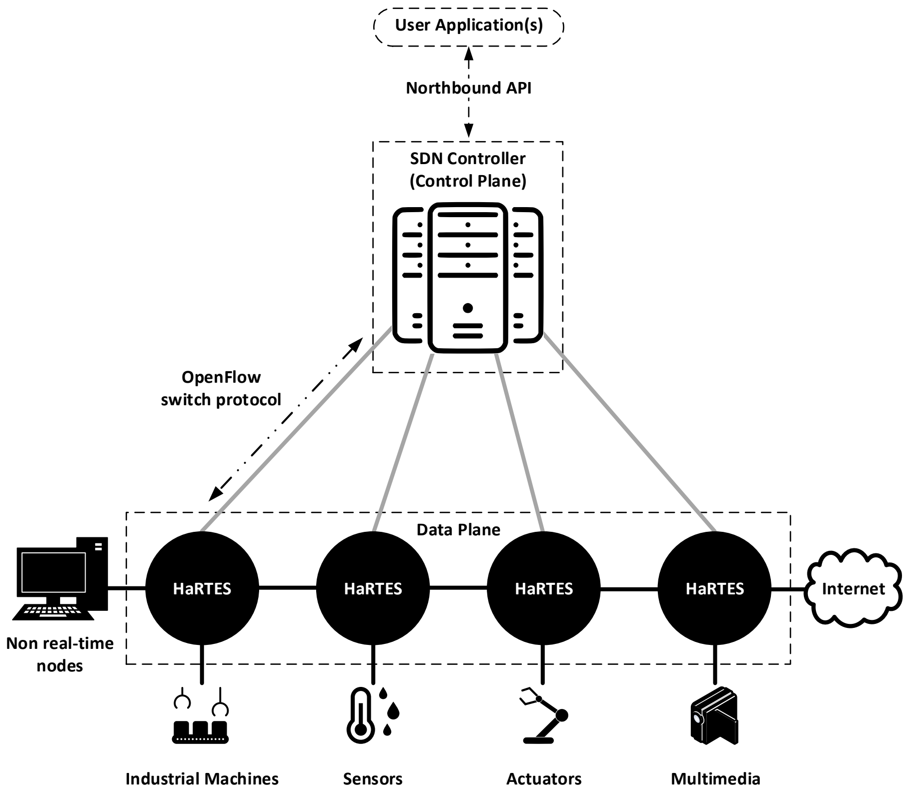
Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2019

Luis Emanuel Moutinho da Silva

Plataforma de Rede Tempo-Real Configurável por Software para Sistemas de Produção Ciber-Físicos

A Real-Time Software-Defined Networking Framework for Cyber-Physical Production Systems



Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Luis Emanuel
Moutinho da Silva

Plataforma de Rede Tempo-Real Configurável por Software para Sistemas de Produção Ciber-Físicos

A Real-Time Software-Defined Networking Framework for Cyber-Physical Production Systems

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Telecomunicações, realizada sob a orientação científica do Doutor Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Luis Miguel Pinho de Almeida, Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Universidade do Porto.

This work has been supported by the European Regional Development Fund (FEDER) through a grant of the Operational Programme for Competitivity and Internationalization of Portugal 2020 Partnership Agreement (PRODUTECH-SIF, POCI-01-0247-FEDER-024541).

This work has been supported by the research unit UID/EEA/50008/2013 (Instituto de Telecomunicações).

This work is funded by FCT/MEC through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2013.

Dedico este trabalho ao melhor da humanidade.

agradecimentos / acknowledgements

A maratona que culminou nesta tese começou em 2013 quando, o meu então futuro orientador Paulo Pedreiras, me perguntou (algo parecido com): "Já andas nesta vida de investigação há uns anos... e porque não iniciar um doutoramento?!". Pois bem, fui para casa, encomendei uma bela *pizza*, e reflecti bastante (não tanto assim) no assunto. Passados poucos dias disse-lhe eu (também algo parecido com) : "Sabes que mais... siga! Aceitas ser o meu orientador?". E assim começou esta autêntica aventura.

Durante este período tive a felicidade de vivenciar imensas experiências que me fizeram crescer como pessoa e também profissionalmente (desculpem o *cliché*, tive de o usar). Uma das grandes oportunidades que tive foi a experiência de docência que começou na Universidade de Aveiro e que actualmente continua na Escola Superior de Gestão de Águeda (ESTGA). Quero agradecer a todas as pessoas e colegas que viram em mim a responsabilidade e o empenho necessário para esta função, e que me deram o seu voto de confiança, apoio e amizade. Em particular, quero agradecer ao Paulo Pedreiras, que me impulsionou para esta oportunidade de valor incalculável. Quero agradecer também ao Válter Silva, à Margarida Urbano, ao Joaquim Ferreira, ao José Manuel, ao Ricardo Marau, e a todos os outros colegas da ESTGA, pela sempre pronta disponibilidade e companheirismo. Por fim, e não tendo menor importância, quero agradecer a todos os meus alunos pelo seu empenho, respeito, e compreensão. Espero ter-vos ajudado a adquirir novas competências e que tenham desfrutado as aulas como eu desfrutei :)

Uma outra, e igualmente importante experiência foi o envolvimento em diversos projectos de investigação. Quero agradecer ao Luis Almeida e ao seu grupo de trabalho que me acolheram pela primeira vez na Faculdade de Engenharia da Universidade do Porto (FEUP) pela orientação destes meus primeiros passos no mundo da investigação científica. Quero também agradecer ao grupo de Sistemas Embutidos do Instituto de Telecomunicações (IT) - Pólo de Aveiro por estes últimos anos fantásticos. Bruno Fernandes, Joana Mendes, João Rufino, João Almeida, e restantes colegas, obrigado por todas as discussões técnicas (e outras menos técnicas) agradáveis, todas as gargalhadas e partidas, pelo espírito de grupo e entreajuda, e ainda mais importante, obrigado pelo apoio nos dias menos bons e pela compreensão nos dias em que exibi um maior grau de má disposição e "rabugice" :) Eu culpo as *deadlines* dos artigos!!

Não posso deixar de dar uma palavra à minha família e amigos:

obrigado à minha família, em particular ao meu irmão Marco Silva pelos momentos "bro" e é claro, à minha mãe Maria Deolinda e ao meu pai Alberto Luís pelo milagre da vida, apoio, compreensão nos momentos difíceis (mesmo sendo menos correcto convosco... nunca foi por mal), e pelo amor incondicional. Que tenhamos muitos anos juntos pela frente!

obrigado à minha namorada, Alice Paiva, por ter tido a resiliência de me "aturar" nestes tempos difíceis, pelas nossas piadas "foleiras", e por todo o amor. Que seja para a vida!

obrigado família Pedreiras (Paulo, Cristina, Sofia, Francisco) por sempre me acolherem de braços abertos e com genuína amizade. Obrigado por me fazerem sentir "parte" da família... espero ser sempre um bom exemplo de "filho do meio" :)

[com a mesma consideração e sentimento] obrigado família Marau (Ricardo, Jeanette e Mariana). Ainda não acredito na honra de ter tido o vosso voto de confiança para ser padrinho da Mariana (onde tinham vocês a cabeça?!). Vou fazer o meu melhor para ser um excelente modelo de pessoa e transmitir os melhores valores para a vossa filha, minha laroca afilhada (padrinho babado, sempre!).

obrigado Nuno Valdemar, Nélson Cunha e Samuel Moreira, pela amizade desde tenra infância que resiste ao teste do tempo, distância, e discussões e "zangas" tontas (iOS é melhor que Android, tenho dito!!). Que tenhamos longos anos deste verdadeiro companheirismo!

obrigado a todos os colegas de escola, das orquestras e bandas nas quais colaborei, e a todas as outras amizades criadas por esse mundo fora!

Como palavras finais, quero novamente agradecer aos meus orientadores, Paulo Pedreiras e Luis Almeida, pela orientação técnico-científica e pessoal que me proporcionaram no decorrer deste percurso e que foi sempre bastante próxima e humana. Obrigado por acreditarem sempre, mesmo quando eu menos acreditava em mim. Sem vós nunca teria conseguido completar este caminho, muito menos com qualidade! Sinto um enorme orgulho de poder contar convosco como meus grandes amigos. Paulo, não poderia pedir melhor orientador e amigo. Obrigado por tudo, foste a minha verdadeira mão direita, braço, pernas,... :)

Palavras Chave

Sistemas de Produção Ciber-Físicos, Redes de Comunicação Tempo-Real, Redes Configuradas por Software, Indústria 4.0.

