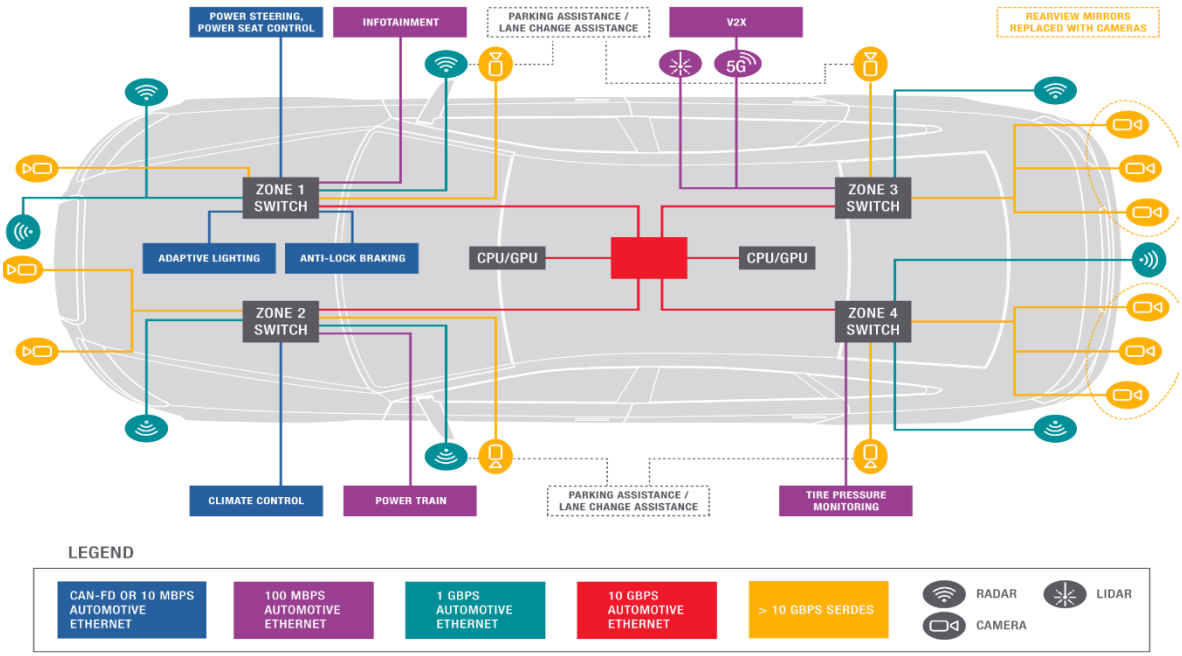


Figure 2. Zonal Architecture of the vehicle.

As we can see in Fig. 2. the vehicle would be divided into several parts, each of them controlled by a switch, which is connected to a central CPU. The intention to do so is to reduce the amount of data load that the CPU of the vehicle would have to process. Also, to increase speed and security in data transmission.

Each switch will be intelligently programmed to forward the essential data to the place it is intended to go, with a millimetric time precision and synchronization at high speeds. This will be suitable for real time applications such as video streaming of the rear camera while you are parking your vehicle, listening to music inside the vehicle, and so on. Furthermore, the switch will also be programmed to store the non-essential data temporarily while the bus is being occupied by essential information. When the bus is free, the non-essential data will be sent through the communication medium.

The great advantage that this zonal architecture has in comparison of the bus-type architecture is that not only we increase the speed with which the information is transmitted (increasing the bandwidth), but also create a Point-to-Point communication between the ECU and the switch, which prevents the collision possibility that we could not avoid in a bus-type architecture.

2.3. Ethernet layer

2.3.1. Link Layer

The Ethernet layer corresponds to the link layer (2) of the OSI reference model. Since the automobile network involves switches and is a "closed" network regarding the connection of internal devices, we will only work at layer 2. Therefore, we will not use protocols such as IP, TCP or UDP because those correspond to higher layers of the OSI reference model.

As far as the medium access is concerned, all the different ECUs will be connected point to point with the switch, which has buffers to store ingress and egress data. Therefore, no collisions will happen when various devices try to transmit at the same time and, consequently, we will exclude the Carrier Sense Multiple Access and Collision Detection medium access from the project (CSMA/CD) and the Ethernet Collision Resolution: Backing Off and the *Truncated Binary Exponential Backoff* (TBEB).

What is essential about link layer in automotive networks are the Ethernet Frame format and the different types of frames associated. The different communicating protocols used in this project will be built over Link Layer, using 100/1000Base-T1 bus as a physical layer.

Ethernet Frame

To delimitate any type of frame, we need the Start of Frame (SOF) and the End of Frame (EOF). In the Ethernet protocol, the SOF consists of a preamble, which is a sequence of 7 bytes of alternated ones and zeros; and a Start Frame Delimiter (SFD), which is 1 byte identical to the Preamble but with the difference that the last two bits are "11". Therefore, every single Ethernet Frame will start with: 1010101010…(until 7 bytes) + 10101011.

Figure 3. Ethernet frame format.

The great advantage of this SOF is that the clock of the source device is being sent implicitly with the preamble and enables the destination device to synchronize with its partner. In the following image we can see the Ethernet Frame format.

Once the Start of Frame is sent, it is being followed by the destination and source MAC address, that are used to identify the device that the message is going for and its sender. The MAC address is formed by 6 bytes, the first three are used to identify the organization, and the last three are used to identify the device.

Continuing with the frame, we have 2 bytes that contain the type of data and, after these bytes, the payload (from 46 to 1500 bytes) that has all the useful information to be transmitted. To end with the frame, a Frame Check Sequence (FCS) is being carried out to detect if any transmission error has occurred. The mechanism used to check the errors is the Cyclic Redundancy Code (CRC). The End of Frame consists of a period of time during which no device will be transmitting.

Type Field	Description
0x0800	Ipv4 packet
0x0806	Address Resolution Protocol (ARP) packet
0x22F0	Audio/Video Transport Protocol (AVTP)

0x86DD	Ipv6 packet
0x8100	VLAN packet
0x88F7	Precision Transport Protocol packet
0x88E5	MACsec packet

Table 1. Types of Ethernet Frames

2.3.2. Virtual Local Area Networks (VLANs)

A Virtual Local Area Network (VLAN) is a network which is characterized by being software-independent from other networks that are connected to the same physical device.

To differentiate and identify where the incoming frames need to be forwarded, a 4-byte tag is being added before the Ethernet type. The first part of the tag is formed by 2 bytes, which in a VLAN-type frame is always 0x8100. The second part of the tag is divided into the following parts: Priority Code Point (PCP), 3 bits to establish the priority of the frame, being 7 the highest priority; Drop Eligible Indicator (DEI), 1 bit to identify if the frame is suitable for being dropped (1) or not (0) in case of congestion; and the VLAN ID, 12 bits to distinguish in which Virtual LAN the frame needs to be forwarded.

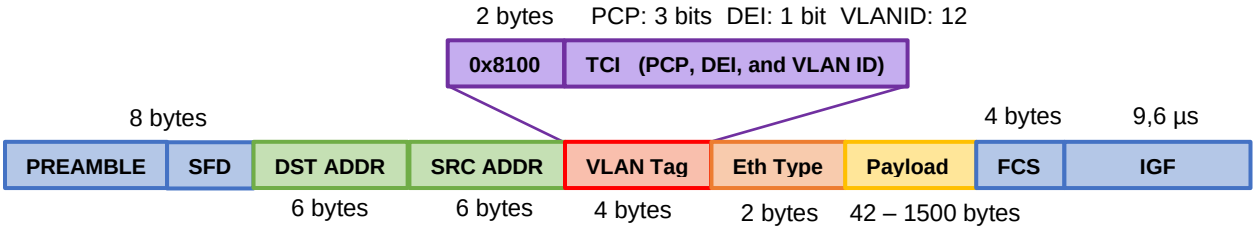


Figure 4. Ethernet frame format with VLAN Tag.

Note: since VLAN ID is formed by 12 bits, we have up to 4096 possible combinations of Virtual LAN identifiers. In the following table there is a summary table of the VLAN tag. Apart from this tag, the rest of the parts of the frame are kept the same. In the following image there is the structure of a VLAN frame:

Field Name (within Tag)		**Size**	**Description**
Type (TPID)		2 bytes	It's the tag protocol identifier, to indicate the frame that is being sent is tagged. The tag type has a value of 0x8100.
Tag Control Information (TCI)	Priority Code Point (PCP)	3 bits	Indicate 8 levels of priority being the 7 the higher priority and the 0 the lower priority.
	Drop Eligible Indicator (DEI)	1 bit	When this flag is set to 1, indicates that a frame has been marked as suitable for being dropped (discarded) in case of network congestion.
	VLAN identifier	12 bits	Values can be from 0 to 4096. It's the identifier of the virtual LAN. The numbers 0, 1 and 4095 cannot be used, however.

Table 2. Structure of the VLAN Tag.

2.3.3. Ethernet as the base for other protocols

The network architecture of a car can be considered as a closed local network. Consequently, there is no need to route information between devices (routers) or use protocols from layer 3 or above such as IP, TCP, UDP and so on. The automobiles architecture is based on Ethernet layer (layer 2 of the OSI model) and the rest of the involved protocols are built above this layer.

In the following chapters of this section, I mention and explain the main protocols involved within a car network that are related to Automotive Ethernet. Those protocols will be:

- Generalized Precision Time Protocol (gPTP).
- Audio Video Bridging (AVB).
- Stream Reservation Protocol (SRP).
- Forwarding and Queuing for Time Sensitive Streams (FQTSS).
- Security Encryption at Layer 2 (MACsec).

2.4. Time Synchronization (gPTP)

When it comes to Time Sensitive Networking, delays as low as 10 milliseconds can be noticed by people. Therefore, in real time systems we need some sort of synchronization.

To solve this previous problem, in 2000 the IEEE 1588 was developed, which was also known as Precision Time Protocol (PTP). This protocol was initially invented so as systems could distribute accurate timing within networks, as well as selecting sources of synchronization (known as grandmaster clocks).

Although PTP was initially created to synchronize large mobile networks, it became a quite complex protocol since those networks have large asymmetrical delays. Due to this, in 2006, another IEEE standard was created known which received the name of IEEE 802.1AS or the generalized Precision Time Protocol (gPTP). This protocol is built above link layer.

Indeed, gPTP is a "sub-protocol" of PTP, but it reduces many of the complex configurations and makes it an easier protocol to manage. Furthermore, it is being used in Time Sensitive Networking, especially in Audio and Video Streaming Synchronization.

2.4.1. gPTP network structure

In every Time Sensitive Network there are three main elements which participate so as the messages can be sent from one device to another. Those elements are:

1. **Grand Master (GM):** is the device that has the most precise clock in comparison to the clocks of the other hardware devices. This GM is responsible for transmitting the time (PTP messages) to the rest of devices within the network.
2. **End Points:** they receive the synchronization source from the grandmaster and they update their clock according to the GM messages. Those endpoints are hardware devices that have less precise clocks than the GM's clock.
3. **Bridges:** they are responsible for propagating the gPTP messages from the grandmaster to the end points. Intelligent switches have the capability of adding a correction factor to the time offset, which is related to the time that is required for the switch to receive, identify and forward the message to another device.

Apart from those previous elements, in every Time Sensitive Network there are three PTP clock types which can be defined. Those are the Ordinary Clock, Boundary Clock and Transparent Clock. Furthermore, each clock has a specified identity (Clock ID).

1. **Ordinary Clock:** this clock is an endpoint clock which can as a master or slave. An ordinary clock can be classified according to three features: *Slave only clock* (i), the endpoint clock will act as a slave; *Preferred grandmaster* (ii), which always acts as a master; and *Master clock or Slave clock* (iii), which can be either master or slave. This last third case normally acts as a slave unless there is no better clock within the network.
2. **Boundary Clock:** the concept of a boundary clock is quite different. It consists of an ingress port, which acts as a slave (CS) from a grandmaster source and the egress port, which acts as a master (CM) for an endpoint connected to that port. Once the PTP message is being passed through the switch, the clock ID is being modified and overwritten with the switch clock ID. Therefore, the grandmaster clock ID and the offset to the GM are not visible from the endpoint.
3. **Transparent Clock:** with a transparent clock, the grandmaster's identity and the offset are being propagated. With this type of clock, we can have two types of connections.

a. *End to End*: bridge only adds the correction factor (offset) of its own. The clock ID is not being modified.
b. *Peer to Peer*: the bridge adds the correction factor and modifies the clock ID.

In the following image there the structure of an AVB network clock hierarchy.

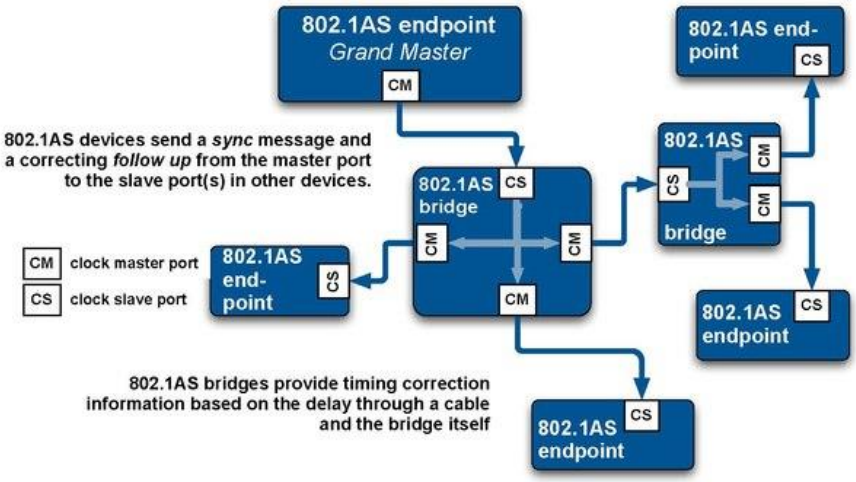


Figure 5. AVB clock hierarchy.

As far as this PTP protocol is concerned, it has a great advantage regarding the cost of the network. If this protocol was not implemented, every single device within the Time Sensitive Network would require having an extremely precise clock. The more precise a clock is, the greater the cost is and, therefore, the overall cost of the network would increase.

Nevertheless, the Precision Time Protocol solves this problem. Only one precise clock is required within the network which transmits PTP messages with time stamp. Consequently, the overall cost of the network is reduced.

2.4.2. Time accuracy and BMCA algorithm

The internal oscillator of an electronic device is not perfect. External factors, such as temperature, radiation and others, might cause the clock to slightly deviate from its perfect accurate value. Consequently, every single oscillator (clock) has associated an error value which can be calculated with the Allan Variance as shown below .

$$\sigma_y^2(\tau) = \frac{1}{2\tau^2}[(x_{n+2} - 2 \cdot x_{n+1} + x_n)^2]$$

In case we desire to calculate the Allan variance for N samples, we just need to do a finite sum and scale it as follows:

$$\sigma_y^2(\tau, N) = \frac{1}{2\tau^2(N-2)} \sum_{i=0}^{N-3} (x_{n+2} - 2 \cdot x_{n+1} + x_n)^2$$

Assuming the oscillator is perfect and the clock keeps an extremely precise time, the inside of the parenthesis in the previous formula would be null. Therefore, we can identify and select which device is suitable for being the grandmaster clock (the one that has the lower variance value).

To select the grandmaster clock, there is a special algorithm known as Best Master Clock Algorithm (BMCA), which chooses the device with the highest precision (lowest variance). The selected device will serve as a source of synchronization (grandmaster or GM) for the other devices within the network.

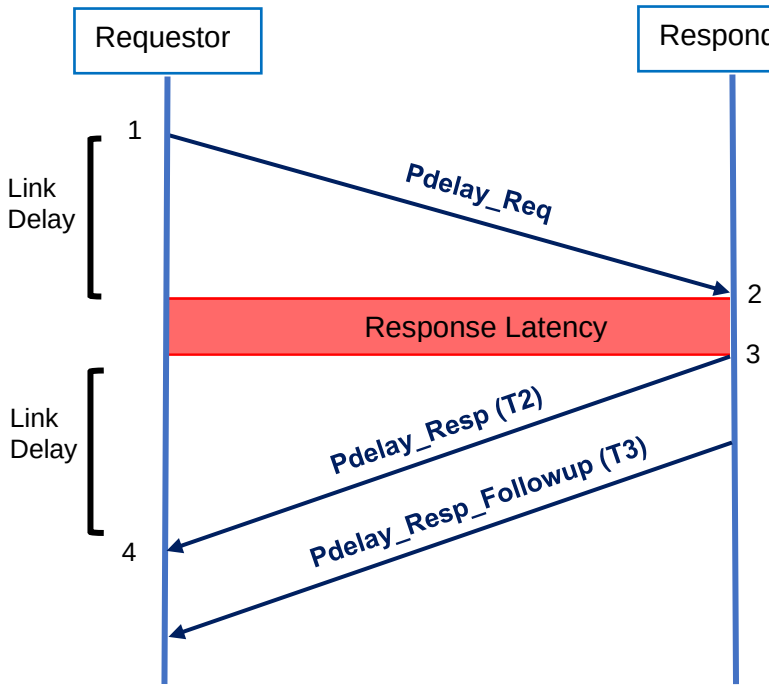
As clocks count time, they can **wander** (delay slightly in time) or **jitter** (advance slightly in time). Jitter can be defined as the short-term variation of the clock and wander as the long-term variation. Those variations are compared with the grandmaster's clock, which is the "ideal" time source.

Once the grandmaster is selected, it starts sending information (gPTP messages) about its time so as to correct the jitter and wander of the clock of the other devices. Those messages are the Announce Message, Sync Message and the Follow Up message, which we will see later on.

2.4.3. Link delay

When a message is transmitted from the master to the slaves, it may be several hops away, which will lead to inaccuracy of the transmission of the message. Therefore, we need to consider the delay of the link between endpoints.

To calculate the link delay, we will use what is known as a two-step clock. The Requestor sends a message (Pdelay_Req) with the time it was sent (T1). Then the Responder will answer with another message (Pdelay_Resp), that will contain the time it received the first message (T2). A third message is sent (Pdelay_Resp_Followup) that contains the time that the Pdelay_Resp was actually sent (T3). Finally, the first device gets the message and saves the time (T4). With those four times we are able to calculate the link delay (LD).



$$\mathbf{LD} = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

Figure 6. Diagram that shows the message structure used for the Link Delay (LD) calculation.

In order to guarantee that different devices emit (in case of audio) the samples at the same time, there will need to be a synchronization. The first endpoints that receive the sample will have to wait until the last devices have gotten the frame in order to emit. Otherwise, we'll find ourselves with audio incurrences. That is why is important to keep in mind the link delay between devices.

The three messages that are being sent have the structure shown in the following tables. They share a common header which will be furtherly explained in chapter 2.4.4.

Pdelay_Req Message	Bytes	Description
Header	34	Explained in chapter 2.4.4.
Reserved	10	Not used.
Reserved	10	Not used.

Table 3. Delay Request Message Structure.

Pdelay_Resp Message	Bytes	Description
Header	34	Explained in chapter 2.4.4.
Receipt Timestamp	10	Corresponds to the T2 time in fig 7. The 10 bytes are divided: 6 bytes for seconds and 4 bytes for nanoseconds.
Requesting Port Identity	10	This field contains the Source Port Identity of the node that this message is responding to.

Table 4. Response Message Structure.

Pdelay_Resp_Follow_Up	Bytes	Description
Header	34	Explained in chapter 2.4.4.
Response Origin Timestamp	10	Corresponds to the T3 time in fig 7. The 10 bytes are divided as in the Pdelay_Resp Message.
Requesting Port Identity	10	The port identity of the node which sent the first message: Pdelay_Req.

Table 5. Response Follow Up Message Structure.

2.4.4. GPTP message structure: Header, Body and TLV

The gPTP messages are defined by the **Ethernet Type 0x88F7**. All the messages that are being sent are built over the Link Layer and, therefore, the Data field of the Ethernet frame contains three main parts: Header, Body and a last parameter which can be Zero or a Type Length Value.

Header	Bytes	Description
Message Type	4 bits	Determines the specific type of the message: **Sync:** 0x0 **Pdelay_Req:** 0x2 **Pdelay_Resp:** 0x3 **Follow_Up:** 0x8 **Pdelay_Resp_Follow_Up:** 0xA **Announce:** 0xB **Signaling:** 0xC
Transport Specific	4 bits	This field exists to keep the compatibility with the IEEE 1588. Even though, this value should always be 0x1.
Version PTP	4 bits	Contains two different versions of PTP. In gPTP this value equals to 0x2.
Reserved	4 bits	Not used.
Message Length	2 bytes	The length of the gPTP message, including the header and the Type Length Value (TLV)
Domain Number	1 byte	The domain number for gPTP is always 0x0.
Reserved	1 byte	Not used.
Flags	2 bytes	Check IEEE 802.1AS standard for more information.
Correction Field	8 bytes	This field contains the correction factor to the timestamp that is being propagated from the grandmaster to the endpoint.

Reserved	4 bytes	Not used.
Source Port Identity	10 bytes	Consists of 8 bytes of Clock identity and 2 bytes of Port Number.
Sequence ID	2 bytes	Due to the fact that various messages can be sent repeatedly, this field is used to match pairs of Sync and Follow_Up messages. The first Folow_Up message cannot be mixed with the second Sync messages, and this field is used to differentiate them.
Control	1 byte	Depending on the type of message, this control field equals to: **0x00:** Sync message **0x02:** Follow_Up **0x05:** Announce, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
Log Message Interval	1 byte	The messages are transmitted periodically. The rate at which those messages are transmitted is included in this header field. The number of this field will be equal to the logarithm (base 2) of the real value. Examples: 5 → 25 = 32 -2 → 2^{-2} = 1/4 = 0,25

Table 6. Header format of the gPTP messages.

The Body is determined by the message type (Announce, Sync and Follow Up) and it has a static structure which will be shown in the following chapters 2.4.5. and 2.4.6.

The Type Length Value field (TLV) can be either zero or contain a TLV. If this table is being added at the end of the frame, then will have the structure below.

Name	Bytes	Description
TLV Type	2 bytes	Determines the specific type of TLV.
Length field	2 bytes	It contains the length of the TLV.
Value field	Length Field	Data of the TLV

Table 7. Type Length Value structure.

2.4.5. GPTP network creation: Announce message

To start the process of selecting the grand master clock, all endpoints must advertise their capabilities via Announce messages. Those messages will be furtherly used to create the gPTP Network domain.

To begin with the process, all endpoints begin to broadcast the Announce messages, which contain the essential information regarding the clock precision and other parameters. Once an endpoint receives an Announce Message from another device, it starts comparing the clock precision with its own. The comparison will result in whether the device has a better or worse clock precision. In the end, if it turns out to be a worse clock precision, the device will stop broadcasting the Announce messages and, eventually, only one device will remain which will contain the most precise clock within the network and will become the grandmaster.

If the grandmaster leaves the network or another device joints it, the newer announce messages can lead to the election of a new grandmaster. In either case, once the grandmaster is selected, it starts to broadcast its time source. Which will be used to correct the jitter and wander of the other clocks in the slave endpoints.

Announce Message	Bytes	Description
HEADER	34 bytes	As seen in chapter 2.4.4.
Reserved	10 bytes	Not used.

<table>
<tr><td>Current Utc Offset</td><td>2 bytes</td><td>Is difference in seconds between the UTC and TAI. In 2012, this value turned to be 35. In 2016, this value became 36. Nowadays (2023) this value equals to 37. Each leap year, a new second is added to this value. Therefore, in 2024 will be 38.</td></tr>
<tr><td>Reserved</td><td>1 byte</td><td>Not used.</td></tr>
<tr><td>Grand Master Priority 1</td><td>1 byte</td><td>255 = the endpoint is not capable of being the grandmaster.
0 = reserved for management operations.
The rest of the values indicate the device capability of being the grandmaster. Lowest numbers indicate better chance of being the grand master clock.</td></tr>
<tr><td>Grand Master Clock Quality</td><td>4 bytes</td><td>It is formed by three subfields:
Clock Class (1 byte)
0-5: reserved.
6: a type of clock synchronized to a primary time source (GPS).
7: a clock that was initially synchronized with a primary time source but is no longer capable to synchronize with it.
8-247: reserved.
248: default value.
249-254: reserved.
255: this value means directly that is not capable of being the GM.

