Universidade de Aveiro
2013

Departamento de Eletrónica,
Telecomunicações e Informática

André
Rodrigues Almeida

Reserva de Recursos em Automotive Ethernet

Resource Reservation in Automotive Ethernet

Universidade de Aveiro
2013

Departamento de Eletrónica,
Telecomunicações e Informática

André
Rodrigues Almeida

Reserva de Recursos em Automotive Ethernet

Resource Reservation in Automotive Ethernet

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor João Paulo Silva Barraca, Professor Assistente Convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Pedro Alexandre de Sousa Gonçalves, Professor adjunto do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais, António Manuel, Ana Maria pelo incansável apoio e à minha irmã Rita.

o júri / the jury

presidente / president

Prof. Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Joaquim José de Castro Ferreira
Professor Adjunto da Universidade de Aveiro

Prof. Doutor João Paulo Silva Barraca
Professor Assistente Convidado da Universidade de Aveiro (orientador)

agradecimentos /
acknowledgements

Agradeço toda a ajuda dos meus orientadores, professores João Paulo Barraca e Pedro Gonçalves pela a orientação prestada e conhecimentos partilhados, essenciais à realização desta dissertação.
Agradeço ao grupo ATNoG (Advanced Telecommunications and Networks Group) e ao Instituto de Telecomunicações de Aveiro por todo o apoio prestado.
A todos os meus colegas e amigos com quem partilhei este meu percurso académico, bem como a toda a minha família.

Palavras Chave Ethernet, AVB, SRP, MSRP, OMNeT++

Resumo

Nos últimos anos, a industria automóvel tem sofrido grandes evoluções, podendo-se destacar não só o crescente desenvolvimento de sistemas eletrónicos em contextos e funcionalidades cada vez mais variados, como também a crescente interacção deste com o condutor e o mundo exterior. Devido ao enorme aumento de tráfego envolvido nas comunicações que compõem esses sistemas, as tecnologias de redes usadas até então deixaram de ser tão apelativas e passaram-se a considerar alternativas económicamente mais competitivas como é o caso da Ethernet. O uso de redes Ethernet em âmbito automóvel levanta alguns problemas, nomeadamente no cumprimento de limites temporais e requisitos de recursos bem definidos.

O aparecimento de protocolos AVB (Audio Video Bridging) vem tentar colmatar vários problemas de gestão dinâmica de Qualidade de Serviço das redes Ethernet no domínio automóvel. O protocol de sinalização SRP (Stream Reservation Protocol) pode ser adaptado para redes Ethernet no contexto automóvel para proporcionar um mecanismo de reserva de recursos.

Para testar a viabilidade de métodos tão recentes, as ferramentes de simulação são de uma importância vital. Este trabalho apresententa uma implemetação do protocolo SRP (Stream Reservation Protocol) em ambiente de simulação OMNeT++. São apresentados os aspectos fundamentais do modelo implemetado bem como alguns testes funcionais de validação deste.

Keywords Ethernet, AVB, MSRP, OMNeT++

Abstract

In recent years, automotive industry has undergone major changes, being able to highlight not only the growing development of electronic systems in increasingly and varied features and contexts, as well as to cope with its growing interaction between with the driver and the outside world. Due to the huge amount of traffic involved in these system communications, networking technologies used so far are starting to be less appealing and the industry began to consider alternatives, economically more competitive as is the case of Ethernet. The use of Ethernet technology in automotive domains faces some challenges, namely with time constraints compliance and well defined resource requirements.

The emergence of AVB (Audio Video Bridging) protocols, is trying to tackle some of these problems of having dynamic Quality of Service management in automotive Ethernet networks. One example of such protocols is the signalling protocol (SRP Stream Reservation Protocol), which could be used for providing a resource reservation mechanism in an automotive Ethernet domain.

To test the feasibility of such recent methods, simulation tools are of paramount importance. This work presents an implementation of the SRP (Stream Reservation Protocol) in Omnet++, taking into account some of its constraints. It is described the fundamental aspects of this model implementation, as well as some functional tests.

Contents

List of Figures

List of Tables

1

Introduction

This first introductory chapter provides an overview of the motivation present for the study and realization of some issues addressed in this work. It's given some insight about the automotive industry evolution and some of its challenges, as well as the structure and organization present in this document.

1.1 Motivation

It is undeniable that in recent decades, the field of electronics has undergone a major expansion and innovation. With the advent of integrated circuits, a new range of microprocessors were made possible and electronic components started to increasingly affect our daily life. The miniaturization of an increasingly array of electronic circuits and its lower power consumption, has allowed the concept of embedded system, with its own characteristics depending on the specific area of application. As such it should come as no surprise the increasingly usage of embedded systems in all kind of equipments, varying from telecommunication devices, industrial machinery, avionics, medical devices, transportation vehicles, and multimedia, with manufacturers making them more sophisticated every day. With this wide spectrum of fields, the usage of large number of different electronic domains in the same system increased and the need to establish communication between those domains through a well designed network which supports certain premises became mandatory. Those premises depend on each field of application and its own requirements and must always be taken into account when developing its topology or communication protocols.

One particular field of application which has a vast number of embedded distributed systems and has particular bandwidth and time constraints is the automotive industry. This industry deals with a lot of real time applications, so it is of paramount importance to research for better solutions in order to deal with ever more complex problems in the automotive world.

For many years the automotive industry has been growing with the increase in production of cars and in-vehicle equipments by entities named Original Equipment Manufacturers (OEMs) . However 2008 was a turning point in this scenario, with OEMs loosing a lot of business with the global crisis. According to [1], car manufacturers will need to shift their production facilities from North America and European Union, due to high-costs, to more emerging markets, such as China, India and South America in order to remain competitive. The goal is to manufacture cars at low-cost and at the same time use the more advanced and cheaper technologies where it was not possible before. Currently the several network technologies used to interconnect the several Electronic Control Units (ECUs) which comprise the various subsystems of an automotive system, are very costly, complex and inflexible [2]. This is due to the automotive specific domain of these technologies and because of that, its OEMs are not able to export or extend them to other fields of application.

Nowadays an automotive system is comprised of many ECUs reaching up to 80 in premium cars [3], that exchange data between them. Those ECUs are grouped in a specific domain with similar characteristics and requirements. Each domain is influenced by a specific kind of function. Taking into account all these aspects each domain is connected with its own specific network protocol like Media Oriented Systems Transport (MOST) , Local Interconnect Network (LIN) , FlexRay and Controller Area Network (CAN) depending on the specific function that it affects. Even more, each domain communicates with each other through a gateway which enables the automotive system to coordinate different kinds of traffic.

Automotive industry is evolving not only in terms of hardware, but in terms of software too. Each ECU is run with more complex algorithms to accommodate increasingly complex applications like infotainment, electronic stability control, engine management system, and therefore uses more and more lines of code. In fact nowadays a premium class car could contain close to 100 million lines of software code [4]. To cope with the more complex nature of automotive functions, bandwidth constraints, upper bounded latency responses and more important cost manufacturing aspects, the usage of Ethernet is considered by the OEMs to be a good alternative either to integrate with existing network technologies or replace them altogether. As legacy Ethernet technology is a source of unpredictability and indeterminism in the medium access by it's Carrier Sense Media Access / Collision Detection (CSMA/CD) mechanism, switched Ethernet must therefore be changed to cope with all this characteristics mentioned above. If all these constraints are taken into account switched Ethernet could be used in the automotive field, which is why several mechanisms appeared in the market to be used in conjunction with switched Ethernet technology. The specification of Ethernet-based communication systems with real-time properties, such as Ethernet Powerlink [5], Profinet [6] or TTEthernet [7] enables Ethernet technology to replace conventional fieldbus technologies in the automotive field. In spite of offering temporal responses guarantees required by real-time systems, these solutions are essentially static or in other words, they are not able to

dynamically adapt its operation to the changing QoS metrics required by an application at a given time.

One technology in particular, the Audio/Video Bridging (AVB) , developed by the IEEE AVB Task Group [8], has shown great promise in dealing with these issues. The AVB technology has been originally conceived with the target market of home entertainment and professional audio installations. However its ability to make resource reservation through a signalling strategy mechanism and its prioritization mechanism specified in the IEEE 802.1Q standard could be used to guarantee performances of the highest traffic class and provide low latency streaming services in an Ethernet based in-vehicle network. Several other mechanisms were envisioned such as time synchronization, forwarding and queuing enhancements and stream reservations essential for obtaining Quality of Service (QoS) guarantees in a switched in-vehicle Ethernet network.

This work intends to present the resource reservation mechanism Stream Reservation Protocol (SRP) as part of the AVB family. The main objective is to conceive and develop a functional model that can implement that protocol in a simulation environment called Omnet++.

1.2 Document's Organization

This dissertation is organised as follows:

Chapter 2 studies the various technological domains of an in-vehicle system with its main fieldbus technologies and the Ethernet motivation as an reliable replacement for these technologies. Study of implicit and explicit resource reservation mechanisms is also made.

Chapter 3 gives an overview of the main AVB bridging standards of the characteristics of its signaling protocol named Stream Reservation Protocol (SRP), for resource reservation in automotive networks taking into account several QoS parameters.

Chapter 4 implements such protocol in a simulator environment OMNeT++, creating new solutions and models with functional blocks, running afterwards simulations with messages exchanging.

Chapter 5 depicts the methodology employed to validate the developed MSRP mode, as well as the recording of some results in a simple scenario.

Chapter 6 presents the main conclusions of this work and some hints for future work.

2

Automotive Networks with Ethernet technology

This chapter describes network technologies used in the various domains of an automotive system, the motivation for the usage of Ethernet and a brief description of resource reservation strategies.

2.1 Automotive Networking Technologies

Today's existing cars are much more sophisticated than the ones built 10 or 20 years ago. The constant replacement of mechanical and hydraulic equipments in an in-vehicle system with electronic ones, namely ECUs, has led to the increasing necessity to control them through communication networks connecting them. And even among the existing communication networks, it became necessary to divide them into networks of a specific in-vehicle subsystem with different requirements and different automotive functions. Some examples of these functions are presented as follow [9]:

- **Engine Control Module (ECM) or Powertrain Control Module (PCM)** - This module is a type of Electronic Control Unit (ECU) that has the role to monitor a vast array of sensors around the car engine to ensure conditions are within normal range of operation. It also controls a series of actuators of the internal combustion engine, such as fuel, air and spark to ensure optimal engine performance. This car component is normally sourced and customized by specialized OEMs according to the auto manufacturers specifications. One of the most recent features delegated to the ECM responsibility in the last decade is the electronic throttle control, which occurred when automotive companies switched it from mechanical ones. This control task is performed by the ECM when an electronic sensor near the accelerator pedal sends an

electrical signal to the ECM, which evaluates the throttle contact and then indicates the engine to adjust fuel dosage. Other functions of this module are the management of anti-skid brakes, cruise control, engine temperature regulation and theft protection and the monitoring of fuel mixture to ensure the car meets emissions standards, fuel injectors to provide fuel at the right moment for optimal power delivery, mass airflow information provided by specific sensors, Exhaust Gas Recirculation (Exhaust Gas Recirculation (EGR)) for emissions and ignition control, which in turn regulate spark plugs and much more. All this is accomplished by monitoring their respective sensor inputs;

- **Anti-lock Braking System (ABS)** - The ABS system is very important because it enables the driver to brake without having the vehicle wheels sliding on the road, making for a safer driving. By keeping the wheels from slipping the driver gains traction and is able to stop with much more safety and steer while slowing down. The main components of this system are the speed sensors, valves, pump and controller. Speed sensors are normally located in each wheel or in a differential position and has the responsibility of informing the ABS system if the wheel is about to lock or not. The valve prevents the pressure from rising above a certain limit if the driver presses the brake harder and is also able to reduce it. Whereas the valve releases pressure, the pump is able to put the pressure back if necessary. Finally the controller acts as a computer to monitor and control speed sensors and valves. The main objective of this system is to slow down the tire at thee same rate as the rest of the car;

- **Electric Power Steering (EPS)** - One important feature present in all vehicles is an EPS system, with the main objective of turning the car smoothly by turning each wheel at a different angle, making the inside wheel turn more then the outside one. This is another example of the migration of mechanic and hydraulic mechanisms to an electronic, sometimes called steer-by-wire. This causes the car to spend less fuel because it doesn't need to pump fluid all the time and the horsepower is not wasted. This steering system would have a lot of sensors to provide output to a control steering system and it should be able to save a lot of space in the engine compartment;

- **Electronic Stability Control (ESC) or Electronic Stability Program (ESP)** - ESC improves driving safety by controlling the car's individual brakes automatically. This system also activates the ABS system to resolve and anticipate safety issues. ESC also uses traction control information monitored by electronic stability control sensors, to try and predict the direction the car takes when the wheels slip from the intended trajectory;

There are many other automotive applications in which Ethernet networks can be used, such as the ignition system, the fuel injection system, cruise control or audio systems. Each of the applications mentioned above can be categorized in different in-vehicle system domains

with different automotive network technologies. A particular subsystem may need to control actuators, others receive feedback from sensor units, others stream audio or video and others may need only to give general information about the vehicle. Everything depends on what kind of requirements and specifications are needed and as such communication between each subsystem is very important to assure the good behaviour of the overall set. Of the existing in-vehicle network technologies, CAN [10], FlexRay [11], LIN [12] and MOST [13] are the most used until now.

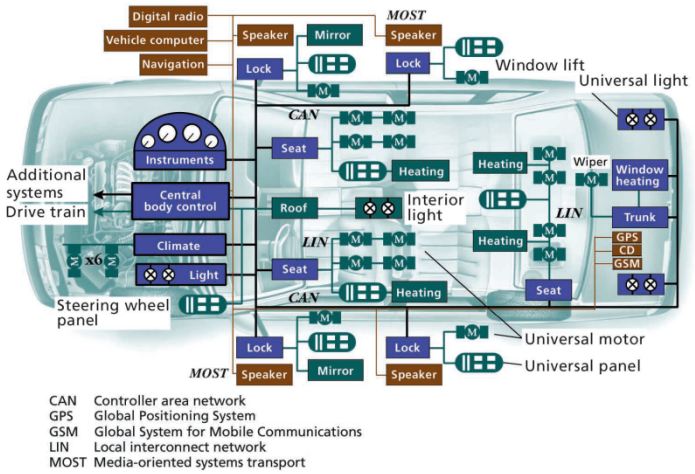


Figure 2.1: One sub-set of a traditional in-vehicle network architecture [14]

Figure 2.1 shows an example of a current in-vehicle network system, with many application specific to a different functions of the car. All the functions of a particular domain are therefore connected by each automotive network technology such as CAN segment, MOST and LIN.

It is also important to note and take into account several functional networking requirements, based on the demands of the more traditional applications for which most of those fieldbus protocols were conceived, when designing those particular networks.

The above mentioned automotive network technologies are now briefly explained in the following sections.

2.1.1 CAN

Controller Area Network (CAN) is a fieldbus technology initially developed for automotive networking and its applications. It first came as a necessity to reduce costs and wiring of an in-vehicle network. It was then migrated to other fields, mainly industrial applications, and served as a first step for the development of other protocols, such as CANopen, DeviceNET and Smart Distributed System(SDS). It was first introduced in 1986 by Robert Bosch GmbH, with CAN 2.0 specification being latter published [10] in 1991. In that time CAN was not the only networking technology competing for the automotive market, however it finally won the

race in the mid-nineties over the A-bus developed by the Volkswagen Group and the French Vehicle Area Network (VAN) developed by the European group [15]. Years later CAN 2.0 divided itself into two versions, the CAN 2.0A with an 11 bits identifier in its message data frame and the CAN 2.0B with 29 bits identifier.

CAN is a short message-based protocol which has its main goal the communication handling in the engine and other control equipment. Some of CAN features are its multi-master capability, which means that no node has a privileged role in communication, a broadcast serial bus for connecting ECUs, which means that each node reads the message sent and must accept it or discard it, depending on its ID, that represents the priority of the message. It is an event-driven technology, which means each node has the ability to react in a certain way when an event is triggered, each node also has the initiative to transmit when the medium is silent. There is no particular coordination between nodes with the possibility of collisions in the medium, it has a control error mechanism and is fault-tolerant.

Protocol Layout

This fieldbus system only uses up to 2 layers of the Open Systems Interconnection (OSI) model, the Physical layer and the Data Link layer.

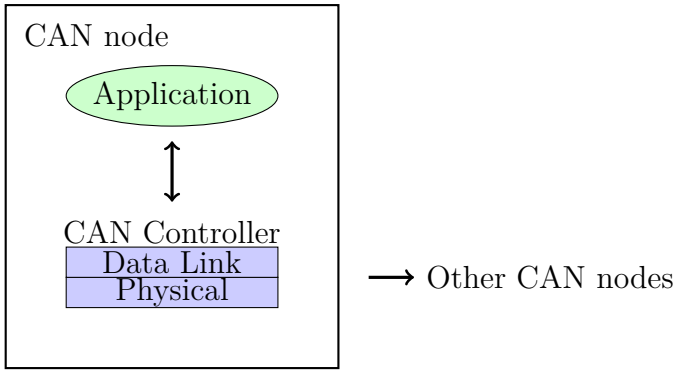


Figure 2.2: OSI Layers of a CAN node

Usually devices such as actuators or sensors are not directly connected to CAN network, but through a host processor and CAN controller. The physical layer makes the connection between the nodes in a network, and translates data drawn from the data link layer into electrical impulses to be transmitted across a copper wire, coax or optic fiber cable, or via wireless. On the receiving end, the physical layer translates those electronic signals back into a data format that is then passed up to the data link layer. Thus the physical layer deals with issues like bit encoding, bit timing and synchronization [16]. A CAN network operates in a quasi-stationary or in other words, all stations must see the same bit value in the line with the bus length smaller than the length that the length travelled by one bit at a certain speed transmission, forcing the network to have relatively small dimensions. Some typical values are 40m @ 1Mbps and 1000 @ 50kbps.

Bit Stuffing

There can be many ways to encode the stream of bits in a digital system. The two most usual ones are the Non-Return to Zero (NRZ) and the Manchester code. In CAN protocol the most obvious choice would be the NRZ because the Manchester code uses signal transitions, it requires to time-slots to encode 1 bit and as such it requires more bandwidth. But NRZ method has a serious disadvantage, which is the difficult to keep two oscillators used with CAN controller, exactly synchronized for a very period of time at the bit rates used by CAN. If a frame has a string of '1's or a string of '0's the signal will remain constant and there is no easy way to tell to where each bit starts or ends. The only way to assure this is to have the oscillators of the transmitter and the receiver exactly synchronized.

To overcome this issue, CAN uses a technique called bit stuffing which inserts a signal transition bit every time five identical bits appear in a row. This signal transition is called a stuff bit and enables the receivers to use it to synchronize their clocks.

Recessive and Dominant bits

In order to provide bus access arbitration to the stations, CAN provides the concept of recessive and dominant bits. The arbitration phase occurs when all bits of each CAN node's identifier are injected to the bus. When the bit to be compared is '0', it is called the dominant bit and the node continues to the next arbitration. If the bit is '1' the node loses the arbitration and let the other nodes sort them self out. The commonly medium access mechanism used by CAN is Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) , with an arbitration scheme based on the message identifier fields to decide which node will be granted the permission to continue transmission. This means that in CAN protocol the node with a lower ID has higher priority in the bus access.

Frame Compatibility

In this layer the data to be transmitted is encapsulated in an adequate data frame and the data to be received is encapsulated. Two different lengths can be used for frames identifiers:

- CAN 2.0A - 11-bit identifier;
- CAN 2.0B - 29-bit identifier.

It is possible use in the same bus CAN 2.0A and CAN 2.0B. CAN 2.0B controllers always accepts messages from CAN 2.0A but the opposite is not possible. Either the CAN 2.0A is active and destroys the message from CAN 2.0B or is passive and ignores it.

Automotive CAN Examples

CAN protocol provides several important features ideally used for automotive applications. It guarantees certain levels of pre determined transmission delay times by using messages of limited lengths and fixed bit rates in each system. It also provides flexibility in the system configuration, since nodes can be added to and removed from the CAN network without changing the software and hardware of the other nodes. To ensure the integrity of the transmitted data CAN offers several error detection mechanisms such as Cyclic Redundancy Check (CRC) , bit stuffing, bus monitoring and message frame check are applied. Corrupted messages are flagged by the node that detects them. They are aborted and retransmitted automatically. The recovery time from detecting an error until the transmission of the next message is at most 29 bits long and also enables high bit rates for small networks sizes. For this reasons CAN is still used for the transmission of control messages in almost all automotive domains. Also, the sensor data for driver assistance services is transmitted by the CAN bus. Several types of the CAN bus may differ in their transmission rate and their upper layer protocols are applied in the car as follows:

- **S-CAN** - safety-relevant CAN segment for automatic cruise control and anti-collision functions;
- **PT-CAN** - power train applications used in engine components that generate power and supply to the rest of the vehicle;
- **FA-CAN** - chassis and power train applications;
- **K-CAN** - CAN segment for driver assistance and infotainment applications;
- **Safety-CAN** - safety-critical driver assistance.

2.1.2 FlexRay

With the increased number of automotive applications and with the introduction of the 6th and 7th CAN bus to a car's network, eventually made this technology to reach its limits in terms of bandwidth. In fact this only resulted in more integration complexity and unpredictable communication delays [17].

With all this in mind FlexRay [11] protocol was conceived by FlexRay Consortium to govern on-board automotive communications and was designed to be an improvement over CAN with high data rates up to 10 Mbps, far superior than CAN. It is a deterministic protocol that supports both event and time triggered communication. It also supports redundancy and fault-tolerant mechanisms with two independent channels connecting each car's ECUs. The second channel could also be used to double the transmission rate when safety-critical applications are not required.

Keeping the requirements for safety-critical applications in mind, FlexRay uses a time triggered network access with a global time synchronization to guarantee message latency times. The Time-Division Multiple Access (TDMA) protocol was chosen as a medium access mechanism to cope with those needs. Each communication cycle of the protocol contains a static segment, a dynamic segment, a symbol window and a network idle time. Within the static segment there are static TDMA with constant time slot periods used to arbitrate transmissions while a dynamic mini-slotting-based scheme has variable length time slots and a position dependent arbitration is used for transmissions. The symbol window is a communication period in which a symbol can be transmitted on the network and the network idle time is a communication-free period that concludes each communication cycle [17].

This technology is currently used today by manufacturers such as Audi or BMW, with BMW X5 model being one of the first ones to use it [17].

FlexRay with its dual redundancy combined with the time triggered communication, could potentially meet the reliability requirements for many automotive applications such as:

- brake-by-wire;
- powertrain;
- chassis applications.

In the end this protocol still has certain disadvantages like lower operating voltage levels and asymmetry of the edges, which leads to problems in extending the network length. Also certain potential substitutes such as Ethernet are on the way to replace FlexRay for bandwidth intensive, non-safety critical applications.

2.1.3 LIN

Local Interconnect Network (LIN [12]) is an automotive network technology first introduced by LIN Consortium in 2000. In this group there were not only automotive manufacturers, such as Volkswagen, Audi,BMW, Daimler Chrysler and Volvo, but also communication specialist Volcano and semi-conductor manufacturer Motorola [18].

LIN is a serial bus similar to CAN but developed to achieve low cost and low complexity capabilities, making it a technology much more versatile than CAN. However LIN bus is not viewed by the OEMs as a competitor of existing CAN networks but rather as a complementary network technology to work together with CAN networks. LIN can be used in many automotive applications where the costs of CAN are to high and where fast data rates are not critical for its function.

Unlike CAN, LIN is a master/slave bus with 1 master and a maximum of 16 slaves. It has a single wire connection fed by the main battery of the car which makes it to operate asynchronously. The media access is realized through a polling mechanism, in which the

master has the responsibility to perform all synchronization of the slaves. The master is able to communicate with all slaves and only when it gives an order to communicate, slaves may talk. Each message passed between those units are made through LIN frames with its headers being generated by the master only. The slave can only respond if the master addresses the slave's header ID. The header is divided into three parts:

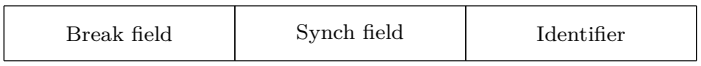


Figure 2.3: LIN Frame Header

The synch break has the function to warn the participant about a new frame sent to the bus followed by a synchronization byte to synchronize the slaves. Finally, the last part of the header is an 8 bits identifier which bit 0 till 5 are the ID bits and bit 6 and 7 are the parity bits. These 6 ID bits result in a range of 0 up to 63 message ID's. The protocol also supports data rates up to 20 kbps with a maximum packet payload size of 8 bytes, however in practice usual speeds of serial communication such as 2400, 9600 and 19200 bits/s are applied [18].

The low cost characteristic of LIN is achieved:

- low cost silicon implementation based;
- self synchronization without a quartz or ceramic resonator in the slave nodes;
- low cost single-wire implementation.