Resumo

Conceitos emergentes como Produção Inteligente, Internet Industrial das Coisas e Indústria 4.0 trazem um conjunto radicalmente novo de requisitos para o modo como os sistemas industriais são projetados. No que diz respeito à infra-estrutura de comunicação, o suporte a ambientes dinâmicos, interoperabilidade e heterogeneidade, combinado com um aumento significativo no número de dispositivos, são apenas alguns dos desafios que devem ser enfrentados.

Redes definidas por software (SDN) é um paradigma de rede disruptivo que surgiu nas redes de campus, mas logo conquistou um interesse considerável da indústria e da academia, e é considerado um salto na gestão de tráfego aberto. Ele exibe dois recursos específicos que são muito adequados para gerir uma rede com requisitos em tempo real: (i) um controlo de recursos centralizado, completamente desacoplado do plano de dados, e (ii), um controlo granular de recursos, trama a trama. No entanto, devido às suas raízes, o modelo de tráfego SDN favorece o rendimento médio da rede com políticas de melhor esforço, eventualmente impondo restrições de largura de banda ou definindo prioridades fixas. Este modelo não é compatível com cenários industriais, que normalmente têm requisitos rigorosos em termos de previsibilidade, pontualidade e tolerância a falhas.

Esta dissertação suporta a tese de que é possível suportar a flexibilidade, a pontualidade e os requisitos de heterogeneidade de aplicações industriais emergentes, aprimorando as tecnologias SDN com os meios para aprovar reservas de recursos em redes que incluem plataformas de comutação com serviços em tempo real baseados em componentes.

Em particular, este trabalho: (i) aprimora uma plataforma de comutação em tempo real com serviços SDN, (ii) desenvolve extensões para o OpenFlow, um protocolo seminal SDN, para que um controlador OpenFlow possa configurar serviços SDN e em tempo real, e (iii) estende um controlador OpenFlow com análise de escalonamento para que possa executar o controle de admissão de novos fluxos de tráfego enquanto mantém o comportamento de prontidão de toda a rede.

Um protótipo da estrutura SDN em tempo real proposto é usado para executar vários experimentos que validam as propriedades desejadas e, consequentemente, a tese.

Keywords Cyber-Physical Production Systems, Real-Time Communication Networks, Software-Defined Networking, Industry 4.0.

Abstract

Emerging concepts such as Smart Production, Industrial Internet of Things, and Industry 4.0 bring a radically new set of requirements to the way industrial systems are engineered. In what concerns the communication infrastructure, support to dynamic environments, interoperability and heterogeneity, combined with a significant increase in the number of devices, are just a few of the challenges that must be faced.

Software-defined networking (SDN) is a disruptive networking paradigm that emerged on campus networks but soon captured considerable interest from industry and academia, and is considered a leap forward in open traffic management. It exhibits two particular features that are very well suited to manage a network with real-time requirements: (i) a centralized resource control, completely decoupled from the data plane, and (ii), a fine granular resource control, down to validating each single frame received in each port of a switch. However, due to its roots, the SDN traffic model favors the average network throughput with best-effort policies, eventually imposing bandwidth constraints or setting fixed priorities. This model is not compatible with industrial scenarios, which typically have strict requirements in terms of predictability, timeliness and fault tolerance.

This dissertation supports the thesis that is possible to support the flexibility, timeliness, and heterogeneity requirements of emerging industrial applications by enhancing SDN technologies with the means to enact resource reservations on networks comprising switching platforms with component-based real-time services.

In particular, this work: (i) enhances a real-time switching platform with SDN services, (ii) develops extensions to OpenFlow, a seminal SDN protocol, so an OpenFlow controller may be able to configure both SDN and real-time services, and (iii) extends an OpenFlow controller with scheduling analysis so it may perform the admission control of new traffic flows while maintaining the timeliness behavior of the whole network.

A prototype of the proposed real-time SDN framework is used to perform several experiments that validate the desired properties and consequently, the thesis.

Contents

List of Figures

List of Tables

Glossary

AFDX	Avionics Full-DupleX switched ethernet
AI	Artificial Intelligence
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CBS	Credit Based Shaper
CFQ	Cyclic Queuing and Forwarding
CIM	Computer Integrated Manufacturing
CLI	Command-Line Interface
CMIP	Common Management Information Protocol
COTS	Commercial-Of-The-Shelf
CPS	Cyber-Physical Systems
CSMA/CD	Carrier-Sense Multiple-Access with Collision Detection
CPPS	Cyber-Physical Production Systems
DDC	Direct Digital Control
EC	Elementary Cycle
EDF	Earliest Deadline First
FCS	Frame Check Sequence
FDB	Filtering Database
FDDI	Fiber Distributed Data Interface
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array (FPGA)
FSM	Finite-State Machine
FTT	Flexible Time-Triggered
FTT-SE	Flexible Time-Triggered Switched Ethernet
HaRTES	Hard Real-Time Ethernet Switch
HSF	Hierarchical Scheduling Framework
ICT	Information and Communication Technology
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
IPG	Inter-Packet Gap
IS-IS	Intermediate System to Intermediate System
IT	Information Technology
ITU	International Telecommunications Union
LAN	Local Area Network
LRP	Link-local Registration Protocol
M2M	Machine-to-Machine
MAC	Media Access Control
MAN	Metropolitan Area Network
MAP	Manufacturing Automation Protocol
MMRP	Multiple MAC Registration Protocol
MRP	Multiple Registration Protocol
MSRP	Multiple Stream Registration Protocol
MSTP	Multiple Spanning Tree algorithm and Protocol
MVRP	Multiple VLAN Registration Protocol
NETCONF	Network Configuration Protocol
NIC	Network Interface Card
NMS	Network Management System
ONF	Open Networking Foundation
OSI	Open Systems Interconnection
PAR	Project Authorization Request
PCI	Peripheral Component Interconnect
PLC	Programmable Logic Controller
QoS	Quality-of-Service
RBS	Reduced Buffering Scheme
RM	Rate Monotonic
RSTP	Rapid Spanning Tree algorithm and Protocol
SDN	Software Defined Networking
SFD	Start Frame Delimiter
SNMP	Simple Network Management Protocol
SPB	Shortest Path Bridging
SRP	Stream Reservation Protocol
STP	Spanning Tree algorithm and Protocol
TDMA	Time Division Multiple Access
TLS	Transport Layer Security
TM	Trigger Message
TOP	Technical and Office Protocol
TS	Timeline Scheduling
TSN	Time-Sensitive Networking
UDP	User Datagram Transport
UML	Unified Modeling Language
VID	Virtual Local Area Network (VLAN) Identifier
VLAN	Virtual Local Area Network
WCET	Worst-Case Execution Time
YANG	Yet Another Next Generation

CHAPTER 1

An introduction

Contents

*P*resently, industry is preparing for what is thought to be the very next big industrial revolution: Industry 4.0. This paradigm foresees an intimate interconnection of three major technologies, Cyber-Physical Systems, Internet-of-Things, and Cloud Computing, in order to seamlessly integrate physical objects, humans, and intelligent machines into a sophisticated information network, commonly known as Industrial Internet-of-Things. With it, a completely digitized industry emerges where the digital and physical environments are unblemished and intelligently connected, and new business models and ways of organizing and controlling value chains during product life-cycles are possible. Factories that are able to autonomously adapt to new configurations in order to meed supply-demand fluctuations or product variations, self-organizing logistics, and self-diagnosing machines, are some examples of the type of new services that are expected to be realized. Besides significant gains in efficiency and cost reduction, Industry 4.0 is expected to bring improved product quality, better manufacturing and logistics planning, and a higher degree of customer satisfaction.

1.1 The problem statement

A cornerstone of Industry 4.0 is the interconnection and availability of data throughout the entire value chain, from field devices at factories' shop floors to logistics and ultimately, the consumer. This vertical (factory levels) and horizontal (value chain) integration pushes forward the boundaries for services that networks must provide, with new applications demanding even more strict requirements in terms of flexibility, heterogeneity, management, and timeliness guarantees. Although the management and heterogeneity aspects have been thoroughly explored by generic data networks such as Ethernet and generic network management frameworks such as Software-Defined Networking (SDN), their main focus has been on throughput, often forsaking the timeliness behavior of the network. On the other hand, industrial network technologies have been developed to meet the strict timing requirements of industrial applications, *e.g.* very low latency and jitter values, with little regard to throughput and online reconfiguration. As such, bridging these two domains is vital to realize the vision of Industry 4.0. This work aims to interconnect both domains, enhancing technologies for highly flexible and heterogeneous networks with the ability to fulfill strict timeliness requirements with analytical guarantees.

1.2 The thesis statement

The thesis supported by the present dissertation argues that:

The flexibility, timeliness, management, and heterogeneity requirements of emerging cyber-physical production systems can be satisfied by a framework that leverages the inherent flexibility of SDN technologies to control a network comprising switching platforms with composable and dynamically configurable real-time services.