Clock Accuracy (1 byte)

<table>
<tr><td>00-1F</td><td>Reserved</td><td>0x27</td><td>100 µs</td><td>0x2F</td><td>1 s</td></tr>
<tr><td>0x20</td><td>25 ns</td><td>0x28</td><td>250 µs</td><td>0x30</td><td>10 s</td></tr>
<tr><td>0x21</td><td>100 ns</td><td>0x29</td><td>1 ms</td><td>0x31</td><td>> 10 s</td></tr>
<tr><td>0x22</td><td>250 ns</td><td>0x2A</td><td>2.5 ms</td><td>32-FD</td><td>Reserved</td></tr>
<tr><td>0x23</td><td>1 µs</td><td>0x2B</td><td>10 ms</td><td>0xFE</td><td>Unknown</td></tr>
<tr><td>0x24</td><td>2.5 µs</td><td>0x2C</td><td>25 ms</td><td>0xFF</td><td>Reserved</td></tr>
<tr><td>0x25</td><td>10 µs</td><td>0x2D</td><td>100 ms</td><td></td><td></td></tr>
<tr><td>0x26</td><td>25 µs</td><td>0x2E</td><td>250 ms</td><td></td><td></td></tr>
</table>

Offset Scaled Log Variance (2 bytes): this value is derived from the Allan variance of the clock.</td></tr>
<tr><td>Grand Master Priority 2</td><td>1 byte</td><td>This field is used to decide the grandmaster clock is case that the Grandmaster Priority 1 and Clock Quality are not enough to do so.</td></tr>
<tr><td>Grand Master Identify</td><td>8 bytes</td><td>Contains the Clock Identity:
First 3-bytes of MAC Address + (0xFF + 0xFE) + last 3-bytes

Example of Clock Identity: 08:00:27 + 0xFF 0xFE + 9D:1A:3C</td></tr>
<tr><td>Steps Removed</td><td>2 bytes</td><td>Contains the amount of hops the message has passed.</td></tr>
<tr><td>Time Source</td><td>1 byte</td><td>Contains the type of time source that will be provided by the GM.</td></tr>
<tr><td>Path Trace TLV</td><td>4+8N bytes</td><td>Contains the path that the Announce message took from source to its destination.</td></tr>
</table>

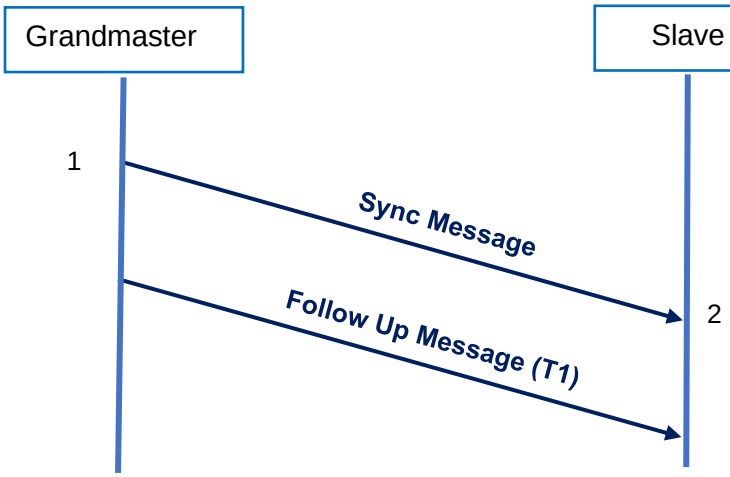
Table 8. Announce message structure.

IMPORTANT NOTE: the announce messages only make sense if the BMCA algorithm is caried out. As far as this project is concerned, the automobile is a closed network and, therefore, the grandmaster and slaves are pre-configured manually. Furthermore, the whole network needs

to be built up within fractions of seconds when the car engine starts, which leads to not using the BMCA and the Announce messages.

2.4.6. Offset synchronization: Sync and Follow Up messages

Once the grand master is established, SYNC messages are sent at a specific rate to keep all he clocks synchronized. The interval at which those messages are sent is configurable.



$$\textbf{Offset} = \mathrm{T2} - \mathrm{T1} - \mathrm{Link\_Delay}$$

Figure 7. Diagram of the Sync and Follow Up messages

The clock synchronization is a two-message process: the Sync and the Follow Up messages. This process of synchronization happens as follows.

1. The grandmaster prepares a Sync message and counts the **T0**, which is the ideal time at which the sync message should be sent through the Phy.
2. However, it takes time for the Phy to actually send the message, which will correspond to the difference between **T0 – T1.**
3. The message is actually sent at **T1**, which is annotated by the sender**.**
4. Once the slave receives the Sync message, will annotate the time **T2**
5. Finally, a Follow Up message is being sent with the information about **T1**.

Once all of those values (T1 and T2) are ready in the Slave, the calculation process begins according to the previous formula. The link delay has been previously calculated by the process explained in chapter 2.4.3.

NOTE: if there is a bridge is between the GM and the slave (a switch for instance), this device adds a correction factor to the offset value. This factor corresponds to the time that it takes for the switch to forward the message from the ingress port to the egress port.

The sync message has a pretty simple structure. It's just a message formed by the 34 bytes of the header and 10 bytes of reserved bytes.

Sync message structure	Bytes	Description
Header	34	Explained in chapter 2.4.4.
Reserved	10	Not used.

Table 9. Sync message structure.

Follow_Up message	Bytes	Description
Header	34	Explained in chapter 2.4.4.

Precise Origin Timestamp	10	This field contains the T1 in the previous image. Its format consists of 10 bytes (6 bytes for seconds and 4 bytes for ns).
Follow Up Information TLV	32	Extra information about the state of the gPTP domain. The structure of this field is: **1.** TLV Type (2 bytes): this value equals to 0x3. **2.** Length Field (2 bytes): This value will be 28. **3.** Organization ID (3 bytes): The value indicating the OUI for the standard. Value will be equal to 00-80-C2. **4.** Organization Subtype (3 byte): the subtype will be 1. **5.** Cumulative Scaled Rate Offset (4 bytes): it compares the ratio between the grandmaster's frequency to the local clock frequency. To calculate this number, the ratio is being subtracted 1 and then multiplied by 2^{41}. **6.** GM Time Base indicator (2 bytes): this field is to inform the other endpoints whether the grandmaster time base has changed or not. **7.** Last GM Phase Change (12 bytes): is the time between the previous grandmaster to the actual grandmaster. It is being expressed as an integral of 2^{-16} ns. **8.** Scaled Last GM Freq Change (4 bytes): is the frequency offset between the previous grandmaster and the actual grandmaster. This value is calculated with the offset, but the value scaled by 2^{41}.

Table 10. Follow Up message Structure.

2.5. MACsec encryption

The Media Access Control Security (also known as MACsec) is defined by IEEE 802.1AE standard and provides data protection at the Link layer of the OSI model. There are other alternatives such as IPsec, TLS, DTLS, SSL and so on which provide protection at higher layers. Indeed, the main advantages of MACsec are:

- Protects all types of data and their integrity.
- Safer architecture: *hop-by-hop*.
- Is faster than other protection methods such as DTLS, TLS and IPsec.
- Less key exchanges are required.
- Protects multicast and broadcast messages.

MACsec can be used in combination with those previously-mentioned protection methods. For example, IPsec provides point-to-point protection between the sender and receiver within a Wide Area Network (WAN), which can be combined with MACsec to protect the data in every hop the frame does between routers or switches.

Although it might seem redundant to combine MACsec with other protection methods, for instance IPsec, it is necessary to protect data and its integrity at the link layer, due to the fact that those protection methods that work at higher layers cannot protect layer 2 data. The only viable solution that can protect all types of data that are being transmitted is MACsec.

Furthermore, MACsec encryption is done by hardware, allowing the encrypting process to be faster than other methods. Therefore, the encryption performance is slightly linear to the Link speed, rather than IPsec which is limited at 40 Gbps approximately, because it done by Software.

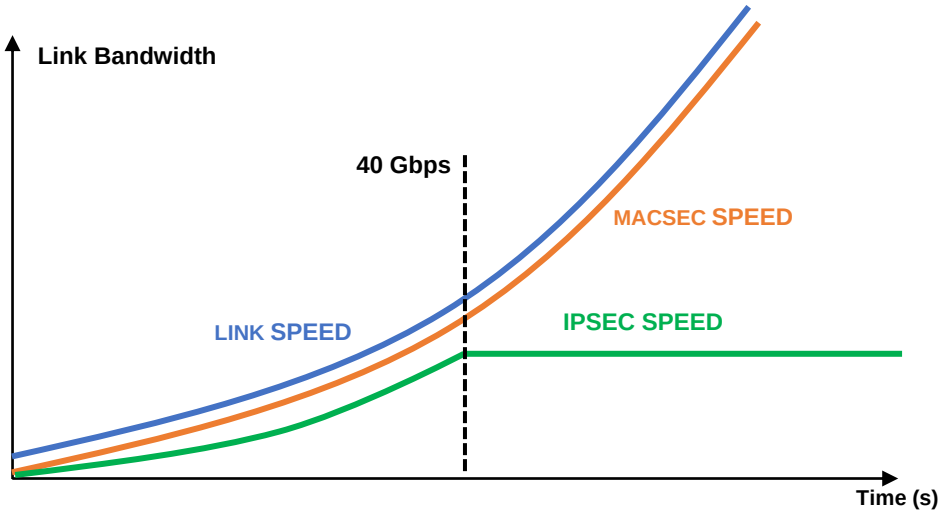


Figure 8. Comparison between Link, MACsec and IPsec speeds.

2.5.1. MACsec frame format

MACsec frame is based on the Ethernet Frame with the difference that an additional Security Tag and an Integrity Check Value (ICV) are being added. After the security tag and before the ICV, all the data is encrypted according to a specific SAK that is passed between sender and receiver.

Field Name		Size	Description
Security Tag	**Type**	**2 bytes**	This corresponds to a MACsec type frame which is 0x88e5.
	TCI	**6 bits (3rd octet)**	Tag Control Information (TCI): contains various information which are the MACsec protocol version and the flags regarding the only integrity mode of the only encryption mode.
	AN	**2 bits (3rd octet)**	Association Number (AN): those 2 bits identify four possible options of Security Associations (SA) within a Secure Channel (SC).
	SL	**1 byte**	Short Length (SL): Indicates the number of octets that are contained between the Security Tag and before the ICV.
	PN	**4 bytes**	Packet Number (PN): those bits are used by the AES-CMAC-128/256 alongside the Secure Channel Identifier (SCI) to construct the initialization vector. This field is also used to prevent replay attacks.
	SCI	**8 bytes**	Secure Channel Identifier (SCI): This field is optional. Each connection corresponds to a port, and each port is designated a specific SCI value so as to make the communication safer.
Integrity Check Value (ICV)		**8/16 bytes**	If the Security tag contains the optional SCI, this field will be 16-bytes long. Otherwise, it will be 8-bytes long. This field is used to check the integrity of the data as well as the MAC source and destination address.

Table 11. MACsec frame structure.

In the following image there is the MACsec frame format of a VLAN tagged frame. A normal Ethernet Frame will be similar with the difference that no VLAN tag will be present, and the payload will be from 46 to 1500 bytes.

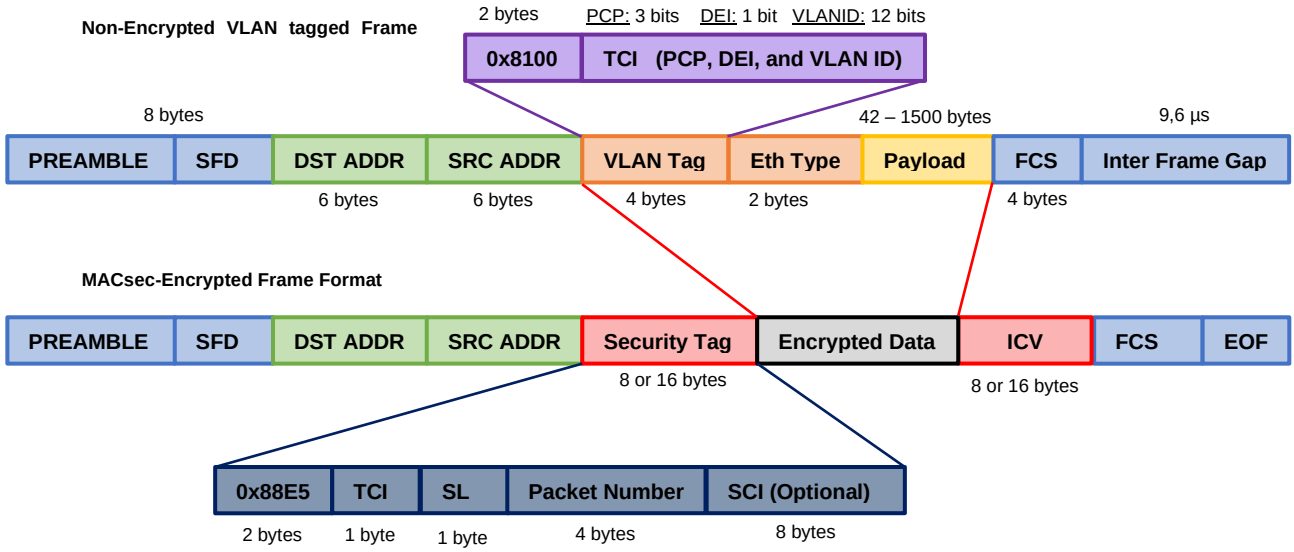


Figure 9. MACsec frame format.

2.5.2. How does MACsec work?

The first requisite before MACsec can be enabled is to establish a stable Ethernet connection (point-to-point) between two devices. After this, a specific *Secure Association Key* (SAK) is exchanged with which every single message passed between the endpoints is encrypted. Both devices need to know the same SAK so as they can encrypt and decrypt respectively the data they mutually send.

If this *Secure Association Key* was always the same, the system would be extremely vulnerable to external attacks. Consequently, the SAK is re-generated after a certain period of time with the use of another key known as *Connectivity Association Key* (CAK). The process by which the SAK is generated from the CAK is divided in two main processes:

- **Authentication:** generates the CAK key.
- ***MACsec Key Agreement* (MKA):** generates the SAK key with the previously generated CAK in the authentication process.

Once the *Secure Association Key* is generated, it is being exchanged between both devices and the encryption process begins. This process is shown in the following image.

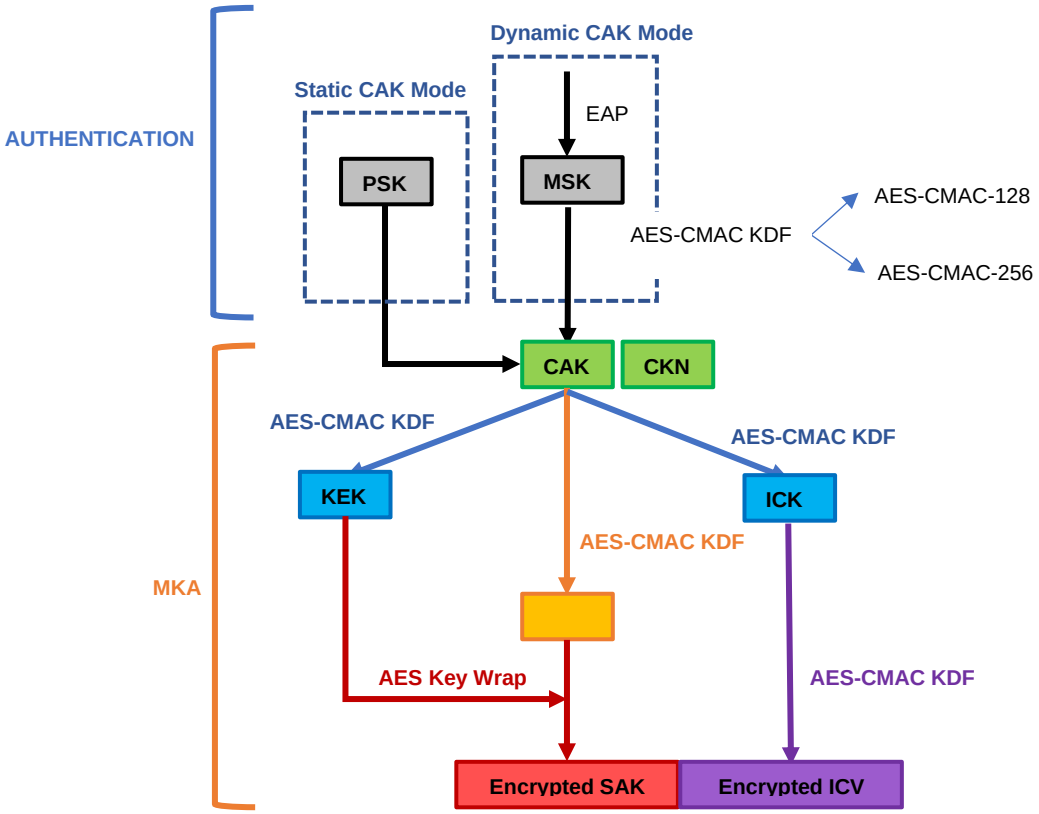


Figure 10. Diagram that shows the CAK and SAK generation.

5.2.2.1. Authentication

To generate the SAK key, a CAK key must be provided. Indeed, this CAK can be generated from two different approaches: Static CAK Mode or Dynamic CAK Mode. In every MACsec connection, one of the following two must be selected according to the security requisites of the application.

Static CAK Mode

This mode is typically used between switch-to-switch links. The authentication process starts with a *Pre-Shared Key* (PSK), that must coincide in both devices, and with a *Connection Association Key Name* (CKN), which must be entered manually in the configuration of both devices.

If this mode is used, the way of generating the CAK from the PSK is as simple as copying the Pre-Selected Key into the Connectivity Association Key.

Although this method is easier to implement, it is highly vulnerable to external attacks and it does not provide enough security to reach the Automotive Ethernet demands. Moreover, any device within the network that had the authentication *Pre-Selected Key* would be able to establish a successful connection. One last thing to consider is that the larger the network gets, the more difficult it is to maintain because it would require to enter the PSK and CKN manually in each device within the network.

Dynamic CAK Mode

As mentioned in the previous paragraph, the Static CAK Mode does not meet the required security in automobiles, which leads to the Dynamic CAK Mode. This mode is based on the IEEE 802.1X standard, also known as the *Extensible Authentication Protocol* (EAP). A few terminology concepts are associated to the EAP:

- **Supplicant:** the device that asks for the access to the network.
- **Authenticator:** the device that allows or denies the access of other devices into the network (a switch for example).
- **Authentication Server (SA):** server that determines if a supplicant is authorized to access the services of the authenticator.

Once a new device (supplicant) joins the network, the authentication process is required. Through this process the EAP will be used to generate a *Master Session Key* (MSK), which will be furtherly used to obtain the CAK key. In order to get the MSK, a series of messages need to be passed between supplicant and the authentication server.

The Dynamic CAK Mode is generally used with a host-to-switch connection, where the host will act as the Supplicant and the switch as the Authenticator. Therefore, is the supplicant wants to communicate with the Authentication Server (AS) it must firstly pass through the switch.

Firstly, the authentication process starts when the Supplicant sends an *EAPoL Start* to the Authenticator. The switch will response to that message with an *Identity Request*, which will force the Supplicant to answer with an *Identity Response*.

Secondly, the Authenticator will communicate the Supplicant Identity to the Authentication Server using RADIUS-type messages. Once this step is completed, it will start the negotiation between the Supplicant and the AS and, if all the process is correct, the server will respond with an *Access Accept* which will allow the host to join the network.

Ultimately, when the AS has accepted the Supplicant request and has joint the network, they both agree and exchange a *Master Session Key* (MSK).

To generate the CAK key from the MSK, the AES-CMAC KDF method will be used. AES stands for *Advanced Encryption Standard*, and KDF for *Key Derivation Function*. This last one is responsible for generating the CAK key with the use of two pseudorandom functions: AES-CMAC-128 and AES-CMAC-256.

Once the *Connectivity Association Key* (CAK) has been generated, the authentication process has finished and the *MACsec Key Agreement* (MKA) is the process that follows. All the devices that share a common CKN and CAK key constitute a *Connectivity Association* (CA).

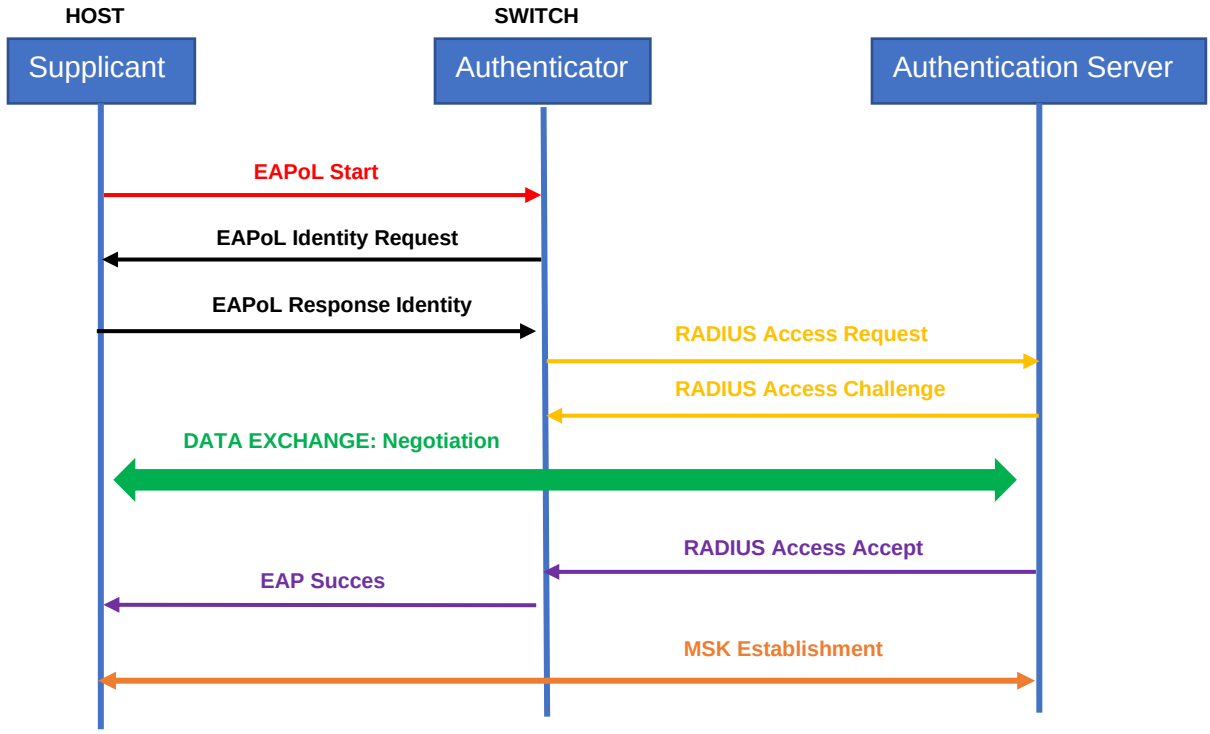


Figure 11. Master Session Key establishment.

5.2.2.2. MACsec Key Agreement (MKA)

Before starting with the MACsec Key Agreement, the election of a **Key Server** must be done. When two CA peers confirm that they share the same CAK and CKN, the process of electing the key server starts. Each station within the *Connectivity Association* (CA) sends a broadcast heartbeat which contains essential information: the Key Server Priority and a list of the active endpoints. Once all endpoints have the active endpoints list, the device with higher Key Server Priority will be chosen as the Key Server.

Once the Key Server has been selected, the key derivation process starts and two news keys are generated: the *Integrity Check Key* (ICK) and the *Key Encrypting Key* (KEK). Furthermore, a third key will be generated by the Key Server: the *Secure Association Key* (SAK). To generate those all of those keys the AES-CMAC KDF (128/256) method is used.

Once the SAK is created, it needs to be broadcasted to the other peers in the CA. However, before sending the generated SAK, it is being applied an encryption method known as AES Key Wrap, which uses the KEK generated key to encrypt the SAK key. Therefore, the Secure Association Key is being sent encrypted to the other members of the CA.

The ICK key is used to generate the Integrity Check Value (ICV). This ICV is being added to the MACsec frame (after the encrypted data) and its main function is to check whether the frame integrity has been compromised or not. If a frame is received and the ICV has some anomalies, then the frame is immediately dropped.

Regarding MACsec protocol, we can assure that not only protects the data from being readable (encryption with the SAK key) but also protects data from being modified and losing its integrity (with the ICV value).

5.2.2.3. Secure Association Key (SAK) regeneration

Once SAK has been successfully transmitted to the other devices, they can start sending information which will be unreadable to external attacks (since its encrypted). However, this method would fail if no SAK key regeneration was taken into consideration.

Therefore, the Key Server will regenerate and distribute a new SAK key as soon as the following scenarios happen:

1. A new device joins the *Connectivity Association*.
2. The maximum uses of the key have been reached. Depending on the *Cipher Suite* used, the keys can be a specific number of times.
3. The *Cipher Suite* changes.
4. A certain amount of time has passed (this time can be changed).

The **Cipher Suite** is the method used to encrypt the different packets that travel across the network. This method is chosen by the Key Server and is broadcasted with the SAK key (encrypted with the KEK). The possible Cipher Suites are the following ones:

1. GCM-AES-128 (Compulsory).
2. GCM-AES-256 (Optional).
3. GCM-AES-XPN-128 (Optional).
4. GCM-AES-XPN-256 (Optional).

2.6. Audio Video Bridging, AVB

In the first decade of the 21st century, having a camera or touchscreen in the car was quite unusual. In the following decade, however, vehicles began to be more complex and multimedia features were introduced. From then, consumer demands have increased and what was previously considered a luxury feature such as video streaming, it is nowadays a common fact which many people use daily.

Audio and especially video require high bandwidth transport mediums. Until that moment, the Control Area Network bus (also known as CAN) had been universally adopted for data transport within the automobile. Nevertheless, as soon as multimedia became a reality it was evident that CAN bus bandwidth was not enough.

Consequently, an Ethernet-based method was developed which adopted the name of **Audio Video Bridging** (AVB). A new concept of network was introduced, which could provide the sufficient bandwidth, Quality of Service (QoS) and keep the cable weight as low as possible.

Audio Video Bridging is not a protocol by itself. Instead, it consists of three foundational standards. Although each of these standards can be used independently, they need to work in coordination if an AVB network is wished to be created. These standards are:

1. Time Synchronization (gPTP)
2. Stream Reservation Protocol (SRP)
3. Forwarding and Queuing for Time-Sensitive Streams (FQTSS)

Those protocols must be working coordinately so as an AVB can be implemented. Furthermore, AVB englobes a certain number of features which include:

1. Priority Levels: some frames (such as PTP) can be prioritized above others.
2. Reservation: a certain amount of bandwidth can be reserved along the path so as the high-priority frames can be transmitted with as low latency as possible.
3. Time Managing Protocols: all the clocks must be synchronized

Apart from the previous features, there are others which are less relevant and have not been included. Moreover, AVB as a network structure has many benefits.

1. Lower cable weight and less fuel consumption.
2. Increased reliability and low latency since AVB is based on Ethernet.
3. Precise synchronization with the gPTP protocol.
4. Globally standardized and scalable.