Many car manufacturers use CAN to serve as a network protocol for motor management and body electronics. LIN on the other hand, with its mentioned characteristics, can be used to replace analogue driving electronics, e.g. the inside interior lighting of a car and the controlling pressure sensors in control systems.

2.1.4 MOST

Information and entertainment applications in a car are constantly evolving with a variety of sophisticated entertainment and information systems that need to communicate and interact with each other and with the human user. The infotainment domain of an automotive system, such as streaming audio, video and packet based data (e.g. Global Positioning System (GPS)), require fast, short-delay networking with continuous bandwidth availability which were not possible with CAN.

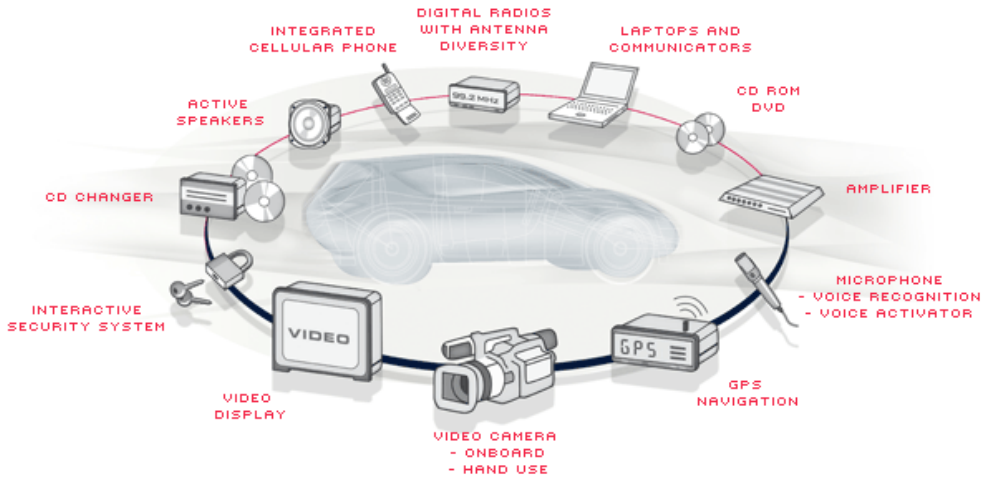


Figure 2.4: In-vehicle infotainment applications [19]

For all these a new in-vehicle network technology called Media Oriented Systems Transport (MOST [13]) was created by MOST Cooperation [19] in 1998, a conglomeration of several OEMs such as BMW, Daimler, Audi, Harman-Becker, Oasis. It was specially designed to interconnect multimedia components for audio streaming in a car. It uses ring topology and a synchronous data communication for audio, video, voice and data signals over an plastic optic fiber, provides a network system based on a multi-master concept at bit rates much higher than other automotive network technologies. There are three kinds of MOST networks: MOST25 with a bit rate of 25 Mbps, MOST50 with a bit rate of 50 Mbps and MOST150 with a bit rate of 150 Mbps.

MOST specification defines all 7 ISO/OSI layers for data communication. The Application Programming Interface (API) of MOST devices is object oriented so that applications can concentrate on their functions.

Three channels are applied in the MOST specification[19]:

- Control channel used to send control messages and set up connections;
- Synchronous channel used to analogue and digital audio and video;
- Asynchronous channel used for internet traffic or information from a navigation system.

The operating mode of MOST in a glance is as follows:

1. The time master node periodically generates frames;
2. One after the other, slaves on the logical ring receive the signal, synchronizes itself with the preamble, parses the frame, processes the desired information, adds information to the free slots in the frame and transmits the frame to its successor;
3. When the frame eventually returns to the time master it synchronizes itself and subsequently generates the next frame in accordance to the frame rate.

In spite of the great evolution and success of this technology as an infotainment in-vehicle network, one serious competitor called Ethernet will tend to take its place because of its greater generality as a standard. Also MOST works in a ring topology, making it difficult to perform tasks outside of the infotainment domain, such as updates of a navigation system for example, in spite of the sufficient bandwidth provided by newer versions.

2.2 Ethernet

Ethernet was first developed in 1972 by *Robert Metcalfe* and his colleagues at the Palo Alto Research Center. It was intended be a Local Area Network (LAN) capable of interconnecting stations, servers and peripheral devices [2]. This technology was standardized into the IEEE 802.3 [20], with coverage of the physical and data link layers of the OSI Model. This technology uses as media access control protocol the carrier sense multiple access with collision detection (CSMA/CD), which means that in order to access the network, each device verifies whether the network is free or not.

In the mid nineties as Ethernet quickly adapted itself to upcoming requirements, succeeded in becoming the dominant LAN technology for companies and since then has been in constant evolution with bit rates increasing from 10 Mbps to 100 Gbps as described in figure 2.5.

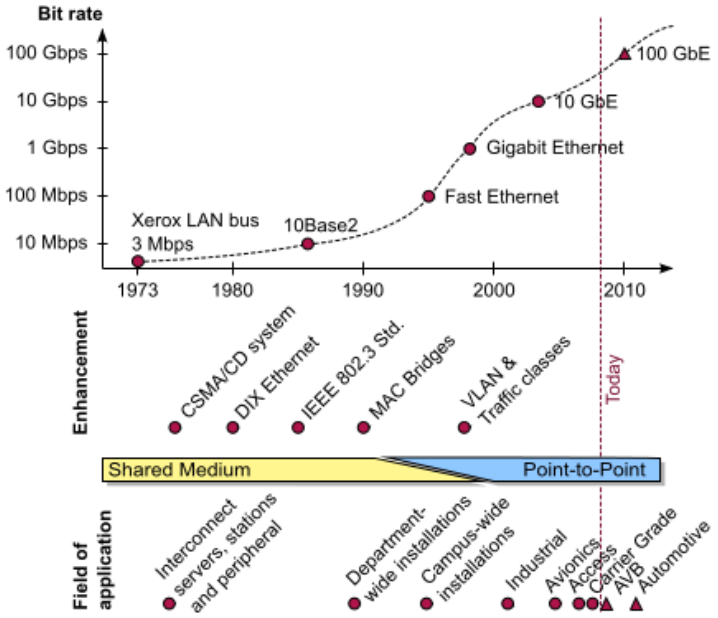


Figure 2.5: Ethernet's evolution and its fields of applications [2]

The topology of the networks also evolved from bus topology or ring topology used in Token Ring, to star topology with one central node and other that could be added and removed. Those nodes are connected with hubs or switches/bridges. Hubs are very simple network equipments that work at the physical layer and act as repeaters of all incoming

frames and output them in every port without any kind of processing. Switches are devices that work at the link layer and connect different LANs in the same broadcast domain. They perform important operations such as storing the frames in queues and forwarding them with the help of a scheduler. They also have an important element indispensable to its proper functioning called Forwarding Database (FDB) tables. They are used to store MAC addresses that have been learned and the ports from which they came from. With this, any Ethernet frame that arrives at the switch will be analysed by its destination MAC address and compared with the ones already in the FDB tables. If the address is present in the table the frame will be forward through the switch port correspondent to the destination Media Access Control (MAC) address. If the address is not found, the frame will be flooded in all ports into the broadcast domain, minus the one from which it came. It is in equipments like bridges and switches that most of the solutions envisioned to cope with real-time and automotive requirements are being made.

This evolution occurred thanks to new technologies implemented in its protocol stack layers. In the physical layer Ethernet evolved towards higher speeds of 100Mbps (FastEthernet) using twisted pair cabling. Besides copper cabling the IEEE [20] also specified single-mode and multi-mode fiber usage, which extended the maximum range of the cables from 100 m to several kilometers. The next step was 1Gbps (Gigabit Ethernet) with IEEE [20] defining long range and short range types of single and multi-mode fibers with a dedicated wave length. All Ethernet modes up to 1 Gbps still allows half-duplex mode, but the 10 Gbps only operates at full-duplex.

In the data link layer, in addition to the various frame formats defined in IEEE 802.3 there is an extension for virtual LANs defined in IEEE 802.1Q [21]. Next figure depicts the common fields of the Ethernet II frame structure, the most common in today's LANs:

Figure 2.6: Ethernet II frame

The first 8 bytes are the *preamble* and exist as an alternative bit sequence for synchronization between the sender and the receiver. The *Start Of Frame (SOF)* byte indicates the start of frame, the *DA* and *SA* with 6 bytes each indicates the destination address and source addresses, which are administered by the IEEE or locally unique addresses assigned by the local network administrator. Broadcast and multicast addresses are also possible. The main purpose of the next two bytes is to differentiate Ethernet from 802.3 version (L) and Ethernet II version (T). The former indicates that the payload is encapsulated in a MAC layer independent Logical Link Control (LLC) frame, which has the advantage of guaranteeing the interworking of different underlying LAN technologies. The latter defines payload type present in the frame as shown in figure 2.6. Finally the last 4 bytes are the *Frame Check Sequence (FCS)* used for error detection.

The next figure depicts an Ethernet frame format with *Virtual Local Area Network*

(VLAN) :

Pre (7)	SOF (1)	DA (6)	SA (6)	T (2)	TPID (2)	TCI (2)	Data (42-1500)	FCS (4)

Figure 2.7: VLAN tagged Ethernet frame

This extension enables the virtual separation of multiple LANs on the same physical infrastructure, for traffic engineering and security reasons. It consists of additional 4 bytes, the *Tag Protocol Identifier (TPID)* corresponding to type/length of the Ethernet II and the *Tag Control Information (TCI)* which contains the VLAN identifier (12 bits), priority information (3 bits) and on standard bit indicator. It is this priority information that indicates the user priority class of the eight class defined in the IEEE 802.1Q.

Ethernet motivation for Automotive Applications

The main motivation for the use of Ethernet in automotive networks lies with the necessity of reducing costs by most OEMs. History of Ethernet has proven that it is a sustainable technology, which has not yet reached its limits and can fulfill upcoming requirements. Due to the fact that current major solutions used in in-vehicle networks like CAN or MOST, are very specific and closed technologies, the costs for improvement of the offered bandwidth are very high. For that reason the use of Ethernet as a low cost network technology, transport capability and simplicity is becoming much more appealing in an in-vehicle network.

Ethernet as a dominant LAN technology and present in most of the computational devices, is leading OEMs to seek and adapt Ethernet main features to cope with real-time communication requirements. The main objective is to reduce development times, use Ethernet as a standardized technology and mass-market components, instead of continuing to invest in automotive-specific technologies. For those reasons low-cost switches could be used to interconnect different sub-networks of automotive domains.

2.2.1 Real-Time Protocols over Ethernet

Nowadays the use of different real-time Ethernet solutions is becoming much more common, specially in the industrial and automotive fields where networking requirements are very constrained in terms of latency and bandwidth.

Due to the different scope of scenarios in which real-time Ethernet can be used, different real-time Ethernet solutions can have completely different purposes and therefore supporting completely different QoS classes. The authors of [2] tell that even though the majority of real-time Ethernet solutions covers completely different and incompatible technologies, they can be classified within a hierarchy of some degree of compatibility with Commercial Off-The-Shelf (COTS) Ethernet technology:

1. *Interoperable homogeneous* - These solutions are build on top of of the lower layers of the

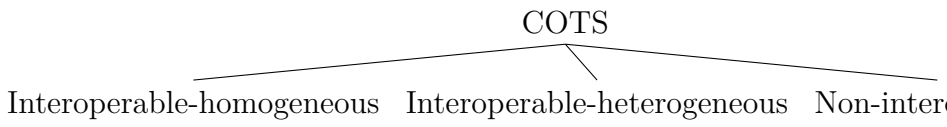


Figure 2.8: COTS Ethernet classification

IEEE 802.3 standards, which makes them compatible with any COTS Ethernet devices. Switches here play an important role, providing a great improvement in performance with the isolation of independent collision domains, reducing the non-determinism of the original shared network and the better prediction of QoS levels, resulting in greater efficiency. The use of switches by themselves and the increased network segmentation provided by virtual LAN's in the network does not make it completely deterministic mainly because of buffer delays and *jitter* due to the different internal architectural solutions of the switch. The most obvious way of resolving this issue is the implementation of a traffic scheduler on top of Ethernet in order to regulate and smooth additional traffic.

2. *Interoperable heterogeneous* - These type of solutions are also able to communicate with IEEE 802.3 devices. However the real-time capability of these solutions are based on exclusive and modified switches, usually with different architectures.

3. *Non-interoperable* - The solutions that fall into this category are the least generic possible, because they hardly depend on very specific timing requirements present only on critical applications such as motion control or hard real-time industrial processes.

Mechanisms to support Ethernet in fault-tolerant networks

In order for Ethernet to support high availability and redundancy, essential to safety and time dependable applications, the need to develop in-vehicle fault-tolerant networks is paramount. This fault-tolerant [22] capability is particularly important in Automotive Ethernet, because of its broadcast characteristics and connectionless oriented kind of scheme. It also needs to be taken into account the possible formation of loops within redundant automotive Ethernet network when a communication problem occurs in the chosen path. Hence the only way traffic is deviated from its original path to a redundant one is through a reconfiguration done by a redundancy control protocol. In the middle of that reconfiguration, some frames could be lost and may need to be resent. The time lost in reconfiguration increases the overall latency that could influence the performance of real-time applications. With all this in mind, many redundancy control protocols were developed to cope with different application requirements.

This redundancy feature can be achieved with two different kind of mechanisms, one with communication interruption and a certain amount of time to recovery and other without any interruption at all.

Mechanisms with communication interruption

The most traditional approach and the first standardized mechanism was the *Spanning Tree Protocol (STP)*, described by IEEE 802.1D [22]. The main point of this protocol is to allow an arbitrary meshed topology of the physical network. Bridges using the spanning tree algorithm are able of pruning the topology into a loop-free tree, however its convergence time is very high, for it to be used in certain real-time applications.

One evolution of this protocol is the *Rapid Spanning Tree Protocol (RSTP)* described by IEEE 802.1D - 2010 revision. Due to its widespread availability and flexibility, this protocol can accommodate industrial level requirements that can tolerate at best a few milliseconds of reconfiguration time. Another important protocol is the *Media Redundancy Protocol (MRP)*, described by [22] and also used in industrial applications with little recovery time-out operation.

Mechanisms with no communication interruption

However, all the mechanisms mentioned above are not able to cope with new Ethernet-based with real-time requirements, because technology evolved in such a way that that certain levels of communication interruption, or any interruption at all could be unacceptable. Some new mechanisms are therefore able to have some kind of interoperability between redundancy control technologies and real-time Ethernet protocols. One such technology is the *Media Redundancy Protocol Duplication (MRPD)* used in Profinet IRT, for example. Other mechanisms that don't have communication interruption, but do not depend directly with a specified real-time protocol are for example the *Parallel Redundancy Protocol (PRP)* and the *High Availability Seamless Redundancy (HSR)* , both described in the international standard IEC 62439-3 [22].

Some other protocols have recently emerged, which are able of taking advantages of not only the layer 2, but also of some characteristics of the routing mechanism of layer 3. Those protocols are the TRansparent Interconnection of Lots of Links (TRILL) and the Shortest Path Bridging (SPB) protocols. Both TRILL and SPB are competing proposals intended to replace the spanning tree protocol. They both use link state routing protocols to calculate optimal paths between nodes on a layer 2 level.

TRILL is a protocol developed by the Internet Engineering Task Force (IETF) to create a large cloud of links, so that IP nodes see it as a single link. Nodes can then move inside the cloud using all layer 3 routing techniques. As such, devices that implements TRILL inside that cloud are called Routing Bridges, or RBridges. Each RBridge calculates shortest paths from itself to each other RBridge by sending frames with a TRILL header containing a ingress RBridge address, egress RBridge address, an hop count and a multi-destination flag bit. The more existing bridges are replaced by these RBridges, the more stability and efficiency in

using the available bandwidth the network inside the cloud becomes, because spanning tree gets smaller [23].

SPB protocol is an IEEE standard 802.1aq [24], providing shortest path routing using link state protocol Intermediate System to Intermediate System (IS-IS) at layer 2 and fast convergence times in the range of 50 to 100ms. SPB aware bridges in the network, learn the topology using MAC addresses, however the core bridged are not aware of the MAC addresses of the edge of the network. This allows scalability without loops.

2.2.2 In-Vehicle QoS Ethernet Requirements

In this section the most important issues related to QoS requirements in automotive communication systems will be addressed. In-vehicle traffic can be categorized in four different classes, based on their QoS requirements [2]. The next table depicts those classes with its most important QoS requirements:

Table 2.1: QoS characteristics

Metrics	**Control Data**	**Interactive Stream**	**Multimedia**	**Best effort**
Expected payload	Few bytes	Mbytes	Kbytes - Mbytes	none
Expected data rate	Low	High	High	none
Max. expected latency	+- 10ms	+- 45ms	+- 100ms	none
Priority	Highest	High	Medium	Low

The first class consists of hard real-time control data which contains sensor information about in-vehicle domains, acting then upon it. This information is only a few bytes long and the data rate is usually low, however, due to its importance for the system integrity it has the one of the higher traffic priorities. One application using such QoS requirements is the ESC system. Typically control domains with safety critical applications are isolated from the rest of the network and uses an additional scheduler to control the incoming traffic. The next two classes called interactive data and multimedia data are usually inserted in the infotainment domain. Interactive applications need strong and well defined delay and jitter requirements because they do not allow buffering of data in the receiving side making the end-to-end delays less tolerable than those of multimedia nature, which often allow buffering and an higher tolerance for latency. One such example could be the data stream coming from a rear view camera. Multimedia category is comprised of video, audio and entertainment applications. Finally there is the best effort data class which only transmits data whenever the network is able to, making this the least priority one. This type can be used to transmit traffic from plugged-in customers personal electronic devices for example.

Another important factor that needs to be addressed in in-vehicle communication is the Electromagnetic Interference (EMI) [2], which plays an important role. For that reason an

automotive Ethernet technology could take advantage of a solution named Polymer Optical Fiber (POF) already installed in actual automotive networks, such as MOST.

The advantages of using Ethernet as an in-vehicle network are its latent characteristics and the ability to adapt itself to all the classes mentioned above, fitting both control and streaming applications. Some of its main features are its flexible frame length, which can be used to support both the transmission of control data of only a few bytes, and multimedia streams with large frame lengths. Although some characteristics of legacy Ethernet have great potential as a reliable in-vehicle technology, they are not sufficient only by themselves. To make sure Ethernet complies with all QoS requirements of table 2.1, switches must be dimensioned. One important aspect is the need to have a low packet loss rate, which is done by the buffer dimension of the switches. However it is important to not forget that the queueing mechanism, as well as algorithm operation of store and forwarding, are important factors for the majority of switching delays. Ethernet also satisfies in most cases the high bandwidth demand, specially in the infotainment domain where quality audio and HD-Video data will play an essential role. Another final aspect that needs to be taken into account is the well defined separation of safety critical domains from other applications. For instance, an unknown source of traffic, coming from plugged-in devices, should not be able to interfere with automotive safety critical systems like ESC or ABS, because of the unknown delay and jitter that could be generated. There are two different ways of resolving this issue. The first one is to physically separate safety critical domains from infotainment ones through a gateway, or further divide them with the use of VLAN tagging, rendering the use of complex gateways dispensable. Another advantage of the Ethernet usage is the ease in which many solutions can integrate with its lower-layer levels and higher-layer protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) , making cars possible to interconnect to the outside world, namely through the internet.

Finally the compatibility issue needs to be taken into account regarding costumers needs to integrate its own current and future electronic devices (e.g. PDA), in the in-vehicle network. As such, next generation automotive communication systems have to be robust with respect to possible future evolutions because the extended life cycle of a car series compared to a consumer electronic lifespan. In order to achieve this, processors and chips of used in those equipments have to be backward compatible to the available car network technology, which could be up to 30 years [2].

Topologies

Topologies in a switched Ethernet network are very important because they can influence traffic QoS requirements in a bad way or a good way depending on the way, such networks are set-up. The authors of [25] propose three different topologies for in-vehicle networks:

- *Star-based* - This topology, as depicted in Figure 2.9, reduces the complexity of cables,

hence reducing maintenance and installation costs, but it has some complexity in the switches, mainly because of the number of interfaces that can emerge.

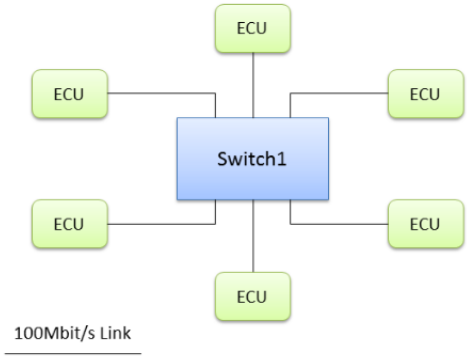


Figure 2.9: Automotive Ethernet star based topology.

- *Daisy-chain based* - This is the simplest topology of the three, as depicted in Figure 2.10. It has 3-port switches, which could be fixed on in-car equipments. The costs of this one are higher because it uses a lot more wiring, but in the other hand its performance is much better then the star-based topology.

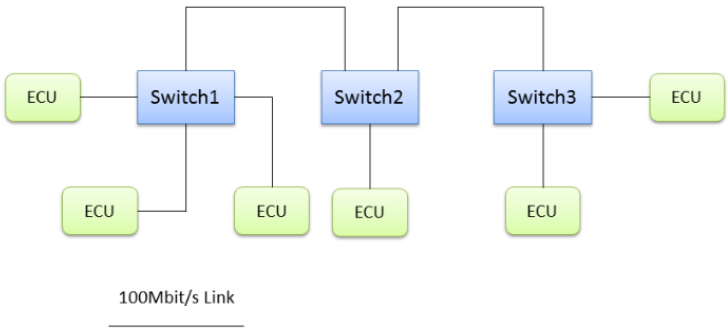


Figure 2.10: Automotive Ethernet daisy-chain based topology.

- *Tree* - The tree topology, as depicted in Figure 2.11, is a combination of the ones mentioned above, which tries to combine the best of two worlds by keeping good levels of performance without sacrificing installation and maintenance costs completely.

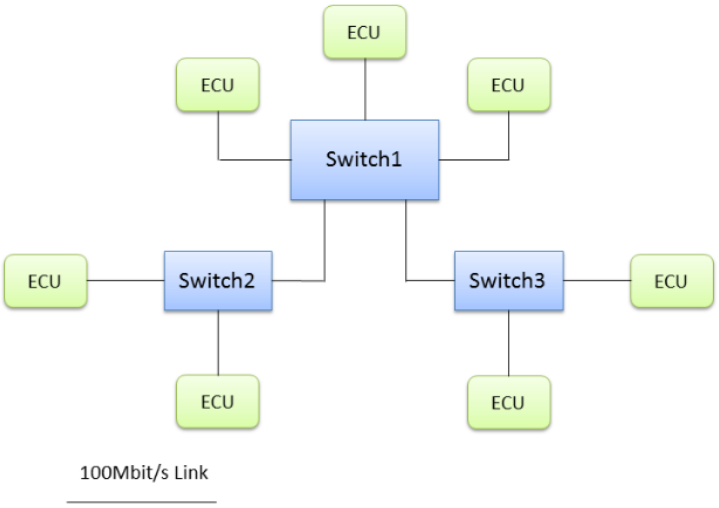


Figure 2.11: Automotive Ethernet tree based topology.

All these topologies can be used in a switched Ethernet network without any modifications the standard IEEE 802.3 [20].

2.3 Resource Reservation Strategies

In order to have resources available whenever a service needs them, several strategies could be employed. The main focus of these strategies is to enable the network to set aside certain resources, so that a specified quantity of QoS can be provided to a particular flow. Such resources can include the share of channel bandwidth or a certain number of buffers present in a switch. Another goal a Resource Reservation mechanisms must be able to resolve is the traffic congestion present in a network, which normally occurs when a number of packets transmitted through the network approaches its capacity. In order to keep the number of packets below a certain level, which if passed could be the cause of a drastic decrease in performance, some kind of mechanism for congestion control must exist. Many techniques exists to deal with this problem and will be addressed in this subsection. However, before that can happen a first look is made into some important issues concerning almost all types of network architectures able to support some kind of QoS.

2.3.1 Traffic Management

The first issue to be addressed is the different ways in which traffic can be managed in a network. Normally exchanged traffic among network entities is done using protocol data units (PDUs), in order to identify it as individualized units of information in the same OSI model layer. There are several different ways of classifying how this traffic can be processed. Some of them are briefly mentioned bellow and are valid both in a bridged network or in a wider context of networks:

- Fairness - provide equal treatment to the various flows, with the adverse affect increased delays and packet losses during congestion;

- Quality of service (QoS) - may want different treatment for different connections, with the setting of different priorities, depending on the type of traffic it pertains. When congestion occurs, each flow is treated based on their QoS requirements.