1.3 The central contributions

The main contributions of this work are:

- The real-time extension of a Software-Defined Networking (SDN) control plane;
- The real-time admission control module for a SDN controller;
- The enablement of a real-time Ethernet technology as a SDN data plane.

The aforementioned contributions, presented in detail in Chapter 5, resulted in the following scientific publications and communications:

1. L. Silva, P. Pedreiras, P. Fonseca, *et al.*, "On the adequacy of SDN and TSN for industry 4.0", in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, May 2019. DOI: `10.1109/isorc.2019.00017`. [Online]. Available: `https://doi.org/10.1109%2Fisorc.2019.00017`

2. L. Silva, P. Goncalves, R. Marau, *et al.*, "Extending OpenFlow with flexible time-triggered real-time communication services", in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, Sep. 2017. DOI: `10.1109/etfa.2017.8247595`. [Online]. Available: `https://doi.org/10.1109%2Fetfa.2017.8247595`

3. L. Silva, P. Goncalves, R. Marau, *et al.*, "Extending OpenFlow with industrial grade communication services", in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, IEEE, May 2017. DOI: `10.1109/wfcs.2017.7991965`. [Online]. Available: `https://doi.org/10.1109%2Fwfcs.2017.7991965`

4. M. Ashjaei, L. Silva, M. Behnam, *et al.*, "Improved message forwarding for multi-hop HaRTES real-time ethernet networks", *Journal of Signal Processing Systems*, vol. 84, no. 1, pp. 47–67, May 2015. DOI: `10.1007/s11265-015-1010-8`. [Online]. Available: `https://doi.org/10.1007%5C%2Fs11265-015-1010-8`

5. L. Silva, P. Pedreiras, M. Ashjaei, *et al.*, "Demonstrating the multi-hop capabilities of the hartes real-time ethernet switch", in *RTSS@Work 2014 Open Demo Session of RealTime Systems*, 2014

1.4 Document outline

This document is structured as follows:

- ***Chapter 2*** presents an overview of the evolution of industrial networks, leading up to the present tendencies. It discusses the communication requirements of the emerging Industry 4.0 and how can they be tackled;
- ***Chapter 3*** introduces basic concepts on real-time communications, scheduling techniques and analyses as well as knowledge on the SDN paradigm and its seminar southbound protocol OpenFlow;
- ***Chapter 4*** presents existing real-time network technologies based on switched Ethernet, existing research, and compares the most relevant technologies in order to find the best candidate for the framework herein proposed;
- ***Chapter 5*** discusses the proposed framework, its components and their development;
- ***Chapter 6*** presents the experiments carried out to evaluate and validate the proposed framework;
- ***Chapter 7*** compares TSN and SDN-based solutions, including the herein proposed framework, in the context of Industry 4.0;
- ***Chapter 8*** concludes the dissertation and draws some conclusions on the carried work and possible lines for future research.

CHAPTER 2

A crash course on industrial (real-time) systems

Contents

*D*uring the past decades, the evolution of computer science and electronic engineering, combined with the ever-increasing power and cost-effectiveness of electronic systems, has significantly influenced all areas of the human society and among them, industrial control systems. This chapter presents an overview of the evolution of industrial communications and discusses the real-time and functional network requirements of the current industrial trend as well as how can they be addressed.

2.1 Industrial systems: the historical evolution

Under a broad-scope view, the industry can be split into two major categories: process and manufacturing. The former alludes to processes or methods, either mechanical or chemical, to modify or preserve a given material or good. Processing can be typically carried out continuously or in batches, handling very large material flows and often having stringent safety requirements. Examples of this type of industries include petrochemical production, ceramics, and plastics. The latter type, the manufacturing industry, is concerned with the process of transforming raw materials, or semi-finished products, into finished goods for consumption or sale. Production is typically done on a large scale in factories, manually or with the help of machines, and normally focuses on attaining maximum quantity throughput of produced goods. Examples of this type of industries include textile and automotive.

Over the course of years, due to economic, social, and environmental factors, both process and manufacturing industries have been striving to improve production efficiency and quality while keeping costs low. As consequence, manual supervision, processing, and operation of machines was incrementally augmented, or even replaced, by automatic equipment in an effort to increase production levels, quality control, and reduce the number of employees. The need to move towards larger plants in order to reap the benefits of economies of scale has further stimulated the aforementioned process. However, the evolution of the industrial systems in the process and manufacturing industry has not been identical. While the former addressed automation within large plant areas with several interconnected machines spread over, manufacturing plants were normally composed of several isolated production stations (cells) and thus, automation was at first limited to the cell boundaries. Nonetheless, their differences in technical solutions progressively diminished and nowadays both use similar electronic systems for closed-loop control, for operator-machine interfaces, and for networking [6]. In this chapter, focus will be placed on the evolution of process plants and their technologies; however, do note that the presented communication technologies and topologies were eventually adopted by the manufacturing industry for both cells and the whole factory.

Early process plants were controlled either manually or through mechanical and/or hydraulic control systems. With the advent of discrete electronics, these systems were replaced by hard-wired control loops circuits comprising electronic sensors and actuators, such as transducers and relays. Process supervision and control was done in analog control rooms, through the use of analog indicators, buttons, and knobs. These systems were extremely complex and difficult to maintain, with changes to control loops requiring hardware to be replaced and kilometers of wiring to be reconfigured manually. With the appearance of computers specially designed for process control, supervisory and control functions were gradually shifted to a single mainframe computer. Point-to-point links connected field devices, *i.e.* sensors and actuators, to the control room mainframe, in a star-like topology. An example of this centralized configuration, common in the 1960s, is depicted in Figure 2.1. With this form of digital control, known as Direct Digital Control (DDC), systems were easier to deploy and maintain, with the possibility to enact changes to control strategies with a simple swap of programs. Still, wiring at the field level was still extensive and complex, and the expensive mainframe represented a single point of failure for the whole system [7], [8].

As computers got more powerful, reliable, and affordable, more and more analog control systems were replaced by computers. Supervisory and control tasks began to be distributed over several devices in an effort to increase the overall system resilience to faults and further reduce costs. This process culminated with the invention of integrated circuitry and microprocessors, which made possible to shift from centralized to distributed control systems by deploying relatively small, powerful, and affordable, digital controllers, *e.g.* Programmable Logic Controller (PLC), throughout the plant. An example of this distributed configuration, common in the late 1970s, is illustrated in Figure 2.2. A Local Area Network (LAN) based on high-speed serial digital networks, such as IEEE 802.4 Token Bus [9] or IEEE 802.3 Ethernet [10], now connected supervisory systems, operator consoles, and other computers at the control room to the distributed controllers that could now be deployed closer to field devices, further reducing the complexity and cost of wiring as the length of cables is significantly shortened. However, communication with field devices was still realized by point-to-point analog technologies, such as the 4-20mA [11] standard for analog sensors or the use of 24Vdc for digital inputs, which still required significant cabling complexity, *e.g.* two wires per half-duplex analog channel or

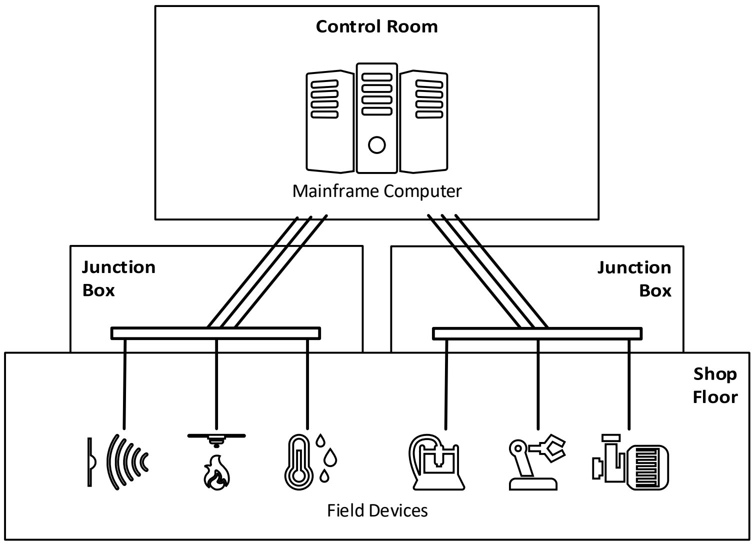


Figure 2.1: Centralized architecture of a process plant

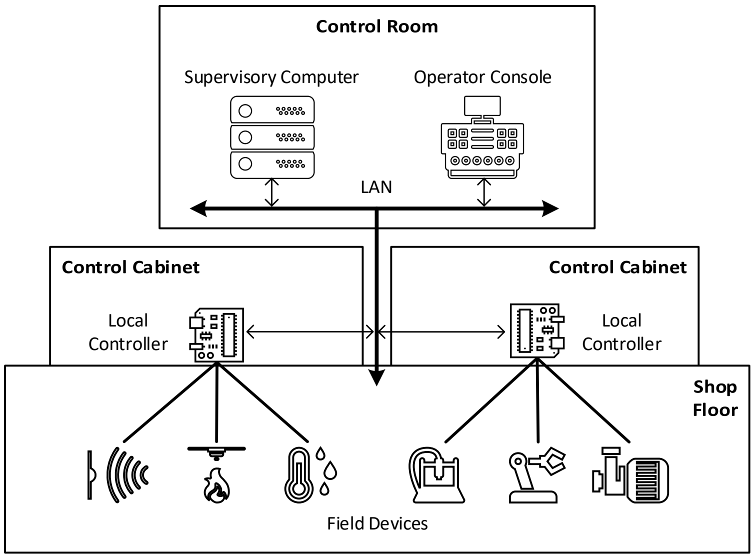


Figure 2.2: Distributed architecture of a process plant

four wires for a full-duplex one, and exhibited low data rates.