2.6.1. AVB Network

An **Audio Video Bridging** network consists of several endpoints (Talkers and Listeners) that are connected into an AVB cloud. This cloud is mainly formed by switches that support AVB and some sort of connection that joins them. Since we are using switches, AVB packets work above the Link Layer of the OSI Model (layer 2). In the following image we can see an AVB network.

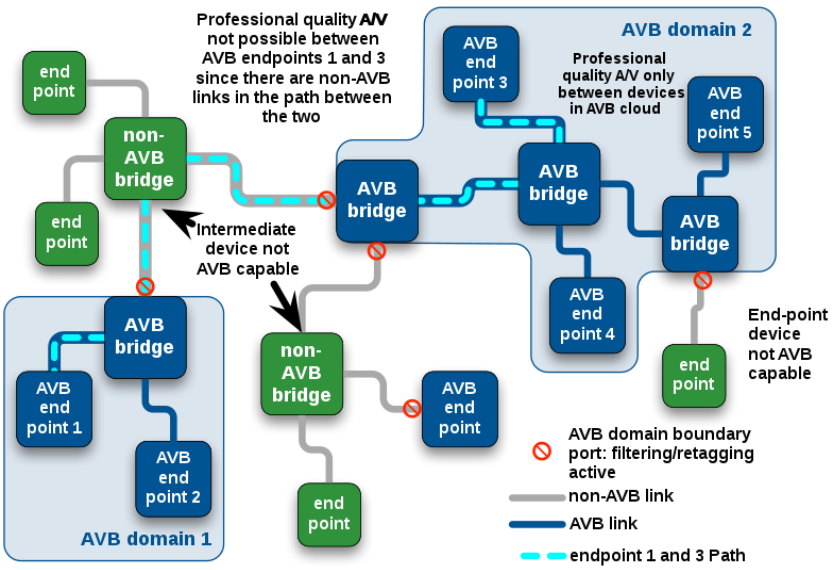


Figure 12. Image of an AVB network. This AVB network consists of several endpoints that can ack as Talkers, Listeners, or both. Furthermore, there are several bridges, which connect al the endpoints within an AVB cloud

A hardware device which is capable of transmitting AVB Streams is known as an Entiny. Those AVB Streams, which can be either audio, video or other types of frames, will always be sent from an AVB Source (Talker) to an AVB Sink (Listener). The AVB-capable bridges will forward those streams from the source to its destination as well as updatting an internal Multicast Forwarding Database (MFDB).

The **MFDG** or **Multicast Forwarding Database** is an internal parameter that the bridges have in order to prevent bandwidth from being wasted. If a frame is being forwarded to a port where no active Listener is connected, the effort of transmitting that frame will be in vain, leading to a waste of bandwidth. Therefore, the AVB-capable switches will contain this database so the device only forwards streams to where an active Listener is connected to, which saves bandwidth.

Each Audio Video Bridging Network has several domains. As not all the switches in the network will be AVB-capable, those streams will only be able to pass through the capable switches, thus creating a domain boundary. Other non-AVB frames such as gPTP messages can pass those AVB boundaries.

As well as gPTP, SRP and FQTSS, the AVB standard defines its own Transport Protocol to exchange audio and video streams. This protocol is the **Audio/Video Transport Protocol** (also referred as AVTP).

The gPTP has been explained in chapter 2.4. and, therefore, in this section there only be an explanation of the SRP and FQTSS.

2.6.2. Stream Reservation Protocol (SRP)

The Stream Reservation Protocol (SRP) is a protocol which reserves a specific fraction of the total's bandwidth of the path between the Talker and the Listener. This standard is referred as the IEEE 802.1 Qat and which guarantees that the least latency is present when a stream is transmitted from source to sink.

Only the required bandwidth of the path between the talker and listeners will be reserved. By default, the maximum amount of reservable bandwidth is the 75%. If any entity is requesting a specific bandwidth which is not available, the device will be notified before any path reservation is established.

The Stream Reservation Protocol is not a protocol as itself. Instead, it works using three other protocols which are:

1. **Multiple VLAN Registration Protocol (MVRP):** a protocol used to facilitate the VLAN treatment and automatic configuration within the switch.
2. **Multiple MAC Registration Protocol (MMRP):** this protocol will be used to register the different multicast MAC addresses in the different bridges (switches) across the network.
3. **Multiple Stream Reservation Protocol (MSRP):** this protocol as itself is the one used for taking care of the bandwidth reservation across the path between several endpoints.

Although the first protocols may be used in many networks, they are not really being used in an Automotive Ethernet network. This is due to the fact that VLANs and the different multicast MAC addresses are being manually entered in the switch configuration. Therefore, the only protocol which will be taken care is the Multiple Stream Reservation Protocol (MSRP).

All the three protocols mentioned above, are built above a base protocol known as **Multiple Reservation Protocol** (MRP). In the diagram on the right there is the structure of how the previously-mentioned protocols are structured on top of each other.

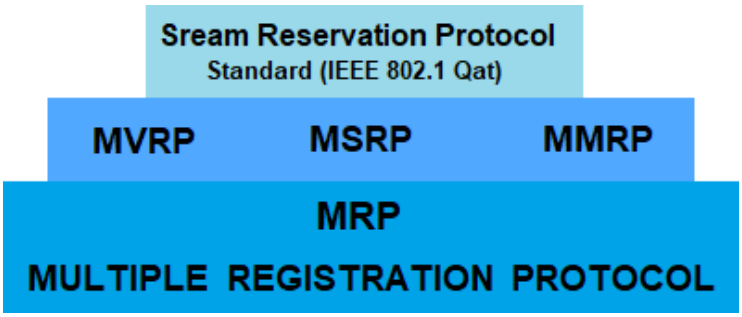


Figure 13. SRP Protocols Structure. MSRP needs MRP.

The reservation process of a stream needs to be done in three main steps:

1. **Establish a Domain.** Every single domain within an AVB network is defined by some boundaries, which are established physically by simply including a non-capable AVB switch in the path.
2. **Talker Advertises a Stream (*Talker Advertise*).** The talker sends a broadcast message in order to discover who would like to listen to his streams. When the message is received in the bridges, it is being replicated to every port that has at least one active

listener. The talker advertise stream contains the different requirements: bandwidth, maximum latency, QoS and so on.

3. **Listener(s) Attach to the Stream (*Listener Ready*).** The device that wants to listen to that stream sends a unicast response to the talker. Once this message has reached the talker's destination, it can begin transmitting.

2.6.2.1. Multiple Reservation Protocol (MRP)

The Multiple Reservation Protocol, also referred as the IEEE 802.1ak standard, is a protocol used for propagating attributes through a LAN. Since all the endpoints within the network need to share the same attributes, whenever a new attribute is declared or withdrawn in a single device, it must be propagated to the rest of the endpoints. Each device is internally responsible for holding all the attributes (i), updating them when a modification is detected from another device (ii), and sent a message to the other devices when an attribute of its own is modified (iii).

To maintain, update or broadcast the attributes, each endpoint must internally hold 4 state machines that take control of the previous functions. Those are the following.

State Machine	Description
Applicant	This state machine has the function of declaring new attributes and propagating them to the other devices connected to the network.
Registrar	This state machine is responsible for recording all the different attributes in the device. It does not propagate them however.
Periodic Transmission	It generates certain periodic events which are used for the other state machines in the device.
Leave All	This last state machine is responsible for preventing the state machines from having an extended failure. Furthermore, it forces other participants to re-register the attributes after a period of time so as non-used events are not kept registered is they are not used.

Table 12. State Machines of the MRP.

Each participant maintains these four state machines. If by any case a device wants to include or remove an attribute, this change must be forwarded to the other devices connected to the network.

2.6.2.2. Multiple Stream Reservation Protocol (MSRP)

The Multiple Stream Reservation Protocol or SRP is based on the previous MRP. The process is the same: different set of attributes are being declared or removed and then propagated to the other devices in the network (and this is when MRP is needed). However, the difference that MSRP introduces is that the attributes are specifically of the bandwidth reservation type.

The Streams can be individually identified by their *StreamID*. This identifier is divided into 48-bit field which corresponds to the MAC address of the device that transmits the stream and the 16-bit field which contains an identifier used to differentiate streams that come from the same Talker.

The reservation process takes place in three different steps mentioned before: establish the domain (i), talker advertising (ii) and listener ready (iii). To do those steps, several types of messages need to be sent: *Domain*, *Talker Advertise*, *Talker Failed* and *Listener Ready*.

Domain

These messages are used for delimiting boundaries in the TSN network. To explain in simple terms, the domain message is used for "telling" the devices within the network that a particular VLAN will be used for streaming AVB messages. With the domain message, the type of AVB streams that will be furtherly transmitted is also included as well as its priority. The selected VLAN ID field will indicate that all the incoming frames will be tagged with that specific VLAN identifier. In the following table there is the structure of the domain message composed by 4 bytes.

Name	Bytes	Description
SRclassID	**1**	It defines the class of the AVB stream. 0x6: Class A stream. 0x5: Class B stream.
SRclassPriority	**1**	Indicates the stream priority. The default priorities are: 011 (3): Class A 010 (2): Class B
SRclassVID	**2**	Indicates the VLAN ID with which the AVB streams will be tagged.

Table 13. Domain message structure.

The main differences between classes A and B are the following:

Class A: tighter latency requirements. Smaller packets which are transmitted frequently. [Interval of 125 μs or 8000 packets/s].

Class B: streaming packets, with lower latency requirements than class A. Those packets are bigger but transmitted at a lower frequency. [Interval of 250 μs or 4000 packets/s].

Talker Advertise

Before streaming AVB messages and reserving any bandwidth, this message type will be sent from Talker to Listener(s). Alongside the message, the specific requirements such as bandwidth reservation, Quality of Service and others will be specified. It has the following structure:

Name	Bytes	Description
StreamID	**8**	Eight bytes divided into two main subfields. A 6-byte MAC address corresponding to the Talker device and a 2-byte subfield to differentiate streams from the same talker.
Destination Address	**6**	The destination MAC address in unicast form (1 Listener) or multicast form (various Listeners).
VLAN Identifier	**2**	The VLAN identifier.
Tspec Max Interval Frames	**2**	Contains the maximum rate (packets per second) at which the stream is transmitted. For class A is 8000 packets/s.
Priority and Rank	**1**	It is used to establish the priority of the stream. This priority will be used is case that multiple streams are being transmitted at the same time. The rank is used to distinguish between streams that carry emergency data (such as emergency calls) or normal streams.

Table 14. Talker Advertise message structure.

Talker Failed

If a Talker Advertise encounters any problem such as no sufficient bandwidth available, no available connection to the Listener or others, the Talker advertise message will be converted

into a *Talker Failed*. The structure of this message will be the same as the previous one but with two additional fields.

Name	Bytes	Description
Talker Advertiser Frame	**24**	Including all the parameters in the table above (Stream ID, Destination Address, VLAN identifier, Tspec Max Frame Size Tspec Max Interval Frames, Priority and Rank and, finally, Accumulated Latency).
Bridge IP	**8**	Contains the ID of the switch that faced the problem and converted the Talker Advertise to Talker Failed.
Failure Code	**1**	It is the code that contains why the problem has been encountered.

Table 15. Talker Failed message structure.

The most common Failure Code errors are:

1. **0x1:** Insufficient bandwidth.
2. **0x2:** Insufficient bridge resources.
3. **0x4:** StreamID used for another Talker.

Listener

When a Talker Advertise is received by a Listener then is means that there is sufficient bandwidth across the path between source and sink. The Listener sends this message with the following structure:

Name	Bytes	Description
Stream ID	**8**	Is the same Stream ID as the stream that was sent by the Talker.
Type	**1**	Depending on the value it can be 4 different types: **Ignore (value = 0)** **Asking Failed (value = 1):** every single path to the Listener(s) has insufficient resources to establish a bandwidth reservation. **Ready (value = 2):** every path from source to Listener(s) has enough resources and has already established a bandwidth reservation. **Ready Failed (value = 3):** At least one path to a Listener(s) has insufficient resources to establish a bandwidth reservation.

Table 16. Listener message structure.

2.6.3. Forwarding and Queuing Enhancements for Time Sensitive Streams (FQTSS)

The Forwarding and Queuing Enhancements for Time Sensitive Streams (FQTSS) is a standard protocol that is used alongside the Stream Reservation Protocol. This standard is also referred as the IEEE 802.1 Qav which was mainly created for Traffic Shaping of the network.

Since the switches may not be able to forward the messages at the speed-rate they receive them, they need to store them in some way. Therefore, for FQTSS to work it is firstly required to have some ingress and egress buffers to store the frames. It is not established how many buffers each port needs to have; it depends on the switch producer.

All the traffic within the automobile network will be tagged with a specific VLAN, which involves a specific Priority Code Point (PCP). When a frame arrives at a certain port, it will be stored in one of the ingress buffers of that port according to the PCP.

The switch will firstly forward the messages in the higher queues and the move on to the lower queues (queue 7 will be the most prioritizing one and queue 0 the least prioritizing one). Even so, there are specific switch configurations which can slightly modify this order. In the following image there is an example of the port layout with the queues.

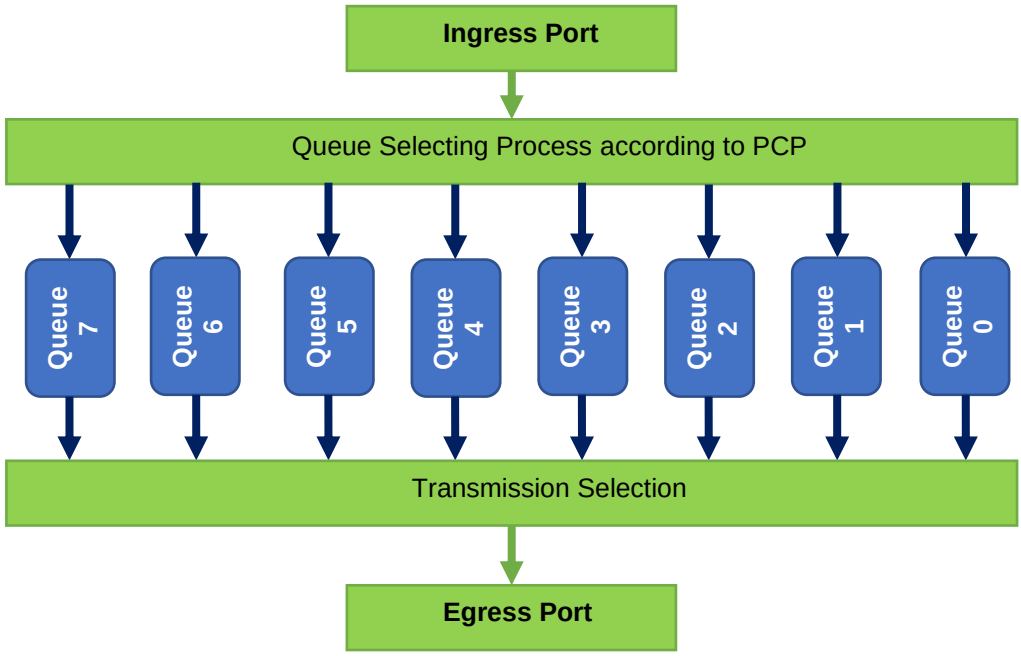


Figure 14. Layout of the queues. In this case we are using 8 different queues, one for each PCP code.

AVB Streams need to have some sort of pacing rules. Let's suppose that a constant flow of AVB frames were send from a talker with a PCP = 7. According to the previous switch rules, all of those frames would be forwarded before the other frames, which could lead to a potential bandwidth waste that could prevent other messages important from being forwarded (such as PTP).

Consequently, a method is developed so as to ensure the Quality of Service (QoS) as well as the pacing rules. This method is known as **Credit Base Shaper** (CBS).

The CBS consists of a simple variable number, the credit, which increases or decreases if an AVB frame is being sent or not. The credit is a number, which correspond to the amount of AVB bytes that can be sent through the port.

This Credit Base Shaper works as follows:

- AVB frames can be sent as long as the credit is 0 or higher.
- When a frame is sent, the credit reduces according to the amount of bytes sent per second (sendSlope).
- If credit reaches negative numbers when a stream is being sent, that stream finishes its transmission and no other AVB stream will be sent until the credit reaches 0 or above.
- If the credit is negative, no AVB streams can be sent (they need to wait in the queue).
- If credit reaches 0 and no AVB frames are ready to be sent, the credit accumulates according to an idleSlope until a specific value is reached (High Limit). If this value is reached, credit no longer accumulates.

To calculate the iddleSlope and the sendSlope we will use the following formulas

$$\mathbf{idleSlope} = \frac{\text{reservedBytes}}{\text{classMeasurementInterval}} = \frac{(\text{perFrameInterval} + \text{MaxFrameOverhead})}{\text{classMeasurementInterval}}$$

$$\mathbf{sendSlope} = \text{iddleSlope} - \text{portTransmitRate}$$

In the following image there is an example of the Credit Based Shaper that is used to decide whether AVB messages in the left queue can be sent or not. The orange message is a non-AVB frame and, while it is being sent, the credit goes up in positive. When the blue AVB frame is sent, the credit goes below 0 and, therefore, the following blue AVB frames frame cannot be sent until the credit gets to 0. When we send the purple and red messages, the credit reaches its limit, where it does not increase anymore.

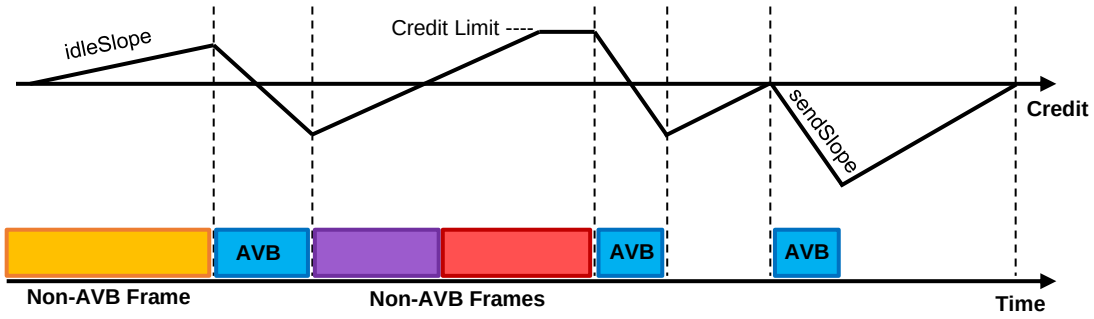


Figure 15. Diagram of the Credit Base Shaper algorithm.

3. DEVELOPMENT OF THE PROJECT

This part consists of the main body of the research project. In the first chapter I explain the development of the initial testbench, its requirements and the achieved results. In the second chapter, I expose the selection process of the switch, its main features and the software and hardware analysis of the board itself. Following this, in the third chapter, I outline the different capabilities that the switch has and the implementation of the different functionalities, which include Virtual LANs, Time Sensitive Networking (gPTP), Audio Video Bridging and MACsec. In the final chapter of the section, I expose the achieved results.

3.1. Testbench Development

This testbench will be used to verify that the switch configuration works properly. The testbench consists of a Server-Client communication, which involves VLAN tag, MACsec and frame generation with a Python extension known as Scapy. This testbench will be faced from two different approaches: a virtual approach, using virtual machines and a virtual switch; and a physical approach, using a real switch and Raspberry Pi's as hardware endpoints.

NOTE: this testbench will only be used for testing purposes with the physical switch. It will not be used in real life because it involves TCP, which makes the communication slower.

Consequently, the different elements used in this testbench (virtual and physical) are:

1. A laptop with Oracle VM Virtual Machine installed. Create two Ubuntu 18.04 machines and install in each one the following programs: Wireshark, Visual Studio Code (or similar), Python interpreter and Scapy extension.
2. A transparent and basic switch.
3. Two Raspberry Pi's with Raspbian OS installed in them.
4. Three full-duplex 100-BaseTX Ethernet cables.

Wireshark: is the application used to analyze all the different TCP/IP segments that are being sent through the network. This app has also the capability to revise each layer of the TCP/IP model separately, which makes a great advantage to debug segments.

Oracle VM Virtual Machine: this Windows application will run the virtual machines.

Scapy: although it's not an application, it needs to be installed as a Python extension. Scapy is allowed to generate custom TCP/IP packets and send them to the network. The use of this extension is mainly for testing purposes, since we can send a fake message coming from a device which is not even connected to the network.

Virtual Server-Client(s) Testbench

Using the Oracle Virtual Machine (VM), several Ubuntu 18.04 machines are created, and they are connected within the same Internal Network. No possible interaction with the organization's Network has to be allowed to the virtual machines and, therefore, if any problem happened it would not affect the rest of the workers in the organization.

Of all the virtual machines, one of them will be used as the server and the others as the different clients requesting to connect and send messages to the server. All the virtual machines will be virtually connected between them with the *NAT Network* configuration in the Oracle VM.

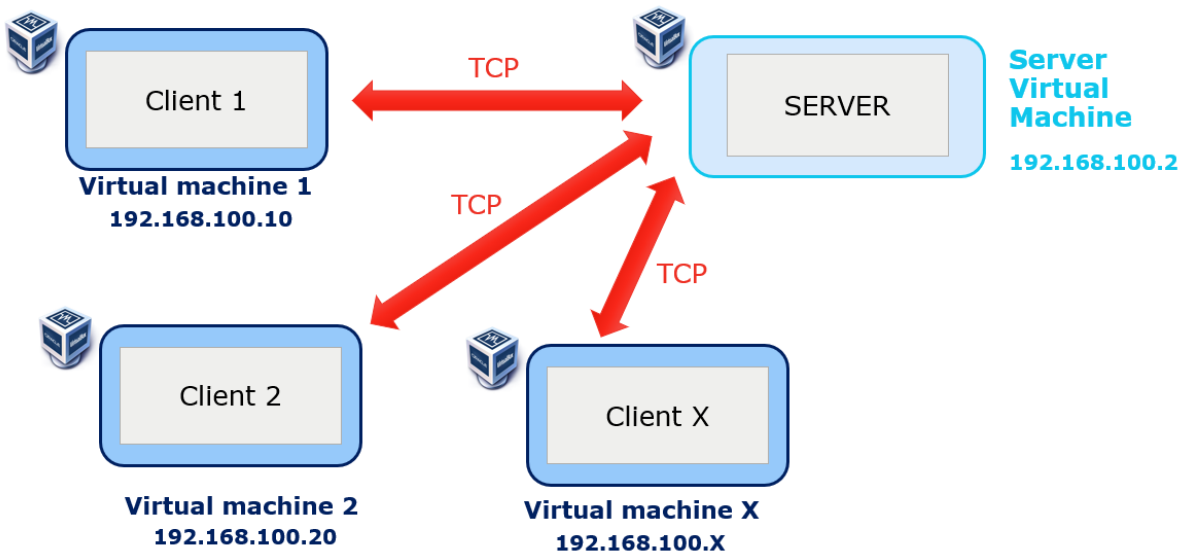


Figure 16. Diagram of the Network structure used in this test. Each virtual machine will be identified with the last number in the IP address and the protocol used for transmitting messages is TCP.

Each virtual machine must be configured with a different IP address (as shown in the figure above). All of them must be connected to the same virtual NAT network and its Classless Inter-Domain Routing (CIDR) will correspond to 192.168.100.0/24, which means that our network can contain up to 255 devices connected. In this test, however, we will only use three.

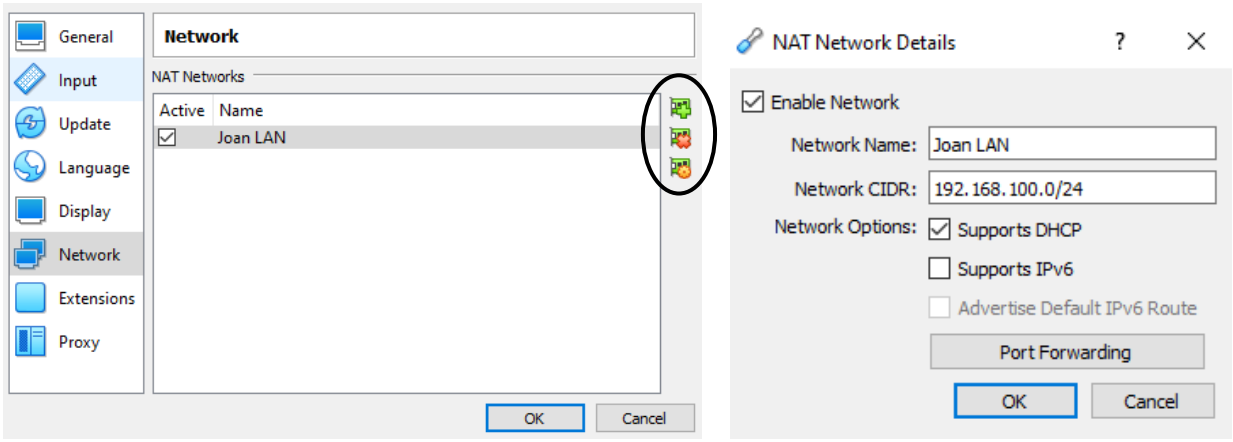


Figure 17. Virtual Machine NAT Network configuration.

Since the NAT network supports DHCP, there is no need to configure each virtual machine IP. Nevertheless, if the *Dynamic Host Configuration Protocol* is not selected, the address has to be configured manually with the following command:

STATIC IP CONFIGURATION UBUNTU 18.04

```
sudo gedit /etc/network/interfaces
sudo ifup enp0s3        # Bring Up the network card
sudo ifdown enp0s3      # Shut down the network card
```

Text to add in /etc/network/interfaces

```
auto enp0s3      #The enp0s3 is the Ethernet card, change it depending on your device
iface enp0s3 inet static
address 192.168.100.2    # The wished static IP
netmask 255.255.255.0
```

In the previous test, all the machines are connected to the same LAN, which makes the communication between them easier. In the next test, however, different VLANs are involved.