- Reservations - this method provides a traffic contract between user and network, occurring during connection setup with the possibility of usage of the admission control concept, where certain QoS requirements are first agreed, before the path along the source and the destination can be established. It is the only way of avoid traffic congestion and to provide assured services. The network agrees to a defined QoS and the user requests its necessities. The network either is able to support that packet transmission, if the traffic exchanged is within contract parameters, or it doesn't have the necessary resources to support the connection, in which case the reservation is denied until a new opportunity arises. Because in the specific context of automotive Ethernet networks there are several restricted requirement demands, such as real time control signals (drive-by-wire) or streaming traffic, the most obvious choice for managing automotive traffic is through resource reservations. One of the most known resource reservation mechanism, designed specifically to run over IP is the Resource Reservation Protocol (RSVP) [26].

Even recurring to the mentioned different types of classification mechanisms, sometimes it is necessary to have a more direct control over the traffic when this is being exchanged between entities of a network. One of the most important parameters to be controlled is the traffic rate. The main features that exists to have more direct control and influence over traffic in a network, and almost always used in combination with a resource reservation mechanism are next described:

- tagging and policing - discriminate between frames that conform to contract and those that don't and monitor a connection to ensure traffic obeys the contracted specification in order to protect network resources from overload by one connection;

- traffic shaping - limits the rate of data sent by network entities. The reason to have this method is mainly to match the incoming flow with the outgoing flow. This discrepancy can happen for several reasons, with the main one being the superior time rate at which the network is able to process a PDU compared to the rate of the incoming ones. In most cases the method used to restrict incoming data is called buffering. In normal situations when the buffer capacity reaches a specific threshold, the packet is discarded. In other situations a policing strategy could enforced to restrict traffic rate. This can be achieved by using a token bucket, in which a packet is only dispatched when a token

is given to it. When the token is depleted of tokens, then packets are discarded until the bucket is filled and a new token is available, as depicted in Figure 2.12.

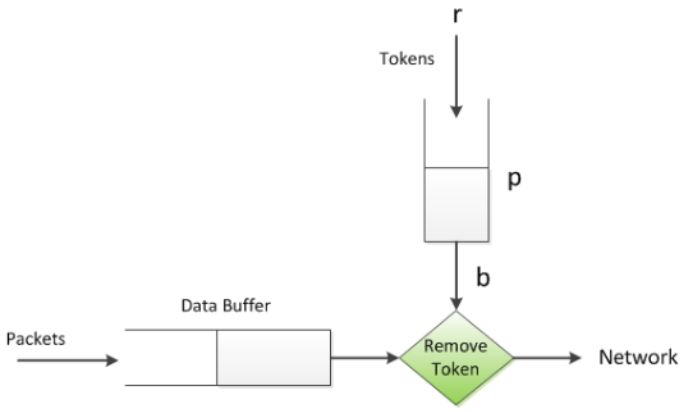


Figure 2.12: Token bucket model.

But if want to preserve the information the shaping mechanism must be able to limit or temporarily halt the incoming traffic from the source. Normally this is not necessary if the specific protocol already uses admission control;

- Queues - will discard tagged cells in preference to untagged frames based on queue-specific thresholds;
- Frame scheduling - gives preferential treatment to untagged frames, where a network switch must decide the order in which each frame is dispatched to the next node. This decision is made based on a scheduling algorithm which determines the QoS the network can provide. There many types of scheduling algorithms, with some of them referred here [27].

Admission Control

Admission control mechanisms are essential, because network's resources are finite. With its limited amount of resources, a network must be able to decide which requests to attend or refuse. The way these decision are made is through an admission control algorithm that determines which reservation requests to grant and which to deny, in order to maintain a certain level of network load. The source uses specific traffic characteristics known as flow specifications, to establish a new flow connection and the network understands it, by selecting a QoS it can support to the respective flow. This connection is only accepted if the network can meet the demand traffic contract specified in the 'flowspecs'. The 'flowspec' is therefore very important to a network architecture, because it specifies an interface applications interact with, hiding from them most of the details and the complexity. Some values can be used as a reference by these specifications, such as the Committed Information Rate (CIR) and the maximum physical data rate possible. The CIR is a rate expressed in bits per second,

representing the value rate a network agrees to support to a particular connection. Data in excess of this CIR is liable to discard in the event of congestion and is not guaranteed, transmitted only if possible. The maximum rate represents a barrier in which all data is discarded.

One example of an admission control algorithm designed to support real-time services is described in the following literature [28].

2.3.2 Congestion Control Mechanisms

With increasing traffic it is possible that network congestion could occur. In order to avoid that, it is essential to use some kind of congestion control strategies. Through the rest of this section, some of this strategies will be discussed, namely the way how it is possible for end nodes to regulate their data flows in order to make a more efficient use of network resources, without overloading the entire system through congestion and resulting throughput collapse.

As technology advances more and more, the problem of congestion will only tend to increase in complexity [29]. Some examples that can cause this phenomenon are the increase of buffers size and the use of increasingly faster connections. The first problem causes the increase of traffic delay inside every network entity that uses buffers, namely switches. Because of that, the period of delay could be a value greater then the source timer value. If that happens, the source would eventually retransmit copies of that packet before the original one reached its destination, thus choking the network physical resources and processing capacity with duplicates during transmission and ultimately contribute to further congestion. The second problem causes the possibility of high levels of congestion in networks with heterogeneous link speeds, due to the fact that it would exist a mismatch of traffic rate between faster networks and slower networks at points interconnection. Because of this disparity, congestion in those points would greatly increase, with the possibility of discarding data if the data stream is constant.

So it is important for the majority of network systems to support and perform congestion control to adapt the sending rate of each data stream source, so that the overall performance of network improves. Allowing a high utilization of the available bandwidth in the network is one example of such improvement.

Some of the strategies most commonly used are depicted next.

Congestion Signalling

This is one strategy in which the network alerts end systems of increasing congestion through special messages. End systems will then take steps to reduce offered load and thus avoid traffic congestion. Some of the strategies used are the implicit signalling and the explicit signalling and are described next.

Implicit signalling

In the implicit congestion signalling strategy the source of the traffic detects congestion at a node if the propagation delays of packets are known. If the delay is longer then the fixed propagation delay and if the source is able of detecting it, then network congestion is detected as an implicit evidence. This may ultimately lead to packet discard [29] if the source does not reduce the traffic flow in intermediate systems that do not need to take any action. This is very useful on connectionless networks, namely IP based networks, for which there is no link-based flow control but there is a logical connection with the TCP as a flow control mechanism that helps implicit congestion signalling.

In short the implicit signalling is known for having the following characteristics:

- Increased transmission delay with greater congestion;
- It is based on acknowledgments;
- The delays are considered end-to-end;
- Responsibility of detecting traffic congestion is pushed to end systems of the network;
- Effective on connectionless networks but could also be used in connection-oriented ones.

One example of implicit congestion signalling usage is the frame relay LAPF (Link Access Protocol - Channel D) control protocol used in an Integrated Services Digital Network (ISDN) for transmission of voice, data, video and graphics, at very high speeds, over standard communication lines.

Explicit signalling

In the explicit congestion signalling mechanism it is not the traffic source that detects congestion, but rather the opposite. In this case network alerts end systems of increasing congestion and then they take steps to reduce offered load. This mechanism can work in one of two different directions. The backward direction notifies the source that congestion avoidance procedures should be initiated in the opposite direction of the requested packet flow. It can also work in the forward direction, in which the destination is notified that the congestion avoidance procedures are to be initiated in the same direction as the requested packet flow. There can also be a method called Explicit Congestion Notification in TCP/IP with the purpose of reacting to congestion in a controlled and fair manner, mainly over connection oriented networks.

Several categories of explicit signalling exist as depicted next:

- binary - This mechanism has the purpose of handling congestion situations before packets actually get dropped. The binary proposal uses a bit set in the packet indicating congestion. With this approach intermediate systems of the network with queueing capability, can mark the incoming packets as congested instead of just discarding them. Then the value of the congestion bit is copied into the acknowledgement packet when it reached the destination and is sent back to the source. The source then changes its transmission window in accordance with the value of the congestion bit causing a reduction of traffic;
- credit based - indicates how many packets the source may send through the usage of explicit credits. Common for end-to-end flow control but also used for congestion control.
- rate based - supply explicit data rate limit applied to a connection with nodes along the path possibly requesting packet rate reduction.

3

AVB Resource Reservation Mechanism

This chapter presents a study of the Audio/Video Bridging Ethernet concept. An overview of several 802.1 AVB [8] standards will be made, namely the IEEE 802.1ak Multiple Registration Protocol, with its protocol operation viewed with some degree of detail. Lastly, the protocols built on top of Multiple Registration Protocol (MRP) [30] will be presented.

3.1 Introduction to IEEE 802.1 Audio/Video Bridging Standards

With the increasing market need for multimedia and streaming content in automotive networks and the tendency to choose Ethernet as its primary technology, IEEE 802.1 AVB standards, currently being developed by the IEEE AVB task group [8], has been conceived originally for home entertainment and professional audio installations. Thanks to its potential of adaptability to support not only audio and video data, but also to provide an inexpensive way of improving real-time behaviour of Ethernet networks, AVB technology was chosen as a viable solution to be integrated with them. As such, the main objective is to get robust QoS enhancements concerning mainly latency and bandwidth. One of the most important AVB metrics that should be analysed is the end-to-end delay in several conditions. IEEE 802.1Q [21] standard tells us that at least 2ms of end-to-end delay is guaranteed with a maximum of seven hops in a switched Ethernet network with a connection rate of 100Mbit/s and an AVB stream reserved as a SR-Class A. In order to have a mechanism that complies with these constraints several AVB protocols are being developed to extend the Media Access Control (MAC) layer and enhance legacy Ethernet technologies. Figure 3.1 depicts an overview of such AVB protocols.

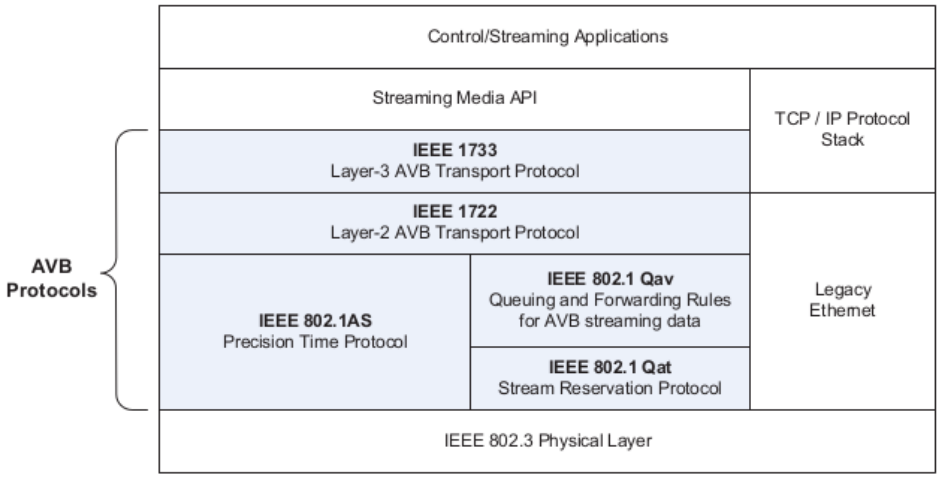


Figure 3.1: Stack of AVB protocols [31]

One of the most important AVB standards in the context of this work is the IEEE 802.1Qat amendment [32], which provides the signalling mechanisms for resource reservations on a bridged network called Stream Reservation Protocol (SRP). All the reservations are done between nodes called end-stations and all the bridges along the entire path, with the protocol also specifying the procedures and objects to be used by higher-layer applications. All reservations are done by means of message exchanges between an end-station called talker and another one called listener. The means by which this is done will be explained with more detail throughout this chapter.

In addition to the resource reservation, a set of rules for frame-forwarding and queueing in bridges is essential in order to comply with high data rates and low latencies of streaming services. This rules also uses methods to separate time-critical and non time-critical traffic into different traffic classes. All this is specified by the IEEE 802.1Qav amendment [33].

Another important feature provided by AVB standards is the time synchronization ability available in the IEEE 802.1AS, essential for applications involving multiple streams delivered to multiple listeners and to control other distributed time sensitive applications [31]. This feature is mainly used in hard real-time systems and is described by IEEE 802.1AS amendment [34].

Finally the IEEE 1722 [35] layer specifies a transport protocol that allows the encapsulation of audio and video frames. This protocol when used in combination with the IEEE 802.1AS ensures synchronization between different listeners.

3.2 IEEE 802.1Qat - Stream Reservation Protocol

The Stream Reservation Protocol (SRP) specified in IEEE 802.1Qat enables the actual reservation of network resources between end stations. This standard is used to register and de-register a streaming application with two different stream reservation classes called

SR-Class A and SR-Class B. This means that streams exchanged between two different end-stations must be done in the same SRP domain, i.e. with the same class priority. This standard also defines the Multiple Stream Reservation Protocol (MSRP) [32], which is the multiple stream version of the SRP, with the possibility of reserving resources for multiple stream flows for one or more end-stations. Each end stations can either be a talker that declares attributes of a specific stream it wants to send, acting as a communication source transmitting stream frames to a destination module called listener, which could first declare attributes that request the reception of those stream frames. Adding to end stations' behaviour description, we also have all the bridges behaviour in the network. Bridges along the path from Talker to Listener process, possibly adjust, and then forward these MSRP attribute declarations to associate a Talker and a Listener via their *StreamID*. MSRP also defines an MSRP application that provides the Stream resource registration service defined in the standard. It is important to note that MSRP makes use and works on top of an Multiple Registration Protocol level in order to have attribute propagation in bridges and attribute declaration, both in bridges and end-stations. All this is described by the IEEE 802.1ak amendment in the next subsection.

Finally Qat allows a maximum reservation of 75% of the total available bandwidth [31] in the channel, leaving the rest 25% to be used by legacy non-AVB Ethernet frames.

3.2.1 IEEE 802.1ak - Multiple Registration Protocol

MRP known as Multiple Registration Protocol is a simple, fully distributed protocol with a many-to-many communication paradigm that supports efficient, reliable and rapid declaration and registration of attributes. Some of the MRP features used by MSRP are the MRP Attribute Declaration (MAD) function, which provides the common state machine descriptions defined for use in MRP-based applications. This process then originates the propagation of said attributes by the MRP Attribute Propagation (MAP) function in all the remaining bridge ports.

Figure 3.2 depicts the example of a declaration, registration and propagation of an attribute:

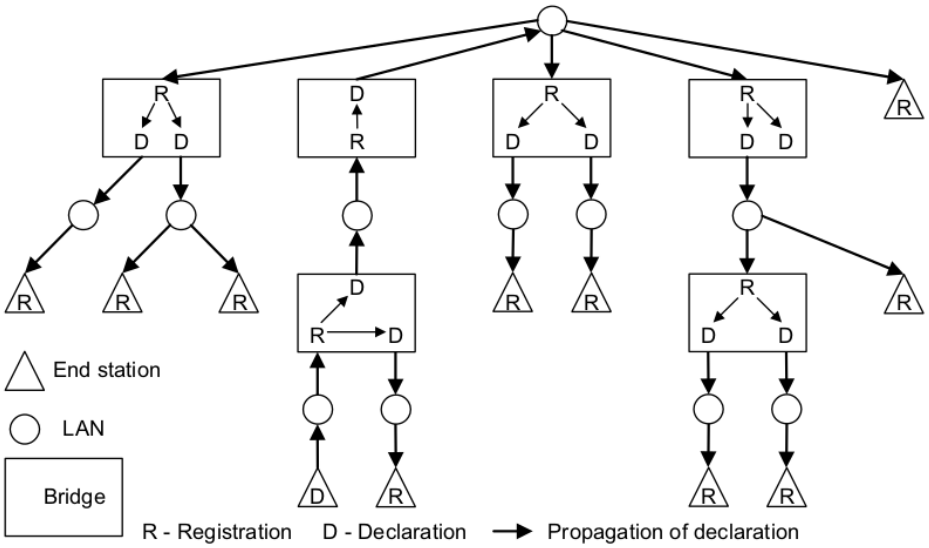


Figure 3.2: Attribute declaration, registration and propagation [30]

The example above illustrates a single station making an attribute declaration. The port of the first bridge in which the attribute arrives, makes a registration of said attribute and declares it in the remaining ports. The attribute is then propagated to the network, as long as all ports are in a forwarding state. Finally, all this process is replicated until the attribute is registered in its final end-station.

The three most important modules belonging to either bridges or end-stations are the MRP application, the MPR Attribute Declaration (MAD) module, and the MPR Attribute Propagation (MAP) module (see Figure 3.3).

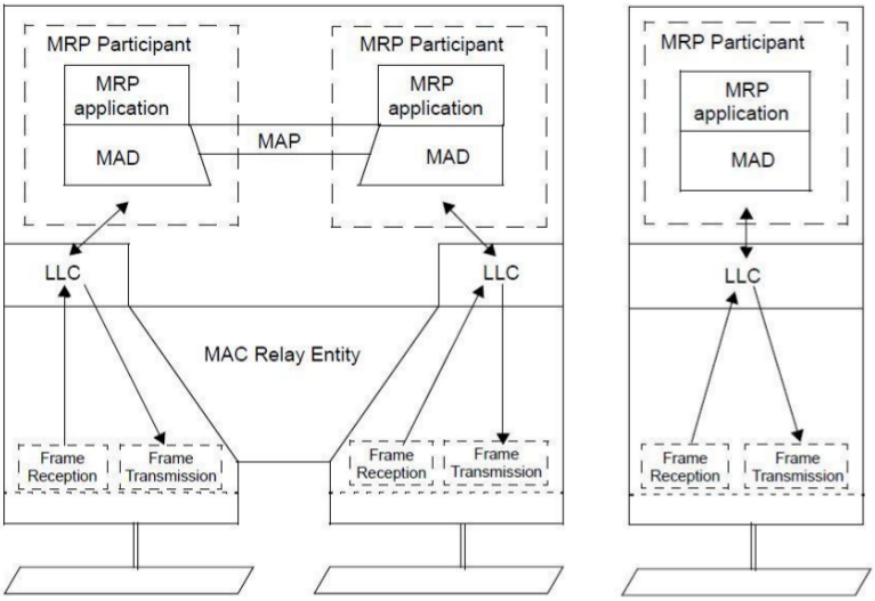


Figure 3.3: MRP architecture of a bridge and an end-station [30]

The MRP application module is responsible for the registration and de-registration of attribute values. Some of those are specified by the standard as follows:

- *Multiple MAC Registrarion Protocol (MMRP)* - this application allows the registration of MAC addresses in order to control MAC address filtering, restricting traffic only in required network segments;
- *Multiple VLAN Registration Protocol (MVRP)* - dynamically registers VLAN membership information;
- *Multiple Stream Registration Protocol (MSRP)* - the actual protocol that allows registration of data streams characteristics and resource reservation.

In order to understand the way each different module interacts with each other, it is important to know first what kind of attribute type they can declare and propagate during the protocol message exchange. There are 4 attribute types in total described as follows:

- **Talker Advertise Vector** - attributes of this type are used to identify a sequence of talker advertisements, indicating that related streams have not been constrained by insufficient bandwidth or resources;
- **Talker Failed Vector** - this kind of attribute identifies talker advertisements, that the associated stream has been constrained by insufficient bandwidth or resources;
- **Listener Vector** - this attribute types identify listener requests for the related stream regardless of bandwidth constraints;
- **Domain Vector** - finally, instances of this attribute are used to identify a sequence of values that describe the characteristics of an SR class.

When an end station wants to make or remove an advertisement of an available stream to another end station, it issues specific primitives of the protocol, as depicted in Figure 3.4.

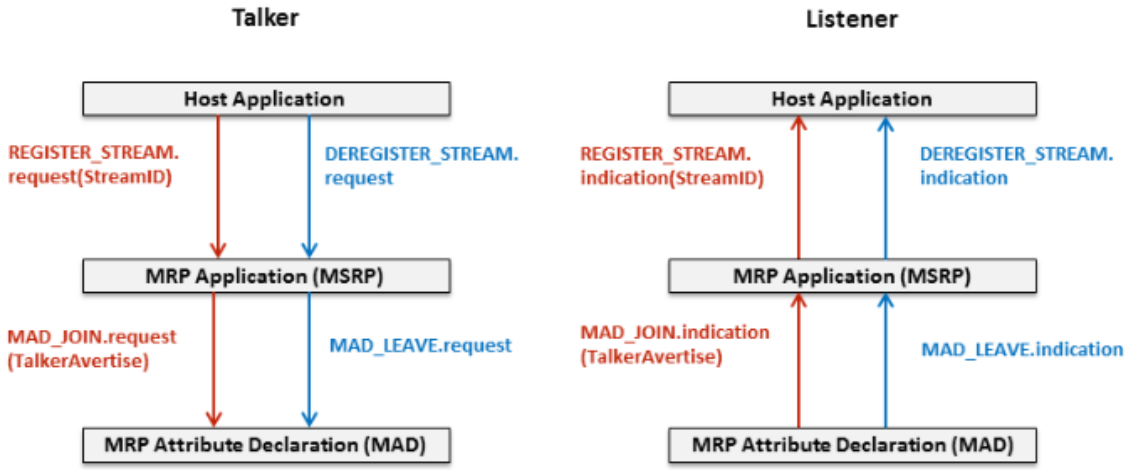


Figure 3.4: Talker advertisement of a stream

A talker application shall invoke the *REGISTER_STREAM.request* primitive to an MRP application, in this case it would be the MSRP. The MSRP will then issue the *MAD_Join.request* service primitive with either the *Talker Advertise* as its attribute type in case of success or the *Talker Failed* type otherwise. Next the listener is notified that a stream is being advertised through the primitives *MAD_Join.indication* to the MSRP and the *REGISTER_STREAM.indication* to the listener application. Finally the de registration of a stream advertisement follows a similar logic.

Instead of making advertisements, an end station could also make a a request of a particular stream. In this process the primitives invoked are presented in Figure 3.5.

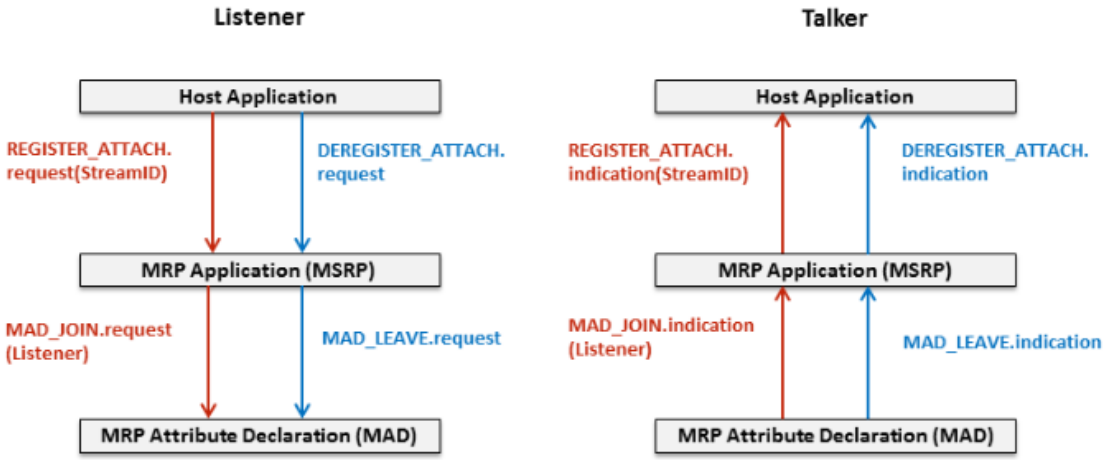


Figure 3.5: Listener request of a stream

A listener application invokes the *REGISTER_ATTACH.request* primitive to the MRSP application. This in turn issues the *MAD_Join.request* service primitive. The de registration process follows the same reasoning as the previews example.

MAD - MRP Attribute Declaration

In order to control and keep track of the existence of many declared and propagated attributes, a mechanism that manages and process all that data is specified by the protocol. That mechanism includes several state machines that preserve those attributes and make decisions upon them using control signals and the previews state the machine was on. This protocol uses 4 state machines in total, all specified by the MAD module, either present in a bridge or an end-station and are responsible for deciding what to do when an attribute is declared and registered. Some of these state machines may influence the behaviour of others depending on the conditions of the network. The four different state machines are presented as follows:

- **Applicant state machine** - This one is used for each attribute declaration made by an MRP Participant to an end-station or bridge port, in order to record it and process it. One Applicant state machine instance exists for each attribute declared;
- **Registrar state machine** - the second one has the role of recording attribute registrations for each MRP participant. Once again, there is one machine instance of this kind for each attribute;
- **Leave All state machine** - the third one is called Leave All state machine and generates events that can affect the operation of all Applicant and Registrar state machines. There is only one machine instance for each MRP participant.
- **Periodic Transmission state machine** - this one is used in environments likely to lose packets and as such can periodically generate messages with event against all Applicant state machines. As the previous one, there is only on instance of this kind o state machine per MRP participant.