Advances in microelectronics led to another revolution in the early 1980s. As field devices became more intelligent and with more embedded functionality, including the capability to communicate over digital links, the previous point-to-point connections to the distributed controllers were phased out and replaced by a single digital data bus: the *fieldbus*. The use of a fieldbus, combined with the advanced capabilities of field devices such as data pre-processing and integrated control/monitoring functions, has several advantages, the most relevant being:

(i) an overall improvement in system modularity that introduced flexibility in the configuration and expansion of the process field, (ii) a significant reduction of installation and maintenance costs due to the substantial reduction in cabling volume and complexity, and (iii), the ability to remotely communicate with field devices for data acquisition, maintenance and configuration purposes [8], [12].

As manufacturing and process factories moved towards a distributed architecture with more computers, controllers, field devices, and communication networks, the existing architecture of a factory began to be logically and physically structured into different levels, *e.g.* management and process floors, each associated to a specific purpose and employing different technologies. During the mid 1970s, the Computer Integrated Manufacturing (CIM) concept is formulated; it recognizes that the development and production of a manufactured product can be accomplished more effectively and efficiently with the use of computer technology, and the importance of a comprehensive, enterprise-wide, information processing system. CIM defines a hierarchical structure for the use of computers and networks at all levels of industrial automation, identifying the varying characteristics, *e.g.* timeliness and volume, of data at each level and the necessary level of abstraction for data exchange across levels. The defined functional model is depicted in Figure 2.3 [12].

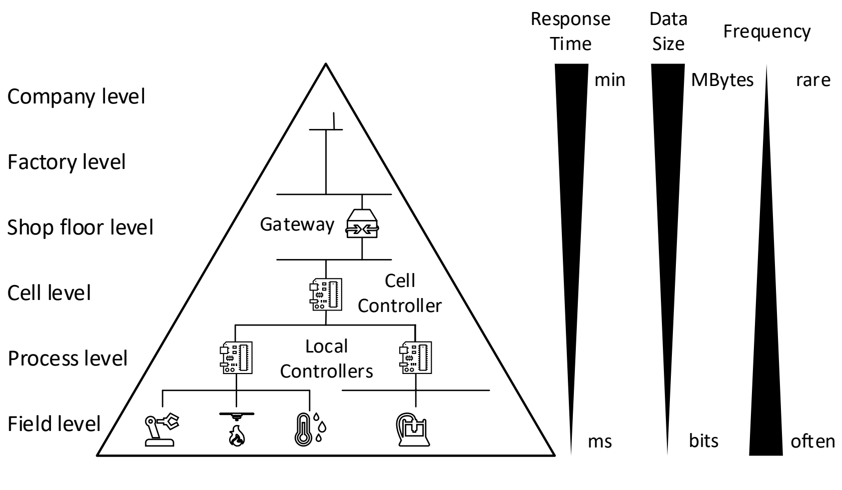


Figure 2.3: CIM automation pyramid

Despite the production advantages foretold by CIM, its implementation, being plagued by interoperability issues between networks and components of different vendors at the different levels in the automation pyramid, was no easy task. With respect to data and traffic characteristics, the 6 layered automation pyramid can be collapsed into three layers: field, cell, and company level [13], [14]. The taxonomy of such model, still used today, is depicted in Figure 2.4 [14]. At the company level, networks span the entire plant and interconnect a large number of computer systems for high-level applications such as stockpile management, engineering, and office applications. Communications at this level concern the exchange of large amounts of bursty data, such as production orders and queries to databases, with lax real-time requirements, e.g. response times of seconds. Cell level networks typically interconnect a moderate number of specialized devices, *e.g.* conveyors and machine tools, over a moderately large area and have to support real-time and non real-time communications comprising moderate amounts of data. Finally, networks at the field level interconnect field devices and controllers for automation purposes and thus, have stringent real-time requirements, *e.g.* response times as low as few milliseconds. Common requirements for industrial applications found at each factory level are presented in Table 2.1 [15]. In addition to performance requirements, a factory communication system has to provide a set of functional services such as acknowledged data transfer, uni-/multi-/broadcast communication, and security features.

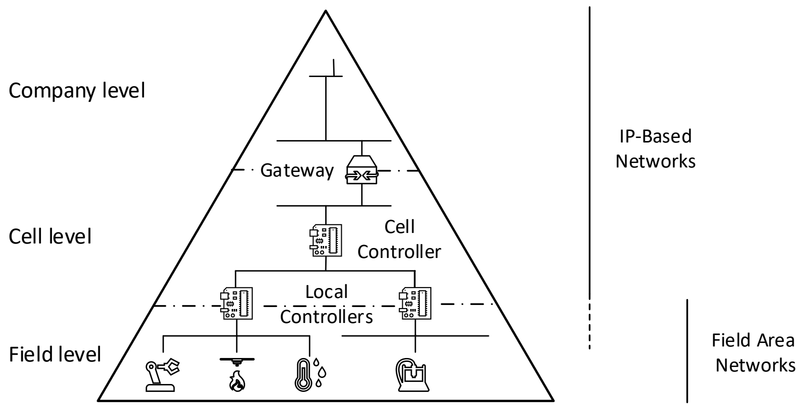


Figure 2.4: Collapsed automation pyramid

Table 2.1: Typical traffic requirements for industrial and multimedia applications

Application	Bandwidth (Mbps)	Data per station (Bit)	Delay (ms)	Jitter (ms)	Frequency (ms)
Company Level	10	4 x10^6	500	-	-
Cell Level	1	800	5	1	10
Field Level	1	16	1	0.01	1
Video (MJPEG)	16	0.5 x10^6	200	1	30
Audio	0.064	8	200	10	0.125

At that time, fulfilling the aforementioned functional and traffic requirements with a single network technology was no trivial task and thus, different, non-interoperable, communication protocols and/or physical media technologies emerged for use at the various plant levels, *e.g.* Asynchronous Transfer Mode (ATM) [16] for the company level, Ethernet at the cell level and a fieldbus technology across the field level. Even within the same level several network technologies could be found to address specific application requirements, or due to the use of machines and devices from different vendors; this is particularly true for field-level networks, where dozens of fieldbus solutions from several companies exist. This technological diversity leads to problems of incompatibility, forcing factories to resort to expensive gateways, adapters and protocol converters in order to integrate such number of heterogeneous systems [14]. An example of such system is depicted in Figure 2.5 [6].

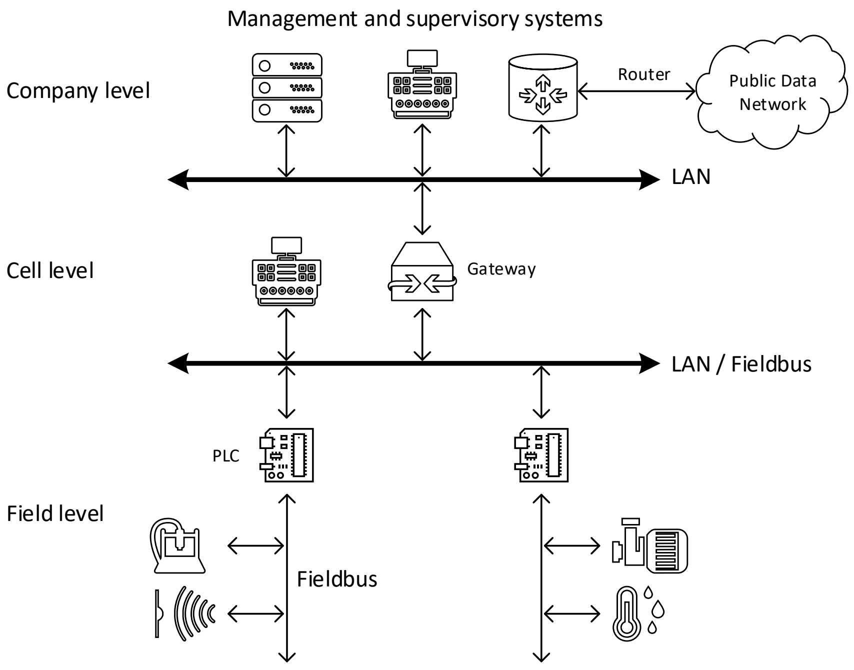


Figure 2.5: CIM-inspired industrial network architecture

In the wake of the CIM idea, several projects sprouted in an effort to tackle interoperability issues across factory levels. Examples of these projects include the Manufacturing Automation Protocol (MAP) [17] and Technical and Office Protocol (TOP) [18] projects for the creation of standard communication profiles within the CIM hierarchy, and the IEC TC 65/SC65C [19] committee the standardization of fieldbus technologies. However, these standardization efforts were an intricate process and limited results were achieved. The discussion of this matter is out of the scope, however, interested readers are referred to [8], [14] for detailed insight.