VLAN Testbench

This test consists of a server that can attend multiple petitions from clients, which are connected to different VLANs. Therefore, the clients can directly "speak" to the server machine, but they cannot do it to other clients from a different VLAN. This concept of "fragmenting" the LAN into Virtual LANS has become essential to Automotive Ethernet because it increases security.

The Virtual Machine configuration has to be left as in the previous test. Moreover, we must configure in each Virtual Machine the corresponding VLAN with the following command:

VLAN Config:

```
sudo ip link add link enp0s3 name enp0s3.10 type vlan id 10
sudo ip addr add 192.168.110.10/24 dev enp0s3.10
sudo ip link set enp0s3.10 up
```

Those previous commands will not be permanent if the device is rebooted or powered of. To solve this, a specific executable file (.sh) can be launched at boot to run those instructions at startup. Furthermore, the configuration of virtual machine must be done as shown in the diagram below:

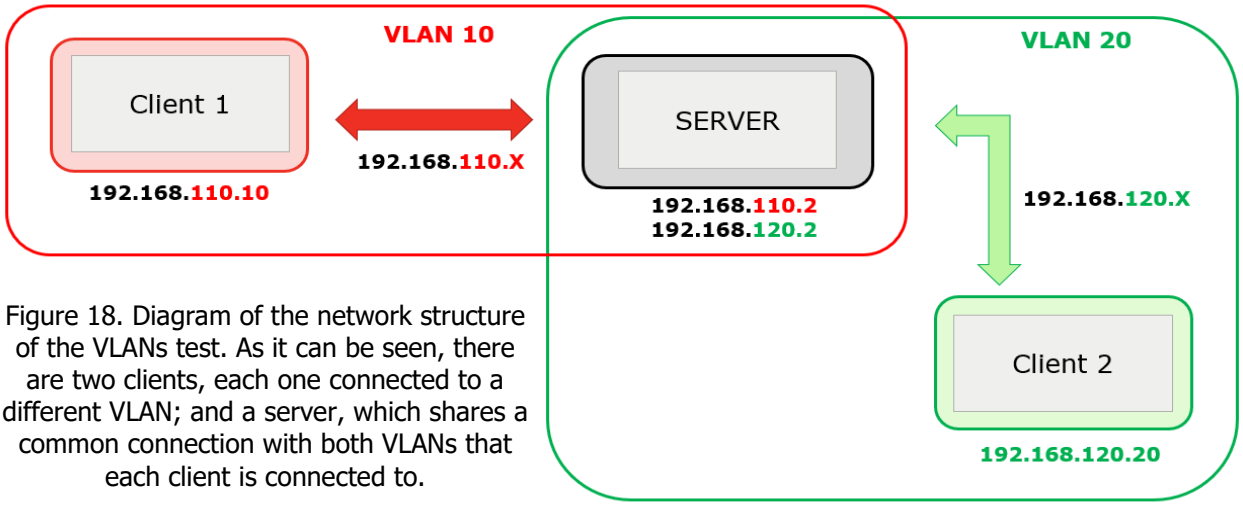


Figure 18. Diagram of the network structure of the VLANs test. As it can be seen, there are two clients, each one connected to a different VLAN; and a server, which shares a common connection with both VLANs that each client is connected to.

Although VLANs work at layer 2, this feature can be abstracted and implement it within IP layer. Consequently, the 192.168.110.X address is associated with VLAN ID 10 and 192.168.120.X

with VLAN ID 20. When a communicating message is sent to that address (ICMP or another type of message) the network card automatically adds the VLAN tag according to which IP address the message is being sent to.

MACsec Testbench

To implement MACsec in the communications, the messages need to be encrypted using a special 128-bit or 256-bit key to increase security and to prevent unwanted people from accessing sensitive information.

As far as this testbench is concerned, MACsec encryption will be implemented within the Ubuntu OS (similar to the previous testbench with the VLANs). A new network interface will be added and, when any message is sent through that interface, the OS will automatically encrypt the message according to the Pre-Selected Key (PSK) established in the configuration.

The configuration of the MACsec interface can be done as follows:

INSTALLATION:

Download zip: https://github.com/shemminger/iproute2 **and extract it to main directory**

```
cd iproute2-main/
./configure
make && make install
modprobe macsec
```

DEVICE 1: **(MAC addr: 08:00:27:b2:8e:be)**

```
sudo ip link add link enp0s3 macsec.30 type macsec port 1 encrypt on
sudo ip macsec add macsec.30 rx port 1 address 08:00:27:9d:1e:4a
sudo ip macsec add macsec.30 tx sa 0 pn 1 on key 01 85fe8cde80cd88fe5364171bb35b2c2b
sudo ip macsec add macsec.30 rx port 1 address 08:00:27:9d:1e:4a sa 0 pn 1 on key
00 a4f256401c20de27727996a114e17c74
sudo ip link set macsec.30 up
sudo ip addr add 192.168.130.10/24 dev macsec.30
```

DEVICE 2: **(MAC addr: 08:00:27:9d:1e:4a)**

```
sudo ip link add link enp0s3 macsec.30 type macsec port 1 encrypt on
sudo ip macsec add macsec.30 rx port 1 address 08:00:27:b2:8e:be
sudo ip macsec add macsec.30 tx sa 0 pn 1 on key 00 a4f256401c20de27727996a114e17c74
sudo ip macsec add macsec.30 rx port 1 address 08:00:27:b2:8e:be sa 0 pn 1 on key
01 85fe8cde80cd88fe5364171bb35b2c2b
sudo ip link set macsec.30 up
sudo ip addr add 192.168.130.2/24 dev macsec.30
```

After finishing all of those steps using virtual machines, we implemented the same testbech using a physical network. The needed Hardware was a transparent switch as intermediate device and a couple of Raspberry Pi's as endpoints.

If this configuration is set up properly, when an ICMP or another type of message is sent to either 192.168.130.10 or 192.168.130.2, the message will be encrypted using the key.

One thing to consider is that this method is only valid for a Static CAK Mode with Pre-Selected Keys (note that keys 1 and 2 are inverted in Rx and Tx channels). If a Dynamic CAK Mode is desired, it will need another type of configuration that exceeds the limits of this testbench.

After those previous tests have been completed successfully, the virtual approach of the whole testbench has reached at its end. The following step will be the physical testbench, which simply consists of repeating the previous steps but using a real Ethernet switch with real hardware devices as endpoints instead of virtual machines.

Physical Testbench

As far as the hardware is concerned, two Raspberry Pi's will be used as endpoints which will be connected to a NETGEAR transparent switch. The network structure needs to be built as follows:

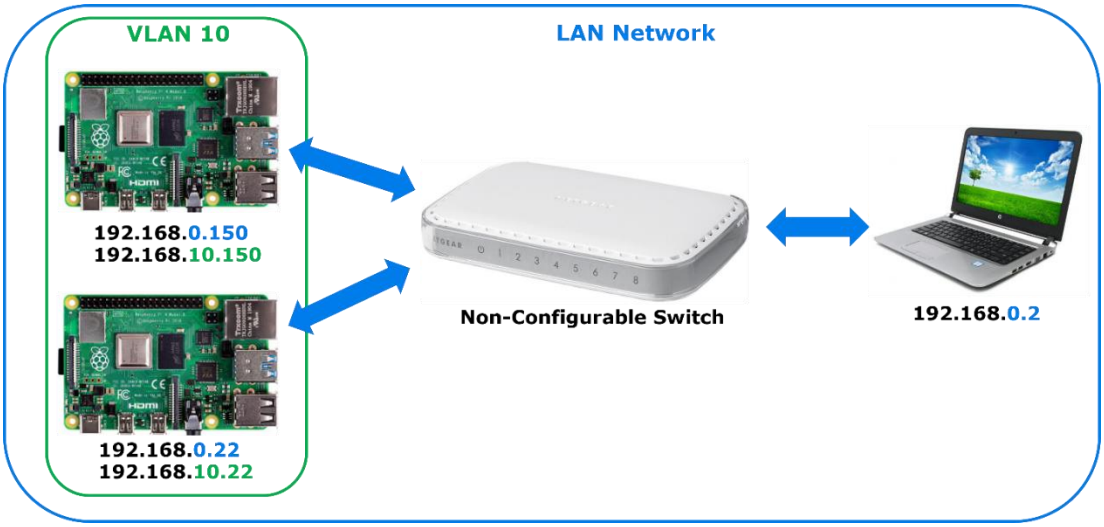


Figure 19. Physical network structure.

The port into which the devices are connected is irrelevant since the switch has its own internal MAC table. Furthermore, frames are forwarded according to this table independently from being tagged (VLAN) or untagged.

AVB Testbench

This last testbench is related to audio and video communications. With the virtual machines, an AVB cloud has to be created (simulating a switch) and some streaming messages will be passed from the source to the listener.

Once the virtual communication is being established, a real approach can be done with the hardware endpoints, such as streaming video from one Raspberry Pi to another.

3.2. Switch Selection

3.2.1. Selection Features and Requirements

In order to select a proper switch, a series of conditions were used to ensure that the selected device met all the needed requirements. In the following paragraphs we'll revise, define and justify why those features are essential to Automotive Ethernet.

Requirements

1000BASE-T1 capability: it can carry data with speeds up to 1 Gb/s. This cable consists of 1 differential pair, which normally could carry one signal of data. However, the great specialty of BASE-T1 is that it can carry two frequency-multiplexed signals: Transmission (Tx) and Reception (Rx). Therefore, we are able to transport twice the amount of data as it would be in a normal bus with just one differential pair. A **differential pair** consists of two cables which carry a voltage differential. This voltage differential is used to identify a HIGH level or a LOW level of voltage, which represents a binary 1 and 0. The great advantage of a differential pair is that no ground reference is needed and, therefore, less errors happen during the transport of the signal.

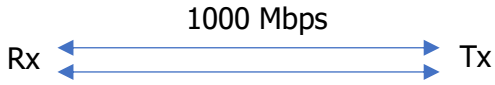


Figure 20. 1000Base-T1 bus. One bidirectional pair.

Integrated PHYs: they are necessary in communications with other devices that imply external ports in the switch. PHYs can be found in Ethernet cards, and they are used to send data faster and further with longer cables. In the switch, all the ports must have incorporated PHYs.

Reduced Giga-bit Media Independent Interface (RGMII): it's a similar port to the SGMII but instead of being serial, it transfers data in parallel. For our application in Automotive Ethernet, only one port of this type is required.

To explain how RGMII works, we will look through first at the Media Independent Interface (MII) and the Reduced Media independent Interface (RMII). The MII started first with 100 Mbps. To do so, at least 10 cables are required since 4 unidirectional cables correspond to the transmission Tx and 4 unidirectional cables correspond to the reception Rx. Furthermore, we need 2 extra cables for the clock control.

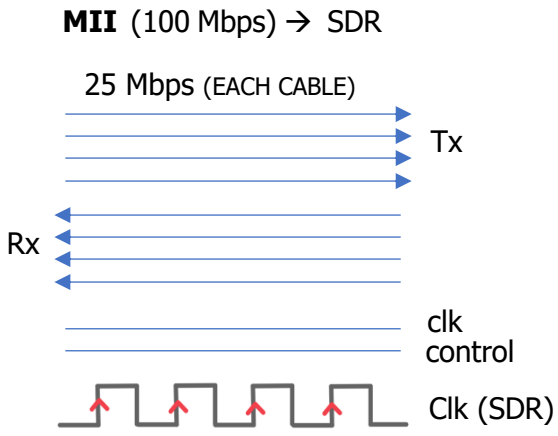


Figure 21. MII communication bus.

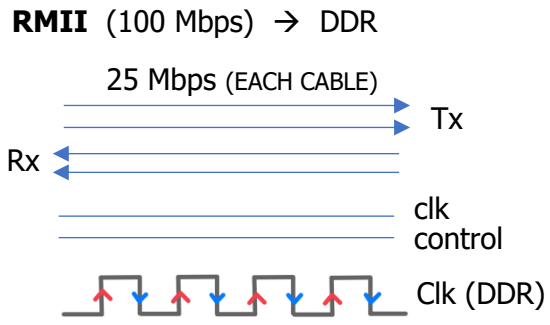


Figure 22. RMII communication bus.

To reduce the number of cables, the RMII was invented. Instead of having 8 cables and sampling the data at Single Data Rate (SDR), we have half of the cables (4) but sampling twice as fast as before (Double Data Rate, DDR).

After explaining those two types of communications, we are able to explain properly RGMII. The concept of the Reduced Giga-bit Media Independent Interface is similar to the RGMII. We have 4 cables for Tx and 4 for Rx, each at 125 Mbps and sampling at DDR.

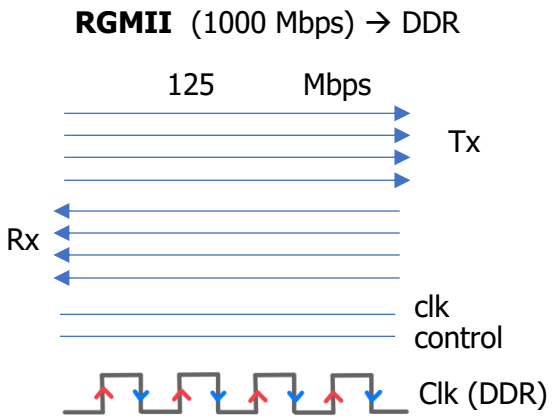


Figure 23. RGMII communication bus.

Serial Giga-bit Media Independent Interface (SGMII): a serial port used for communications. Unlike with the PHYs, not all the ports are required to have SGMII, just a few of them.

The SGMII is a complex communication since it requires a more complex hardware and is more expensive. However, we can transfer data up to 1000 Mbps with just two cables. This SGMII also has implicit clock and is reliable and fast.

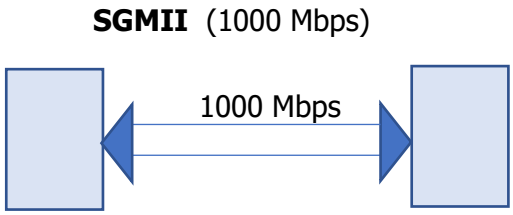


Figure 24. SGMII communication bus. One differential pair.

Peripheral Component Interconnect Express (PCIe): it's a serial bus which serves as an expansion port for many external devices (such as Solid-State Drives, Ethernet Adapters and so on). This port has a high data throughput and, for our project, we need at least one of them.

MACsec: it's a layer 2 security mechanism that is used for encrypting the data that goes through the Ethernet cables and ports. Since we are working with switches, IPsec will not work regarding the data protection and, therefore, we need to use MACsec. This is the strongest hardware requirement that our device needs to have, since all the ports must include MACsec capability.

AVB capability. Our switches will work as AVB bridges for streaming data and audio, thus AVB capability is essential to be supported in our device. Furthermore, we also looked for the maximum amount of AVB protocols that the switch could support such as SRP, FQTSS and gPTP.

Security Mechanisms. Apart from MACsec, we also looked for alternative security mechanisms such as Secure Boot, Network Filter…

Other Features considered: Mirroring capability Hardware Acceleration mechanisms and debugging ports (JTAG). Also 100BASE-TX port for other purposes and 10BASE-T1S.

All those features mentioned above were summarized into an Excel, which was used to identify whether the switches were optimal or not. By the end of the selection, we were able to find out a total of 11 devices which were candidates to be a proper switch. The possible devices were:

- **Supplier 1:** 3 devices
- **Supplier 2:** 5 devices
- **Supplier 3:** 3 devices

We finally decided to choose X, from supplier 1.

3.1.2. Why choose the selected switch?

Confidential.

3.1.3. Evaluation Board configuration

Confidential.

3.3. Switch Basic Programs

Confidential.

3.3.10. Tests with basic Switch configurations

After all those configurations have been documented, there are a few tests which can be run so as to test whether the configurations are working properly or not. Those tests mainly consist of:

1. **Basic Configuration Test 1:** communicate untagged port with tagged ports.
2. **Basic Configuration Test 2:** port mirroring with VLANs.
3. **Basic Configuration Test 3:** re-tagging (VLAN ID Override)

3.3.10.1. Basic configuration test 1

BASIC CONFIGURATION TEST 1	
Aim of the Test	Communicate Untagged Port with Tagged Ports within a VLAN.
Specifications	1. Port 6: Untagged with Default VID 10 (set register with offset 0x07, SMI address 6 and value 0x100A) 2. Ports 1-5: Tagged with VLAN ID 10 3. RPi 1: Remove VLANS, RPi IP = 192.168.0.22 and add an alias eth0:1 with IP 192.168.10.23 4. RPi 2: Configure VLANS with ID 10 and 20 5. Disable the forwarding of Untagged Messages (Discard Untagged, Offset 0x08 = 1)
Run Test	Make a ping from RPi 1 to RPi 2 (Command: ping 192.168.0.150)
Results	The switch receives an untagged message (ICMP) from RPi 1 and, as it has been configured with the default VID, it automatically adds the specified tag (10) and it sends it through the switch. The RPi 2 receives the tagged ICMP and sends a response with the tag as well. Then, when it reaches port 6, the switch automatically removes the tag and sends the response back to RPi 1.

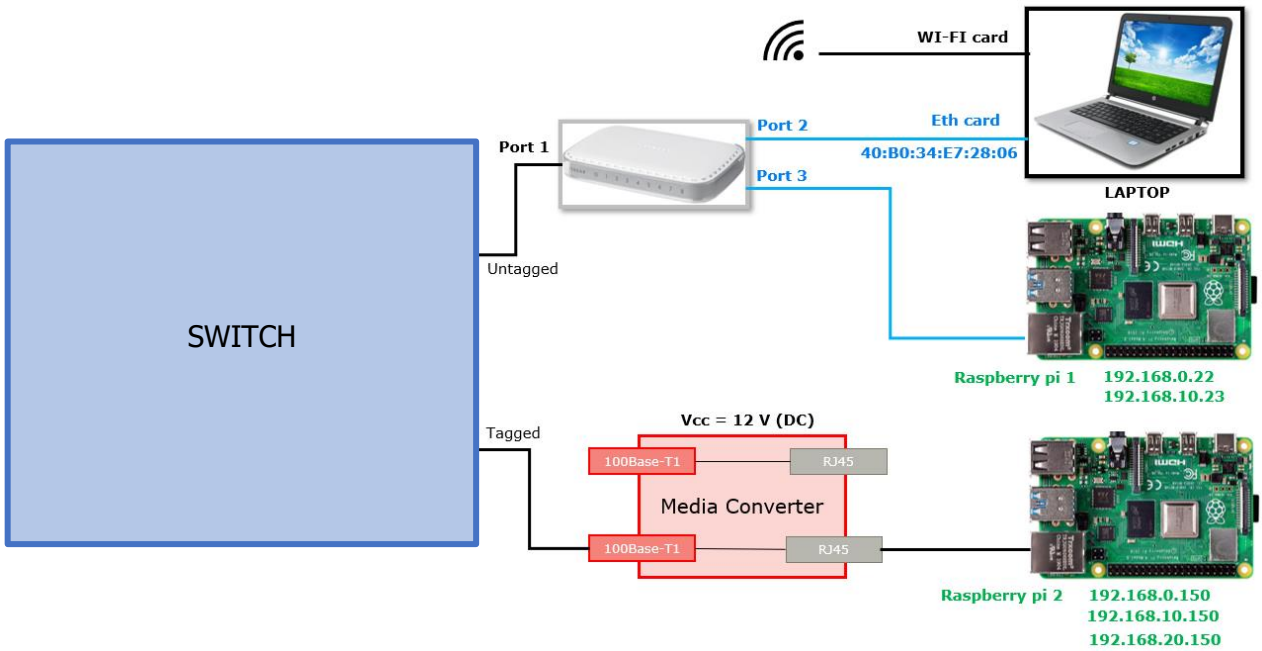


Figure 25. Diagram of the Basic Configuration Test 1.

3.3.10.2. Basic configuration test 2

BASIC CONFIGURATION TEST 2	
Aim of the Test	Port mirroring with VLANS.
Specifications	1. Virtual Machine: bridged adapter, IP = 192.168.0.2 2. Disable any configured VLANS in the RPi 1. 3. Add an alias in the RPi 1 , with name eth0:1 and IP = 192.168.10.23 (sudo ifconfig eth0:1 192.168.10.23 up). 4. VLAN, ID = 10, Configure all ports (1, 2, 3, 4, 8, 9) as Tagged, except 5 and 6, which are Untagged. 5. Mirror Ports 1, 2, 3, 4, 5, 6, 8 and 9 (Ingress and Egress) to port 7. 6. Default VID = 10 with ports 5 and 6 (that correspond to the Untagged ones).
Run Test	Make a ping from RPi 1 to RPi 2 (ping 192.168.10.150) Then open Wireshark in Windows and sniff the mirrored traffic in port 7.
Results	The RPi 1 and RPi 2 communicate with an ICMP message, and the traffic is mirrored into port 7, where Windows is sniffing the traffic with Wireshark and detects the messages that are being sent between RPi 1 and the RPi 2.

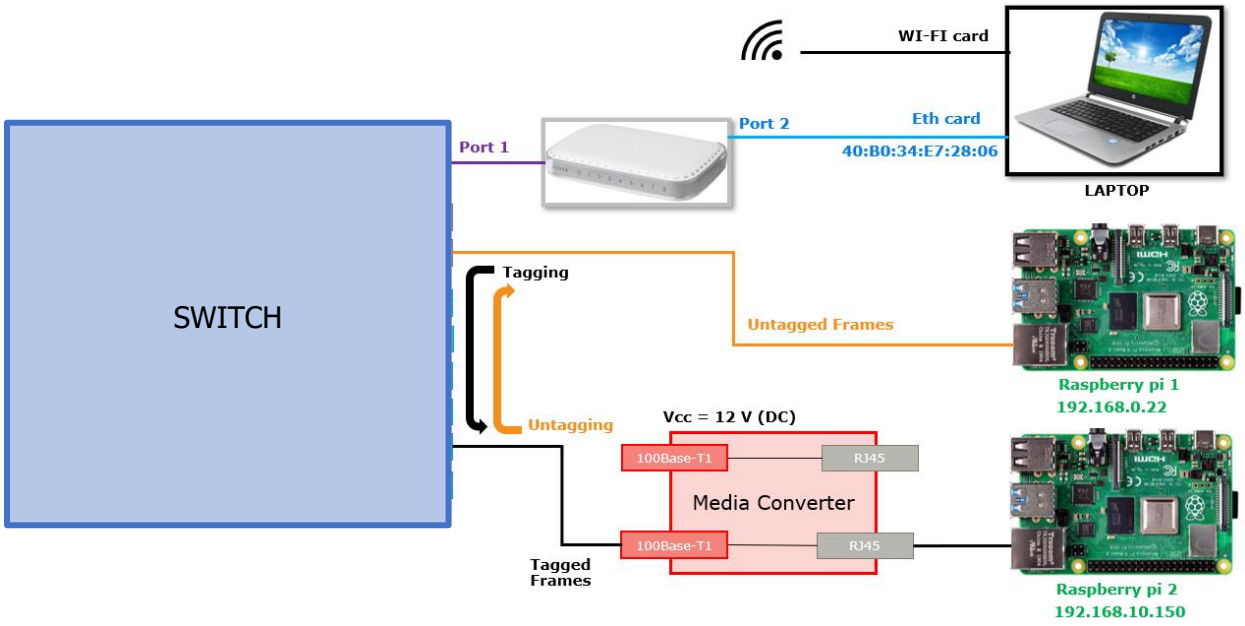


Figure 26. Diagram of the Basic Configuration Test 2.

3.3.10.3. Basic configuration test 3

BASIC CONFIGURATION TEST 3	
Aim of the Test	Retagging (VLAN ID Override).
Specifications	1. Configure VLAN (eth0.10) in the RPi 1. VLAN ID = 10 and IP = 192.168.10.22/24 2. Add an alias in the RPi 1 , with name eth0.10:1 and IP = 192.168.20.23 (sudo ifconfig enp0s3:1 192.168.10.23 up). 3. VLAN, ID = 10, Configure ports 1, 3, 4, 8, 9 as Tagged, except 5 and 6, which are Untagged. 4. VLAN, ID = 20, Configure port 2 as Tagged. 5. Mirror Ports 1, 2, 3, 4, 5, 6, 8 and 9 (Ingress and Egress) b. 6. Default VID = 10 with ports 5 and 6 (that correspond to the Untagged ones).
Run Test	Make a ping from RPi 1 to RPi 2 (ping 192.168.20.150) Then open Wireshark in Windows and sniff the mirrored traffic in port 7.
Results	The RPi 1 sends a message from 192.168.20.23 (alias of eth0.10) and with VLAN ID = 10. Then the switch overrides the tag with VID = 20 and sends it to Port 2. The RPi 2 answers with Tag 20 and the switch overrides again the Tag to send it to RPi 1 with previous VLAN Tag = 10.