The implementations of these four state machines are the core of the control mechanism and decision center of all the traffic of the MSRP protocol messages. With that said, it is possible to have different kind of participants in the network depending on how many of these state machines are implemented in the MAD module. When there is an Applicant and Registrar state machine implemented per attribute and a LeaveAll and Periodic Transmission state machines for the participant as a whole, we are faced with a *Full Participant*. When the same state machines are implemented with the exception of certain states and actions we are faced with a *Full Participant point-to-point subset*. When only the Applicant and PeriodicTransmission state machines are implemented we have an *Applicant-Only Participant*.

Finally when we have the same state machines implemented as the Applicant-Only Participant with the exception of certain states we have an *Applicant-Only point-to-point subset*, also referred as *Simple-Applicant Participant*.

Depending on the machine state and the triggered event, an action may be performed. The way a certain event is triggered in the context of this protocol is through protocol timers. The timers designation and description are presented as follows:

- *jointimer* - one instance of this timer exists per port, in each MRP participant and has the goal of controlling the interval between transmit opportunities that are applied to the Applicant state machine. This timer has its timeout initiated with 0.1s;
- *leavetimer* - this timer controls the period of time that the Registrar state machine will wait in the LV state before transiting to the MT state and one instance of this timer exists for each state machine tha is in LV state. Its timeout should be initialized with a range between 0.6s and 1s;
- *leavealltimer* - one instance of this timer exists per port in each MRP participant controlling the frequency of LeaveAll events generated by the Leave All state machine. Its timeout should be set to 10s;
- *periodictimer* - finally the last timer controls the frequency of *periodic!* events generated by the Periodic Transmission state machine and then applied to the Applicant state machine. One instance of this timer is required on each port and it's timeout is set to 1s when it starts.

MRP operation is not critically dependent on the protocol timer values just mentioned, however the protocol operates with more efficiency if those values are used. It is important to note that the time of *leavetimer* should be at least twice the maximum time of *jointimer* and the time of *leavealltimer* should be large compared to the *leavetimer* to minimize the volume of rejoining traffic generated.

Next, it is explained in more detail the operation and behaviour of each state machine.

Applicant state machine

There are two main goals in the Applicant state machine. The first one is to ensure that a participant's declaration is correctly registered by other participants' Registrars. The second is to prompt other participants to re-register after one withdraws a declaration. Changes in the states of a state machine in general can cause a determined action to be made. Changes in the Applicant state machine's variables trigger the transmission of MRPDUs to communicate attribute declaration or it's withdrawal.

The Applicant state machine is able to categorize the participants that use it in three distinguishable manners. These are called: Active Participants, Passive Participants and Observers. An Active Participant is the participant that has sent an MRP message or messages to make a declaration of a certain attributes. Passive Participant is a participant that requires a registration without having to declare the attribute so far to register it. The Observer state occurs when a participant do not requires registration at present, but needs to keep track of the attributes' registration in case he does or in case he changes to Passive Participant. In order to keep track of the participant's state for a certain attribute, four distinct attribute events are used to communicate with it. They are as follows:

- *Empty* - reflects the non-existence of any attribute declared or registered;
- *In* - attribute registered but not declared;
- *JoinEmpty* - attribute declared but not registered;
- *JoinIn* - attribute declared and registered.

Besides these, the Applicant state machine also implements some auxiliary messages such as *Leave*, which means an attribute was previously registered but then withdrawn, and *New* that declared a completely new attribute. There is also a special attribute event called *LeaveAll* which means that all registrations will shortly be de-registered and all participants will need to re-register them. *LeaveAll* messages are transmitted to ensure that any failure to withdraw a declaration does not result in an unwanted permanent registration. Attributes registered by the MRP Participant transmitting the *LeaveAll* as well as those receiving it are therefore timed out.

The Applicant state machine uses twelve different states (as mentioned in appendix A) and its transition to another state depends on the occurrence of a certain event from the sixteen displayed. Some of the most important events that could occur in this state machine are the transmission opportunity without a Leave All attribute signalled by the Leave All state machine, called *tx!* and the transmission opportunity with a Leave All signalled by it, called *txLA!*. As a consequence, besides the swapping of states, each of this events may trigger a certain action to be performed. Some actions can include, for example, an encoding of a specific attribute event in an Multiple Registration Protocol Data Unit (MRPDU) frame to be sent into the network. Those actions are described as follows:

- *s* - this action verifies if the Registrar state machine is in the *IN*, *MT*, or *LV* states. If it is in *IN* state, then the attribute event value encoded in the MRPDU message is *In*. If, on the other hand the state is either *MT* or *LV*, then the attribute event value is *Empty*;

- *sJ* - this action also verifies the Registrar state and if it is *IN*, then the attribute event value *JoinIn* is encoded in the protocol message. If it is *MT* or *LV*, then the value is *JoinEmpty*;
- *sN* - this action is very simple and only encodes the *New* value into the MRPDU message;
- *sL* - finally this action encodes the *Leave* value.

In short, a full MRP Participant maintains a single instance of the Applicant state machine for each attribute value for which the participant needs to maintain state information.

Registrar state machine

The main role of the Registrar machine state is to preserve all attribute declarations done by an MRP participant by registering them. As such, this machine state does not send any protocol message, leaving it to the Applicant state machine to do it. There are three distinctive states in this machine. The first one is is called *IN*, which indicates an attribute registration, the second one is called *LV*, which means an attribute was previously registered, but is now being timed out and the last one is called *MT* that indicates that no attribute was registered.

Similarly to the Applicant state machine, the Registrar state machine executes some kind of action depending on the previous machine state and the triggered event. Some actions consist only in a change os state, others consist in starting or stopping the *leavetimer* and others perform actions such as New, Join, and Lv. These three actions will be explained in the MSRP operation section of this work.

LeaveAll state machine

This machine state has the important role of generating *LeaveAll* events against all Applicant and Registrar state machines associated a specific participant and port. This event has the objective of de-registering all attributes, including failed withdrawn attributes, which otherwise would be considered by the Registrar as a permanent unwanted registration. All other registrations are re-registered periodically to ensure their right propagation. The *leavealltimer* ensures the correct elapsed time between two consecutive transmissions of the *LeaveAll* event to every participant that declared the same attribute.

This state machine has two states of operation, the *Active* state and the *Passive* state. The active state is reached when the *leavealltimer* expires and the passive is associated with *LeaveAll* messages sent by other participants to the same attribute. The way actions are performed by this machine is very similar to the other two. There can be a change of state,

the starting or stopping of the *leavealltimer* and the execution of another specific action. One such action is called *sLA* and has the role of performing the encoding of the *LeaveAll* event in the MRPDU message and give rise of the *rLA!* event against all instances of the Applicant or Registrar state machines associated with the a MRP participant.

Periodic transmission state machine

This state machine exists in case the protocol operation is susceptible of losing packets, assuring that all attributes are registered through the generation of periodic events. Like the LeaveAll state machine, this one also has two states, one *Active* and one *Passive*. The actions performed by this state machine are the change of state, the start of the *periodictimer* and the execution of the periodic action. This last one has the main role of causing a *periodic!* event against all Applicant state machines associated with the participant.

MAP - MRP Attribute Propagation

MRP Attribute Propagation (MAP) is an important module present in all bridges that follow the SRP protocol. This particular module enables the propagation of attributes registered on the bridge ports to other participants in the network and also decides when to forward or filter a particular data stream. It is possible for each MRP application to specify a certain operation for this module. Such applications are the MMRP, MVRP and MSRP. For the purposes of this work, only the MSRP context is mentioned in MAP function.

Any protocol message issued by an application module (MSRP application) and associated with a given port in the set is received by MAP and propagated to the MAD instance or instances associated with any remaining port of the set. More specifically, if either a *MAD_Join.indication* or a *MAD_Join.request* message is received, a *MAD_Join.request* is propagated. On the other hand if the MAP receives a *MAD_Leave.indication* or a *MAD_Leave.request*, the attribute is propagated as a *MAD_Leave.request* only when there is no registration of that attribute on any other port except the one it is associated with. Generally speaking, the rules just mentioned enable the propagation of either attribute registrations or attribute de-registrations when all other ports in the set are de-registered in that moment.

The way MAP propagates a talker attribute is described in the following table 3.1:

Table 3.1: Talker attribute propagation after merging

	(none)	Listener Ready	Listener Ready Failed	Listener Asking Failed
Talker Advertise	Talker Advertise or Talker Failed	Talker Advertise or Talker Failed	Talker Advertise or Talker Failed	Talker Advertise or Talker Failed
Talker Failed	Talker Failed	Talker Failed	Talker Failed	Talker Failed

This information means that if no talker attribute, associated with a particular *streamID*, is registered, then this attribute is only propagated to its respective destination port. However, if either a *Talker Advertise* or a *Talker Failed* attributes are registered, then both ought to be propagated to all other non-blocked ports of the switch. In case of *Talker Advertise* attributes, further processing is required. More specifically, MAP would need to analyse available bandwidth, in order to determine if the outbound port has enough resources available to support the stream. If sufficient resources and bandwidth exist, then the *Talker Advertise* attribute is propagated. Otherwise, MAP will propagate a *Talker Failed* on the outbound port and add appropriate failure information (annex).

Now it will be given a closer look on how MAP deals with listener's attribute propagation. The table 3.2 represents just that.

Table 3.2: Listener attribute propagation after merging

	(none)	Listener Ready	Listener Ready Failed	Listener Asking Failed
Talker Advertise	(none)	Listener Ready	Listener Ready Failed	Listener Asking Failed
Talker Failed	(none)	Listener Asking Failed	Listener Asking Failed	Listener Asking Failed

Unlike talker attributes, listener attributes from several listeners on different ports can be merged into a single one according to the following rules:

1. processing of incoming listener attribute registration on a port based upon the status of the associated talker attribute registration. If no listener attribute is registered on a particular port then no Listener attribute will be propagated. Also if a *Listener Asking Failed* attribute is registered on a port, then it will be merged with other listener attribute before it is propagated.

2. merging of all individual listener attributes gathered on the first step into a single attribute and its propagation.

Finally it is important to mention that if no talker attributes are associated with any listener attribute, then the listener attribute will never be propagated. It is also acceptable to have more *Talker Advertise* declarations without associated *Listener Ready* declarations

MSRP Application

Alongside MAD, the MSRP application component is also present in a MRP participant. This component is present in the point of the network where attributes are declared or registered, which means a MSRP application exists in each end station and each bridge port. This module is responsible of managing attribute values and issuing request primitives to the

MAD and receiving indication primitives from it. The primitives issued by this module can have the attribute_type, attribute_value and new arguments as shown bellow.

- MAD_Join.request (attribute_type, attribute_value, new);
- MAD_Leave.request (attribute_type, attribute_value);
- MAD_Join.indication (attribute_type, attribute_value, new);
- MAD_Leave.indication (attribute_type, attribute_value).

The *attribute_type* specifies the type of the declaration, the *attribute_value* specifies the instance of that type and the *new* indicates an explicit new declaration. This module is also responsible of obtaining MAP contexts, set a group MAC addresses and *EtherType* values for a given protocol between application participants.

This module maintains a set of state variables that are used internally:

- Port Media Type - this variable defines the way the medium is accessed by the port. It can be defined with *Access Control Port* value in which the transmitter controls the access to the medium or with *Non-DMN shared medium Port* in which the transmitter is attached to the shared medium, but does not have control over it;
- Direction - this variable indicates whether this is a talker or listener declaration;
- Declaration Type - this variable is derived from the attribute type definition and the *fourPackedEvent* value. Indicates the specific kind of talker or listener with the following values:
  (a) talker advertise - represents an advertisement of a stream with no bandwidth constraints. The talker advertise attribute will continue to be declared as long resources continue to be available;
  (b) talker failed - represents an advertisement of a stream with bandwidth constraints;
  (c) listener ready - this means one or more listeners are requesting an attachment and there are sufficient resources in the network to receive the stream;
  (d) listener ready failed - this means that two or more listeners are requesting an attachment and at least one of them has sufficient resources to receive the stream;
  (e) listener asking failed - one or more listeners are requesting attachment and none of them have sufficient resources to receive the stream.
- SRP parameters - this parameters describe important values which are used to make the allocation of resources.

Structure of MRP messages

During a session between two MRP participants several information must be exchanged, mainly the attributes advertised with all its values during requests for a transmission opportunity in the Applicant state machine. The way this is done is through the use of messages encoded in an *MRP Protocol Data Unit (MRPDU)*. Every MRPDU follows a specific structure defined by the protocol, with different content depending on the application to which it pertains. In this case, it will only be described the MRPDU structure within the context of the MSRP as shown in figure 3.6:

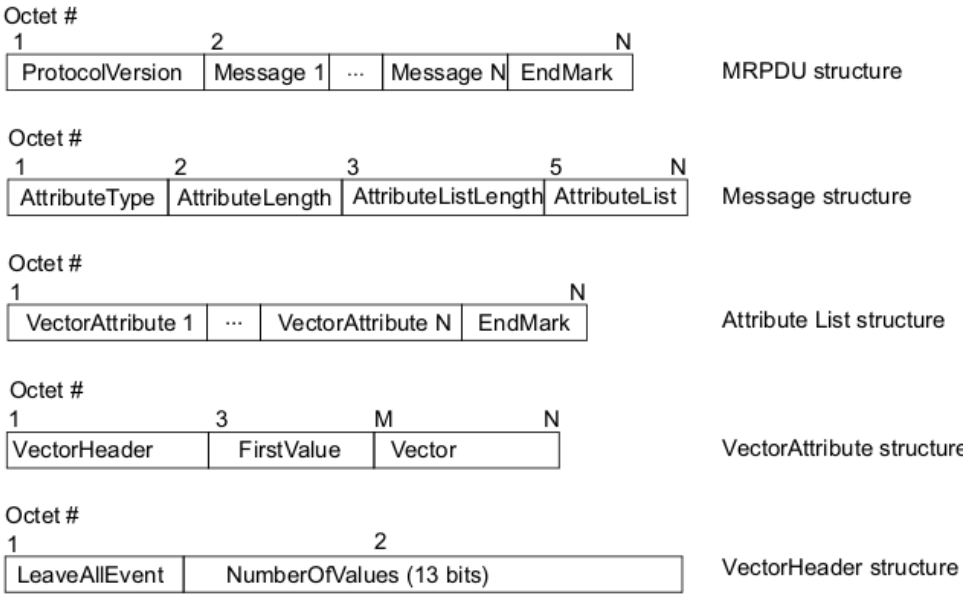


Figure 3.6: Format of the MRPDU [30]

The compacted way in which all potential attributes of the MSRP are encoded, allows for the MRPDU to be further encapsulated in a single IEEE 802.3 frame, promoting efficiency. The MRP application is chosen through the *EtherType* of said frame. Every field of this structure is now explained in further detail.

- *Protocol Version* - the first byte identifies the protocol version used and is included in the structure to provide the ability to identify future enhancements to MRP applications;
- *MSRP message* - all the bytes that follow are reserved for the protocol message, with the exception of the last two bytes called *end mark*, that are used to terminate the structure. Each message contains the following header fields: one byte with the *attribute type*, one byte with an *attribute length*, two bytes with an *attribute list length* and a structure with the attribute values called *attribute list*, that acts as the body of the message;
- *Attribute Type* - the attribute type identifies the type of attribute to which the message applies. As told before this field can take 4 possible values: the *talker advertise vector*,

talker failed vector, *listener vector* and the *domain vector*;

- *Attribute Length* - this field indicates the the length in bytes of the *first value* field;
- *Attribute List Length* - this field represents the length in bytes of the *attribute list* to which the message concerns;
- *Attribute List* - this field is composed of a list of structures called *vector attributes* each containing the information of the attribute used in the state machines. Each *vector attribute* contains three more fields called *vector header*, *first value* and *vector*;
- *Vector Header* - this field is used to encode two values, each called the *LeaveAllEvent* and the *NumberOfValues*. The value of *LeaveAllEvent* can mean either a *LeaveAll* operator, which is interpreted on receipt as a MAD Leave All event applied to all state machine of all attributes specified with the same *Attribute Type*. If not, this value causes no event process and is encoded as a *NullLeaveAllEvent* for efficiency purposes. This values is added to the second value called *NumberOfValues*, which represents the number of events encoded in the *Vector* field.
- *First Values* - the first value represents a structure with N bytes, depending on the attribute type specified by the MRP application. Figure 3.7 depicts all possible fields that this structure could have.

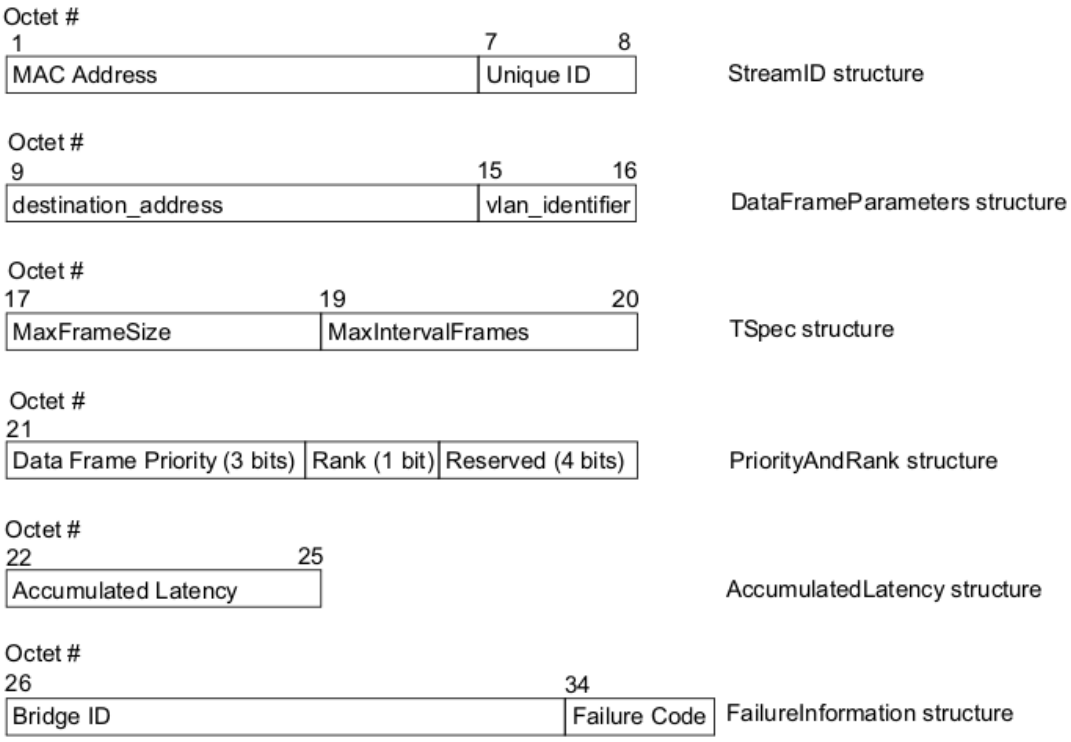


Figure 3.7: Format of the First Value structure

It is important to note that not all fields can be filled, with some of them dependent on the attribute type encoded in the MRPDU packet. As such the *FirstValue* of a

Listener attribute is only composed of the first 8 bytes of the *StreamID*. The *Talker Advertise* uses all fields down to *AccumulatedLatency* and the *Talker Failed* uses all fields contained in the *FirstValue* including the last bytes of *FailureInformation*.

It is now described in further detail each field of this structure:

(a) *StreamID* - this field is comprised of a 6 byte length MAC address, and is used to match different end stations and the 2 other bytes describing a unique stream ID used to distinguish multiple streams sourced by the same end station. With the combination of these two components there can be only unique Stream IDs across the entire bridged network. These values can be offered by a device controlling the entire system.

(b) *DataFrameParameters* - this field contains the parameters common to all frames belonging to the data stream for which the MAD is reserving resources and is also used to create Dynamic Reservation Entries during the queueing and forwarding process. Those parameters are the destination MAC address and the VLAN identifier. By filtering stream destination addresses, the protocol is able to reserve bandwidth guarantees for stream that do not have a reservation. Systems with VLAN awareness can use any valid VID from 1 through 4096.

(c) *TSpec* - this field contains the traffic specification associated with a stream, namely the *MaxFrameSize* and the *MaxIntervalFrames*. The first component represents the maximum frame size the talker will produce, excluding any overhead for media specific framing. The *MaxFrameSize* is specifically used to allocate resources requested by talker declarations. When the amount of bandwidth to be reserved is calculated, the media specific framing overhead on the egress port is also determined and added to the *MaxFrameSize*. The second component of this field is called *MaxIntervalFrames* and it represents the the maximum number of frames that the Talker may transmit.

(d) *PriorityAndRank* - this field specifies two components: a 3 bit data frame priority and a single bit rank. The priority is used to determine, on a particular egress port, which queue the frame is placed into. The rank is used to decide which streams can and cannot be served, when the MSRP registrations exceed the capacity of a Port to carry the corresponding data streams. As such this value is used to help decide which stream or streams can be dropped, with a lower numeric value representing a greater importance than a higher numeric one.

(e) *AccumulatedLatency* - this parameter is used to determine the worst-case latency that a stream can encounter in its path from the talker to the listener. This value is intended to never increase beyond a certain threshold during the life time of the reservation. If that happens then the MSRP will change a *Talker Advertise* to *Talker Failed*.

(f) *FailureInformation* - this parameter is filled when it is issued a talker failed declaration. The listeners know the cause of the failure by checking this field when

Talker Failed attribute is issued by the switch.

- *Vector* - this field has the role of encoding events associated with an attribute to be applied to the state machines of the MRP protocol. They can be of two different types and take several event values as shown in figure 3.8.

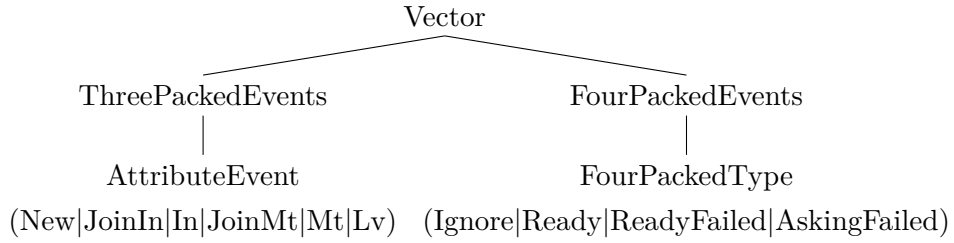


Figure 3.8: Contents of the Vector field

The number of encoded event values byte of the *ThreePackedEvents* type is determined by the nearest upper integer of the *NumberOfValues* devided by 3. The following equation shows an example on how to encode a *ThreePackedEvents* with three possible event values.

$$\begin{aligned} ThreePackedEvent = ((((firstAttributeEvent \times 6) + secondAttributeEvent) \times 6) \\ + thirdAttributeEvent) \end{aligned} \tag{3.1}$$

Each of the *attributeEvents* values relates to a different state machine of the originating MAD and are all common to the following attribute types: *Talker Advertise*, *Talker Failed* and *Listener*. In case there is a value of *NumberOfValues* that is not a multiple of three, then the last one or last two event values are encoded with the numeric value 0 and are ignored on the MRPDU.

The *FourPackedEvents* values enables the distinction of different types of listener declarations such as *Listener Ready*, *Listener Asking Ready* and *Listener Asking Failed*. According to the protocol, these values can be encoded by successively adding a *FourPackedType* value and multiplying the result by 4, as shown by the following expression:

$$\begin{aligned} ThreePackedEvent = ((firstFourPackedType \times 64) + (secondFourPackedType) \times 16) \\ + (thirdFourPackedType \times 4) + fourthFourPackedType) \end{aligned} \tag{3.2}$$

Using the same reasoning as above it is possible to know the number of encoded bytes by rounding up the result of *NumberOfValues* divided by 4.

3.2.2 Impact of IEEE 802.1Qat redundancy in AVB networks

Of all the protocols specified by the AVB standard, SRP and MRP are the ones that most influence redundancy because they are the ones that actually perform the resources

reservation for streams of data. Contrary to standard Ethernet networks, where frames are forwarded to their destination according to learned ports of the FDB tables entries, SRP and MRP exchange messages to perform a path selection according to the resources available, before sending the actual stream frames. In other words, in a network without AVB protocols it's possible to pre-determine a worst case upper bound value for the recovery time lost in topology reconfiguration. This recovery time is calculated though:

$$t_{rec} = t_{rec\_protocol} + t_{FDB\_flush} \tag{3.3}$$

Here the $t_{rec\_protocol}$ is the time expended by a redundancy protocol, e.g. Redundancy Spanning Tree Protocol (RSTP) , to reconfigure a new physical path and the t_{FDB_flush} is the time lost in clearing out the content of the FDB tables, so that the new path can be learned. In networks with AVB protocols in place however, the total amount of time spent in reconfiguring the network is not foreseeable [22]. Its time could be calculated by the following equation:

$$t_{rec} = t_{rec\_protocol} + t_{FDB\_stream} \tag{3.4}$$

however there is no easy way of knowing the value of t_{FDB_stream} because AVB technology was not developed to support fast and deterministic stream reconfiguration. To resolve this issue the authors of [22] have proposed a method to enhance the SRP protocol, in which there is no communication interruption and where redundancy path registration for streams are independent from the operation of redundancy protocol used.