While both manufacturers and factories struggled towards an harmonization of the industrial communication infrastructure, Ethernet was quickly becoming the *de facto* standard for home and office networks worldwide and it was not long before it was considered for industrial use. Ethernet, being a mature and widely used technology, offers a significant number of advantages over other technologies. The main advantage comes from the great availability of manufacturers, which translates into cheap networking hardware, *e.g.* Network Interface Card (NIC) boards and hubs, and thus, lower deployment costs. Moreover, since it is a widespread and open technology, there is a large number of technicians and/or network administrators who are familiar with the technology; expenses due to proprietary software licensing and personnel training can be significantly decreased. The next significant advantage comes from the massive bandwidth, typically higher than that offered by fieldbus networks, with transmission speeds that can be in the range of Gbps, *e.g.* Gigabit Ethernet [20]. This fact opens the possibility for new advanced applications such as the use of audio and video for process surveillance [21]. Finally, the use of Ethernet across the entire plant, from offices connected to the Internet at the company level to field devices at the field level, would create a ubiquitous access to all devices in the plant, allowing controllers to communicate directly with each other, with system servers and field devices.

Despite being successfully adopted at high levels of the industrial network hierarchy, *i.e.* company and cell level, standard Ethernet could not be used at the field level. The main obstacle is its destructive and non-deterministic arbitration mechanism, Carrier-Sense Multiple-Access with Collision Detection (CSMA/CD) [10], which prevents Ethernet from providing deterministic channel access times and convey real-time traffic within guaranteed deadlines, particularly for high network traffic loads [22]. Although switched Ethernet [23] alleviates the impact of the non-determinism inherent to CSMA/CD with the absence of collisions at the switches' ports, providing real-time communication services is still not trivial due to multiple factors such as the limited number of priority levels and output queues, queuing build-up and memory overflow issues [22], [24], [25]. Thus, research began in order to support real-time traffic on switched Ethernet. Alas, despite standardization efforts from some committees, *e.g.* IEC 61158, the result has been a trend towards multiple Ethernet-based fieldbus protocols, similar to the fieldbus development process in the 1980s. Examples of such real-time Ethernet (RTE) protocols include EtherNet/IP [26], PROFINET IRT [27], TT-Ethernet [28], HaRTES [29], and Time-Sensitive Networking (TSN) [30].

Fast-forward to today, the industrial network landscape is fragmented between traditional fieldbus and real-time Ethernet technologies. Nonetheless, Ethernet-based networks, in particular networks employing switching technology (Figure 2.6 [31]), have overtaken traditional fieldbuses, and now account for 52% of market share *versus* the latter's 42% (Figure 2.7 [32]). Moreover, one can expect Ethernet to rapidly attain a dominant position due to its significantly higher annual growth of 22% *versus* the observed 6% for traditional fieldbuses.

Despite the important improvements in terms of flexibility, scalability, cost and performance, and the potential for a vertical integration of all factory levels, there is more to be reaped from the use of Ethernet in automation. Since its inception, Ethernet, combined with the TCP/IP network stack, quickly became the most popular technology for networks in offices, homes and commercial facilities all over the world. According to Beck [33], more than 95% of all LAN networks are Ethernet-based. As costs of network infrastructure is steadily being driven down, more and more devices are being connected. This trend is further fuelled by recent concepts, the most prominent one being the Internet of Things (IoT), in which is envisaged that it will be possible to connect almost anything with everything, anywhere, with the aid of the Internet technology. Industry and automation are no exception to the aforementioned trend, as attested by the recent uprise of similar concepts such as Industry 4.0 and Industrial Internet of Things (IIoT). The next section will present these on-going trends and the expected changes on the industrial landscape.

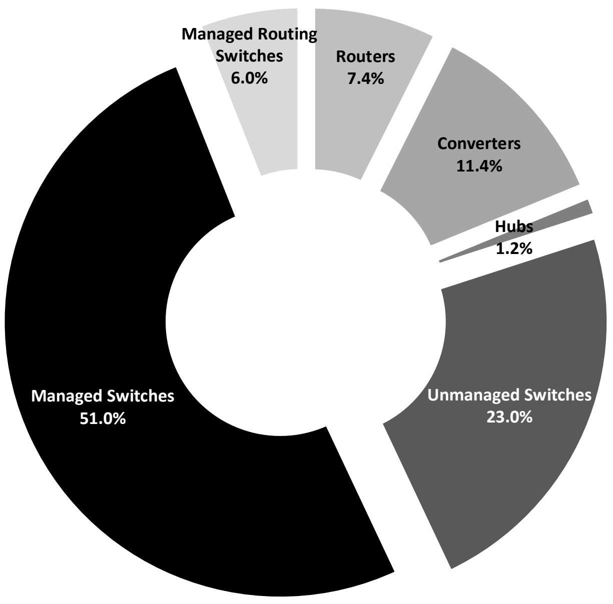


Figure 2.6: Industrial Ethernet equipment market share

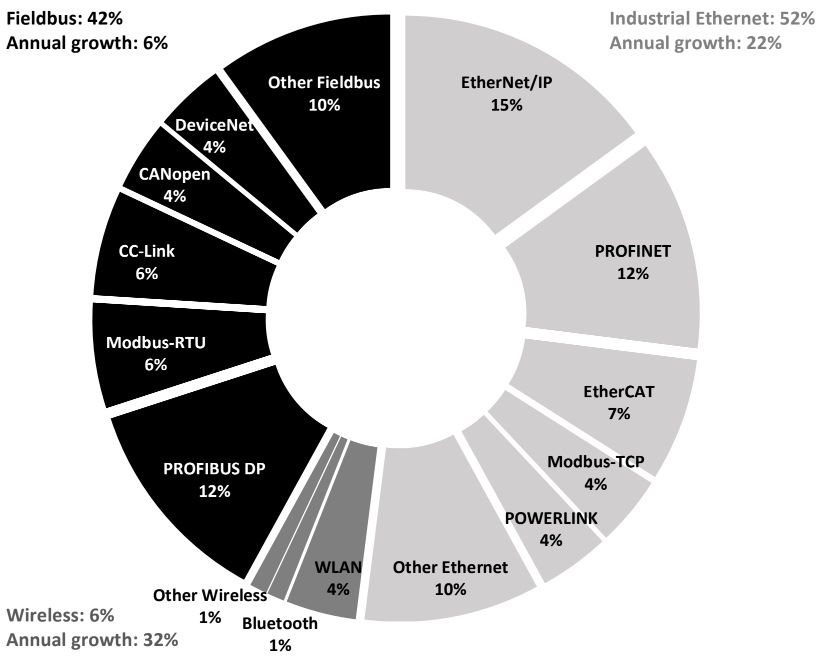


Figure 2.7: Industrial network market shares

2.2 The rise of a digitized industry

Industry 4.0, a term first coined as *Platfform Industrie 4.0* [34] by the German government as part of its strategic development road-map "High-Tech Strategy 2020", is poised as the very next big industrial revolution. While the first three revolutions were clearly marked by the advent of a new disruptive technology, be it the steam machine of the first revolution, the electricity and mass production techniques found in the second, or the advent of electronics and automation in the third, Industry 4.0 is all about leveraging and interlinking existing technologies from diverse technological areas, *e.g.* Cyber-Physical Systems (CPS), Machine-to-Machine (M2M) communications, the Internet-of-Things, *Big Data*, and 3D printing. Its main goal is to seamlessly integrate physical objects, humans, and intelligent machines into a sophisticated information network commonly known as IIoT, and with it bring forth a digitized industry, where the digital and physical environments are unblemished and intelligently connected, and new business models and ways of organizing and controlling value chains during product life-cycles are possible [35], [36].