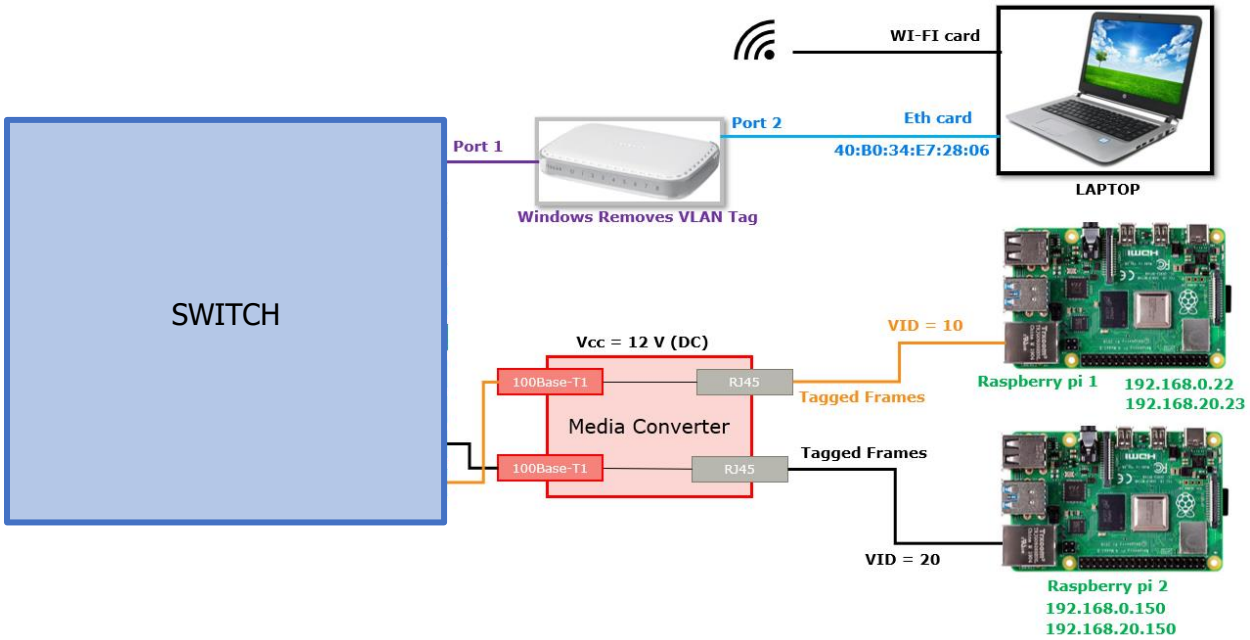


Figure 27. Diagram of the Basic Configuration Test 3.

3.4. Implementation of AVB and SRP

Once the basic configurations of the switch have been implemented (VLAN, Port mirroring and others), it is time to start working with the audio and video capabilities of the switch.

Aim of the Test: this test consists of creating a network structure formed by a Talker, a Listener, and a Network Saturator and test whether the Stream Reservation is working properly or not.

The tool used for streaming video and audio is *GStreamer*. All the information regarding the installation and basic steps of this tool can be found in the Annexed information at the end of the project. The Operating System used for launching GStreamer is Ubuntu 22.04.

3.4.1. Network structure

Since all the traffic within the automobile is tagged, different VLANs need to be configured in the switch as shown in the following diagram:

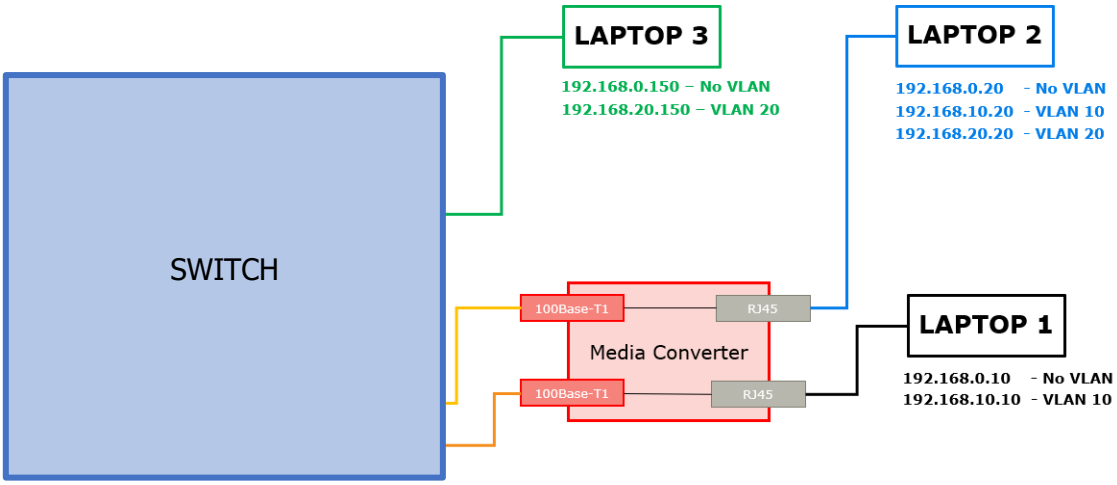


Figure 28. Network connections and configuration for all the tests.

The previous diagram shows a general overview of the network structure. Each individual test requires a specific one, which can be exclusively programmed. Nevertheless, if the network is configured entirely as seen in the previous picture, it will be suitable for being used in all the following tests.

To configure VLANs, the following commands can be typed (is recommended to use a batch file):

```
sudo ip link add link enp0s31f6 name enp0s31f6.10 type vlan id 10 egress 0:6
sudo ip addr add 192.168.10.10/24 dev enp0s31f6.10
sudo ip link set enp0s31f6.10 up
```

The VLAN ID will be indicated in the `vlan id 10` and the Priority Code Point will be indicated with the `egress 0:6`. In this particular case, the VLAN ID = 10 and the PCP=6. Once all the network is up, ready and configured, the testing part can be started. Four tests are being done with different configurations.

3.4.2. GStreamer – High quality streaming (1920x1080)

Before trying to stream video over the network, several initial tests were done. It was discovered that if the video is lower than 2000x2000 pixels, the JPEG encoding can be used. However, if the video is from higher quality (4K – 3840x2160) this type of encoding will not work properly since it has a limit of two thousand pixels wide or high. Nevertheless, the video streams within an automobile network will not be higher than 1920x1080 (even less than that). Consequently, the JPEG encoding is a fine encoding method for this purpose.

The network structure required in this scenario is shown in the previous section (3.4.1). Once all the devices are properly configured, the following commands must be launched in the shell in the talker and listener:

Talker

```
gst-launch-1.0 filesrc location=
/home/jfonts/documentation/test7_results/example_videos/test_5.mp4 ! decodebin !
videoconvert ! jpegenc ! rtpjpegpay ! udpsink host=192.168.0.20 port=8080
```

Listener

```
gst-launch-1.0 udpsrc port=8080 ! application/x-rtp,encoding-name=JPEG,payload=26 !
rtpjpegdepay ! jpegdec ! autovideosink
```

Use of launch.sh file

The contents of the .sh file are the following. The parameters that are passed are two, the first is the number of times that the video will be streamed in a loop (default value is 2). The second parameter corresponds to the video identifier (1, 2, 3… 5, 6, A, B….).

```
x=0
rep="${1:-2}";   #Takes default value as 2 if no argument is passed
                    "${sys_argv_number + :- + default_value}"
test_run="${2:-5}";

echo "Repeating process $rep times";


while [ $x -lt $rep ]   # -lt ("<")   and   -le ("<=") -- lower than / lower or equal
do
      gst-launch-1.0 filesrc
location=/home/jfonts/documentation/test7_results/example_videos/test_$test_run.mp4 !
decodebin ! videoconvert ! jpegenc ! rtpjpegpay ! udpsink host=192.168.0.20
port=8080;
      x=$((x+1));
      echo "Times executed: $x";
done
```

Launch command: ./documentation/test7_results/launch.sh 50 6 (video 6, 50 times)

3.4.3. Switch Configuration for SRP and AVB

Confidential.

3.4.4. AVB - Test 1

This test must be done with the use of iperf, which is a Linux tool used to measure the efficiency of a network. It can also send packets between two devices within a network. The aim of using *iperf* in this case is to saturate the network, to test if the Switch configuration is working properly.

1. **Laptop 1:** Iperf client, VLAN ID of 10, PCP=6 and port 5001
2. **Laptop 2:** Iperf Server at port 5001 and Iperf Server at port 5002
3. **Laptop 3:** Iperf client, VLAN ID of 20, PCP=6 and port 5002

```
iperf -u -s -p 5001 -i -1    #Laptop 2, shell 1
iperf -u -s -p 5002 -i -1    #Laptop 2, shell 2
iperf -u -c 192.168.10.20 -p 5001 -i 1 -b 100m -t 10000    #Laptop 1
iperf -u -c 192.168.20.20 -p 5002 -i 1 -b 100m -t 10000    #Laptop 3
```

The switch configuration parameters are the same as shown in the GUI configuration (regarding Global and Port Settings). Nevertheless, the iperf UDP traffic if of a different length and, consequently, the CBS limit has to be changed. In this case, the Iperf traffic is 1516 bytes long. Adding the Preamble and IGF, the resulting number is 1536 bytes.

In this case the Stream Reservation is a total of 1720·32000 = **55.04 Mb/s**

Checking if the reserved bandwidth is correct

```
jfonts@ricardo-HP-ProBook-640-G1:~$ iperf -u -s -p 5001 -i 1
------------------------------------------------------------
Server listening on UDP port 5001
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 192.168.20.20 port 5001 connected with 192.168.20.150 port
 46587
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total
 Datagrams
[  1] 0.0000-1.0000 sec  2.53 MBytes  21.2 Mbits/sec  31.616 ms 0/1807
 (0%)
[  1] 0.0000-1.0000 sec  1 datagrams received out-of-order
[  1] 1.0000-2.0000 sec  0.000 Bytes  0.000 bits/sec  31.616 ms 0/0 (-
nan%)
[  1] 2.0000-3.0000 sec  7.59 MBytes  63.7 Mbits/sec   0.181 ms 17947/
23364 (77%)
[  1] 3.0000-4.0000 sec  11.4 MBytes  95.4 Mbits/sec   0.370 ms 0/8116
 (0%)
[  1] 4.0000-5.0000 sec  11.4 MBytes  95.4 Mbits/sec   0.139 ms 14/813
0 (0.17%)
[  1] 5.0000-6.0000 sec  11.4 MBytes  95.5 Mbits/sec   0.148 ms 61/817
9 (0.75%)
[  1] 6.0000-7.0000 sec  10.6 MBytes  88.8 Mbits/sec   0.197 ms 573/81
25 (7.1%)
[  1] 7.0000-8.0000 sec  8.40 MBytes  70.5 Mbits/sec   0.488 ms 2128/8
119 (26%)
[  1] 8.0000-9.0000 sec  5.28 MBytes  44.3 Mbits/sec   0.475 ms 4376/8
142 (54%)
[  1] 9.0000-10.0000 sec  2.59 MBytes  21.8 Mbits/sec   0.433 ms 6349/
8200 (77%)
[  1] 10.0000-11.0000 sec  1.39 MBytes  11.7 Mbits/sec   0.317 ms 7284
/8278 (88%)
[  1] 11.0000-12.0000 sec  3.96 MBytes  33.2 Mbits/sec   0.254 ms 5398
/8233 (66%)
```

```
jfonts@ricardo-HP-ProBook-640-G1:~$ iperf -u -s -p 5002 -i 1
------------------------------------------------------------
Server listening on UDP port 5002
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 192.168.10.20 port 5002 connected with 192.168.10.10 port
43823
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total
 Datagrams
[  1] 0.0000-1.0000 sec  2.08 MBytes  17.5 Mbits/sec   0.189 ms 6513/7
998 (81%)
[  1] 1.0000-2.0000 sec  4.32 MBytes  36.2 Mbits/sec   0.162 ms 4994/8
076 (62%)
[  1] 2.0000-3.0000 sec  7.31 MBytes  61.4 Mbits/sec   0.156 ms 2861/8
078 (35%)
[  1] 3.0000-4.0000 sec  10.4 MBytes  87.0 Mbits/sec   0.144 ms 682/80
77 (8.4%)
[  1] 4.0000-5.0000 sec  9.01 MBytes  75.6 Mbits/sec   0.286 ms 1652/8
078 (20%)
[  1] 5.0000-6.0000 sec  5.11 MBytes  42.8 Mbits/sec   0.204 ms 4320/7
963 (54%)
[  1] 6.0000-7.0000 sec  2.09 MBytes  17.5 Mbits/sec   0.212 ms 6556/8
044 (82%)
```

Figure 29. No Stream Reservation configured. Bandwidth variable.

In the first case where no Stream reservation is configured, the bandwidth is not stable. As can be seen in the image below, it changes its value constantly.

However, if the Stream Reservation is configured, the result is successfully achieved. As can be seen in the image below, the bandwidth of the laptop 3 (left part of the image) is at maximum capacity (95 Mbps) but, when Laptop 1 (Talker) starts transmitting video, the Switch immediately recognizes the AVB frames and reserves bandwidth of 52 Mbps approximately for laptop 1. Consequently, the other device has no other option but to lower the streaming down to 43 Mbps.

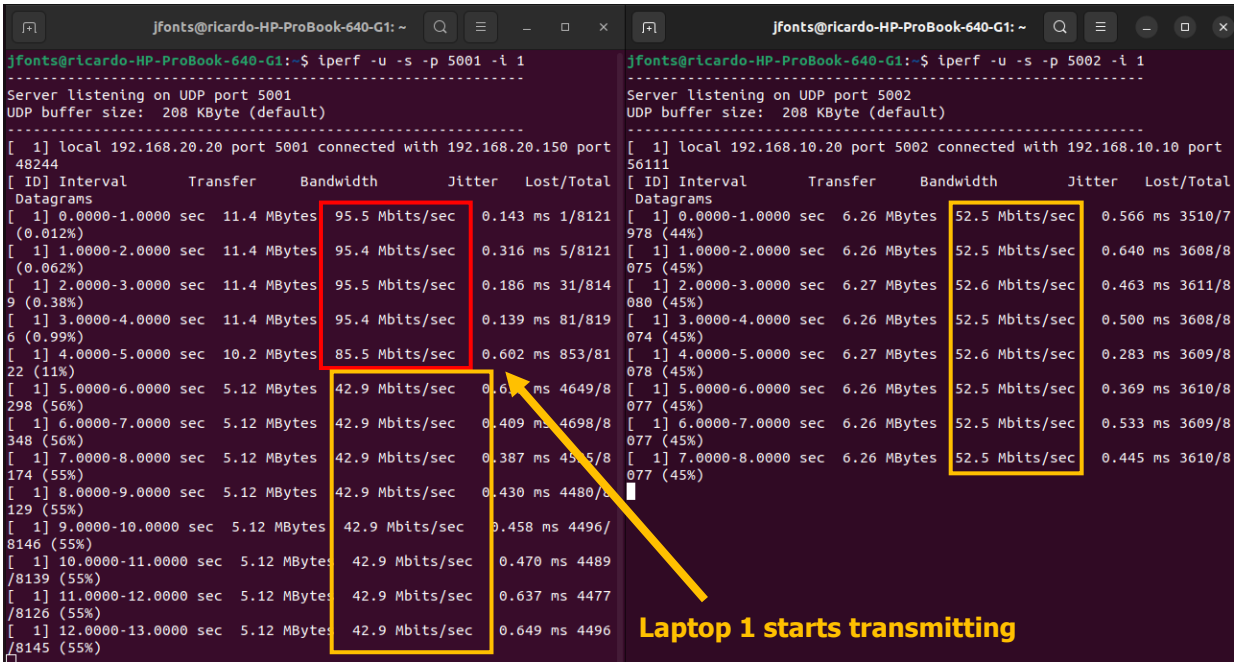


Figure 30. Stream Reservation configured. Fixed bandwidth.

Checking if the CBS is correct

After a certain amount of time, two frames are sent. When the credit reaches the top value (1536) an AVB frame is sent, which reduces the credit down to 0. However, a frame can be sent as long as credit ≥ 0. Therefore, a second frame is sent, and the credit becomes negative. A period of time needs to pass until credit reaches again its previous value and newer AVB frames can be sent.

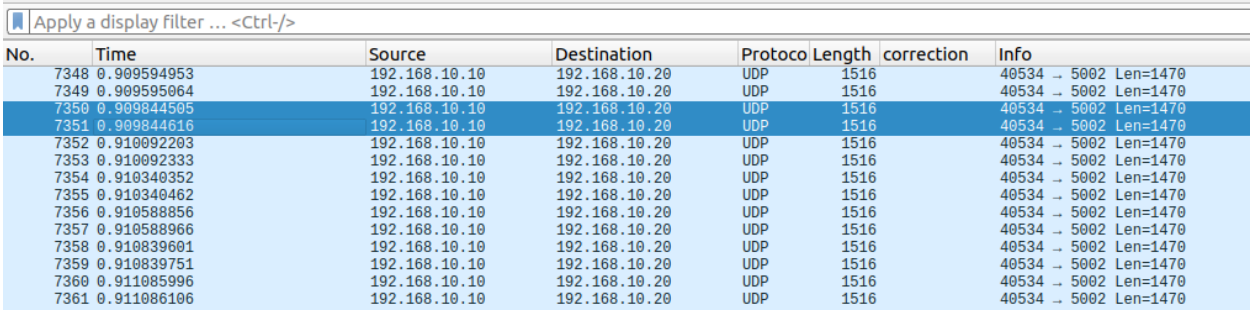


Figure 31. Credit Base Shaper checking. Two frames are sent consequently and after this, a small period of time passes until newer frames can be sent. An example of those "pair of frames" are selected in the image in a dark blue color.

3.4.5. AVB - Test 2 (No SRP)

Once test 1 has been completed, we are in good shape to start streaming video. In test 2 and 3 the same proceedings are done but, instead of generating a fake video streaming with iperf, an actual video will be sent.

1. **Laptop 1:** Talker of video, VLAN ID of 10, PCP=6
2. **Laptop 2:** Listener of video and Iperf Server
3. **Laptop 3:** Network saturator, Iperf client, VLAN ID of 20, PCP=0

```
#Laptop 1, check launch.sh file
./documentation/test7_results/launch.sh 50 6          #Change IP to 192.168.10.20
#Laptop 2, shell 1
iperf -u -s
#Laptop 2, shell 2
gst-launch-1.0 udpsrc port=8080 ! application/x-rtp,encoding-name=JPEG,payload=26 !
rtpjpegdepay ! jpegdec ! autovideosink
#Laptop 3
iperf -u -c 192.168.20.20 -p 5002 -i 1 -b 100m -t 10000
```

In this particular scenario, no Stream Reservation will be configured. Therefore, when the network is being saturated with a third device, the video streaming has insufficient bandwidth. Thus, creating a subtle stop in the video (due to insufficient bandwidth). In the following image there is the diagram that shows the network structure used in test 2 and 3.

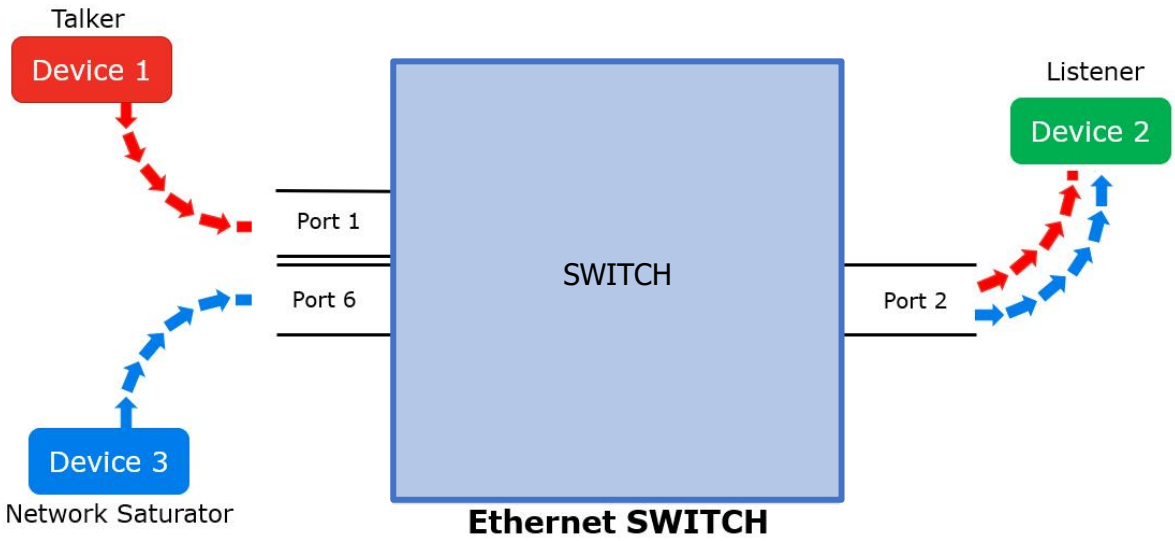


Figure 32. Diagram of the Network structure used in test 2 and 3.

The red arrows correspond to video streaming, and the blue arrows correspond to useless UDP traffic which its only purpose is to saturate the network, to check whether the switch is capable of reserving a specific bandwidth or not.

Ports 1 and 2 have been chosen arbitrarily since it seemed easier to connect Laptop 1 to port 1 and Laptop 2 to port 2. However, if another port was desired to be used, there would be absolutely no problem, as long as the Ethernet Switch is properly configured.

Results

After configuring this type of test, we achieved the following results:

1. Video is streamed perfectly as long as the Network is not saturated.
2. Video can be streamed as long as Network is saturated but leaves sufficient bandwidth for the video to transmit.
3. If the Network is saturated at maximum capacity, the video stops since it has insufficient bandwidth.

Although these results are difficult to check without a video, there are a couple of images showing its functionality:

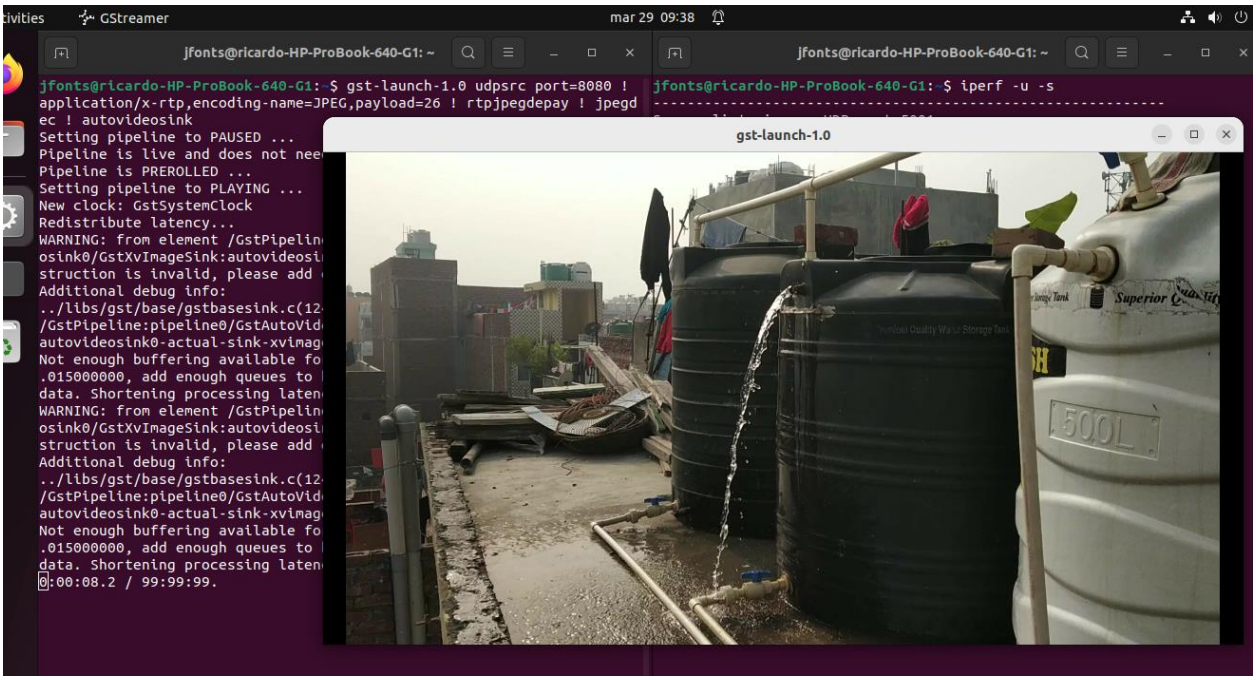


Figure 33. Screenshot from the Listener's Laptop.

3.4.6. AVB - Test 3 (with SRP)

This test is identical to the second test, with the difference that Stream Reservation Protocol (SRP) is being enabled. To enable SRP in the Switch, refer to section 3.4.3. with the following parameters:

AVB

Confidential.

Once all those parameters have been established, the switch is properly configured.

Results

After configuring this test 3, we achieved the following results:

1. Video is streamed perfectly as long as the Network is not saturated.
2. Video can be streamed perfectly even if Network is saturated.

3. If a third device tries to saturate the Network, the switch itself will reduce the amount of traffic it produces to ensure sufficient bandwith for the video streaming.

Note: This result cannot be visualized in this written documentation. Therefore, it will be shown in the final presentation of the project.

3.4.7. Stream without encoding

After doing a few tests it was realized that every time that the video was being sent, it was previously encoded. Therefore, a new way was encountered to send the video directly without encoding it.

The encoding was previously done from a MP4 to a H264 type. To prevent this step, a video of this last format was downloaded. That way, no encoding was required since the video was already in the desired format (H264). The following commands launched were:

Talker

```
gst-launch-1.0 filesrc location=/home/jfonts/big_buck_bunny_1080p_h264.mov !
qtdemux name=d ! rtph264pay config-interval=-1 pt=96 ! udpsink host=192.168.10.20
port=8080
```

Listener

```
gst-launch-1.0 udpsrc port=8080 ! application/x-rtp,media=video,clock-
rate=90000,encoding-name=H264 ! rtph264depay ! avdec_h264 ! autovideosink
sync=false
```

The result of this streaming can be seen in the following image. **Note:** the sync=false parameter is essential to ensure a fluid transmission. Otherwise, the video will result in a laggy streaming.