3.3 IEEE 802.1Qav - Forwarding and Queuing

In order to assure the distinction of different classes of traffic, the IEEE 802.1Qav specifies a set of rules to ensure that all marked streams of a particular class are forwarded in the switch with the appropriate QoS levels. IEEE 802.1 also includes the priority ingress metering to ensure a correct handling of time sensitive traffic and also defines a credit-based shaping mechanism to smooth out the traffic from the egress ports. This scheduler mechanism is used in AVB switches to reduce latency for specific data streams with high real-time requirements [22].

This amendment is intimately related with the MAP module, because besides propagating attribute declarations, the MAP module is also responsible for deciding which stream has higher priority and for determining the switch resources available to them. For that the MAP module makes use of Dynamic Reservation Entries, which specifies whether a particular stream must be forwarded or filtered, as well as its associated *streamAge*. Each of these entries

could be identified by some or all kind of the following elements: group MAC address or an individual one, for which the dynamic reservation information was registered or a reservation port map. The last one consists of a control element for each outbound port, identifying forward (reservation information exists for this port) or filter (reservation information does not exist for this Port) states of frames. There can only exist one entry for a given identifier and its initial value should be filtering.

MSRP protocol makes use of the *PriorityAndRank* field present in the first value of the exchanged protocol messages to determine each stream priority and importance. More specifically the stream *Rank* is compared to decide the stream importance, with the use of the *streamAge* as a tiebreaker in case two streams have the same *Rank*. If both of them are equal then the most important stream is decided by comparing *StreamIDs* and making the numerically lower one the most important.

According to the IEEE 802.1Qav [33] the next expression represents the value of the actual bandwidth needed to support a given stream.

$$actualBandwidth = (perFrameOverhead + assumedPayloadSize) \times maxFrameRate \tag{3.5}$$

By knowing the protocol stack that supports the bridge port, such as the MAC framing that is added by the underlying MAC service, it is possible to determine the overhead that is added to the frame payload and with that determine the *perFrameOverhead* value. The *assumedPayloadSize* is equal to the *MaxFrameSize* making the sum of these values the total frame size. Finally the *maxFrameRate* is determined by the following expression 3.4, where the *classMeasurementInterval* depends on the SR class associated with the stream, with the assumed value of 125µs for class A and the value of 250µs for class B.

$$maxFrameRate = MaxIntervalFrames \times (\frac{1}{classMeasurementInterval}) \tag{3.6}$$

Some parameters used in the equations just mentioned are specified by the *TSpec* field, such as the *MaxIntervalFrames* and the *MaxFrameSize*. The *TSpec* field, doesn't take into account the effect of the per-frame overhead associated with the transmission of the protocol messages. However the determination of this value is necessary, so that the *operIdleSlope(N)* parameter corresponding to the actual bandwidth, in bits per second, that is currently reserved for use by the queue associated with traffic class N. This value is used by a credit-based shaper algorithm as the idle Slope for the corresponding queue. It is also important to notice that the math expressions just mentioned have as an assumption about the constant size of a stream data, so that a maximum value of payload size can be determined.

4

Omnet++ Implementation

In this chapter it will be presented the main concept and model for the AVB protocols implemented in the simulator environment Omnet++. A brief introduction to Omnet++ is first given and then all requirement assessments, modules description and the implementation of the proposed resource reservation method is depicted.

4.1 Model Specifications

In order to implement the SRP resource reservation mechanism, several specifications must be defined. As such, the work developed is based on the following premisses:

- Implementation of a MSRP end station capable of declaring and register attributes, initiating a transmit opportunity and exchange MSRP protocol messages;
- Implementation of a MSRP Switch capable of declaring register and propagate attributes present in the protocol messages. Reserve resources (bandwidth) along the path between a talker and a listener, according to a mechanism of traffic classification into several levels of priority.
- Implementation of specific modules, with MAD module being the first and most important one. It is this module that will contain the core functionalities built from scratch, including the various state machine algorithms of the MRP protocol and the specification of the several messages.
- Implementation of a module called MAP that has the role of propagating attributes to the ports leading to their destination and making it known to the remaining ports of the switch. This module should also enquire the switch about its resources, when it wants to make a reservation.

It is therefore necessary to conceive a model implementing these modules in order to reach the following goal: simulation of a resource reservation mechanism between talker and a listener in a simple switched network.

Next, it will be depicted an overview of the simulation environment used in this dissertation.

4.2 Simulation Environment

In this section, the main reasons for choosing Omnet++ [36] as the environment for simulating the MSRP model, are referred.

Omnet++ is an IDE that provides a simulation of discrete events based on C++ with modular type architecture. It has also the purpose of complement simulation tools already on the market such as open simulator NS-2, or commercial market simulators like Opnet. Released to the public in 1997, OMNeT++ is available on most popular platforms of the market, such as Linux, Mac OS/X and Windows, using to that end the compilation tool GCC or C++ Microsoft Visual. Omnet++ is also a tool whose source code is open and available for academic or research purposes by non profit institutions. However it is also used on many research projects of many commercial companies such as IBM, Intel, Cisco, Thales or Broadcom. Omnet++ was developed from a basic idea, to be as generic as possible, allowing the development of simulations rather than being itself a specific simulator. So instead of directly providing components of specific models for running simulations, it provides the basic mechanisms and tools to write our own models and therefore our own simulations. This environment tool has been used to build models to the most diverse areas of communication networks, multiprocessors and other types of distributed systems. Among the simulation study of communication networks it is included the development of models for wireless networks, ad-hoc networks, peer-to-peer networks, sensor networks, and IP networks IPv6, MPLS, wireless channels, storage area networks (SAN), optical networks, networks of queues, file systems, high speed interconnections (InfinitiBand), and others [37]. In the context of this dissertation the framework to be used is the INET framework. INET corresponds to an extension of Omnet++ itself, with various network protocols implemented, especially in the intermediate layers of the OSI protocol stack and some applications. INET's main goal is to support the simulation development of Ethernet networks and small mobile networks.

One of the main advantages of developing software in Omnet++ is the possibility of enabling great modularity and reusability in its software, meaning that it could be used in other simulations with completely different contexts [38]. In order to achieve that, the models built need to follow a given structure and architecture. A module in Omnet++ can be classified into two kinds: simple modules, also called active modules, which are written in C++ and compound modules, that are grouped into one or more single modules or other compound modules. This allows us to organize our models according to a hierarchical architecture,

without any limitation as the number of levels that can be achieved. There is also a special kind of compound module called network modules without any input or output to the exterior, where other modules are combined like building blocks. Each module communicates between them by sending messages to each other through an input interface and an output interface connected together, called gates. These connections can be parametrized with several properties such as propagation delay, transmission rate or error rate. The messages must be necessarily transmitted and received by simple modules. All compound modules work as a transparent box that just automatically redirects those messages between the exterior and interior [39]. All model definitions in Omnet++ (module structure and their connections) are made from a description language topology called NED. This language is used to make simple module declarations, *i.e.* a description of its input and output interfaces to enable compound modules definitions. With this it is possible to simplify the writing of our models with only its topology definition in the NED files and to push the complexity of their behaviour into the C++ files. The code present in these files is usually structured in a standard way for all Omntet++ developments. Normally the operating system invokes in the first place an *initialize* method. This method can be called in different stages, in order to give omnet++ the opportunity to first set up all network description features present in NED files, into the first stage; and the initialization of classes and variables present in C++ files into the second stage. After the initialization phase, it is usually necessary to invoke the Omnet++ standard method handle message. This method has the responsibility of receiving and processing incoming messages either from the same module or from another module. Sometimes it can also be used the finalized method, for statistical purposes.

4.3 General Architecture

In this work an MSRP model supporting the main features of Stream Reservation Protocol was built. Some modules were built from scratch, such as the *MAD*, the *MAP* and the *MAD Relay* modules, while others, like the *Ethernet Encapsulation*, *Switch Relay unit* or *Queues*, were extended by taking advantage of modules already existing in the INET framework, as shown in the next figure. Any other module used derived directly from the INET framework. Of the modules shown in figure 4.1 *MSRP services* was the only one not implemented.

This model description shows an assembly of two end stations and a switch, connecting them both. Each of the end stations represents either a talker or a listener, with a MSRP service, MAD module, containing the core implementation of the MSRP protocol, an Ethernet encap / decap module, responsible of encapsulating incoming packets from upper modules into Ethernet II frames, or decapsulating them when they come from a MAC interface connected to the host physical link. The switch is comprised of one MAP with multiple gates, each connected to the MAD Relay unit, responsible of relaying MSRP messages to the respective MAD module, created per switch port. Finally there is also one Ethernet encap / decap module and a switch relay unit connected to each switch port MAC interface.

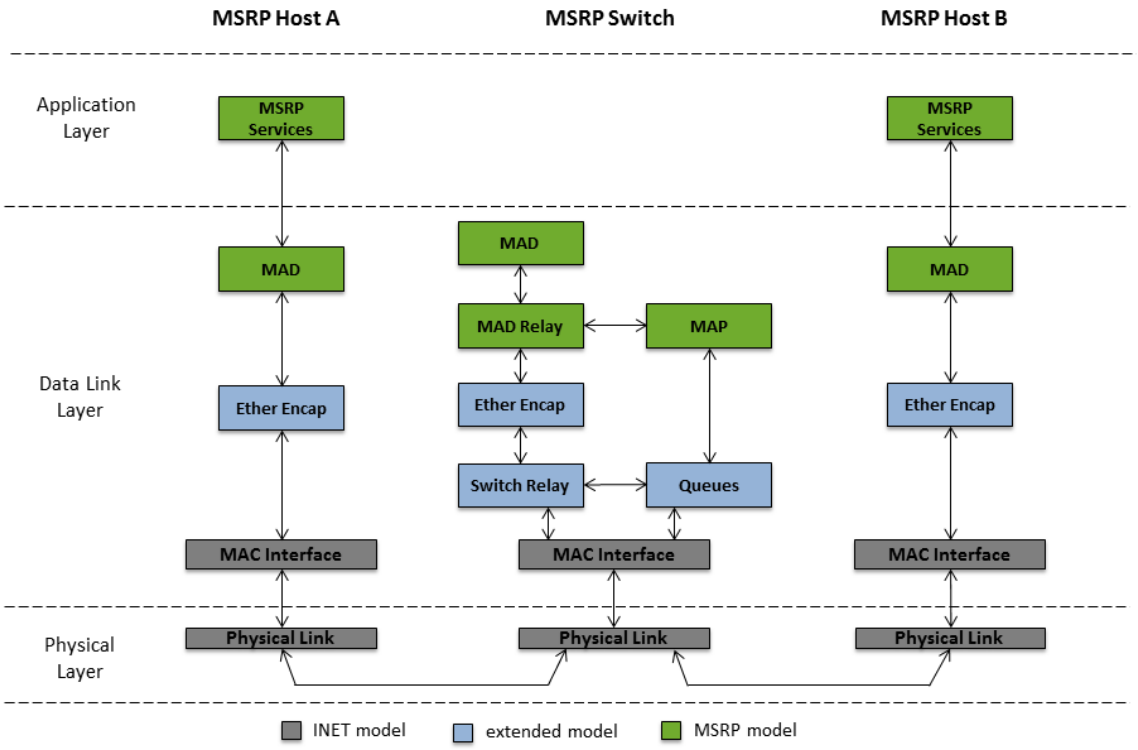


Figure 4.1: MSRP model integration in INET

The rest of this chapter will be dedicated to the review of the proposed model implementation. It will be presented the general architecture of this system and the behaviour and implementation of each relevant model.

4.4 Implementation Details

The moment the simulation starts running, each *initialize* method pertaining all simple modules are invoked. In the first stage of this initialization all NED files composing the network infrastructure, including its elements, are initialized. Even simple modules and internal connections inside compound ones must be initialized in this phase. Several different C++ classes were created or extended from ones already existing in INET. Of those created the ones that stand out are *MAD*, *MAP*, *MADRelay*, *MRPDU*, *MSRPTimer* and *StateMachine* classes.

Figure 4.2 represents an example of an end station used in this work. It contains a MAD, an Encap and a MAC interface with an optional internal queue interface. By default this MAC interface is configured to use the half-duplex CSMA/CD as a medium access protocol.

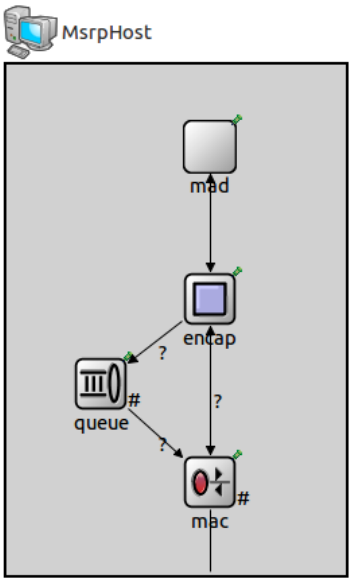


Figure 4.2: Inner details of an MSRP end station

Figure 4.3 represents an MSRP switch with all modules created, extended or used from the INET framework.

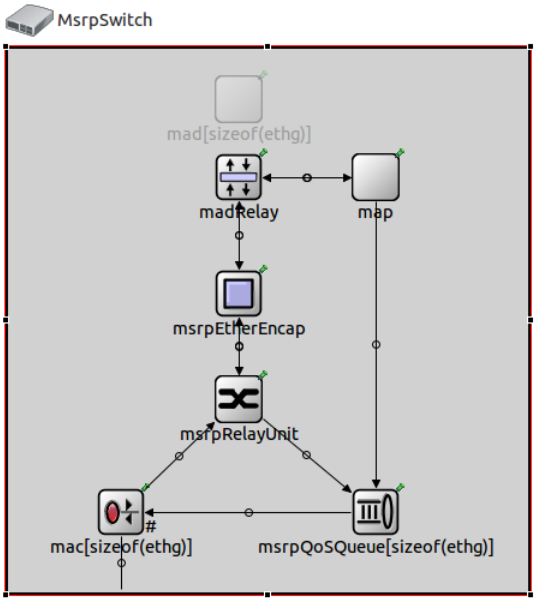


Figure 4.3: Inner details of an MSRP Switch

Finally, figure 4.4 depicts the inner details of the MSRP queue, with a classifier module, drop tail queues and a priority schedule.

The classes that were not directly visible in figures 4.2, 4.3, 4.4, such as *MRPDU*, *MSRP-Timer* and *StateMachine* do not implement the logic of any kind of module. Instead they are used by them to ensure the proper functioning of the model. The *MRPDU* class contain

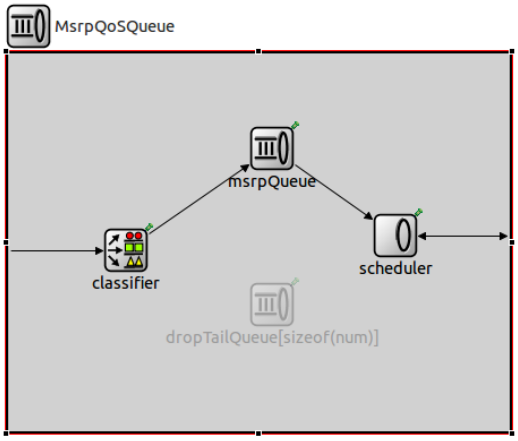


Figure 4.4: Inner details of an MSRP Queue.

all fields and structures present in an MRPDU packet specified by the protocol, as well as *MAD_Join* and *MAD_Leave* messages as show in the code just below:

```
packet MRPDU
{
    char ProtocolVersion enum(Protocol_Version);
    MsrpMessage msrpMsg;
    short EndMark enum(End_Mark);
}
packet MAD_Join
{
    MACAddress dest_add;
    Stream_ID StreamID;
    int declarationType enum(DeclarationType);
    Data_Frames_Parameters DataFrameParameters;
    T_Spec TSpec;
    char PriorityAndRank;
    int AccumulatedLatency;
    Failure_Information FailureInformation;
}
packet MAD_Leave
{
    MACAddress dest_add;
    Stream_ID StreamID;
    int declarationType enum(DeclarationType);
}
```

The *MSRPTimer* class, containing all four types of timers used by the *Applicant*, *Registrar*, *LeaveAll* and *Periodic* state machines are respectively the following: *join* timer, *leave* timer, *leavall* timer and *periodic* timer.

```
class MSRPTimer extends cMessage
{
    char timerKind enum(MSRPTimerType);
}
```

Finally the *StateMachine* class contains four more extended classes each describing each of the four state machines specified by the protocol, as it is depicted just bellow:

```
class StateMachine
{
}
class ApplicantSM extends StateMachine
{
    int state enum(Applicant_State);
}
class RegistrarSM extends StateMachine
{
    int state enum(Registrar_State);
}
class LeaveAllSM extends StateMachine
{
    int state enum(LeaveAll_State);
}
class PeriodicTransmissionSM extends StateMachine
{
    int state enum(PeriodicTransmission_State);
}
```

Because the last three classes mentioned, only needed generic setting and getting methods, they were described in .msg files and generated automatically into C++ code by Omnet++. These classes are depicted in the code just below.

It will be now explained in further detail the way each developed module works.

4.4.1 MAD

This is a simple omnet++ module and is the most important piece of the whole system. Because it contains most of the MSRP protocol implementation, its NED code is comprised of one input gate, one output gate and the definition of several parameters, which are are loaded by omnet++ configuration files or are initialized with default values in the NED file itself. These parameters are in turn used by the implemented module MAD C++ class and represent values not only used to control the simulation environment, but also to be used by the protocol algorithm itself. Some of the most important simulation parameters are the starting time, stopping time and destination address. Model parameters include session number, attribute type, reservation request value and protocol timers values. Some of these parameters, like *session*, *attribute type* and *reservation request* are hardcoded because of the absence of a MSRP application.

When the simulation starts the first thing the MAD class does is invoke the *initialize* method. This method gets all parameters, specified by the INI configuration file and the default ones present in NED files, in case they are necessary. It is also in this method that all state machines are first created or updated, according to the session number (unique

stream ID) parameter. The state machines corresponding timers are also initialized here. Normally MAD_Join or MAD_Leave messages would be the triggering mechanism for which the reservation would take place, but since there are no MSRP services this action must be triggered externally. In this scenario a reservation request is made in the INI files and a *transmitting opportunity* occurs with a *talker advertise vector* as an attribute type value encoded in a MRPDU packet.

Next we have the *handleMessage* method of the MAD class, which can be be invoked in two situations. The first one occurs when the MAD module receives an incoming message from another module connected to it. According to this implementation this happens when there are incoming MRPDU, MAD_Join or MAD_Leave messages from the lower modules (*Ether encap* in case of end stations or *MAD relay unit* in case of switches. The second situation occurs when the MAD receives a message coming from it self, also known as self messages. These self messages represent in this case the expiration of one of the four MSRP protocol timers (*join* timer, *leave* timer, *leaveAll* timer and *periodic* timer).

If a self message arrives then the *handleSelfMessage* is called. Because there are more then one timer in the same Omnet++ class, the kernel has no way of determining which timer has arrived. In order to resolve this issue, each timer when is scheduled must have a specific context assigned. As described by the protocol, each timer was assigned the context of the state machine associated with it, or in other words the *join*, *leave*, *leaveAll* and *periodic* timers were associated their respective *applicant*, *registrar*, *leaveall* and *periodic* state machines. According to each timer triggered self message, each respective process method is invoked. The figure 4.5 represents the flow in which the module's methods are called.

Since there must be, according to the MSRP protocol, one instance of the *applicant* and *registrar* state machine for each attribute value, a standard template library (STL) container named *map* was used. This *map* has a first value called key used to identify its second value. In the context of this work the first value corresponds to the *unique stream ID* used to identify the current stream. The second value of the *applicant* or *registrar* map, corresponds to another structure, called *ApplicantEntry* or *RegistrarEntry* respectively. Each entry contains its respective state machine and timer. As such, when a message triggered by the *join* timer arrives the *handleApplicantStateMachines* tries to find its respective state machine in the *Applicant* std::map. If it is not there, a new one is created and the *join* timer is scheduled with the *BEGIN* event. If the state machine is found, then the vector attribute is decoded and the *leaveAllEvent* value present in the vector attribute header analysed. If the value corresponds to a *NULL_LEAVE_ALL_EVENT*, the join timer is scheduled with a transmitting opportunity event (*TX*). If it is *LEAVE_ALL* the timer is scheduled with transmitting opportunity event with leaveAll (*TXLA*). Finally if the values corresponds to *LEAVE_ALL* and the MRPDU packet is full, then the timer is scheduled with a transmitting opportunity with leaveAll and packet full event (*TXLAF*). The *handleRegistrarStateMachine* method has a similar behaviour, with the difference that only

the *TXLA* event can be scheduled in a *leave* timer. Figure 4.6 illustrates the process described.

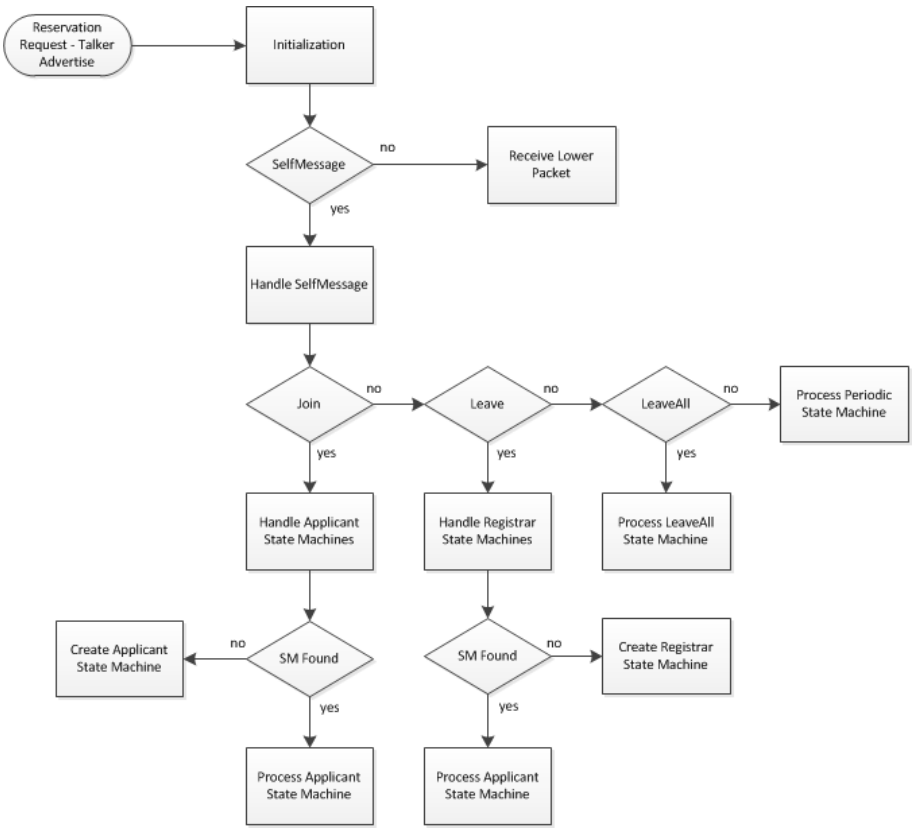


Figure 4.5: MAD processing flow

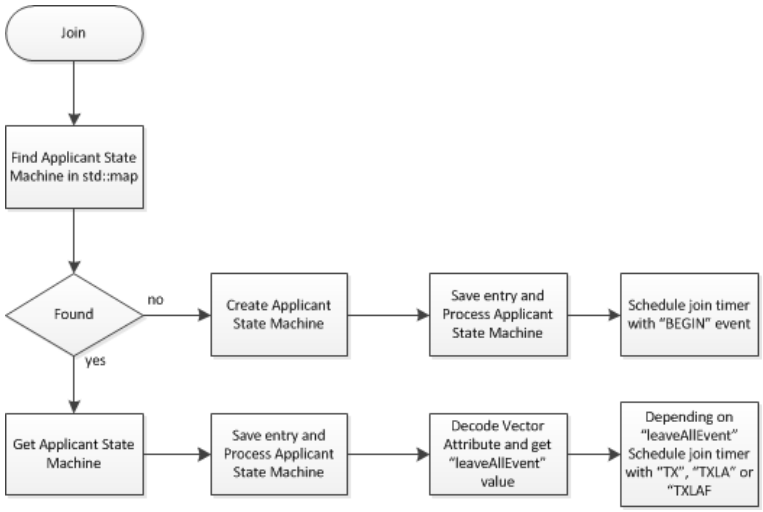


Figure 4.6: Applicant state machine processing flow

Each state machine behaviour present in [30] and in the tables of the appendix A of this document, is described by methods with prefixed by *process* in their names. All the state machine classes, with their own set of operating methods, were automatically generated by Omnet++. In a general way these methods execute protocol actions or change the state of

the machine according to the previous state the event triggered. These actions can perform, for example, the sending of *MAD_Join* or *MAD_Leave* messages to the MAP module or the start or end of specific timers.