Following the Industry 4.0 paradigm, the new industrial landscape will exhibit the following key characteristics [35], [36]:

- ***Smart robots and machines***. Robots and machines will exhibit a higher level of sensorization and intelligence, being able to interact and cooperatively work with humans and other machines on complex and/on interlinking tasks. Moreover, they will also be able to autonomously perform diagnostics and foresee necessary maintenance interventions, alerting human operators before breakdowns may happen;

- ***Vertical networking of smart production systems***. Although current production systems already embrace electronics and Information and Communication Technology (ICT), these systems will be far more interconnected and controlled in real-time. CPS will now spawn over all levels of the traditional factory pyramid (Figure 2.3), and incorporate machines across processes, storage and supply systems, as well as be able to communicate with suppliers and customers alike, giving way to the so called Cyber-Physical Production Systems (CPPS). With this vertical integration, factories will be able to react rapidly to changes in demand, stock levels, fluctuations in quality and machinery breakdowns, and production can be dynamically tailored, for example, according to customer-specific requests;

- ***Horizontal networking across value chains***. Entire value chains will be interconnected locally and/or globally thought the use of Internet and a new generation of real-time optimized networks. By networking inbound logistics, warehousing, production, marketing, sales, and outbound logistics, new cooperation and business models involving both customers and business partners are possible. For example, a customer is able to issue specific requests, not only for production but as well for the ordering, distribution, or even development of the product, and keep track of the product status in real-time.

Moreover, value chains will be also able to better cooperate and facilitate a global optimization, *e.g.* adjust production schedules among factories, and stock piles;

- ***Big Data.*** Due to the huge scale of sensorization and interconnection between the diverse systems and factories, a significant amount of data can be collected. This rich data set may be leveraged by cloud computing and Artificial Intelligence (AI) entities, for example, to further optimize and coordinate operations across the entire value chain;
- ***Incorporation of exponential technologies.*** Future factories will exploit exponential technologies to further improve product individualization, production flexibility, and cost savings. For example, AI and advanced robotics technologies can improve the cognitive ability and autonomy of machines and robots, enhancing production speeds, improving tasks such as the management of warehouses by autonomous vehicles, or even enabling new ones such as flying maintenance robots in production lines. Another example of the application of an exponential technology is the use of 3D printing for quick prototyping, inventory reduction, or even the reduction of supply chain partners.

An overview of the different Industry 4.0 components and entities, and their interactions, is depicted in Figure 2.8 [35]. All in all, the foundation of Industry 4.0 can be condensed as a tight relationship and interconnection of three major technologies [37]:

- ***Cyber-Physical Systems.*** These systems are defined by the control of physical processes through the use of computer-based algorithms and networking technologies. Examples include smart grid and autonomous vehicles;
- ***Internet of Things (IoT).*** Although there is no clear-cut definition for the term "Internet of Things", the International Telecommunications Union (ITU) defines it as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [38]. In summary, IoT can be described as a network technology that allows physical devices, vehicles, buildings and other electronic objects to collect and exchange data virtually anywhere;
- ***Cloud Computing.*** Cloud computing provides ubiquitous, on-demand access to remote computing resources, *e.g.* applications, storage or raw processing power, and data over the Internet. It can be efficiently leveraged, for example, as enabler of Big Data applications.

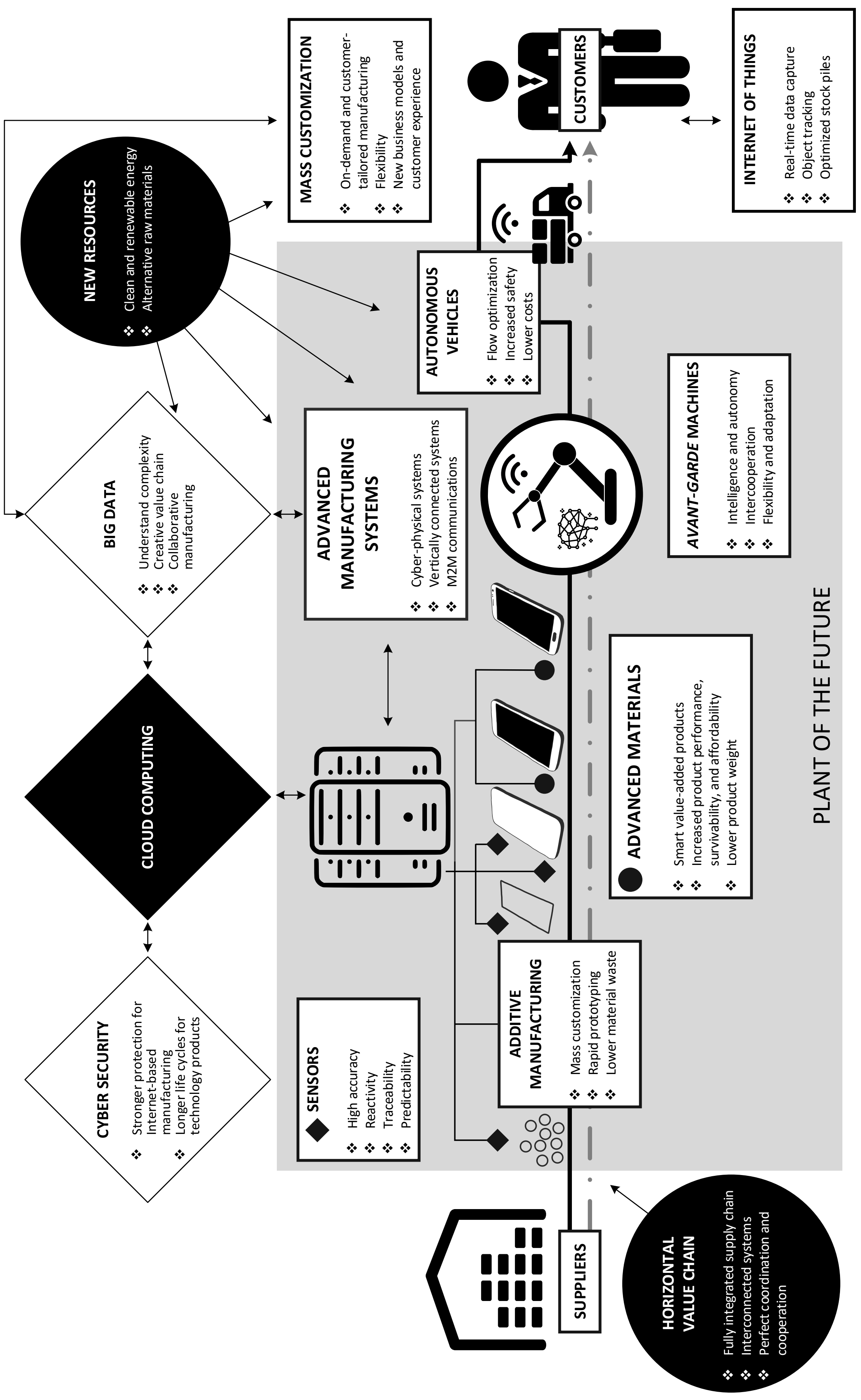


Figure 2.8: Industry 4.0

Besides the obvious significant initial gains in efficiency and cost reduction, which according to [39] can reach values of 18% and 14% respectively over a period of five years, the fourth industrial revolution is expected to bring [39], [40]: (i) better planning and control for manufacturing and logistics, (ii) a higher degree of customer satisfaction, (iii) greater manufacturing flexibility and faster time to market, (iv) improved product quality, (v) improved safety in the working environment, (vi) reduction of the consumption of raw materials, energy and water, as well as lower pollution emissions, and (vii), entirely new business models, with product oriented services and the individualization of products.

Although the vision and potential of Industry 4.0 is somewhat easy to understand, its development may be viewed as a daunting task due to its tremendous technological complexity and significant number of involved industrial, business, and political stakeholders. To facilitate the specification, development, and standardization of services, interfaces, and infrastructure, the *Plattform Industrie 4.0* consortium introduced the *Industrie* 4.0 reference architecture model (RAMI 4.0). RAMI 4.0 [41], depicted in Figure 2.9, spawns beyond previous models used for traditional industrial systems, *e.g.* CIM, and incorporates not only the functional hierarchical aspects of factories/plants (y axis) but also information on the life-cycle (type) and service life (instance) of I4.0 components (x axis), *i.e.* products and production systems. Layers in the vertical axis represent the various perspectives, such as hardware, communications, functional descriptions and data maps.