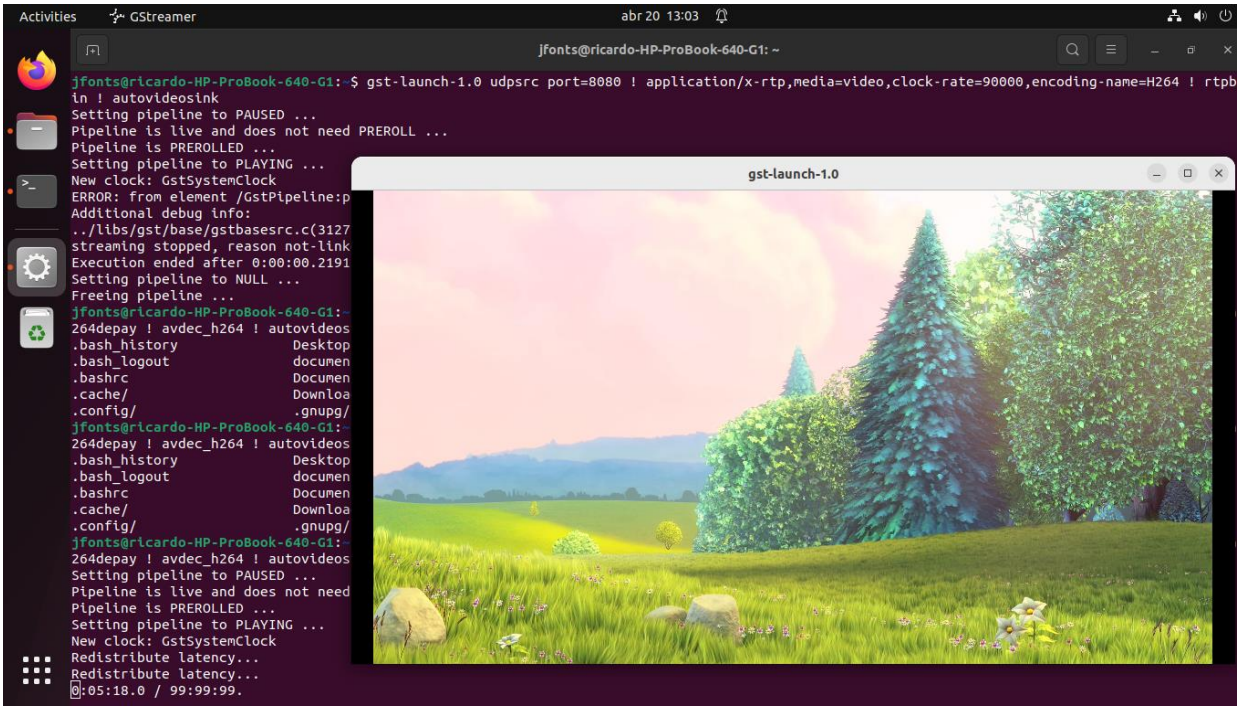


Figure 34. Streaming directly without encoding the video.

3.4.8. Constant streaming bitrate

After doing all the previous tests, there is only one handicap left to be solved: the constant bitrate streaming. Is essential that the talker transmits at a constant bitrate since the configuration of the SRP into the switch depends on it.

The problem is due to GStreamer, which instead of transmitting at a constant bitrate of, for example, 10 Mbps, it transmits video of 25 Mbps and then does not stream for a while (which the average number results in the same value as if it was transmitting at a constant bitrate). The image below show the desired result (on the left) and the actual non-desired result (on the right):

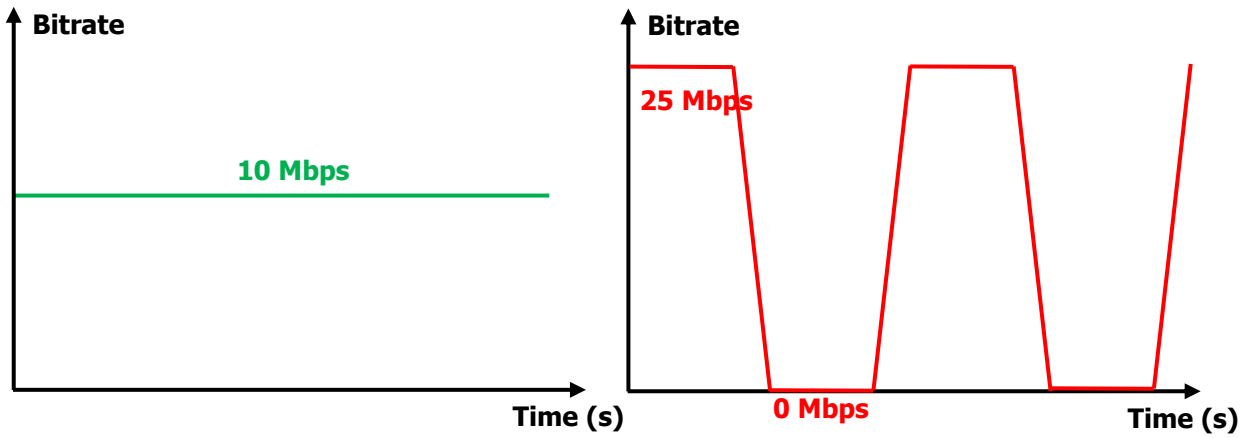


Figure 35. Desired result vs Obtained result.

Peak value and debugging with Wireshark

To debug this error, Wireshark will be used. As it can be seen in the image below, GStreamer streams video until second 1.62 and, after this, it quiets until second 1.66. Therefore, it is being validated that the video is being transmitted as shown in the red curve.

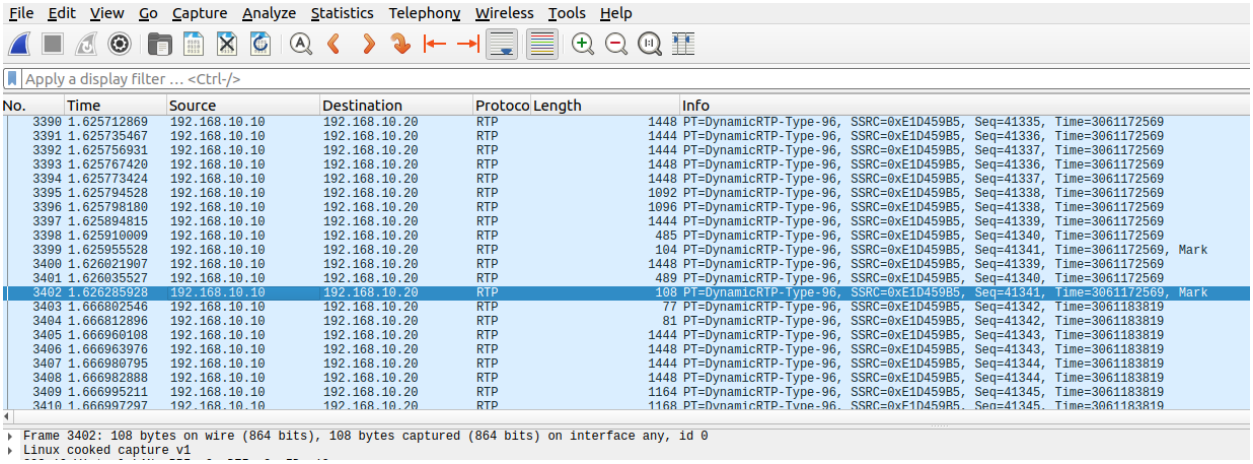


Figure 36. Frames read from Wireshark.

To calculate the peak value, it can be done easily with Wireshark. Into the "Statistics" section, the "I/O Graphs" options can be selected. Then, a graphic should pop up as in the image below and the bitrate can be seen graphically.

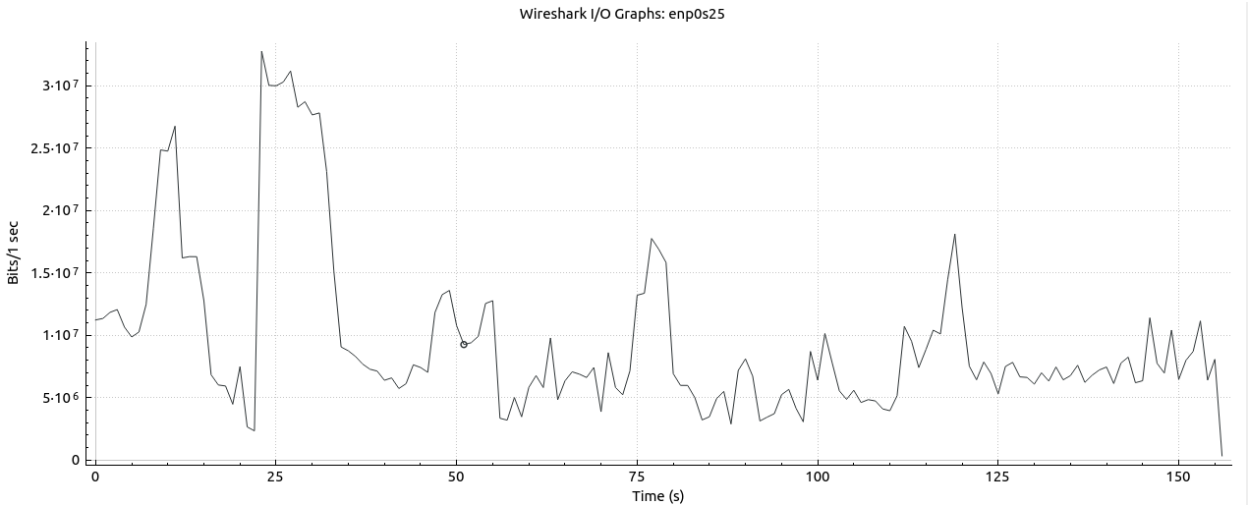


Figure 38. Peak Bitrate graphic.

The peak bitrate value is around the beginning of the transmission in 25th second. The peak bitrate value at that point is around $3.3 \cdot 10^7$ bits/s which is **33 Mbps**.

Find average Bitrate with Wireshark

Even though the streaming is in pulses, the average value can be also obtained with Wireshark. To do so, open Wireshark and go to "Statistics" section and choose "Conversations". Select UDP option since the streaming frames are of this type. A screen should appear as shown below:

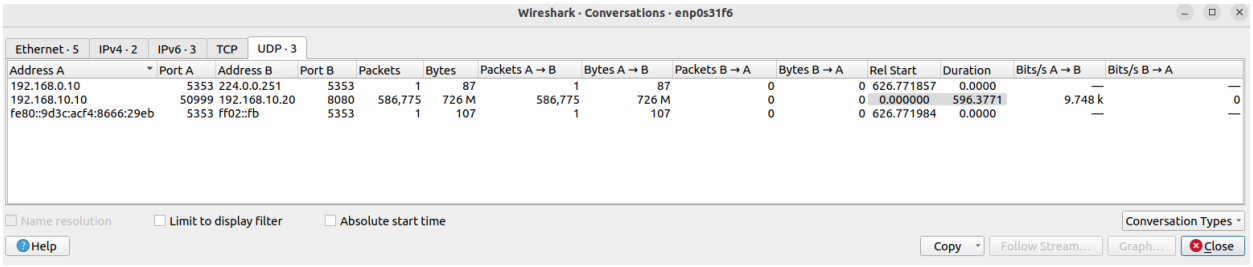


Figure 39. Conversations in Wireshark. The average value can be found under the tab "Bits/s A ▪ B" which, in this case, is 9.75 Mbps.

As can be seen, the peak value is very different from the average value (around three times its value). Even though, a solution has not been found yet to solve this problem. Therefore, when a Stream reservation is configured, it **must be done with the peak value calculated in the Wireshark graphic.**

3.5. Implementation of Time Synchronization (gPTP)

Time synchronization is essential in Automotive Ethernet so as to guarantee that all the ECUs share the same time reference. To synchronize all the different endpoints the messages must be time stamped, and this process can be done in two different ways:

- **Hardware Timestamping:** the time stamping is done in the Ethernet driver.
- **Software Timestamping:** the time stamping is done in the SO software.

Regarding the HW Timestamping, highly accurate timings can be achieved and it's far faster than the Software Timestamping. Nevertheless, the Ethernet card must support this feature of HW Timestamping, whereas the SW Timestamping can be supported in all the devices. Event though, the desire is to achieve highly accurate precisions, leading to choose **HW Timestamping.**

As far as Raspberry Pi's are concerned, they do not support HW Timestamping. Hence, new devices need to be used as endpoints. Two laptops with Ubuntu OS and HW Timestamping capabilities are used for the following tests.

After getting the required hardware, the Linux PTP library must be downloaded. Github link:

```
https://github.com/richardcochran/linuxptp
```

To run PTP execute the following command, specifying the network card with **`-i`** (enp0s5 in this case), specify the file and its location with **`-f`**, and specify the step_thereshold.

```
sudo nice --20 ptp4l -i enp0s5 -m -f /usr/share/doc/linuxptp/configs/automotive-
slave.cfg --step_threshold=1
```

Note: the PTP command is run with the highest priority using the `nice` option. The slave config will be run in one laptop and the master config in the other laptop (`automotive-slave.cfg` or `automotive-master.cfg`).

Note 2: although all the traffic within an Automobile Network should always be tagged, an exception needs to be taken care of. Regarding the IEEE 802.1AS standard, which refers to the Time Synchronization, all the (g)PTP messages shall not use an VLAN tag. Therefore, all the traffic within the Switch will be tagged except for the PTP messages. The Switch itself detects if the message if PTP-type and will automatically assign it to the higher queue possible (Queue 7).

The standard IEEE 802.1AS can be found at the following page: https://1.ieee802.org/wp-content/uploads/2019/03/802-1AS-rev-d8-0.pdf [Last seen 03 May 2023].

3.5.1. GPTP network structure

The network configuration stays the same as with AVB and SRP. The diagram is the following:

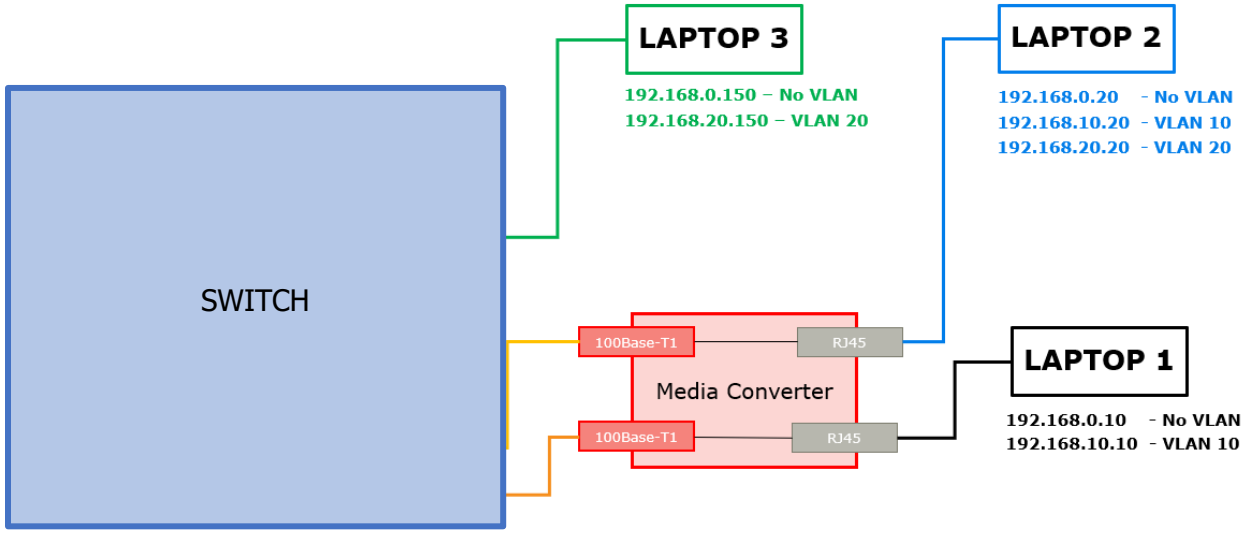


Figure 40. Network structure for PTP tests.

In the previous network, laptop 1 will serve as the master of PTP synchronization (Grandmaster clock). The laptop 2 will receive the time-stamped messages and synchronize according to the gPTP messages. Finally, the laptop 3 serves as a network saturator to test whether the switch configuration is working properly or not.

3.5.2. GUI configuration for gPTP

Confidential.

3.5.3. Test 1 – without PTP configuration

The aim of this test is to check how the Switch reacts to network saturation. As seen in the title, no PTP configuration will be used at all.

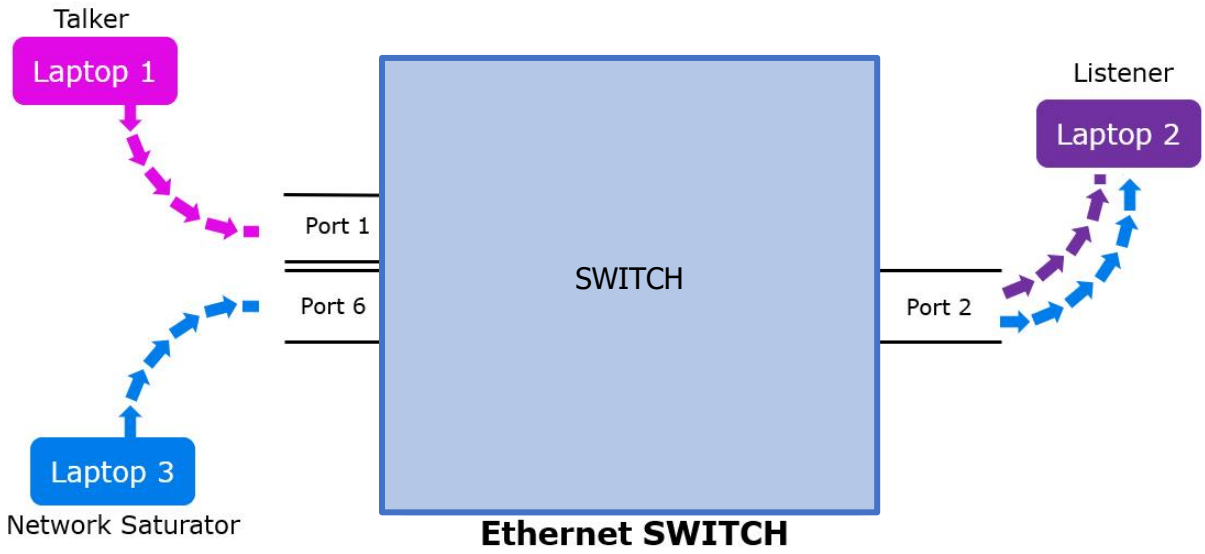


Figure 41. Network Structure for PTP tests (1 and 2).

The network structure is similar to the AVB test. The only difference is that instead of using a talker and a listener, a PTP Master and Slave are used.

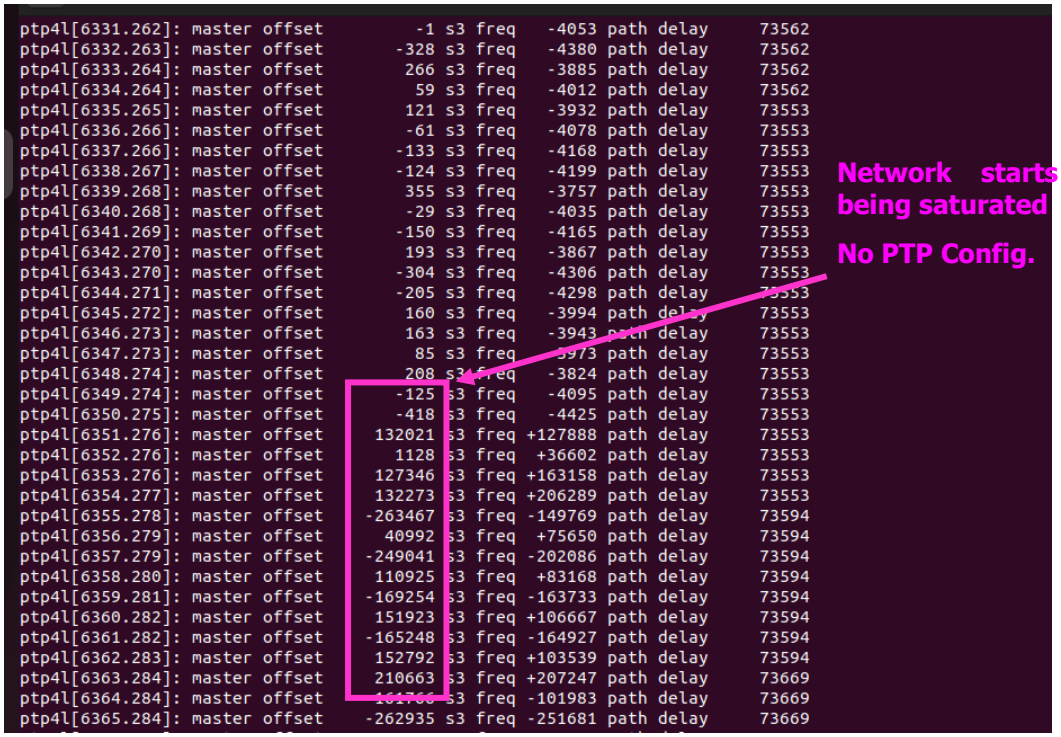


Figure 42. Test 1 result - No PTP configuration. Units of offset are in nanoseconds.

As can be seen in the previous image, if the network is not saturated, the result of the offset is less than 1000 ns. However, if no PTP is configured, the clock offset increases considerably when the network is saturated.

3.5.4. Test 2 – PTP configured

The network structure is exactly the same as in test 1. However, the difference in this latest test is that PTP is properly configured. Hence, time accuracy will be greater.

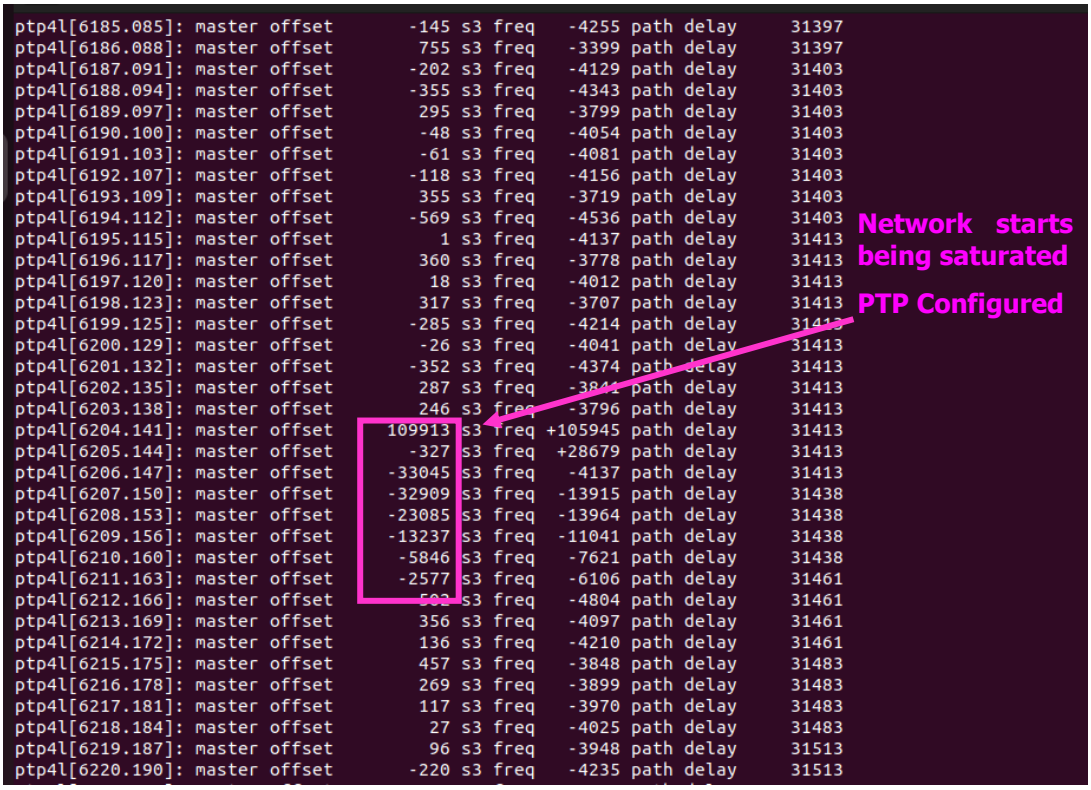


Figure 43. Test 2 result - PTP has been configured. Units of offset are in nanoseconds.

3.6. Mixing AVB, SRP and gPTP

After testing all the previous parts separately, it is time to join them within the same network. The device with the most precise clock (laptop 1) acts as the source of synchronization or Grandmaster clock as well as a video Talker. A second device (laptop 2) acts as a Listener of device's 1 streaming and a Slave of the clock source. Finally, a third device (laptop 3) connects to the network with the only purpose of saturating it to check whether the configuration is working or not. In the following image there is the network structure used:

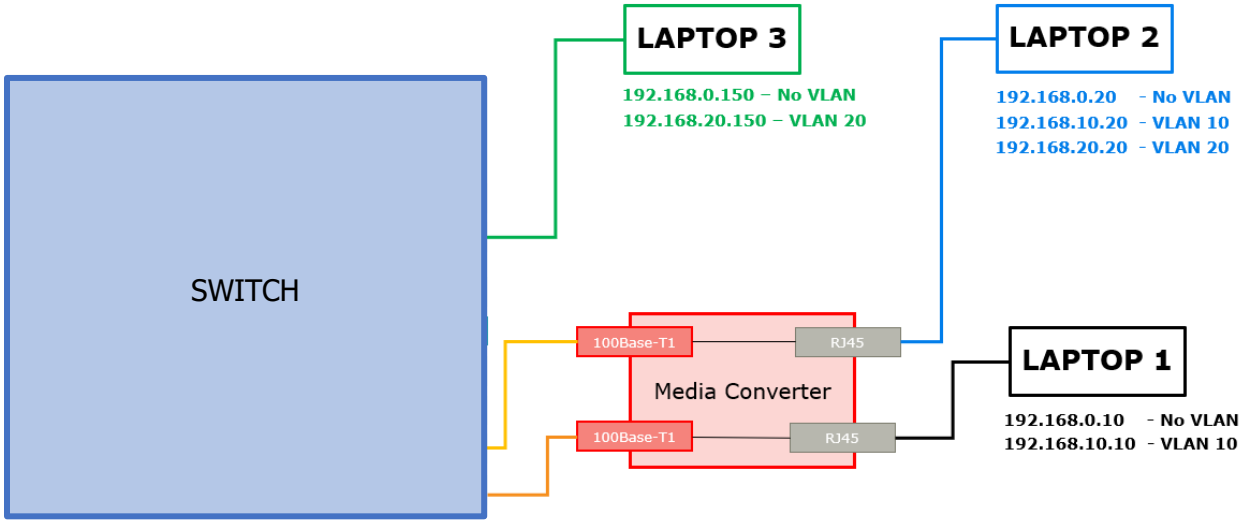


Figure 44. Network structure used for this test. It is the same structure as in the PTP and AVB individual tests.