The second situation that can occur in MAD is when it receives a message from an external module. When this is the case the message handler invokes the *receiveLowerMessage* and determines what kind of message is dealing with. Figure 4.7 depicts just that.

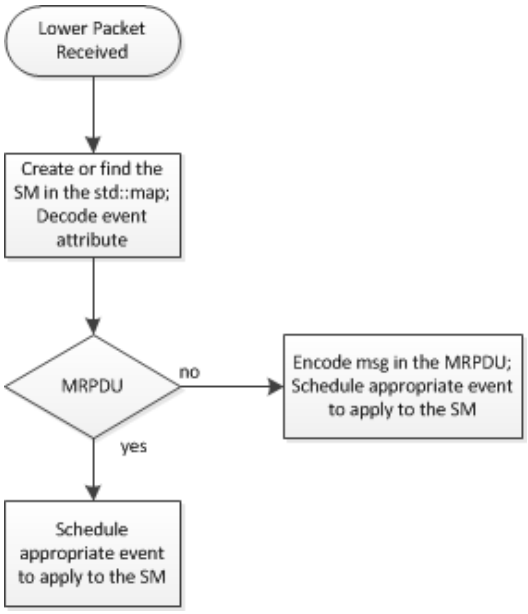


Figure 4.7: Receiving packet processing flow

If it receives a *MRPDU* packet the method determines its validation by checking the packet stream ID and MAC address. If the packet is not valid, then MAD ignores it. If it is valid then the *applicant state machine* and the *Registrar state machine* are either created or found in their respective maps. Now not only the *leaveAllEvent* is checked, but also the *attribute event* value present in the *three packed events* of the vector attribute. Depending on the value of the *attribute event* the state machines respective timers will be scheduled with the appropriate protocol event described in the table A.2 of the appendix A. The table 4.1 shows more specifically what event is scheduled when an *attribute event* is received.

Table 4.1: Scheduling of state machine events

Attribute Event (*threePackedEvent*):	**Scheduling of the following SM event:**
ATTRIBUTE_EVENT_NEW	rNew!
ATTRIBUTE_EVENT_JOIN_IN	rJoinIn!
ATTRIBUTE_EVENT_IN	rIn!
ATTRIBUTE_EVENT_JOIN_MT	rJoinMt!
ATTRIBUTE_EVENT_MT	rMt!
ATTRIBUTE_EVENT_LV	rLv!
Attribute Event (*vectorHeader*):	**Scheduling of the following SM event:**
LEAVE_ALL	rLA!

If the MAD receives either a *MAD_Join* or *MAD_Leave* messages, then the *declarationType* field is inspected and a new event is scheduled. In case of the *MAD_Join* the attribute is declared by the the schedule of the *New!* or *Join!* events. In case of the *MAD_Leave* the declaration is removed by the schedule of the *Leave!* event.

MRPDU Messages

In the MAD module, there are also some methods to encode MRPDU messages. These methods encode three types of messages per attribute type. Each method uses a STL container called *std::vector* to represent each MSRP message and each vector attribute. In case of a MRPDU message with a *talker tailed* attribute encoded, the *first value* field present in the *vector attribute* contains all its fields, while a *talker advertise* contains all except for the *failure information* field. Both these two messages do not contain the *fourPackedEvent* field present in the *vector attribute*, however it is present in a MRPDU message with the *listener* attribute present.

4.4.2 MADRelay

MADRelay was a module developed to redirect each MSRP message to the their respective MAD module, so that they may be processed by its respective state machines. This module is connected to three different kind of modules, the MAP, the MAD and MSRP Encap. As such the NED file that describes this module structure must have as many input and output gates per interface as the number of switch ports. The name of the gates connected to the MAD, MAP and Ether Encap modules are called respectively *madRelayUpperLayerIn[]* and *madRelayUpperLayerOut[]*, *madRelayMapIn[]* and *madRelayMapOut[]*, *madRelayLowerLayerIn[]* and *madRelayLowerLayerOut[]*.

When a message arrives at MADRelay the *handleMessage* method must check from which interface it came. If the message arrived from *madRelayLowerLayerIn*, then the message received was an MRPDU packet and so it should be dispatched to its respective MAD module. The input port is updated into a table (std::map) with the message source MAC address as its key. A copy of the message is made and the *handleMads* method is invoked. If the message arrives from the *madRelayUpperLayerIn* gate, then the message is checked for its kind. If it is a *MAD_Join* or a *MAD_Leave* message then the output port is obtained in the address table and the message is redirected to the MAP module for further validation of the reservation request. If the output port does not exist in the table, then the message is broadcasted to all ports in the switch except for the one it came originally. In case an MRPDU packet arrives, then the MRPDU is sent to the lower module Ether Encap. Finally, if either a *MAD_Join* or a *MAD_Leave* messages arrives in the *madRelayMapIn* gate, then the *handleMads* method is invoked and the message is redirected to its respective MAD module.

It is in the *handleMads* method that Omnet++ knows which MAD module to call or

to create, depending if it exists or not. The solution reached is very similar to the one implemented by MADs to handle its state machines. If a MAD module doesn't exists then the module is created or saved in a std::map with its content being a *cModule* Omnet++ class and the number of the switch port as its key. As such *MADRelay* is also responsible of handling dynamically created MAD modules as shown in Figure 4.8.

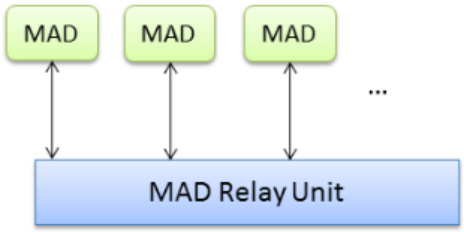


Figure 4.8: MADs dynamically created.

4.4.3 MAP

This module has as its main job the propagation of vector attributes from one MAD to another. It also has the role of deciding whether a certain switch port has sufficient resource to support incoming traffic or not. In this implementation this module does not enable the determination of resources present in the switch or bandwidth available. Therefore it is a very simple module, which propagates every message without any bandwidth constraints, from an ingress gate interface to an egress gate with the same index.

When the *handleMessage* method is invoked the message is checked if it is either a *MAD_Join* message or a *MAD_Leave* message. In either case the module input index obtained and the message are propagated to the output gate.

4.4.4 MSRP Queue

The MSRP queue contains several simple modules: as a classifier, a queue and a scheduler. The classifier is used to discern to which queue the incoming traffic is sent, by differentiating incoming traffic to its proper queue according to a certain degree of traffic priority. This module receives frames from an altered switch relay, adapted to simply dispatch the incoming encapsulated frame to the same output gate as the input one. This classifier was built from scratch to accommodate the following methods: the *initialize* and the *handleMessage*. The first method gets the number of queues parameter established by the user in the configuration file. The second one verifies if the incoming Ethernet II frame contains any MSRP messages or any other type of traffic. If it contains MRPDUs then the message is sent to a drop tail queue of higher priority called *msrpQueue*. Otherwise the traffic is sent to dynamically created queues. This queues are then saved in a std::map with a data type string configured by the user as its key.

The queue used uses the drop tail queue algorithm which discards incoming packets when the queue reaches its full capacity. Finally the scheduler uses a priority scheduler algorithm, which means that packets arriving at a lower gate will have higher priority then the ones arriving at higher gates, and so they will be dispatched first.

5

Analysis & Evaluation

This chapter presents experimental results obtained in OMNeT++. An explanation on how the experiments were prepared is first presented, followed by some results with the purpose of validating the system implemented. These experiments were performed in Ubuntu 12.04, Linux based operating system and in Omnet++ version 4.2.2.

5.1 Experimental Setup

As it was described in the last chapter, several modules were implemented in C++ so that the MSRP protocol could be supported. It is the main purpose of these experiments not only to perform architectural validation of the implemented system, but also an operational validation for the state machines present in the MAD module.

According to the MRP protocol, it is possible for one MRP participant to establish, maintain, and withdraw, its presence from the LAN, through the dissemination of attribute declarations and registrations. As such it is necessary the existence of exchanged messages between each participant, and a correct validation of state machines status.

The following experiments consist of a talker performing its *Talker Advertise* attribute advertisement to the network, and the resulting listener attachment through the *Listener Ready* message. After this the connection established is broken by forcing the the leave event in the exchanged messages along the path between the talker and the listener.

Because there isn't a MSRP application entity per MRP participant, most of the MSRP protocol implementation was push down to the MAD module. With that said, some protocol parameters and actions were hardcoded in the experiment, which would otherwise be managed by the MSRP application. Some examples of such parameters are the talker and listener MAC

address values, the session number or the state machine timer values. Nevertheless, these details do not invalidate the behaviour of the system.

The next set of code contains some initializations made in the configuration INI file used to run the simulation:

```
*.Talker.mad.startTime = 0.1s
*.Talker.mad.stopTime = 5s

*.Talker.mad.srcAddress = "Talker"
*.Talker.mad.destAddress =  "Listener"
*.Talker.mad.attributeType = 1
*.Talker.mad.reservation = true
*.Talker.mad.endStation = true

*.Switch.numberOfQueues = 1
**.Switch.mad[*].attributeType = 0
**.Switch.mad[*].endStation = false
**.Switch.msrpQoSQueue[*].dropTailQueue[*].frameCapacity = 10

*.Listener.mad.destAddress = "Talker"
*.Listener.mad.srcAddress = "Listener"
*.Listener.mad.attributeType = 3
*.Listener.mad.endStation = true
```

The values used to configure the state machine timers were the same values recommended by the standard and they are present in the *mad.ned* file as it is show below:

```
simple MAD
{
    parameters:
        double jointimerTimeout @unit(s) @prompt("JointimerTimeout time") = default(0.2s);
        double leavetimerTimeout @unit(s) @prompt("LeavetimerTimeout time") = default(0.8s);
        double leavealltimerTimeout @unit(s) @prompt("LeavealltimerTimeout time") = default(10s);
        double periodictimerTimeout @unit(s) @prompt("PeriodictimerTimeout time") = default(1s);
        ...
}
```

As for the delay time between each network element, the value was configured in its correspondent network ned file as depicted bellow:

```
network _1sw
{
...
    connections:
  Talker.ethg <--> cable {  delay = 1ms; } <--> Switch.ethg[0];
        Switch.ethg[1] <--> cable {  delay = 1ms; } <--> Listener.ethg;
}
```

It is important to note that the delay value of 1ms is a relative value to do the simulation. The processing time inside each network element was ignored.

5.2 First Scenario

In this scenario an end station is connected to a switch and this in turn is connected to another end station, as it is shown in figure 5.1. The end station on the left is the one that starts the message sequence, and for that it was named *Talker*.

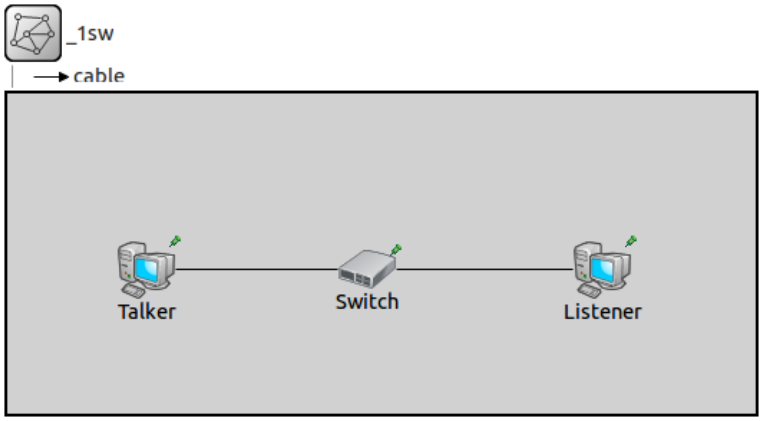


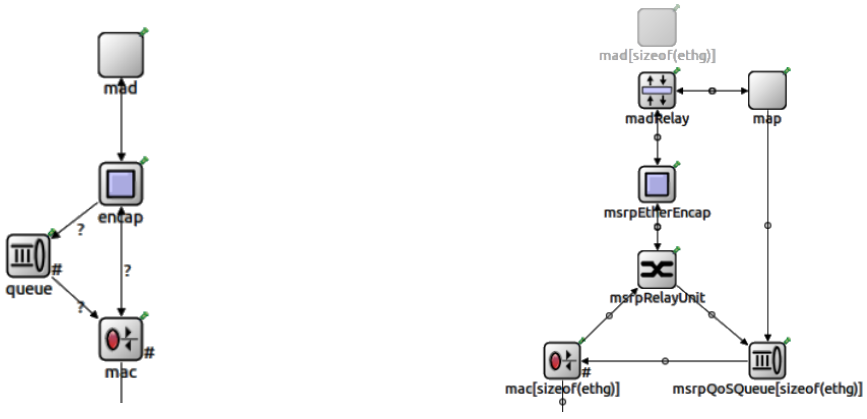
Figure 5.1: Network scenario with one switch

When the simulation starts all interfaces composing the modules are built and each module loads its configuration parameters. One of those parameters is a flag that triggers a joining message advertisement to the network at a time value of 1s plus the internal time value the simulator was on when the this event was scheduled. This message has a *talker advertise* attribute in it, to be declared and registered in all the MADs Applicant state machines. In turn the listener was configured to always respond with an *listener ready* attribute in the response message. After the talker notices the listener existence in the network a new signalling processing is started by the talker with the advertisement of a leaving message.

5.2.1 Message sequences

In this subsection all message sequences diagrams are presented based on the simulation results of the Appendix B of this document. Figure 5.2 represents both internal structures of each element (end station and switch), present in each message sequence diagram.

Figure 5.3 show a MRPDU being sent by the *MAD* module of the talker to the *encap* where it is encapsulated into an Ethernet II frame. The frame is then dispatched to the *mac* module to the switch. It is possible to see the delay of 1 ms configured initially. The *EndIFG* and the *End Transmission* are messages issued to the same module that issued them. This first one represents the inter frame gap period issued when the channel is in idle state. The second notifies the *mac* module that the frame transmission has finished. The message is then



(a) End station internal structure

(b) Switch internal structure

Figure 5.2: Network elements

dispatched to the listener and all the reverse operations, such as the reception of the frame, the decapsulation of the frame and the reception of the MRPDU by the listener's MAD.

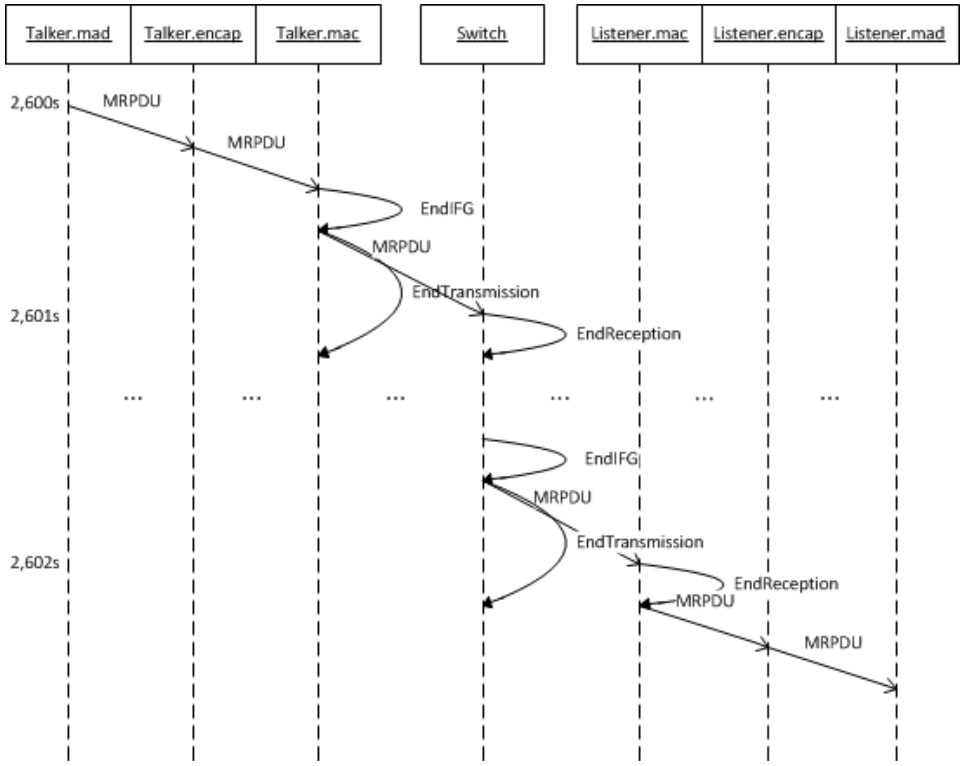


Figure 5.3: Talker advertisement(a)

Figure 5.4 depicts the path travelled, by the same joining message inside the switch. In this diagram the MRPDU message travels from the first port of the switch all the way up to the *madRelay* module. This module has a vector of gates as its input and output interfaces. The message arrives at the first element of this vector, and the *madRelay* learns the source MAC address by updating a table with the MAC address corresponding to its input port.

The message is then relayed to a dynamically created MAD module. The MAD will update its state machines by declaring and registering the attribute present in the MRPDU. After that the attribute is propagated to the MAP module via the *madRelay*, according to the event triggered by the arriving message. The attribute is finally propagated to the MAD of the second port of the switch and then the MRPDU message makes the inverse path to the switch output port.

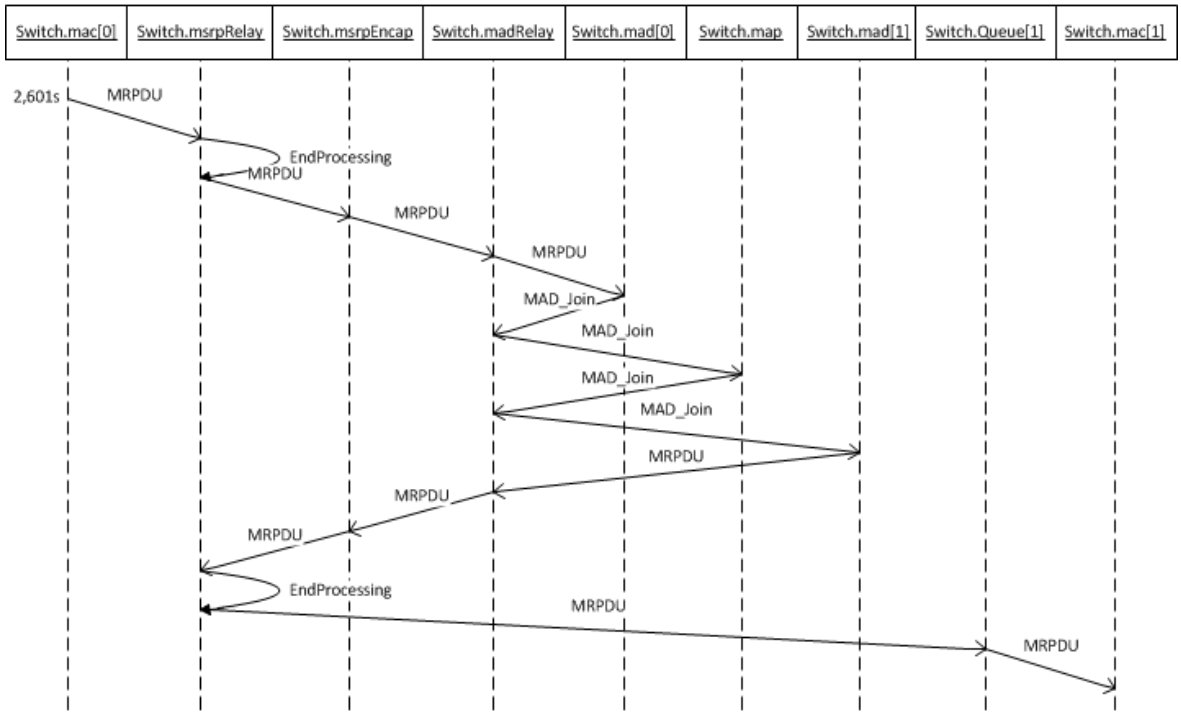


Figure 5.4: Talker advertisement(b)

Figures 5.5 and 5.6 represents the listener response to the talker, with the joining event. All the steps performed by messages are identical to the previews ones. The MRPDU leaves the listener with a *Listener Ready* as its attribute type value, and arrives at the second port of the switch. The message then is sent to the talker by the other port.

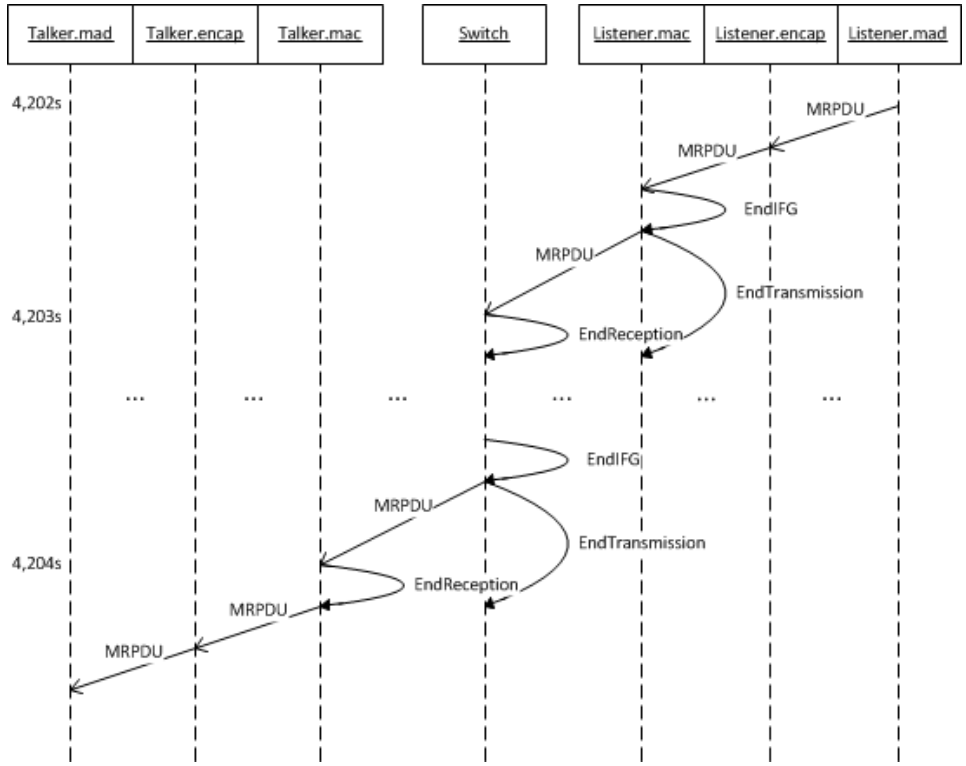


Figure 5.5: Listener attachment(a)

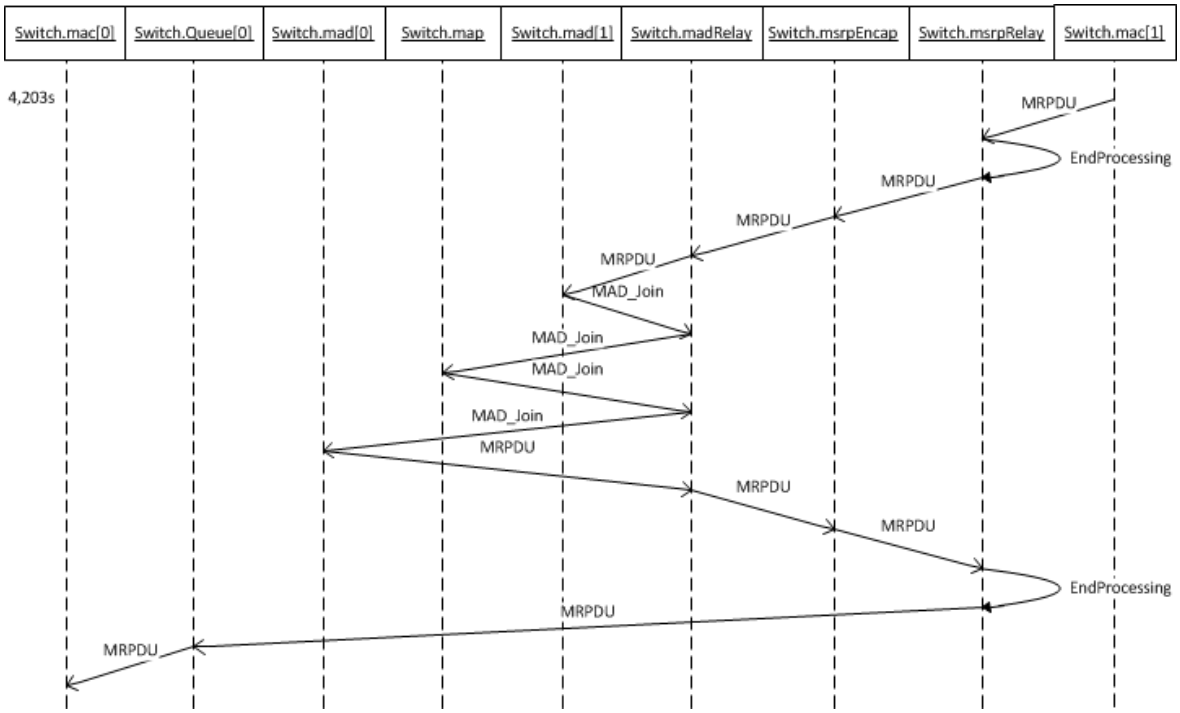


Figure 5.6: Listener attachment(b)

At this point both the talker and listener are attached to the network and the talker could start to transmitting streams to the listener.