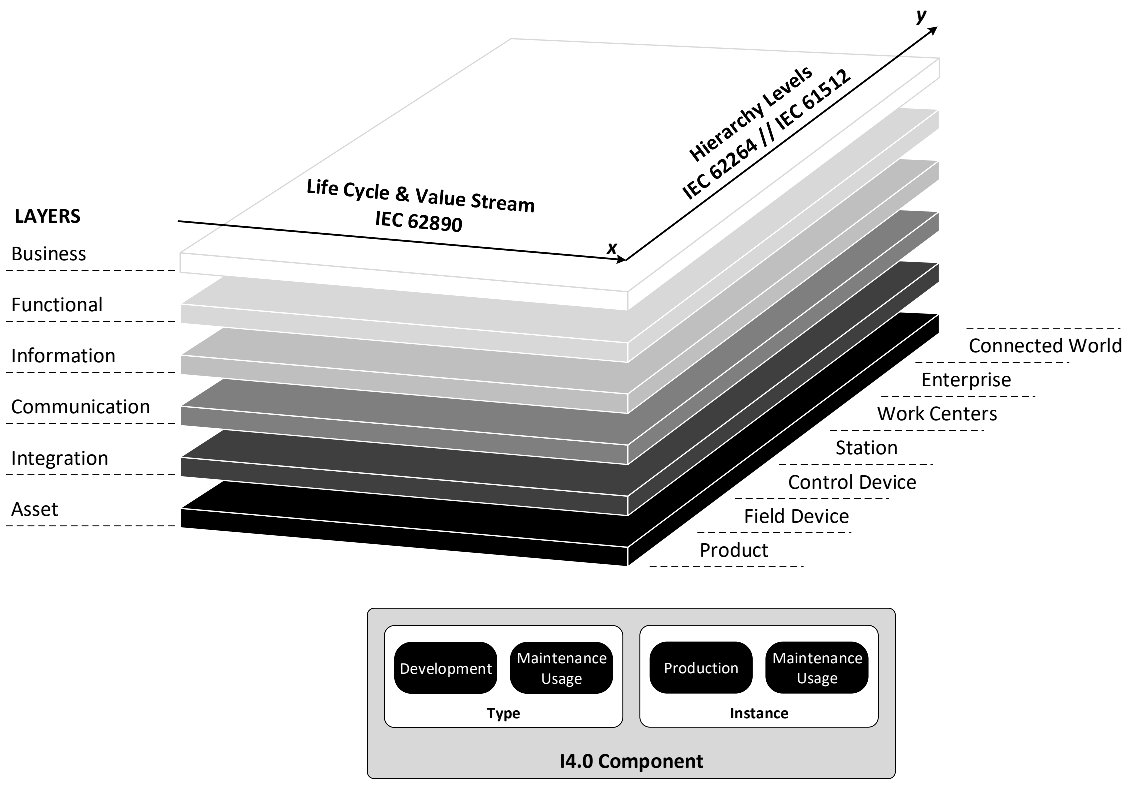


Figure 2.9: *Industrie* 4.0 reference architecture model (RAMI 4.0)

At the bottom of the model, the Asset Layer represents the reality which includes physical components, *e.g.* documents, raw materials, software and hardware, as well as humans. Moving up, the Integration Layer provides a digital representation of assets, making the properties of the real world accessible to computer systems. The Communication Layer deals with protocols and the transmission of data, providing uniform data formats to the Information Layer, which pre-processes, e.g. executes integrity checks and event-related rules, and stores the gathered data. The Functional Layer formally defines functions and services for remote access and horizontal integration. Finally, the business layer deals with relevant business processes, while also dealing with legal and regulatory aspects [41].

To help defining the set of interfaces and interactions among distinct entities of the RAMI 4.0 model, as well as to identify functional and qualitative requirements, nine application scenarios, described in detail in [42], are pinpointed:

- ***Order-Controlled Production (OCP).*** The existence of a network extending autonomous production capabilities beyond factory and company boundaries may allow companies to offer available production services to other companies in a way to increase the utilization of their own machinery. Third party companies may then access the aforementioned capacities to expand their production capability temporarily and in an ad-hoc fashion;

- ***Adaptable Factories (AF).*** With a network of intelligent and interoperable production modules that are capable to autonomously adapt to new configurations, an individual factory may rapidly change its production capabilities and throughput to meet supply-demand fluctuations or product variations;

- ***Self-organizing Adaptive Logistics (SAL).*** By taking advantage of the networking of adaptive logistic systems and autonomous transport vehicles, the distribution of goods and materials across the entire value chain can be done in a more decentralized and agile way, ensuring consistent flows between warehouses, production lines, and pick-up/delivery sites, while autonomously adjusting according to production conditions and material needs;

- ***Value-Based Services (VBS).*** By collecting data related to production, *e.g.* the machines and plants required to manufacture a certain good, product status information, and even the characteristics of processed raw materials or product parts, Information Technology (IT) platforms may provide new individualized services such as, for example, the provision of the correct process parameters for a production task being requested, or optimized maintenance schedules and procedures;

- ***Transparency and Adaptability of delivered Products (TAP).*** Manufacturers will be able to automatically and inexpensively collect information on sold products during their entire life cycle, in real-time. Akin to the business practices found in the software industry of today, manufacturers will be able to reconfigure/update some features of the product according to current operating conditions, provide after-sales services, and enhance/optimize future products by taking into account how consumers use a certain product;

- ***Operator Support in Production (OSP)***. The increase in digitization and networking, combined with new human-to-human/machines interfaces, will enable new services to assist human operators in the production process. For example, the real-time feedback from experts via remote video can provide quick assistance in the analysis and fixing of complex problems, or the training of new employees, which could be customized according to personal profiles, *e.g.* previous experience and language, and augmented with on-site 3D video-based tutorials;

- ***Smart Product Development for Smart Production (SP2)***. The availability of material data from suppliers, product specifications and functions, in the form of transparent data and virtual designs, may enable new collaborative and seamless product engineering work flows;

- ***Innovative Product Development (IPD)***. By leveraging Internet-based cooperation, new forms of collaborative product development may be created, with product vendors involving the right partners and technologies beyond site and company boundaries;

- ***Circular Economy (CRE)***. Technologies of Industry 4.0, in particular sensor technology and connectivity, will assist the return, restoration, and re-usability of products and components. For example, RFID technology can be used to clearly identify products with information regarding material composition and reuse possibility, and machines may monitor their components in real-time and order spare parts, while returning defective ones directly to proper recycling facilities.

The RAMI 4.0 model and its application scenarios pose significant technological challenges, in particular, the availability of secure information in real-time across the entire value chain is not easily fulfilled by existing communication technologies and demands the research of new solutions [42]–[44]. These network challenges are discussed in Section 2.3. Besides technological challenges, there are other barriers to the uptake of Industry 4.0 [37], [39]:

- ***IT security risk***. The connection of the value chain to the Internet may give room to security breaches and cybertheft which can have serious costs and reputation loss for enterprises;

- ***Data privacy***. The significant level of data collection and sharing found in such interconnected industry raises transparency and legal issues, with questions like who owns the data, what and how can it be shared, and for which purposes. This may lead to privacy concerns for both companies and consumers alike;

- ***Initial capital investment***. Companies will have to adapt or completely revamp existing installations, for example, to accommodate the necessary complex IT systems. This can be prohibitively costly, specially for smaller businesses;

- ***Lack of skilled workers***. The digital transformation to Industry 4.0 demands workers with new technical skills for both operation and maintenance tasks. This may require a significant retraining of the work force and/or hiring of highly qualified workers which can be scarce;
- ***Low maturity level of required technologies***. Industry typically demands proven and mature technology, and isn't willing to take unnecessary risks that may arise from bleeding-edge technology.

2.3 Networks for the industry of tomorrow

Considering the use cases and application scenarios envisaged for Industry 4.0, which foresee a significant increase in volume, reaction times, and variety of information, it is clear that networks will have to exhibit a high level of adaptability and throughput while having to satisfy demanding timing and reliability requirements [40], [44]. Moreover, requirements concerned to network safety, security, and ease of management and monitoring are also expected [40], [45]. This work focus on the requirements concerning heterogeneity, timeliness, and management, namely:

- **Requirement 1 (R1): Support simultaneous applications with heterogeneous QoS requirements.** Table 2.2 presents typical QoS requirements, particularly timing, of classes of applications that will co-exist in Industry 4.0 and which need to be fulfilled jointly [40], [45].
- **R2: Meeting precisely applications' QoS requirements.** Over-provisioning of network resources must be avoided to achieve efficient use of the network capacity.
- **R3: Dynamic message and reservation sets.** Order-controlled production and adaptable factories [42], leveraged by MAS and Service-Oriented Architectures (SOA) [46], [47] imply changing the communication requirements online. This change is itself time-constrained, *e.g.*, in the seconds range [46].
- **R4: Real-time network monitoring and diagnostics.** Precise communications and network state monitoring is key to react promptly to changes in communication requirements and potentially harmful situations [40], [48].
- **R5: Consistent set of open management tools.** The current practice of using vendor-specific and non-interoperable sets of management tools limits heterogeneity and jeopardizes the benefits of Industry 4.0.