Since this test is quite complicated, a message flow diagram has also been included. This diagram shows the direction of each type of message and the source and destination.

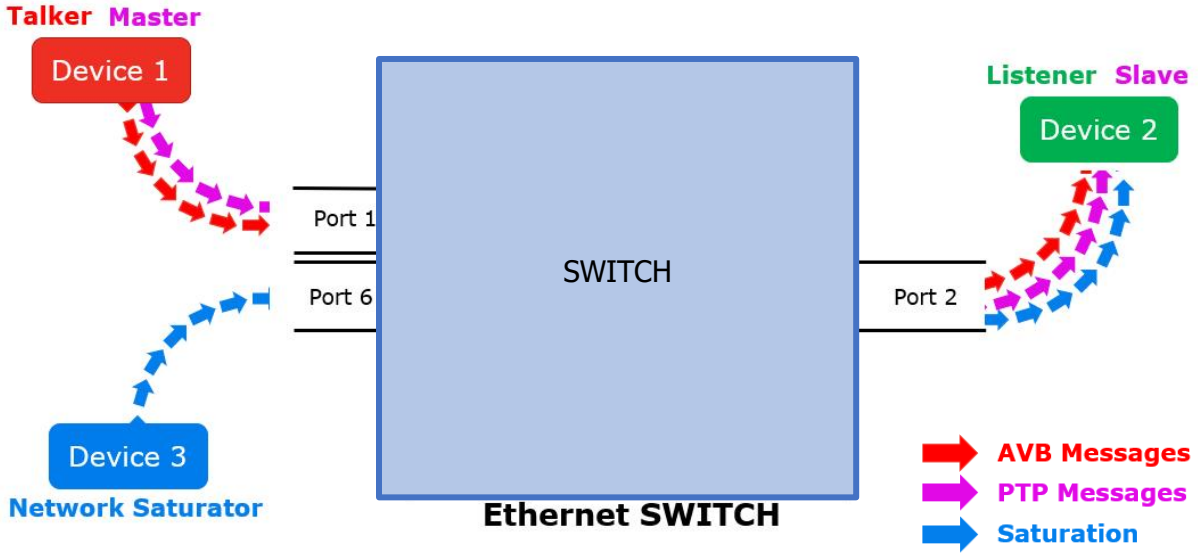


Figure 45. Flow diagram of the messages.

Launch

Note: before launching any video, make sure that the respective VLANs are configured as seen in the first picture in this chapter. Otherwise, no results will be seen.

```
#Laptop 1, (2 shell tabs)
sudo nice --20 ptp4l -i enp0s31f6 -m -f /usr/share/doc/linuxptp/configs/automotive-master.cfg --step_threshold=1

gst-launch-1.0 filesrc location=/home/jfonts/big_buck_bunny_1080p_h264.mov ! decodebin ! x264enc ! rtph264pay ! udpsink host=192.168.10.20 port=8080
```

```
#Laptop 2, (3 shell tabs)
sudo nice --20 ptp4l -i enp0s31f6 -m -f /usr/share/doc/linuxptp/configs/automotive-master.cfg --step_threshold=1

gst-launch-1.0 udpsrc port=8080 ! application/x-rtp ! rtph264depay ! avdec_h264 ! autovideosink

iperf -u -s
```

```
#Laptop 3 (1 shell tab)
iperf -u -c 192.168.20.20 -p  -i 1 -b 100m -t 10000
```

When all the programs have been launched, the result obtained is successful and can be seen in the following images:

Laptop 1

In the previous image there are the two programs that are launched in the first laptop. On the one hand, the PTP Master of synchronization (grandmaster clock) which corresponds to the shell tab of the left. On the other hand, the AVB Talker which corresponds to the right tab of the shell.

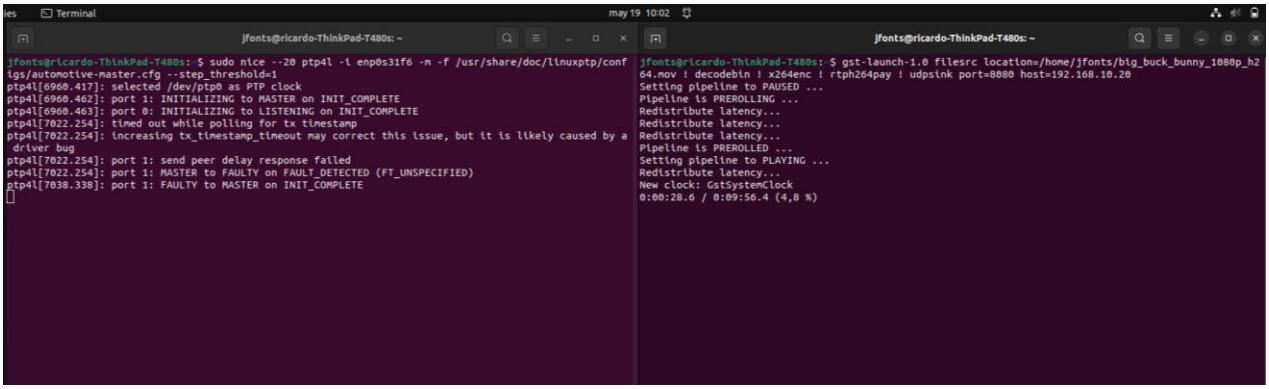


Figure 46. Test launched in laptop 1.

Note: remember to configure the switch as seen in the previous chapters and remember to use the peak bitrate value (obtained with Wireshark graphics) to configure the AVB features.

Laptop 2

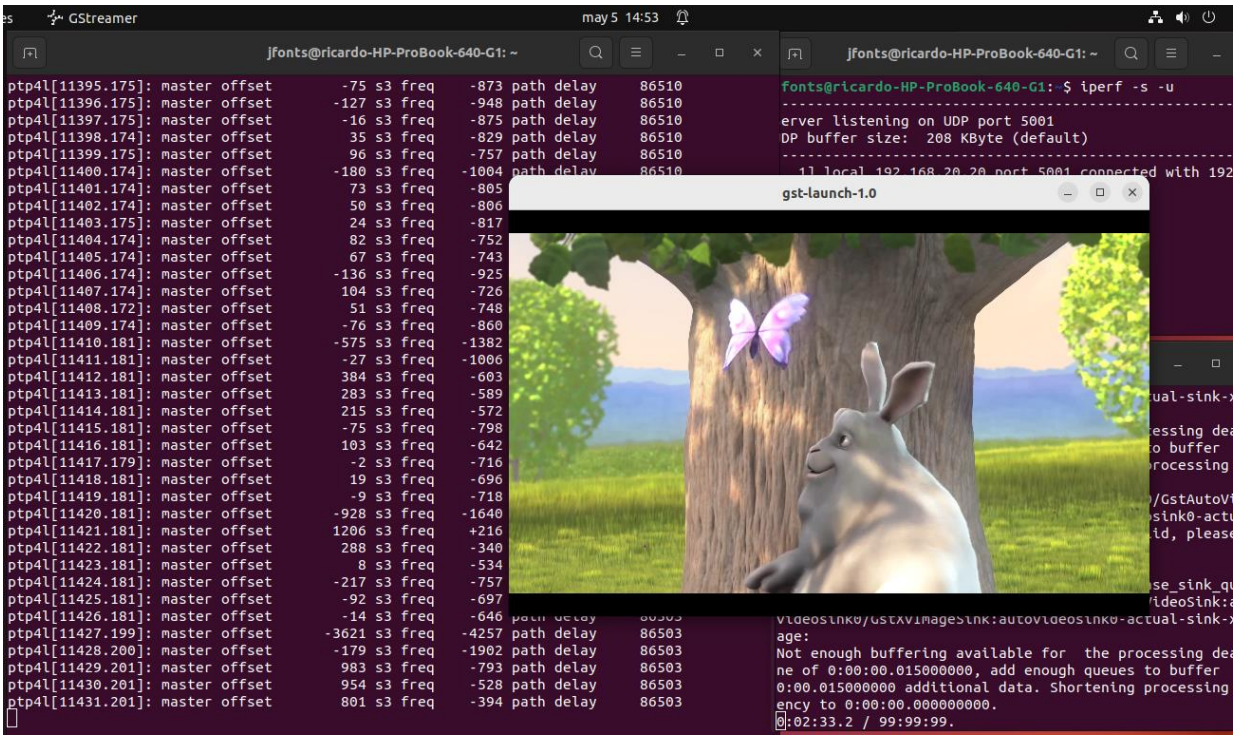


Figure 47. Launched programs in laptop 2.

The image shows all the different programs that have been launched in the second laptop. Firstly, the PTP Slave on the left tab, which has an offset of just a few nanoseconds since the Switch has been configured properly. Secondly, the AVB Listener tab on the bottom-right tab, which takes care of receiving the video frames and play them in the screen (*gst-launch-1.0*). And, thirdly, the *iperf* server on the top-right tab, which receives the frames of the network saturator.

Note: when the network starts being saturated, the offset of the PTP Slave increases slightly. However, since PTP tab if configured, the Slave adjusts the offset until it gets back down to hundreds of nanoseconds. After a few hours of trying to solve this issue, the conclusion was reached that the problem was due to Ubuntu OS and the Ethernet card.

Note 2: even though the network is saturated, the offset does keeps at a good value and, although in an image cannot be appreciated, the video is not laggy at all.

Laptop 3

This laptop has the only function of saturating the network to test if the configuration works correctly. Since the media converter is limited to 100 Mbps, the laptop 3 sends UDP useless traffic to the Port 2 (with IP = 192.168.20.20 and VLAN ID = 20) and with a max capacity of 100 Mbps. In the image below it can be seen that the bandwidth is between 95 – 100 Mbps, which is barely the whole bandwidth capable.

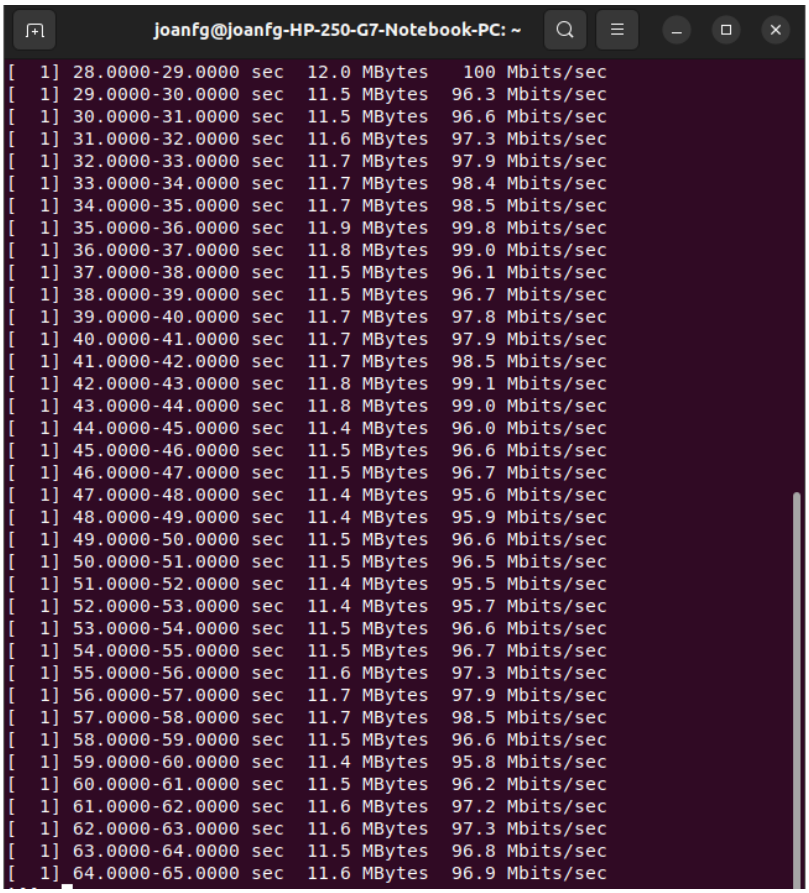


Figure 48. Image of iperf saturating the network.

3.7. MACsec implementation

Encryption in Automotive Ethernet has become essential in the last years. Therefore, it is the last but not least step in order to complete this project. Since the actual version of the switch does not support MACsec, this test will be implemented between two endpoints (laptops) which will be directly wired-connected.

In the first section, the MACsec encryption is generated from a Pre-Shared Key (PSK) which coincides between both endpoints. Nevertheless, in the second section the MACsec encryption is generated using the Transport Layer Security protocol (TLS) which exchanges public keys and digital certificates between endpoints.

In the future uses of the project, the switch would be added to the network with the proper configuration to decrypt and forward messages.

3.7.1. MACsec with a Pre-Shared Key (PSK)

The method used in this section is the *Static CAK Mode*, where a Pre-Shared Key generates the Connectivity Association Key (CAK) and the Connectivity Association Key Name (CKN). Those keys are the base of the generation of the KEK, ICK and SAK, which are exchanged and used to encrypt all the messages.

The configuration of the network interface can be done with the `wpa_supplicant.conf` file.

wpa_supplicant.conf:

```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=3   #EAPOL protocol version
ap_scan=0
fast_reauth=1  #Enable Fast Re-authentication (1) or not (0)

network={
     key_mgmt=NONE  #List of accepted protocols for key management
     eapol_flags=0  #1: unicast WEP keys   2: broadcast WEP keys   3: both
     macsec_policy=1

     #16-byte and 32-byte CAK and CKN keys (must be the same in both endpoints)
     mka_cak=a9ff6781a38ffedaaf9b10a085a087c2
     mka_ckn=85aa384f266aa61e2e65cbb36a80eb08895aa54afb651869586b32a8982ec0d1
}
```

Once this file has been configured, the run command is the following. Remember to change the "`-i`" field for the corresponding Ethernet interface used.

RUN CMD: sudo wpa_supplicant -i enp0s31f6 -Dmacsec_linux -c wpa_supplicant.conf

This file of configuration must be launched in both endpoints and, after this, a new interface is created with the "macsec0" name.

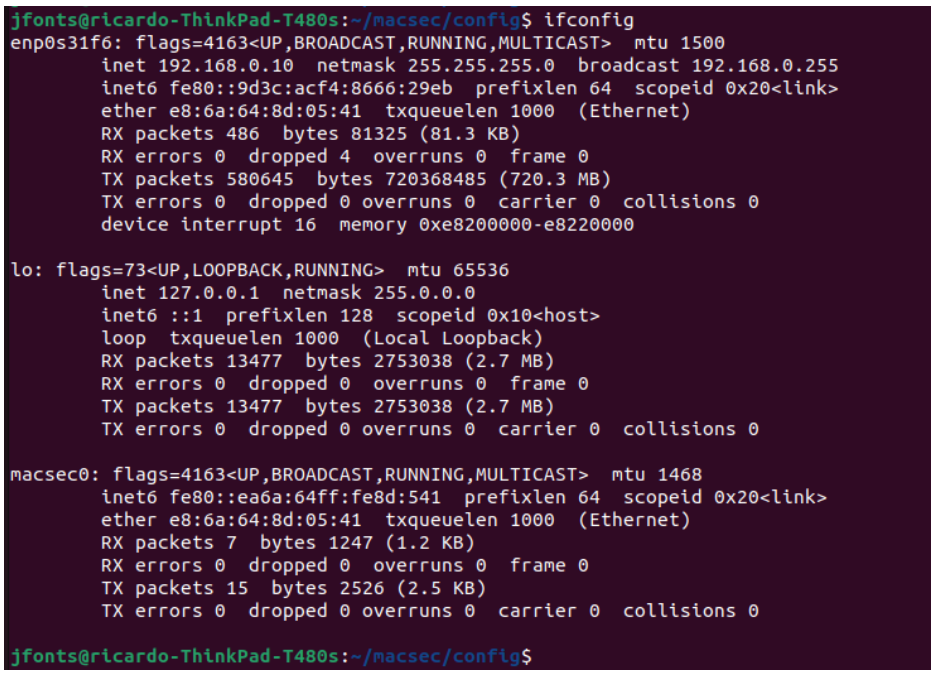


Figure 49. MACsec interface created.

As can be seen in the previous image, the newer interface has no IP associated and, therefore, the following command is used to assign an IP address.

ADD IP: sudo ip addr add 192.168.1.10/24 dev macsec0

Note: a different subnet is used with the macsec0 interface since it must be differentiated from the other interfaces. Furthermore, different IP addresses are used for Laptop 1 (192.168.1.10/24) and Laptop 2 (192.168.1.20/24).

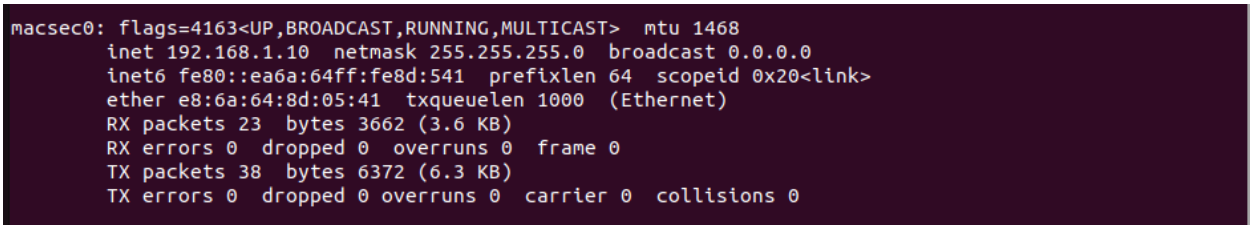


Figure 50. Added IP address to the macsec0 interface of Laptop 1.

Once the launch has been successfully done, the MACsec Key Agreement (MKA) starts between both endpoints. All the messages that are desired to be encrypted, must be sent through this interface. Otherwise, the messages will not be encrypted.

To test whether the encryption works, an ICMP-type message is used as shown below:

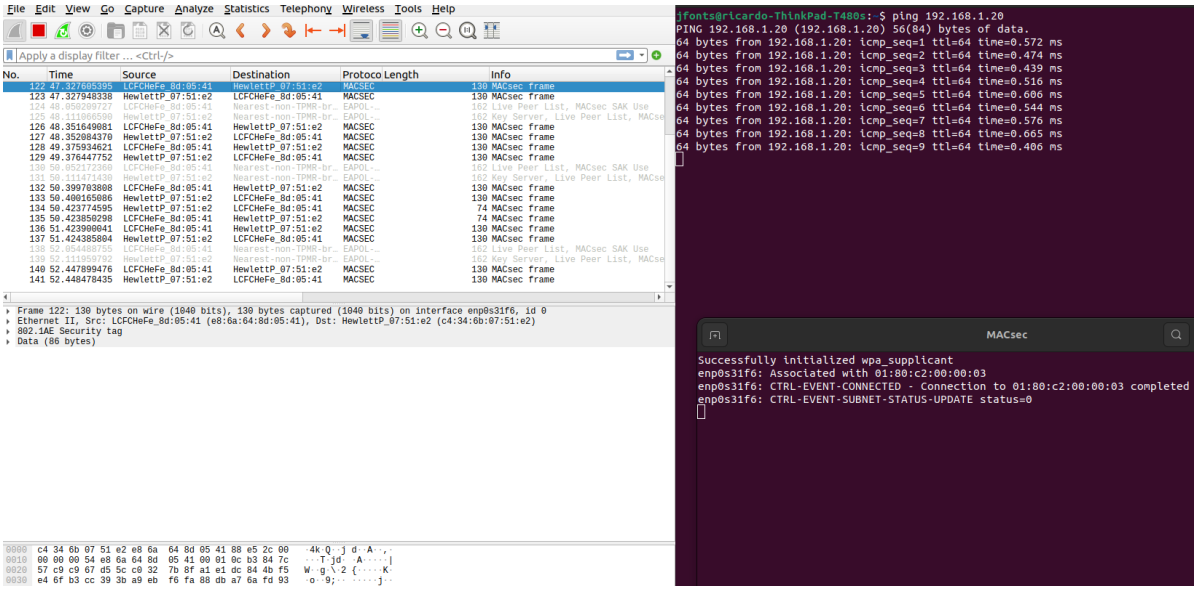


Figure 51. Working MACsec encryption between laptops with a PSK.

In the previous image there are several results which can be explained. On the left side of the image, the Wireshark is used to sniff all the MACsec packets. On the right side, is can be observed the ICMP message sent to the 192.168.1.20 address (note that the subnet corresponds to the macsec0 interface) and also the MACsec wpa_supplicant that has started.

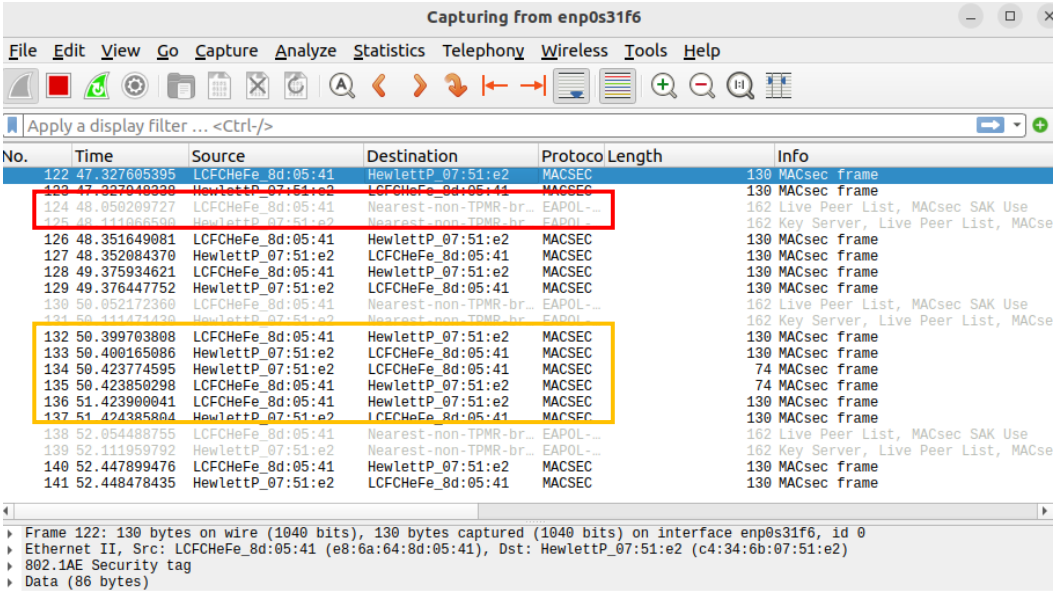


Figure 52. Amplified image of the Wireshark MACsec packets.

In this last figure, MKA messages are altered with the encrypted ICMP messages. Since we used the fast_reauth option, the MKA messages (red square) are sent more frequently. The orange square represents the encrypted ICMP messages which are sent from the Laptop 1 to the Laptop 2.

3.9.2. MACsec with TLS

The method used is the *Dynamic CAK Mode*, where both devices exchange the MKS (Master Session Key) with the use of asymmetric encryption. Both endpoints also exchange certificates.

The configuration of the network interface can be done with the `wpa_supplicant.conf` file.

wpa_supplicant.conf:

```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=2
ap_scan=1
fast_reauth=1

network={
     ssid="macsec0"
     scan_ssid=1
     key_mgmt=WPA-EAP
     pairwise=CCMP TKIP
     eap=TLS
     identity="laptop_1@macsec.com"
     ca_cert="/etc/cert/ca.pem"
     client_cert="/etc/cert/user.pem"
     private_key="/etc/cert/user.prv"
     private_key_passwd="123456789"
}
```

Generate certificate:

```
sudo apt install openssl

sudo openssl req -newkey rsa:4096 -x509 -sha512 -days 365 -nodes -out ca.pem -keyout
usr.prv
```

Note: after several tries, this configuration did not work properly. In future uses of this project this section would require more investigation.

3.9.3. Comparison between PSK and TLS

After testing both configurations, it can be assured that the second method is far more secure than the first one. If a PSK is used, it can be hacked more easily than the second method and, therefore, the TLS is preferably used.

4. CONCLUSIONS

After finishing the project and reaching the final results, many conclusions can be extracted. In this section, not only the results and conclusions will be explained, but also the next steps that the project is required to have in order to continue its development.

Objectives

Regarding the objectives that were defined in this project, all of them have been achieved successfully according to the hardware capabilities.

The first objective consisted of studying all the protocols related to Audio Video Bridging, Time Synchronization and MACsec encryption as well as researching the TCP/IP model. This research has been done successfully and all the documentation has been written in the second chapter of the project. Many information sources have been compared in order to obtain the highest accuracy possible in explaining and summarizing all the protocols. Furthermore, I would like to remark the huge knowledge that has been acquired during this research. Not only did I research about those protocols separately, but also understand how they are built within an Automobile framework, which has been essential to complete the further objectives. The respective testbenches have also helped during this process of comprehension. Therefore, it can be concluded that the first objective has been achieved.