The next set of message sequence diagrams represent the leaving of both the talker

and listener from the network. With same operations performed previously except now the *MAD_Leave* message issued by the MAD is triggered by a leave event instead of a joining one.

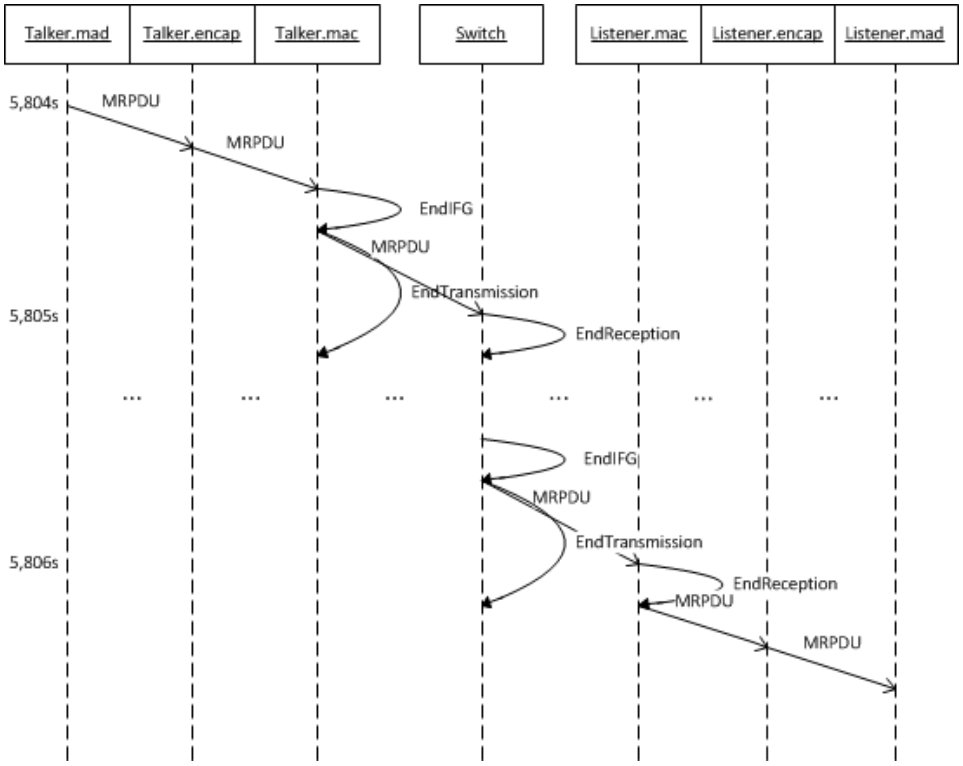


Figure 5.7: Talker leaving(a)

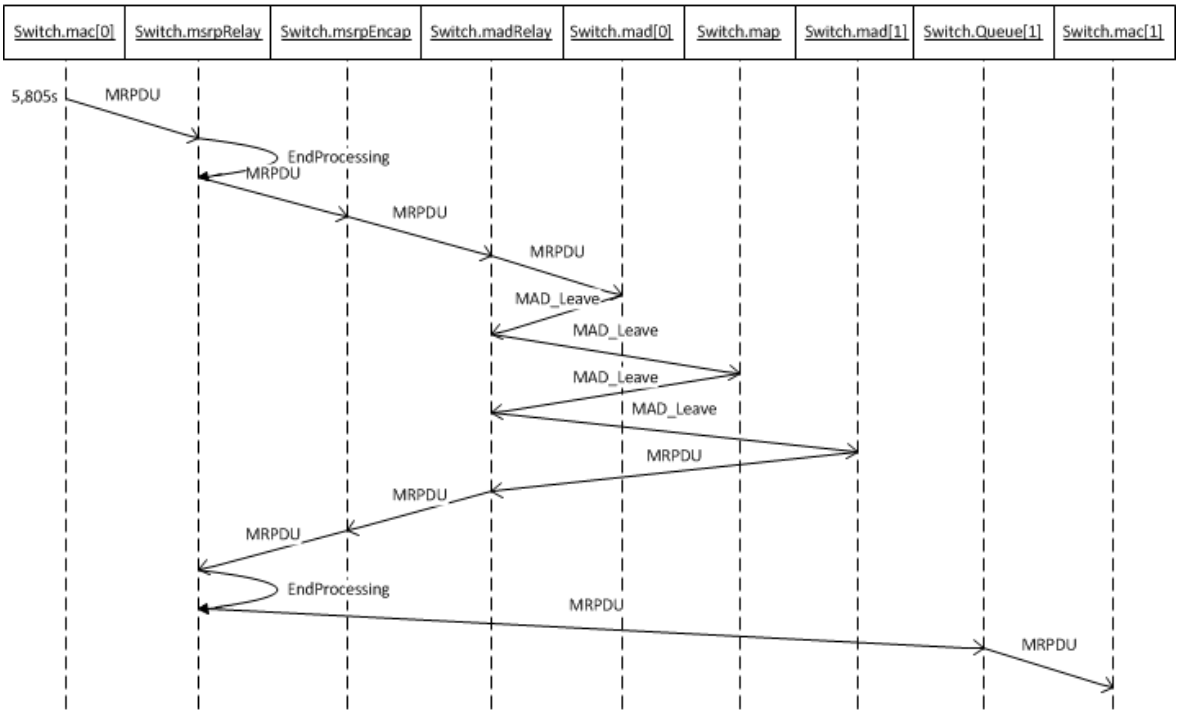


Figure 5.8: Talker leaving(b)

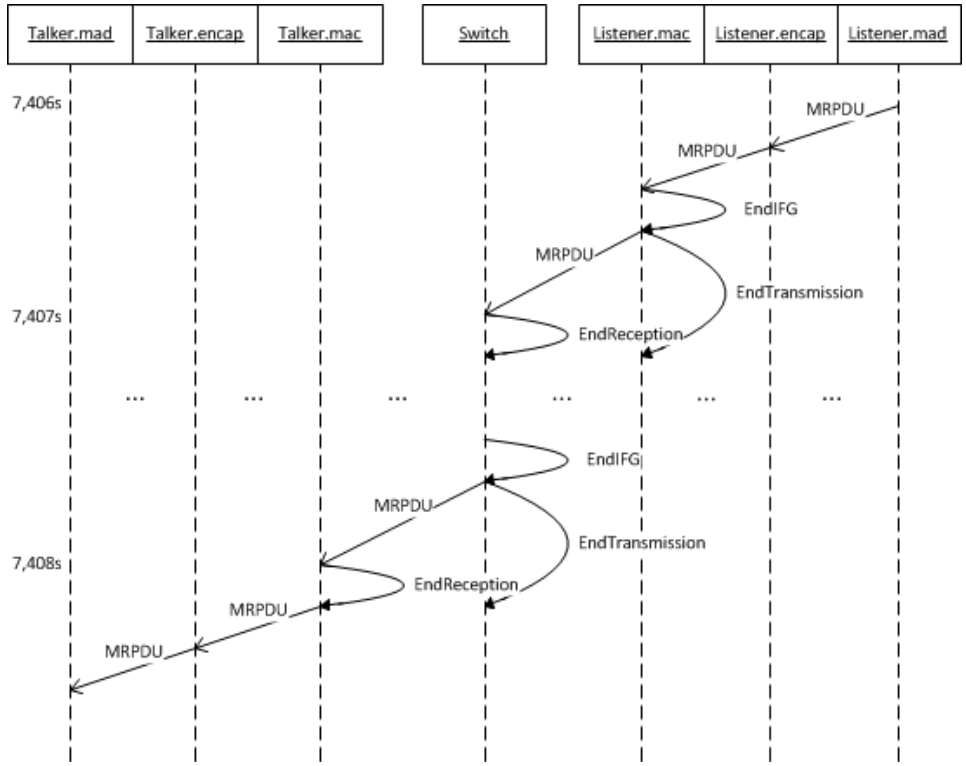


Figure 5.9: Listener leaving(a)

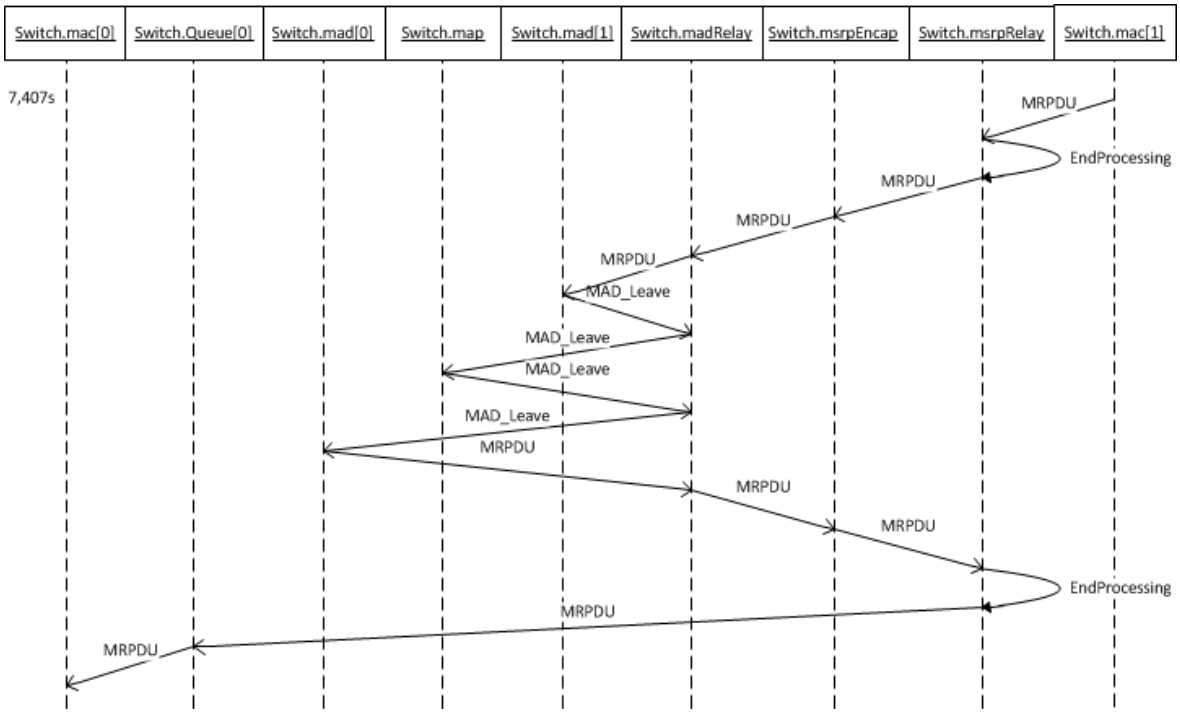


Figure 5.10: Listener leaving(b)

5.2.2 State machines validation

In this subsection, some results of the operation of the state machines inside the MAD modules are described. The purpose is to highlight both the Applicant and Registrar state machines operation during the simulation. Results can be collected in two ways: directly from C++ code, using the simulation library or the definition of signals inside the code as shown bellow:

```
simsignal_t MAD::stateApplicantSignal = SIMSIGNAL_NULL;
simsignal_t MAD::stateRegistrarSignal = SIMSIGNAL_NULL;

  int stateA = a_entry->applicantSt->getState();
  int stateR = r_entry->registrarSt->getState();
  emit(stateApplicantSignal, stateA);
  emit(stateRegistrarSignal, stateR);
```

The definition of these signals is important, because it allows the recording of results in a much more easy manner, without requiring other heavy tools or continuous tweaking of the simulation model.

These results were recorded in the *MAD.ned* file, with the following instructions:

```
simple MAD
{
    parameters:
     ...
        @signal[statApp](type=cMessage);
        @signal[statReg](type=cMessage);
        @statistic[statApp](title="Applicant"; source=statApp; record=vector; interpolationmode=none);
        @statistic[statReg](title="Registrar"; source=statReg; record=vector; interpolationmode=none);
        ...
}
```

The state values were recorded as integers. In order to make it easier to analyse the machine state symbols, the Table 5.1 presents the mapping of each state integer values into its correspondent symbol, for both the Applicant and Registrar state machines. The meaning of these values are depicted in Appendix A.

Figure 5.11 represents both the applicant and registrar state machine changes applied to the talker, during the lifetime of the simulation.

The talker applicant state machine goes through the following states: VO, VP, AA, LA and VO. The Applicant starts as a very anxious observer when the state machine is initialized, which means the applicant is not declaring any attribute, and has not received a *JoinIn* message since the state machine was initialized. Then it changes to a very anxious passive state, making an attribute declaration, for the first time since the state machine was

Table 5.1: Applicant and Registrar states

Applicant Symbol	**Applicant State**	**Applicant Symbol**	**Applicant State**
VO	1	LA	7
VP	2	AO	8
VN	3	QO	9
AN	4	AP	10
AA	5	QP	11
QA	6	LO	12
Registrar Symbol	**Registrar State**	**Registrar Symbol**	**Registrar State**
IN	1	MT	3
LV	2	-	-

Item#	Event#	Time	Value
0	0	0	1.0
1	1	0.3	1.0
2	7	1.3	2.0
3	11	2.6	5.0
4	87	4.504002688	7.0
5	96	5.804002688	1.0
6	177	7.708005376	1.0

(a) Applicant states

Item#	Event#	Time	Value
0	0	0	3.0
1	3	0.9	3.0
2	8	1.3	1.0
3	88	4.504002688	2.0
4	93	5.304002688	2.0
5	178	7.708005376	3.0

(b) Registrar states

Figure 5.11: Talker state machines

initialized, or since it receive a *leave* or *leaveAll* attribute event. It is followed by an anxious active state, where the applicant is declaring an attribute and a transmit opportunity of a join message occurs. From there the MRPDU is dispatched and the talker attached to the network. When it is time to leave the state changes to an leaving active state, triggered by the expiring of the *leave timer* of the registrar state machine. Finally the state changes back to its original state.

In the registrar case, the states covered are as follows: MT, IN, LV and MT. The state machine is started in the *not registered* state, and changes to *registered* after the attribute is declared. When it is time for the talker to leave, then the *leave timer* is scheduled and the state is changed to a *leaving* state. Finally the registrar returns to its original state.

In both Figure 5.12 and Figure 5.13, it is depicted all applicant and registrar state machine changes applied to each switch port.

As it can be seen both state machine transitions are the same in the two ports. The applicant is started and then changes to a an anxious observer triggered by the receipt of the *JoinIn* event. In the end the state machine changes to a leaving observer state. The registrar performs the same state changes as the the talker and listener examples.

Finally, Figure 5.14 illustrates both the applicant and registrar state machine changes applied to the listener.

Item#	Event#	Time	Value
0	21	2.601000672	1.0
1	41	2.801000672	8.0
2	85	4.503002016	8.0
3	126	6.00500336	12.0
4	175	7.707004704	12.0

(a) Applicant states

Item#	Event#	Time	Value
0	21	2.601000672	3.0
1	42	2.801000672	1.0
2	86	4.503002016	1.0
3	127	6.00500336	2.0
4	136	6.80500336	2.0
5	176	7.707004704	2.0

(b) Registrar states

Figure 5.12: Switch[0] state machines

Item#	Event#	Time	Value
0	25	2.601000672	1.0
1	43	2.901000672	8.0
2	83	4.403002016	8.0
3	130	6.10500336	12.0
4	171	7.607004704	12.0

(a) Applicant states

Item#	Event#	Time	Value
0	25	2.601000672	3.0
1	44	2.901000672	1.0
2	84	4.403002016	1.0
3	131	6.10500336	2.0
4	137	6.90500336	2.0
5	172	7.607004704	2.0

(b) Registrar states

Figure 5.13: Switch[1] state machines

Item#	Event#	Time	Value
0	0	0	1.0
1	2	0.3	1.0
2	45	2.902001344	2.0
3	53	4.202001344	5.0
4	132	6.106004032	7.0
5	141	7.406004032	1.0

(a) Applicant states

Item#	Event#	Time	Value
0	0	0	3.0
1	4	0.9	3.0
2	46	2.902001344	1.0
3	133	6.106004032	2.0
4	138	6.906004032	2.0

(b) Registrar states

Figure 5.14: Listener state machines

As it can be seen here the changes are in everything similar to the ones verified in the talker, with both the declaration and registration of the arriving attributes in the *joining* and *leaving* messages.

5.3 Second Scenario

In this scenario a sequence of three switches were considered, between each end station (see Figure 5.15). The delay between each network element was configured with 1 second, in order to allow a better observation of the message sequence displayed in Figure B.5 and Figure B.6 of Appendix B. This simulation was scheduled to run during 35 seconds.

All message sequences are equal to the ones obtained in a network with only one switch. These message sequences are displayed by the results of the appendix B.

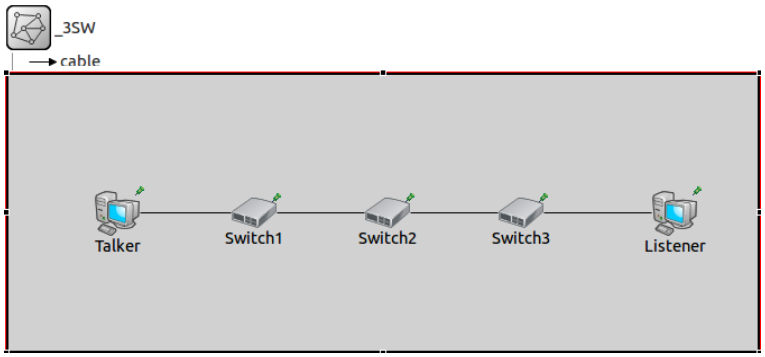


Figure 5.15: Network scenario with three switches

5.3.1 State machines validation

Just as it was done in the previews scenario, it was possible to record the state machine changes recurring to the help of table 5.1. In figures 5.16 and 5.17 it is shown as an example, both state machines of the talker and one of the third switch ports.

Item#	Event#	Time	Value
0	0	0	1.0
1	1	0.3	1.0
2	7	1.3	2.0
3	11	2.6	5.0
4	245	12.500005376	7.0
5	264	13.800005376	1.0
6	532	23.700010752	1.0

(a) Applicant states

Item#	Event#	Time	Value
0	0	0	3.0
1	3	0.9	3.0
2	8	1.3	1.0
3	246	12.500005376	2.0
4	255	13.300005376	2.0
5	533	23.700010752	3.0

(b) Registrar states

Figure 5.16: Talker state machines in a network with 3 switches

Item#	Event#	Time	Value
0	83	5.600002016	1.0
1	101	5.900002016	8.0
2	161	9.40000336	8.0
3	376	17.100007392	12.0
4	439	20.600008736	12.0

(a) Applicant states

Item#	Event#	Time	Value
0	83	5.600002016	3.0
1	102	5.900002016	1.0
2	162	9.40000336	1.0
3	377	17.100007392	2.0
4	389	17.900007392	2.0
5	440	20.600008736	2.0

(b) Registrar states

Figure 5.17: Second port of the third switch state machines

The resulting state changes are as expected the same as those previously recorded with one switch. This means that the model is capable of declaring and registering attributes in each port along the path between two MRP participants. At this point the network could support streams of data from the talker to the listener.

6

Conclusions

In recent years automotive industry and its main OEMs have undergone major changes in the way they design vehicles internal networks. The demand of lower cost production while maintaining the same efficiency is leading automotive manufacturers to support more general technologies like Ethernet, instead of more specific ones used until. In this work several automotive network technologies were described and their specific domain of application was emphasized. Then an overall description of the Ethernet LAN technology was made, with the main focus on the automotive QoS Ethernet requirements and resource reservation strategies.

In order to achieve the same or better levels of QoS for a vast array of applications, several improvements were needed. One of these improvements is the usage of a resource reservation signalling mechanism named SRP, specially developed for bridged networks and also described in this work. This mechanism is based on Audio/Video international standards like the Multiple Registration Protocol and the Stream Reservation Protocol. These protocols not only allow the reservation but also the registration and propagation of attributes through some complex state machines. This mechanism is ideal to be used in an automotive Ethernet environment because it can assure a relative high levels of QoS to several applications at a low cost, without compromising the vehicle vital functions.

In the development stages of this work a general architecture was conceived to support these protocols. Several modules were built for that purpose giving special focus to the implementation of the protocol state machines.

Some experiments were realized to ensure the correct exchange of protocol messages through out the modules and state machines operation.

6.1 Future Work

All the work realized during this dissertation focuses on the MRP and SRP protocols. Its complete implementation would allow the complete support of the resource reservation signalling mechanism. In order to acomplish that several other modules and functionalities would need to be developed, such as:

- A network redundancy mechanism with no recuperation time, like TRILL or SPB;
- An MSRP application module to allow the management of all SRP protocol parameters and its full integration with the MAD module to allow complete control over the state machines;
- Development of an admission control mechanism in the MAP module to allow the switches to manage its own resources;
- Use in combination with the full resource mechanism of the SRP, a traffic conditioning strategy for optimization purposes.

7

Glossary

ABS	Anti-lock Braking System
API	Application Programming Interface
AVB	Audio/Video Bridging
CAN	Controller Area Network
CIR	Committed Information Rate
COTS	Commercial Off-The-Shelf
CRC	Cyclic Redundancy Check
CSMA/BA	Carrier Sense Multiple Access/Bitwise Arbitration
CSMA/CD	Carrier Sense Media Access / Collision Detection
ECM	Engine Control Module
ECUs	Electronic Control Units
EGR	Exhaust Gas Recirculation
EMI	Electromagnetic Interference
EPS	Electric Power Steering
ESC	Electronic Stability Control
ESP	Electronic Stability Program
FCS	Frame Check Sequence
FDB	Forwarding Database
GPS	Global Positioning System
HSR	High Availability Seamless Redundancy
ISDN	Integrated Services Digital Network
LAN	Local Area Network
LIN	Local Interconnect Network
LLC	Logical Link Control

MAC	Media Access Control
MAD	MPR Attribute Declaration
MAP	MPR Attribute Propagation
MMRP	Multiple MAC Registrarion Protocol
MOST	Media Oriented Systems Transport
MRP	Multiple Registration Protocol
MRPD	Media Redundancy Protocol Duplication
MRPDU	Multiple Registration Protocol Data Unit
MSRP	Multiple Stream Reservation Protocol
MVRP	Multiple VLAN Registration Protocol
NRZ	Non-Return to Zero
OEMs	Original Equipment Manufacturers
OSI	Open Systems Interconnection
PCM	Powertrain Control Module
POF	Polymer Optical Fiber
PRP	Parallel Redundancy Protocol
QoS	Quality of Service
RSTP	Redundancy Spanning Tree Protocol
RSVP	Resource Reservation Protocol
SOF	Start Of Frame
SPB	Shortest Path Bridging
SRP	Stream Reservation Protocol
TCI	Tag Control Information
TCP/IP	Transmission Control Protocol/Internet Protocol
TDMA	Time-Division Multiple Access
TPID	Tag Protocol Identifier
TRILL	TRansparent Interconnection of Lots of Links
VLAN	Virtual Local Area Network

References

[1] Z. Equity Research. (Feb. 2013). Auto industry outlook and review, [Online]. Available: http://www.nasdaq.com/article/auto-industry-outlook-and-review-feb-2013-industry-outlook-cm216470.

[2] J. Sommer, S. Gunreben, F. Feller, M. Kohn, A. Mifdaoui, D. Saß, and J. Scharf, «Ethernet–a survey on its fields of application», *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 2, pp. 263–284, 2010.

[3] H. Zinner, J. Noebauer, T. Gallner, J. Seitz, and T. Waas, «Application and realization of gateways between conventional automotive and ip/ethernet-based networks», in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, IEEE, 2011, pp. 1–6.

[4] R. N. Charette, «This car runs on code», *IEEE Spectrum*, vol. 46, no. 3, p. 3, 2009.

[5] V. Kaczmarczyk, «Ethernet powerlink simulator»,

[6] P. W. Group *et al.*, «Profinet, architecture description and specification», *Norma Profinet elaborada pela Associação PROFIBUS Internacional, Version*, vol. 1, 2001.

[7] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, «An extension of the omnet++ inet framework for simulating real-time ethernet with high accuracy», in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011, pp. 375–382.

[8] (2012). Audio/video bridging task group, [Online]. Available: http://www.ieee802.org/1/pages/avbridges.html.

[9] (). Auto @ONLINE, [Online]. Available: http://auto.howstuffworks.com/.