As far as the second objective is concerned, a total of 47 Ethernet Switches have been analyzed. Consequently, the second objective of the project has been completed successfully.

Regarding the third objective, the Ethernet Switch was programmed successfully. The Virtual LANs, Audio Video Bridging, Stream Reservation and Time Synchronization have been accomplished and the launched tests have worked as expected. The endpoints used have been laptops with Ubuntu OS and hardware Time-Stamping, which definitely improved the offset in Time Synchronization and leading to better results. As far as MACsec within the switch is concerned, I will mention it further on.

Setbacks

Comparing the consumed time in all the parts of the project, this last objective has been by far the most time-consuming task to do. To launch the tests with Ubuntu, much research had to be done previously through the Internet. Not only was the information scarce, but also contradictory.

Before solving a problem, a theoretical and schematic approach was done. In networking is essential to have the network structure clear in order to achieve de desired results. However, even though having all the structure clear and configuring every endpoint correctly, many protocols still did not work. That was due to lack of information regarding the Switch component or the internal procedures of the Ubuntu OS. Therefore, a few problems were solved by brute force, repeating all the possible solutions and uploading them (if it implied the Ethernet Switch) to the board until it worked. Once the problem was solved, a proper documentation was written to ensure that if anyone had that same issue, they require less time to solve it.

I would like to mention that the datasheets of the hardware component were not quite well explained. Moreover, the information was not prioritized and distributed equally which also increased the spent time while working with it.

As said before, the Internet research was not an easy part too. For instance, the GStreamer installation was quite a headache, since many different possible ways were encountered. One of them broke the entire OS which had to be reinstalled.

Apart from the previous issues mentioned above, after couple of months of work with the evaluation board, many problems were also encountered. On the one hand, the silicon version of the hardware was "A", which did not support MACsec encryption. During the project, the newer version "B" of the silicon was released which did support the encryption. Even though, the firmware to program the Switch has not been developed yet and, therefore, there is no use in changing the hardware from A to B since MACsec would still be disabled.

Apart from the MACsec issues, many other features of the Switch need to be furtherly developed. To implement Stream Reservation Protocol, the tab in the GUI did not work and ATU and AVB tab had to be used. Although the SRP did work properly, in my opinion I believe that many improvements to the firmware need to be done. Nevertheless, I believe that Ethernet Switches are a very recent topic and other similar problems would have been found if another brand switch had been selected.

Even though, and considering our hardware and software capabilities, all the objectives have been accomplished successfully, even all those setbacks.

Next Steps

The projects prospects are evident. Nowadays, Automotive Ethernet is being introduced within the automobiles and that is why it will be developed during the following years. Although the time limit of the project has been reached, I would like to mention the next steps that would be carried out in order to continue the developing of the project.

The media converter, which converts the basic Ethernet connector RJ45 to Automotive Ethernet BASE-T1 and vice versa, is limited to 100 Mbps. A further step would be to test how the communications work when reducing the whole bandwidth down to 10 Mbps. This is an interesting step to check how SRP and AVB work when reaching low bus speeds.

On the contrary, another step would be to increase the bus speed up to 1000 Mbps. This process would also require changing the media converter to another one which supported that speed. It would be possible to escalate the whole network to a more realistic approach.

Laptops with Ubuntu OS have been used as endpoints. Those laptops have many functions and processes to run and, even though the PTP and AVB (Talker and Listener) are run with high priority, there is still a small delay associated to other OS processes. Therefore, to improve this a real automotive hardware would be used with time-critical ECUs.

The third next step would be to improve MACsec. This step can only happen as long as the silicon and the firmware support this feature. As seen in the theoretical section of the project, the switch will act as an Authenticator and the endpoints as Supplicants. Furthermore, a RADIUS server would be implemented to communicate with the Ethernet switch and the corresponding certificates need to be exchanged.

Although those next steps would be the ones immediately after the project, the prospects of the Automotive Ethernet are evident. Many brands are now moving forward to implementing Ethernet within their automobile's framework. Moreover, the Ethernet within the automobiles is becoming a reality which is being introduced in the cars we will drive in the near future.

Lear

This project has been done with the collaboration of Lear Corporation. Working at this organization has been extremely rewarding and has given me the chance to work with state-of-the-art technologies.

During this period of time, I have been in constant improvement and always learning new ways of solving the problems. The project involved lot of hard-working periods and well as presentations, which definitely improved my communication skills.

The organization also allowed me to gain work experience while working in a good working atmosphere. I would also like to remark the people networking that has been done in Lear.

Summary

To sum up, this project involved hard-work, improvement and learning. The newest and latest technologies were used to bring the project up to the highest level possible, allowing me to get done all of the results. With problem-solving skills, the different objectives were successfully accomplished. Lear corporation is a great organization to work in which helped to achieve the objects with the perfection it was required. As far as the project prospects are concerned, it is clear that many automobiles in the near future will use this technology and Automotive Ethernet will become essential in the vehicles that are coming in the next years.

5. REFERENCES

[1] KOZIEROK, Charles, CORREA, Colt, BOARIGHT, Robert and QUESNELLE, Jeffrey (2014). *Automotive Ethernet: The Definitive Guide.* USA: Harman and Intrepid Control Systems, INC. 1127. p.

[2] Website with information about TCP/IP model. Available: www.tcpipguide.com [Last seen 17 August 2022].

Scapy

[3] Download Scapy. Available: https://github.com/secdev/scapy/releases [17 Aug 2022].

[4] Implementation of MACsec with Scapy. https://fossies.org/linux/privat/scapy-2.4.5.tar.gz/scapy-2.4.5/test/contrib/macsec.uts [09 Sep 2022].

[5] Documentation of Scapy. https://scapy.readthedocs.io/en/latest/api/scapy.html [09 Sep 2022].

[6] Use Scapy with Shell. https://scapy.readthedocs.io/en/latest/usage.html [09 Sep 2022].

[7] Three-way Handshake with Scapy. https://stackoverflow.com/questions/26480854/3-way-handshake-in-scapy [09 Sep 2022].

Ubuntu

[8] Create .sh file. Available: https://www.cyberciti.biz/faq/run-execute-sh-shell-script/ [08 Sep 2022].

[9] Configuration of VLAN. Available: https://wiki.ubuntu.com/vlan [24 Aug 2022].

Time Synchronization (gPTP)

[10] *Time Synchronization in Embedded Systems, overview of gPTP (Precision Time Protocol)*. YouTube, uploaded by Intrepid Control Systems, 4th February of 2021. Available: https://www.youtube.com/watch?v=ZrujSridNpQ

[11] *Introduction to Precision Time Protocol (PTP)*. YouTube, uploaded by Cisco IoT TME TV, 11th August of 2021. Available: https://www.youtube.com/watch?v=ovzt3IUFbyo

[12] *How a PTP slave syncs with a PTP master*. YouTube, uploaded by David Gessner, 28th March of 2015. Available: https://www.youtube.com/watch?v=Forh3XfD_Ec

[13] *Precision Time Protocol (PTP) Clock Types*. YouTube, uploaded by Cisco IoT TME TV, 13th August of 2020. Available: https://www.youtube.com/watch?v=rbb9DcIGLKY

[14] PARSONS, Glenn. *IEEE P802.1AS-Rev/D8.0 – Timing and Synchronization for Time-Sensitive Applications.* Institute of Electrical and Electronics Engineers, Inc, 2019. Available: https://1.ieee802.org/wp-content/uploads/2019/03/802-1AS-rev-d8-0.pdf

[15] GOETZ, Franz-Josef. *IEEE 802.1 AS, gPTP – One Step Issues.* Siemens AG. Geneva, Switzerland, 13th July of 2013. Available: https://www.itu.int/en/ITU-T/Workshops-and-Seminars/ethernet/201307/Documents/S3P2_Franz_Goetz.pdf

[16] GOETZ, Franz-Josef. *High Available Synchronization with IEEE 802.1AS*. Siemens AG. Orlando, USA, 19th March of 2013. Available: https://www.ieee802.org/1/files/public/docs2013/asbt-goetz-HighAvailableSync-0319-v01.pdf

[17] GARNER, Geoffrey M. *State Machines and Message Formats for IEEE 802.1AS.* Samsung, 8th March of 2008. Available: https://www.ieee802.org/1/files/public/docs2007/as-garner-protocol-state-machines-frame-formats-0307.pdf

[18] CORREA, Colt. *Time synchronization in embedded systems*. COO – Intrepid Control Systems, July of 2013. Available: https://cdn.intrepidcs.net/events/Webinars/Time_Sync_Embedded_Systems_gPTP_20210125.pdf

[19] Clock Types of the gPTP IEEE 1588 Standard. https://blog.meinbergglobal.com/2013/10/21/ieee-1588-clock-types/ [08 February 2023]

MACsec

[20] VÖLKER, Dr. Lars. *Starting Up MACsec for Automotive ethernet. 7th International VDI Conference – Cyber Security for Vehicles.* June of 2021. Available: https://automotive-network-security.com/papers/2021-06-22_VDI_CyberSecurityVehicles-DrLarsVoelker_v.1.0a.pdf

[21] HILL, Craig and ORR Stephen. Innovations in Ethernet Encryption (802.1AE – MACsec) for Securing High Speed (1-1000GE) WAN Deployments. Cisco, 2016. V: https://www.cisco.com/c/dam/en/us/td/docs/solutions/Enterprise/Security/MACsec/WP-High-Speed-WAN-Encrypt-MACsec.pdf

[22] *Recomendaciones de Seguridad para MACsec.* Centro Criptológico Nacional, 2006. Available: https://www.ccn.cni.es/index.php/es/docman/documentos-publicos/boletines-pytec/470-pildorapytec-abr2021-seguridad-macsec [06 February 2023].

[23] Understanding Media Access Control Security (MACsec) and the Static and Dynamic Mode. https://www.juniper.net/documentation/us/en/software/junos/security-services/topics/topic-map/understanding_media_access_control_security_qfx_ex.html [22 August 2022].

[24] Introduction to MACsec in the Automotive E/E. YouTube, uploaded by VECTOR, 8th July of 2021. Available: https://www.youtube.com/watch?v=j4OZhV2CB3E

[25] Information about the MACsec 802.1AE Header and its format and structure. https://infocenter.nokia.com/public/7750SR217R1A/index.jsp?topic=%2Fcom.nokia.Interface_Configuration_Guide_21.7.R1%2Fmacsec_802-1ae_-ai9emdynxf.html [13 September 2022].

[26] Information about MACsec Frame Format. https://docs.commscope.com/bundle/fastiron-08092-securityguide/page/GUID-333630FE-363D-43F1-A4C9-0EDD0D0D53E2.html [13 September 2022].

[27] MACsec config (Option 1). https://developers.redhat.com/blog/2016/10/14/macsec-a-different-solution-to-encrypt-network-traffic# [08 Sep 2022].

[28] MACsec config (Option 2). https://nextheader.net/2016/10/14/macsec-on-linux/ [08 Sep 2022].

Stream Reservation Protocol (SRP)

[29] Bandwidth Reservation. YouTube, uploaded by Kane Zhang, 20th September of 2017. Available: https://www.youtube.com/watch?v=7lqsmMXuIXU

[30] Milan AVB (English). YouTube, uploaded by Meyer Sound, 1st June of 2020. Available: https://www.youtube.com/watch?v=hydphfAgJvQ

[31] Audio Networking Webinar: Dante. AVB, and MADI. YouTube, uploaded by RME USA Audio – Synthax, 10th March of 2021. Available: https://www.youtube.com/watch?v=ymouSeTBYvM

[32] PEARSON, Levi. *Stream Reservation Protocol. Revision 1.0.* AVnu Alliance™ Best Practices, November 2014. Available: https://avnu.org/wp-content/uploads/2014/05/AVnu_Stream-Reservation-Protocol-v1.pdf

[33] FENG, Felix. IEEE802.1Qat: Stream Reservation Protocol. Samsung Electronics, 2007. Available: https://www.ieee802.org/1/files/public/docs2007/at-feng-srp-summary-0707.pdf

[34] FENG, Felix. GARP-based Simple Reservation Protocol. Samsung Electronics, 2005. Available: https://grouper.ieee.org/groups/802/3/re_study/public/200507/feng_1_050719.pdf

[35] SEAMAN, Mick. *MRP State Machines.* 11th June of 2004. Available: https://www.ieee802.org/1/files/public/docs2004/ai-seaman-MRP-machines-09.pdf

[36] GUNTHER, Craig and INTERNATIONAL Harman. SRP in Automotive. AVnu, 8th January of 2010. Available: https://avnu.org/wp-content/uploads/2014/05/AVnu-AAA2C_SRP-in-Automotive_Craig-Gunther.pdf

[37] Performance Analysis of the IEEE 802.1 Ethernet Audio/Video Bridging Standard. https://www.researchgate.net/publication/262323002_Performance_Analysis_of_the_IEEE_802_1_Ethernet_AudioVideo_Bridging_Standard [09 September 2022].

Forwarding and queuing Enhancements for time Sensitive Streams (FQTSS)

[38] FREDETTE, Andre. *"How big do my pipes need to be?" – Traffic Shaping & Infrastructure planning.* AVnu Alliance, 16th May 2013. Available: https://avnu.org/wp-content/uploads/2014/05/AVnu-AABAC_Traffic-Shaping-Infrastructure-Planning_Andre-Fredette.pdf

[39] REGEV, Alon and TENEA Bogdan. *Is AVB really much better than classic IEEE 802.1Q queuing?* Keysight Technologies, Paris, France, 22nd September of 2016. Download link: https://www.keysight.com/us/en/assets/7019-0405/technical-overviews/Is-AVB-Better-than-Classi.pdf

[40] *FQTSS Overview*. AVnu Testing Consortium, 2011. Available: https://www.iol.unh.edu/sites/default/files/knowledgebase/avnu/FQTSS_OVERVIEW.pdf

AVB

[41] Install GStreamer on Linux. https://gstreamer.freedesktop.org/documentation/installing/on-linux.html?gi-language=c [05 March 2023].

[42] Use of gst-launch-1.0. https://gstreamer.freedesktop.org/documentation/tools/gst-launch.html?gi-language=c [20 March 2023].

[43] Information regarding the use of Gstreamer and different types of encoding. https://blueye-robotics.github.io/blueye.sdk/docs/video/gstreamer-for-video-streaming/ [20 March 2023].

[44] Talker and Listener (Server and Client) example with MJPEG and H264 encoding. https://gist.github.com/reinzor/812a1dadd62dcf2a309c9c99af92244f [20 March 2023].

[45] Downloading of example video. https://download.blender.org/peach/bigbuckbunny_movies/ [05 March 2023].

6. GLOSSARY

General

AVB: Audio Video Bridging.

CRC: Cyclic Redundancy Code.

CPU: Central Processing Unit.

CSMA/CD: Carrier Sense Multiple Access with Collision Detection.

DIP Switch: Dual In-Line Package. Used to refer to the small switches used for the bootstrap configuration. Nomrally they are of red and white colours.

DDR: Double Data Rate.

DTLS: Datagram Transport Layer Security (UDP).

EOF: End of Frame.

FCS: Frame Check Sequence.

FQTSS: Forwarding and Queuing for Time Sensitive Streams.

GMII: Giga-bit Media Independent Interface.

IP: Internet Protocol.

IPsec: internet Protocol Security.

MACsec: Media Access Control Security.

MII: Media Independent Interface.

OSI: Open Systems Interconnection.

PCIe: Peripheral Component Interconnect Express.

PTP: Precision Time Protocol.

QoS: Quality of Service

RGMII: Reduced Giga-bit Media Independent Interface.

RMII: Reduced Media independent Interface.

Rx: used to refer to the reception port.

SDR: Single Data Rate.

SGMII: Serial Giga-bit Media Independent Interface.

SRP: Stream Reservation Protocol.

SOF: Start of Frame.

TCP: Transport Control Protocol.

TLS: Transport Layer Security.

TSN: Time Sensitive Networking.

Tx: used to refer to the transmission port.

UDP: User Datagram Protocol.

VLAN: Virtual Local Area Network

VM: Virtual Machine.

Encryption

AES: Advanced Encryption Standard.

CA: Connectivity Association.

CAK: Connectivity Association Key.

CKN: Connectivity Association Key Name.

EAP: Extensible Authentication Protocol.

EAPoL: Extensible Authentication Protocol over LAN.

ICK: Integrity Check Key.

ICV: Integrity Check Value.

KDF: Key Derivation Function.

KS: Key Server.

MKA: MACsec Key Agreement.

MSK: Master Session Key.

PSK: Pre-Selected Key.

SA: Authentication Server.

SAK: Secure Association Key.

Communications

Phy: the electronic element used to communicate with external devices. It is necessary to give current to the signal to ensure it reaches its destination.

100Base-TX: communication of 100 Mbps that does not require to cross cables between transmitter and receiver.

1000Base-T: same as Base-TX but with speeds of 1000 Mbps

1000Base-T1: used in automation. It transmits data at speed rates of 1000 Mbps with a differential pair. The Rx and Tx are multiplexed in frequency.

File Types (name.type)

lua: the type used as a dissector in wireshark.

bin: the file that is being flashed into the evaluation board.

xml: file that saves the tool configuration of the selected features of the switch.

7. ANNEXED INFORMATION

7.1. GStreamer

7.1.1. Installation

To stream some video/audio, it is firstly required to install GStreamer. Go to the official page: https://gstreamer.freedesktop.org/documentation/installing/on-linux.html?gi-language=c

```
sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-
good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav
gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-
gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
```

If some error shows up regarding unmet dependencies, the following command solves it:

```
sudo apt-get -u dist-upgrade
sudo apt-get install libgbm1   //This command is not tested if it works
```

To check whether the installation has worked or not, this command can be used:

```
gst-launch-1.0 videotestsrc pattern=0 ! videoconvert ! autovideosink
```

Figure 53. Example to test if GStreamer has been installed successfully.

Pipeline: the information of the previous instruction is passed over the next one. For example, in the previous command, the `gst-launch-1.0 videotestsrc pattern=0` generates a video with the videotestsrc field and then it is passed to the `videoconvert` which transforms the raw video so other elements can read it. Finally, the `autovideosink` generates the window so the video can be played on a visible format.

videotestsrc: to generate the video which is seen in the screen. It's the source of the video generated in raw format.

pattern: to change the type of video which is being sent (change 0 to other numbers to see the different types of videos).

videoconvert: converts video from raw format to the specific type needed for the following elements in the pipeline. The `videoconvert` is necessary before the `autovideosink` field.

autovideosink: to create the window into which the video generated will be seen.

7.1.2. Play downloaded file

Download file (.mkv) at the following link: https://gstreamer.freedesktop.org/media/. It is already downloaded in the documentation folder.

```
gst-launch-1.0 filesrc location=
/home/jfonts/documentation/test7_results/example_videos/sintel_trailer-480p.mkv !
decodebin ! videoconvert ! autovideosink
```

When using the `filesrc` field, the location must be specified and shown in the previous box. Furthermore, the field `decodebin` must be added in the pipeline so as to transform the .mkv source into a suitable format to pass to the `videoconvert` field. The final step is to show the video with the `autovideosink`.

decodebin: this field is used for automatically detect the source file type (.mkv in this case) and transform it into a raw format. This raw is a type of format where no conversion or compression has been applied yet, all the video bits are added one after the other as soon as they are read. The raw format will be used for the `videoconvert` field in the pipeline.

There is another way to play a video (.mkv) without using the playing field. This is with:

```
gst-launch-1.0 playbin uri=file:///home/jfonts/
documentation/test7_results/example_videos/sintel_trailer-480p.mkv
```

The field `uri=file://path_to_the_video` is used to indicate where the video is located.

playbin: a plugin of GStreamer that allows to playback any type of video. It merges the `decodebin` (seen before), the `videoconvert` and the `autovideosink` together.

If everything succeeds, in both cases a trailer video should pop up. It will look similar to this:

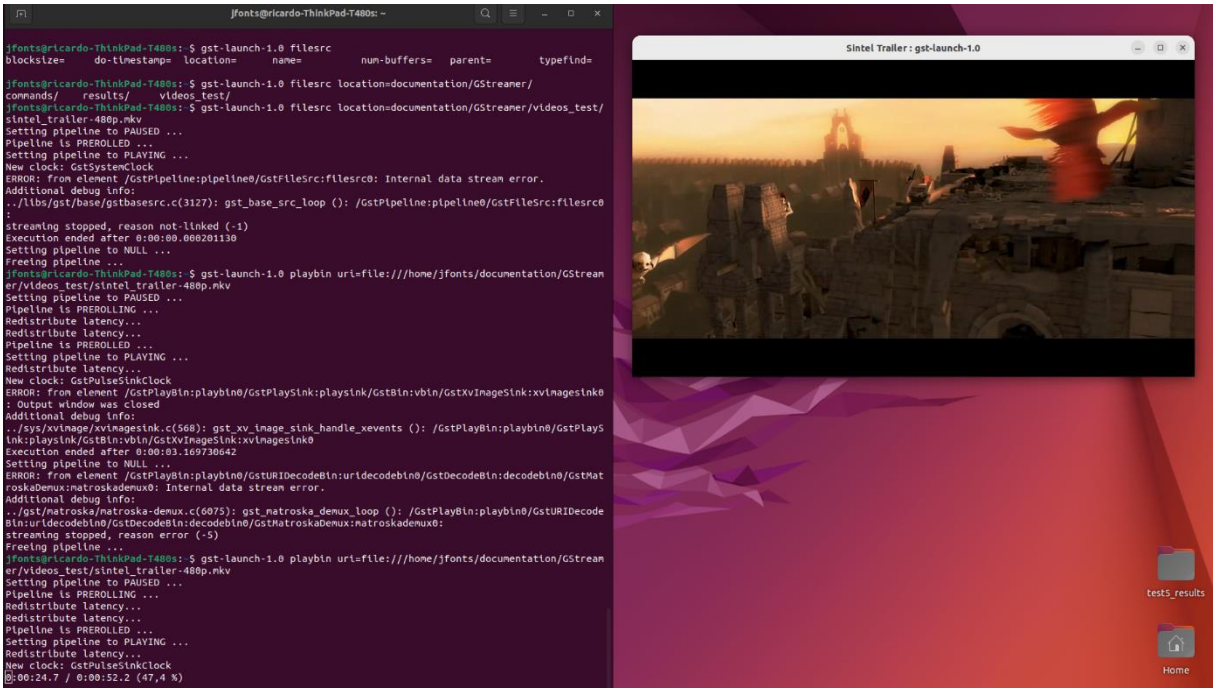


Figure 54. Result of playing video with "playbin".

7.1.3. Stream Video using Ethernet

To try this configuration, both endpoints (talker and listener) must be physically connected with an ethernet cable. The type of connection will be point to point between both devices or, if desired, a transparent switch can be added between endpoints.

7.1.3.1. JPEG encoding

The server will stream the packets which will be received by the client. The commands to run this type of streaming are:

Talker

```
gst-launch-1.0 filesrc location=
/home/jfonts/documentation/test7_results/example_videos/sintel_trailer-480p.mkv !
decodebin ! videoconvert ! jpegenc ! rtpjpegpay ! udpsink host=192.168.0.20
port=8080
```

The host is the IP address of the stream destination. If the stream needs to be tagged with a specific VLAN, it must be sent through another interface which is configured as tagged.

jpegenc: is used to encode the output coming from the videoconvert into .jpeg images. Those images will be furtherly used to stream the video.

rtpjpegpay: transforms the jpeg images generated before into RTP frames. The Real Time Protocol (RTP) frames are used for audio and video transmissions over UDP layer.

udpsink: used to transmit the frames coming from the `rtpjpegpay` (RTP frames) into UDP datagrams which will be sent to a specific socket (specifying the `host` and the `port`).

Listener

```
gst-launch-1.0 udpsrc  port=8080 ! application/x-rtp,encoding-name=JPEG,payload=26 !
rtpjpegdepay ! jpegdec ! autovideosink
```

The `rtpjpegdepay` and `jpegdec` are the same as the `rtpjpegpay` and the `jpegenc` but instead of encoding, they are used for decoding. In the client pipeline, they must be written the other way around since the process is inverted.

rtpjpegdepay: decodes RTP frames into .jpeg images.

jpegdec: transforms the .jpeg images into the right format which autovideosink uses.

The `application/x-rtp,encoding-name=JPEG,payload=26` converts the RTP payload into a format specified in the `encoding-name` field.

7.1.3.2. H264 encoding

In the previous test it only mattered that the frames could be transmitted from source to sink. However, if the Stream Reservation in the switch needs to be implemented, the frame size and frame rate have to be specified. Therefore, those parameters must be added to the command line and the codification will be changed from jpeg to H264.

Talker

```
gst-launch-1.0 filesrc location=
/home/jfonts/documentation/GStreamer/videos/sintel_trailer-480p.mkv ! decodebin !
videoconvert ! x264enc ! rtph264pay ! udpsink host=192.168.0.20 port=8080
```

x264enc: this field is responsible for converting the raw data coming from the `decodebin` into compressed H264-type frames. The different options available are:

rtph264pay: encapsulates the H264-type frames into RTP messages that will be sent to a specific host and port.

Listener

```
gst-launch-1.0    udpsrc    port=5000    !    application/x-rtp,media=video,clock-
rate=90000,encoding-name=H264 ! rtph264depay ! advec_h264 ! autovideosink
```

rtph264depay: converts the RTP messages into H264-type frames.

avec_h264: converts the H264-type frames into a suitable format that is being passes to the window to show it.

The application/x-rtp,media=video,clock-rate=90000,encoding-name=H264, converts the RTP payload into a format specified in the encoding-name field.

27th of May 2023

Joan Fonts Gómez