[10] R. Bosch, «Can specification version 2.0», *Robert Bosch Gmbh, Stuttgart*, 1991.

[11] F. Consortium *et al.*, «Flexray communications system protocol specification version 2.1», *Request online: http://-www. flexray. com/specification request. php*, 2005.

[12] L. Consortium, L. S. Package, *et al.*, «Technical report version 2.1», *LIN Consortium, November*, 2006.

[13] M. Cooperation, *Most specification rev 3.0*, 2008.

[14] G. Leen and D. Heffernan, «Expanding automotive electronic systems», *Computer*, vol. 35, no. 1, pp. 88–93, 2002.

[15] C. in Automation (CiA). (). Can history, [Online]. Available: `http://www.can-cia.org/index.php?id=systemdesign-can-history`.

[16] J. Rufino, «An overview of the controller area network», in *Proceedings of the CiA Forum CAN for Newcomers*, 1997.

[17] D. M. T. Consulting. (). Flexray technology introduction, [Online]. Available: `http://www.millinger-consulting.com/dmtc/index.php?option=com_content&view=article&id=3:flexray-technology&catid=1:dmtc-english&Itemid=3`.

[18] J. Ploeg. (). Lin, [Online]. Available: `http://www.specifications.nl/lin/lin_UK.php?requestedpage=1`.

[19] M. cooperation. (). Most, [Online]. Available: `http://www.mostcooperation.com`.

[20] IEEE. (). Ieee 802.3 ethernet working group, [Online]. Available: `http://www.ieee802.org/3/`.

[21] I. C. Society, «Media access control (mac) bridges and virtual bridged local area networks», *IEEE Std 802.1Q*, 2011.

[22] O. Kleineberg, P Frohlich, and D. Heffernan, «Fault-tolerant ethernet networks with audio and video bridging», in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, IEEE, 2011, pp. 1–8.

[23] D. E. H. T. Radia Perlman Intel Labs. (Sep. 2011). Introduction to trill @ONLINE, [Online]. Available: `http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_14-3/143_trill.html#reference1`.

[24] «Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks–amendment 20: shortest path bridging», *IEEE Std 802.1aq-2012 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, and IEEE Std 802.1Qbf-2011)*, pp. 1–340, 2012. DOI: `10.1109/IEEESTD.2012.6231597`.

[25] H.-T. Lim, L Volker, and D. Herrscher, «Challenges in a future ip/ethernet-based in-car network for real-time applications», in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, IEEE, 2011, pp. 7–12.

[26] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, «Rsvp: a new resource reservation protocol», *Network, IEEE*, vol. 7, no. 5, pp. 8–18, 1993.

[27] D. Karger, C. Stein, and J. Wein, «Scheduling algorithms», in *Algorithms and theory of computation handbook*, Chapman & Hall/CRC, 2010, pp. 20–20.

[28] S. Jamin, S. Shenker, L. Zhang, and D. D. Clark, «An admission control algorithm for predictive real-time service», in *Network and Operating System Support for Digital Audio and Video*, Springer, 1993, pp. 347–356.

[29] K. S. Reddy and L. C. Reddy, «A survey on congestion control mechanisms in high speed networks», *IJCSNS-International Journal of Computer Science and Network Security*, vol. 8, no. 1, pp. 187–195, 2008.

[30] I. C. Society, «Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 7: multiple registration protocol», *IEEE Std 802.1ak*, 2007.

[31] H.-T. Lim, D. Herrscher, M. J. Waltl, and F. Chaari, «Performance analysis of the ieee 802.1 ethernet audio/video bridging standard», in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 27–36.

[32] I. C. Society, «Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 14: stream reservation protocol (srp)», *IEEE Std 802.1Qat*, 2010.

[33] (). Ieee 802.1qav - forwarding and queueing enhancements @ONLINE, [Online]. Available: `http://www.ieee802.org/1/pages/802.1av.html`.

[34] (). Ieee 802.1as - timing and synchronization @ONLINE, [Online]. Available: `http://www.ieee802.org/1/pages/802.1as.html`.

[35] «Standard for layer 2 transport protocol for time sensitive applications in bridged local area networks», *IEEE Std 1722, year=2011, publisher=*,

[36] (). Omnet++ home page @ONLINE, [Online]. Available: `http://http://www.omnetpp.org/`.

[37] A. Varga and R. Hornig, «An overview of the omnet++ simulation environment», in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[38] A. Varga *et al.*, «The omnet++ discrete event simulation system», in *Proceedings of the European Simulation Multiconference (ESM'2001)*, sn, vol. 9, 2001, p. 185.

[39] L. M. Feeney, «Managing cross layer information in omnet++ simulations», 2009.

Appendix - A

This appendix contains all the state machines of the MRP protocol and the meaning of their values.

Table A.1: MRP protocol triggered events [30]

Protocol Events	**Description**
Begin!	Initialize state machine
New!	A new declaration
Join!	Declaration without signaling new registration
Lv!	Withdraw a declaration
tx!	Transmission opportunity without a LeaveAll
txLA!	Transmission opportunity with a LeaveAll
txLAF!	Transmission opportunity with a LeaveAll, and with no room (Full)
rNew!	Receive New message
rJoinIn!	Receive JoinIn message
rIn!	Receive In message
rJoinMt!	Receive JoinEmpty message
rMt!	Receive Empty message
rLv!	Receive Leave message
rLA!	Receive a LeaveAll message
Flush!	Port role changes from Root Port or Alternate Port to Designated Port
Re-Declare!	Port role changes from Designated to Root Port orAlternate Port
periodic!	A periodic transmission event occurs
leavetimer!	leavetimer has expired
leavealltimer!	leavealltimer has expired
periodictimer!	periodictimer has expired

Table A.2: Applicant States [30]

State	**Description**
VO-Very anxious Observer	The applicant is not declaring the attribute, and has not received a JoinIn message since the state machine was initialized, or since last receiving a Leave or LeaveAll.
VP-Very anxious Passive	The applicant is declaring the attribute, but has neither sent a Join nor received a JoinIn since the state machine was initialized, or since last receiving a LeaveAll or Leave.
VN-Very anxious New	The applicant is declaring the attribute, but has not sent a message since receiving a MAD Join request for a new declaration.
AN-Anxious New	The applicant is declaring the attribute, and has sent a single New message since receiving the MAD Join request for the new declaration.
AA-Anxious Active	The applicant is declaring the attribute, and has sent a Join message, since the last Leave or LeaveAll, but either has not received another JoinIn or In, or has received a subsequent message specifying an Empty registrar state.
QA-Quiet Active	The applicant is declaring the attribute and has sent at least one of the required Join or New messages since the last Leave or LeaveAll, has seen or sent the other, and has received no subsequent messages specifying an Empty registrar state.
LA-Leaving Active	The applicant has sent a Join or New message since last receipt of a Leave or LeaveAll, but has subsequently received a MAD Leave request and has not yet sent a Leave message.
AO-Anxious Observer	The applicant is not declaring the attribute, but has received a JoinIn since last receiving a Leave or LeaveAll.
QO-Quiet Observer	The applicant is not declaring the attribute, but has received two JoinIns since last receiving a Leave or LeaveAll, and at least one since last receiving a message specifying an Empty registrar state.
AP-Anxious Passive	The applicant is declaring the attribute, and has not sent a Join or a New since last receiving a Leave or a LeaveAll but has received messages as for the Anxious Observer state.
QP-Quiet Passive	The applicant is declaring the attribute, and has not sent a Join or a New since last receiving a Leave or a LeaveAll but has received messages as for the Quiet Observer state.
LO-Leaving Observer	The applicant is not declaring the attribute, and has received a Leave or LeaveAll message.

Table A.3: MRP protocol actions [30]

Protocol Actions	**Description**
New	send a New indication to MAP and the MRP application
Join	send a Join indication to MAP and the MRP application
Lv	send a Lv indication to MAP and the MRP application
sN	send a New message
sJ	send a JoinIn or JoinMT message
sL	send a Lv message
s	send an In or an Empty message
[s]	send an In or an Empty message, if required for optimization of the encoding
[sL]	send a Lv message, if required for optimization of the encoding
[sJ]	send a Join message, if required for optimization of the encoding
sLA	send a Leave All message
periodic	Periodic transmission event
leavetimer	Leave period timer
leavealltimer	Leave All period timer
periodictimer	Periodic Transmission timer
-x-	napplicable event/state combination. No action or state transition occurs in this case.

Table A.4: Registrar States [30]

State	**Description**
IN	Registered
LV	Previously registered, but now being timed out
MT	Not registered

Table A.5: Recommended timer values [30]

Timer	**Values(centiseconds)**
JoinTime	20
LeaveTime	60-100
LeaveAllTime	1000
PeriodicTime	100

EVENT	STATE											
	VO[11]	**VP**[6]	**VN**[6]	**AN**[6]	**AA**[6]	**QA**	**LA**[6]	**AO**[3,11]	**QO**[3,11]	**AP**[3,6]	**QP**[3]	**LO**[6]
Begin!	—	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO	VO
New!	VN	VN	—	—	VN	VN	VN	VN	VN	VN	VN	VN
Join!	VP	—	—	—	—	—	AA	AP	QP	—	—	VP
Lv!	—	VO	LA	LA	LA	LA	—	—	—	AO	QO	—
rNew!	—	—	—	—	—	—	—	—	—	—	—	—
rJoinIn!	AO[4]	AP[4]	—	—	QA	—	—	QO	—	QP	—	—
rIn!	—	—	—	—	QA[5]	—	—	—	—	—	—	—
rJoinMt! ‖ rMt!	—	—	—	—	—	AA	—	—	AO	—	AP	VO
rLv! ‖ rLA! ‖ Re-declare!	LO[1]	—	—	VN	VP[9]	VP[9]	—[10]	LO[1]	LO[1]	VP	VP	—
periodic!	—	—	—	—	—	AA	—	—	—	—	AP	—
tx![7]	[s] —	sJ AA	sN AN	sN QA[8]	sJ QA	[sJ] —	sL VO	[s] —	[s] —	sJ QA	[s] —	s VO
txLA![2]	[s] LO	s AA	sN AN	sN QA	sJ QA	sJ —	[s] LO	[s] LO	[s] LO	sJ QA	sJ QA	[s] —
txLAF![2]	LO	VP	VN	VN	VP	VP	LO	LO	LO	VP	VP	—

Notes to the table:

[1] Applicant-Only participants exclude the LO state, and transition to VO.

[2] These events do not occur for Applicant-Only participants.

[3] Point-to-point subset participants exclude the AO, QO, AP, and QP states.

[4] Ignored (no transition) if point-to-point subset or if operPointToPointMAC is TRUE.

[5] Ignored (no transition) if operPointToPointMAC is FALSE. See MRP Design Notes below.

[6] Request opportunity to transmit on entry to VN, AN, AA, LA, VP, AP, and LO states.

[7] If the MRPDU is full and cannot convey a required message there is no change of state and an additional transmit opportunity is requested if that has not been done already.

[8] QA if the Registrar is IN, and AA otherwise. See MRP Design Notes below.

MRP design notes:

[9] On shared media the receipt of In does not confirm registration by all Participants, and the In could have been sent by an Applicant-Only participant.

[8] Since New messages do not convey registrar state, a Leave could have been received without an Empty or JoinEmpty prompt being sent, the transition to AA guards against loss of that Leave by another Applicant.

[9] The design accepts a small possibility of a continued registration (after rLv! if a Lv! occurs before a further Join is sent) in return for not accumulating many Active participants when Join!s and Lv!s are frequent. rLv! processing is deliberately not optimized for point-to-point.

[10] If a Leave has been received, the Registrar for the transmitting participant is very probably IN, as this Applicant has not yet sent a Leave, so the pending Leave is required. The small savings from avoiding transmission of Leaves pending on receipt of LeaveAlls does not merit distinguishing the rLv! and rLA! cases.

[11] The VO, AO, and QO states represent states where the attribute is neither being declared by the Participant nor being registered by any other station on the LAN. In implementations where dynamic creation and discarding of state machines is desirable, the state machine can be discarded when in any of these states, pending a future requirement to declare or register that attribute value.

Figure A.1: Applicant state machine

		STATE		
		IN	**LV**	**MT**
EVENT	**Begin!**	MT	MT	MT
	rNew!	New IN	New Stop leavetimer IN	New IN
	rJoinIn! || rJoinMt!	IN	Stop leavetimer IN	Join IN
	rLv! || rLA! || txLA! || Re-declare!	Start leavetimer LV	-x-	-x-
	Flush!	MT	Lv MT	MT
	leavetimer!	-x-	Lv MT	MT

Figure A.2: Registrar state machine

		STATE	
		Active[a]	**Passive**
EVENT	**Begin!**	Start leavealltimer Passive	Start leavealltimer Passive
	tx!	sLA Passive	-x-
	rLA!	Start leavealltimer Passive	Start leavealltimer Passive
	leavealltimer!	Start leavealltimer Active	Start leavealltimer Active

[a]Request opportunity to transmit on entry to the Active state

Figure A.3: LeaveAll state machine

		STATE	
		Active	**Passive**
EVENT	**Begin!**	Start periodictimer Active	Start periodictimer Active
	periodicEnabled!	-x-	Start periodictimer Active
	periodicDisabled!	Passive	-x-
	periodictimer!	Start periodictimer periodic Active	-x-

Figure A.4: Periodic state machine

Appendix - B

This appendix contains all message sequences captured by omnet++, exchanged between a talker, a switch and a listener.

First scenario

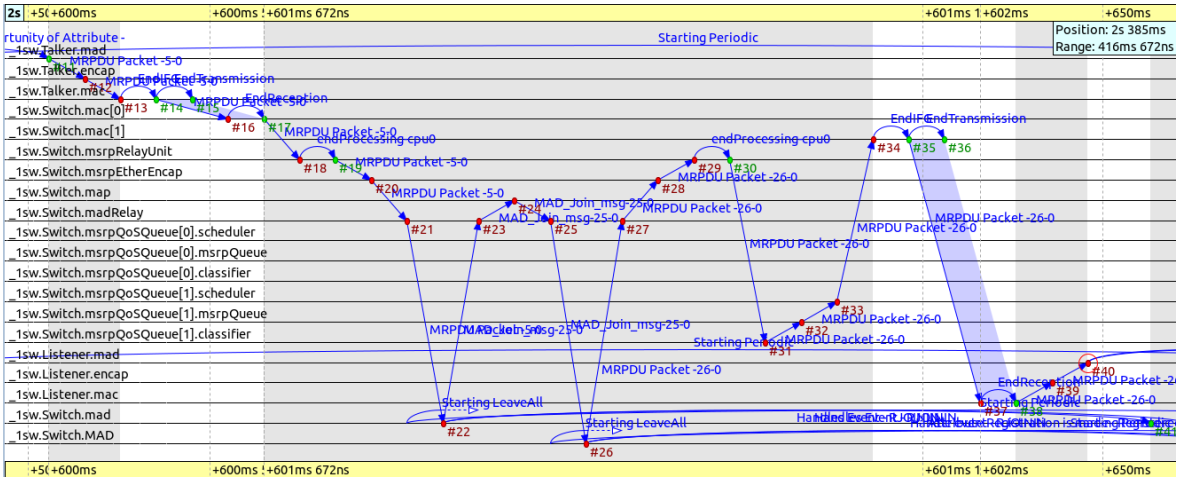


Figure B.1: Joining talker message sequence advertisement

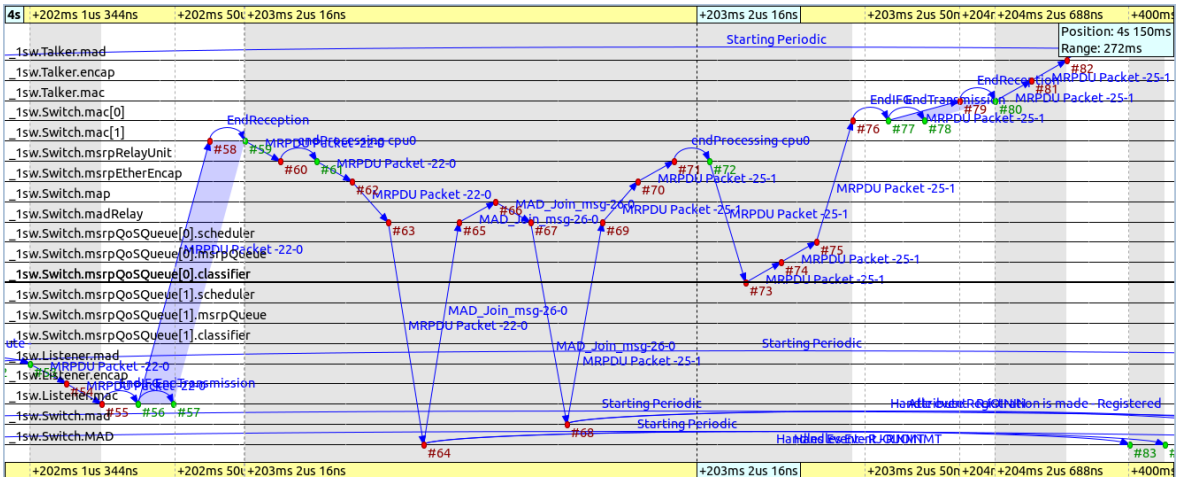


Figure B.2: Joining listener message sequence response

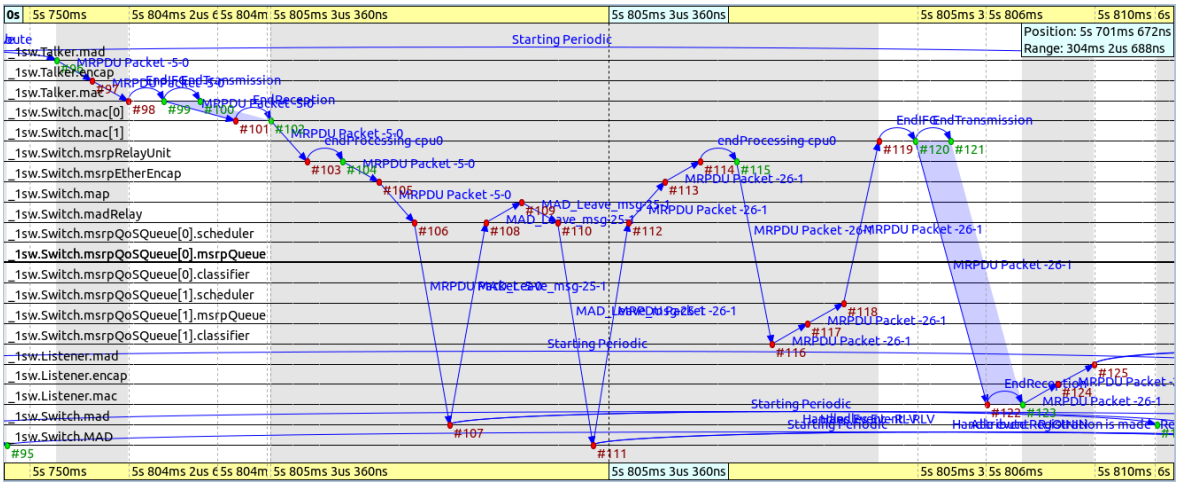


Figure B.3: Leaving talker message sequence advertisement

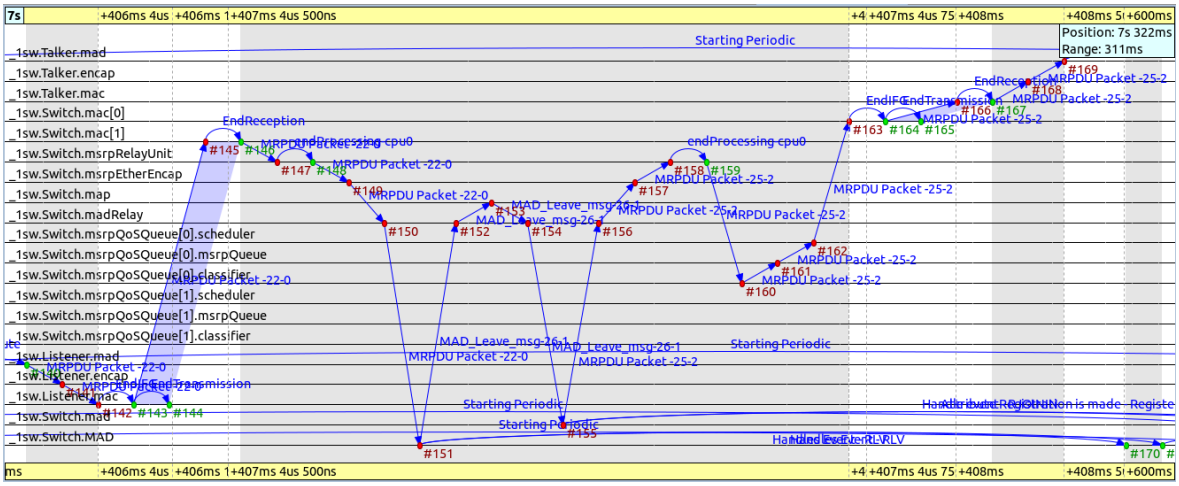


Figure B.4: Leaving listener message sequence response

Second scenario

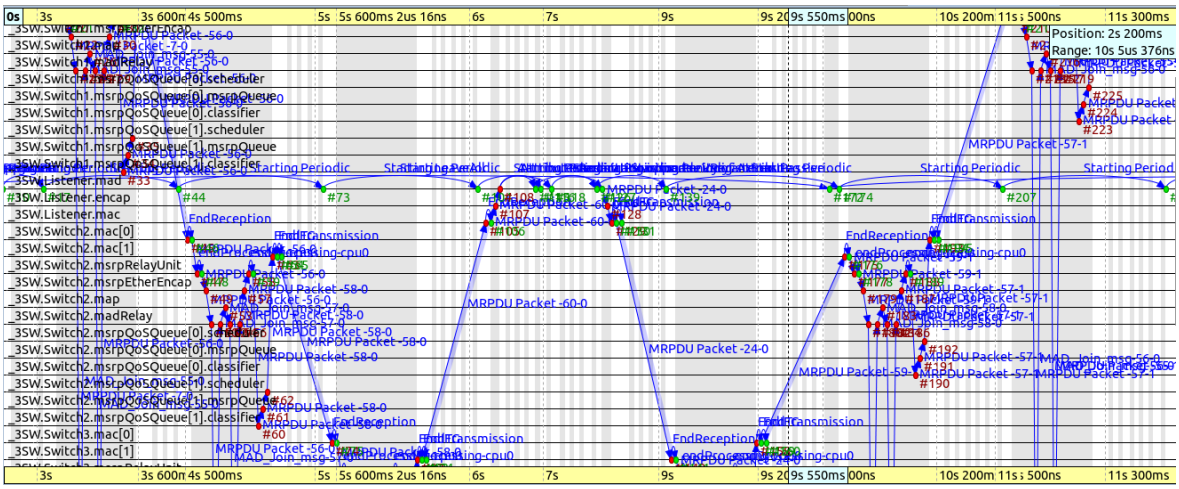


Figure B.5: Event log (1)

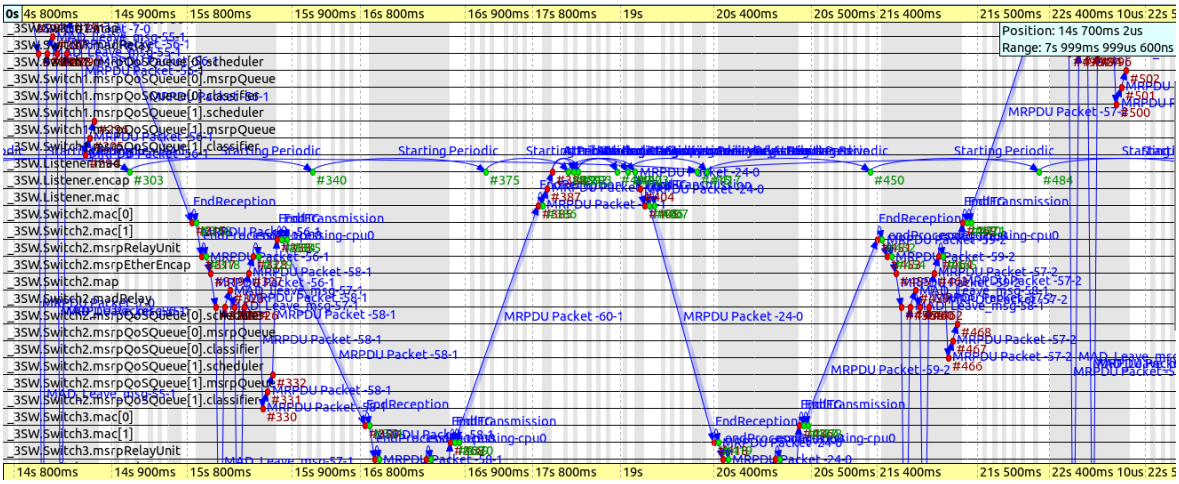


Figure B.6: Event log (2)