During the last years, Software-Defined Networking (SDN), explained in detail in Chapter 3, has garnered considerable attention from industry and academia and is considered an important step for an open and flexible management of generic data networks, in particular for data center and carrier grade networks. Its success is based on two particular aspects:

Table 2.2: Typical QoS requirements of Industry 4.0 applications [40], [45]

	Motion Control	Cell Control	Augmented Reality	HMI*
Latency / Cycle Time	$250\mu s$ to $1ms$	$1ms$	$10ms$	$100ms$
Jitter	$\leqslant 1\mu s$	$\leqslant 1ms$	- - -	- - -
Reliability**	$1e^{-}8$	$1e^{-}8$ *to* $1e^{-}5$	$1e^{-}5$	$1e^{-}5$
Data rate	kbit/s	k-Mbit/s	M-Gbit/s	M-Gbit/s

- - - Not specified

*Human-Machine Interface

**Packet Error Rate

(i) a centralized resource control, completely open and decoupled from the network devices hardware/software, and (ii), a fine granular resource control, down to the validation of each single received frame. These two characteristics are also very interesting and suitable for a flexible and dynamic management of resources in real-time networks (meets requirements R4-R5). Alas, due to its roots, the development of SDN technologies prioritizes forwarding and throughput management and has no suitable services and APIs to support real-time applications and provide a proper configuration of real-time resources (misses R1-R3).

On the other side of the spectrum, the development of industrial networks have intensively pursued high determinism and extremely low latency and jitter values. Although there is a plethora of industrial network technologies (discussed in Chapter 4) which are able to fulfill strict timeliness requirements, these commonly exhibit limitations in either throughput, operational flexibility, and concurrent support for traffic with distinct requirements and activation patterns (limited fulfilment of R1-R3). Moreover, it is also common to have different network technologies across factory levels, where each technology exhibits different degrees of determinism, throughput, and flexibility, and requires different tools for the management and configuration of its resources (fails R4-R5).

In conclusion, future networks will have to address the aforementioned issues and reconcile the determinism of low level industrial networks with the openness of Internet-like generic data networks, leading to significant research and technological challenges. In fact, this is the problem addressed by this thesis work. Herein, it will be shown that it is possible to conciliate the flexibility and heterogeneity of SDN technologies with deterministic real-time communications offered by suitable Ethernet technologies and meet all the previously identified requirements.

CHAPTER 3

Theoretical foundation

Contents

*T*his chapter aims to provide basic knowledge on real-time communications, scheduling techniques, and on software-defined networking technologies. This knowledge is essential for the understanding of concepts introduced throughout this document.

3.1 On real-time systems

Some systems, commonly associated to physical processes, have to interact with the external environment in order to perform a given function, *e.g.* control an engine's fuel combustion. This interaction is usually done through the use of sensors, actuators and other input-output interfaces. These systems can be found, for example, in industrial automation systems, automotive applications, flight control systems and military applications [49]. Since the environment has inherent temporal dynamics, in order to properly interact with it these systems not only have to produce logically correct solutions but also produce and apply them within a specified time interval, known as *deadline*. Systems, where the correctness of the system behaviour depends on both the logical computations and the physical time instant when they are produced and applied are called ***real-time systems*** [49].

3.1.1 Taxonomy of a real-time system

Real-time systems are usually composed by computational activities, *i.e.* tasks, which execute specific functions and have stringent timing constraints that must be met in order to achieve proper system behaviour. A typical timing constraint on a task is the deadline, *i.e* the instant before which a task should complete its execution without impairing the system. Depending on the effects of a missed deadline, tasks can be categorized as [49]:

- **Non real-time**. Task has no time constraints and always contributes to the system whenever it completes its execution;
- **Soft**. Task's output still has some utility to the system after missing its deadline, however, the system's performance is degraded;
- **Firm**. Task's output has no utility to the system after a deadline miss, however, it does not cause catastrophic consequences on the system behaviour;
- **Hard**. Task only contributes to the system if it completes within its deadline. A deadline miss may cause catastrophic consequences, *e.g.* overall system failure with human and/or material losses.

According to the type of tasks executing in the system and the consequences that may arise from deadline misses, real-time systems can be categorized as [49]:

- **Soft real-time systems**. Systems that only integrate soft and/or firm tasks are categorized as soft real-time. In these systems, deadline misses may induce overall performance degradation without catastrophic consequences. A typical example for this type of system is video and sound streaming in which a deadline miss typically results in minor image/sound glitches.

- **Hard real-time systems**. Systems that contain at least one hard task are categorized as hard real-time. In these systems, a deadline miss may result in a system failure with catastrophic effects, *e.g.* material and/or human losses. A typical example for this type of system is a nuclear power plant control in which a deadline miss could result in the failure of the nuclear reactor;

Besides timing constraints, tasks may also have precedence relations, *i.e.* they can't be executed in arbitrary order, and resource constraints. Readers are referred to [49] for more information.

A real-time system may have one or more resources that have to be shared among several tasks, *e.g.* CPU and network. An important matter is to determine how tasks access shared resources and whether all tasks can execute and meet their requirements, in particular during peak-load. To that end, a real-time system commonly employs a *scheduler* entity which runs *scheduling algorithms* to determine how tasks access a given resource, *e.g.* task's order of access, and performs *schedulability analysis* to assess if a given task-set can be executed while fulfilling its timeliness requirements. To perform these operations, a real-time system describes tasks using mathematical abstractions (*task models*) that convey, for example, their timeliness requirements. These components will be briefly described in the following sections.

3.1.2 Task model

Tasks can be triggered (activated) by several sources, from events generated at specific time instants, to occurrences derived from the execution of other tasks or by external events, *e.g.* sensor reading. The time instant in which a task is activated is known as *task arrival*, while the duration of time between two arrivals of the same task is known as *task inter-arrival time*. Depending on the regularity of its activation, a task may be categorized as periodic, aperiodic, or sporadic. Tasks that are activated regularly and with a fixed inter-arrival time are called *periodic*. Conversely, tasks whose arrival is not periodic but exhibit a minimum inter-arrival time between any consecutive actions are known as *sporadic*. Finally, tasks which don't have a predictable and bounded activation pattern are defined as aperiodic.

The model for a set of periodic tasks Γ can be formally defined as:

$$\Gamma = \{\tau_i \mid \tau_i = \{C_i, T_i, \phi_i, D_i, Pr_i\} \quad, i = 1, 2, .., n\} \tag{3.1}$$

where:

- C_i is the worst-case computation time required to complete task τ_i, also known as Worst-Case Execution Time (WCET);
- T_i is the period of task τ_i;
- ϕ_i is the initial phase, i.e., the release time of the first instance of task τ_i;
- D_i denotes the relative deadline of task τ_i;
- Pr_i denotes the priority of task τ_i.

The release (activation) instant ($r_{i,k}$) and absolute deadline value ($d_{i,k}$) of a generic kth instance of the periodic task τ_i can be computed as follows:

$$r_{i,k} = \phi_i + (k-1) \times T_i \quad, \ k \in \mathbb{N} \tag{3.2}$$

$$d_{i,k} = r_{i,k} + D_i \quad, \ k \in \mathbb{N} \tag{3.3}$$

There are other parameters typically defined for periodic task-sets:

- ***Hyperperiod.*** Minimum interval of time after which the schedule repeats itself. For a set of periodic tasks synchronously activated at time $t = 0$, the hyperperiod is equal to the least common multiple of all task periods;

- ***Job response time.*** Time elapsed between the instant when the task is ready to execute to the time when it finishes its execution (single instance);
- ***Task response time.*** Maximum response time among all task jobs;
- ***Task critical instant.*** Activation time that produces the largest task response time.

The notation of Equation 3.1 can also be applied for sporadic tasks by foregoing the initial phase ϕ_i and by considering the minimum inter-arrival activation time $Tmit_i$ instead of the regular period T_i. For sporadic tasks, the activation and the absolute deadline instants are computed as:

$$r_{i,k} \geq r_{i,k-1} + Tmit_i \quad , \; k \in \mathbb{N} \tag{3.4}$$

$$d_{i,k} = r_{i,k} + D_i \quad , \; k \in \mathbb{N} \tag{3.5}$$

3.1.3 Task scheduling

As previously mentioned, the resource scheduler is the system component that decides the order in which tasks access a given shared resource. The procedure of selecting the task that executes at a given point in time is called *scheduling* and the set of rules that determines the order by which tasks are executed is known as *scheduling algorithm* [49]. A common taxonomy used to categorize the scheduling algorithms is depicted in Figure 3.1.

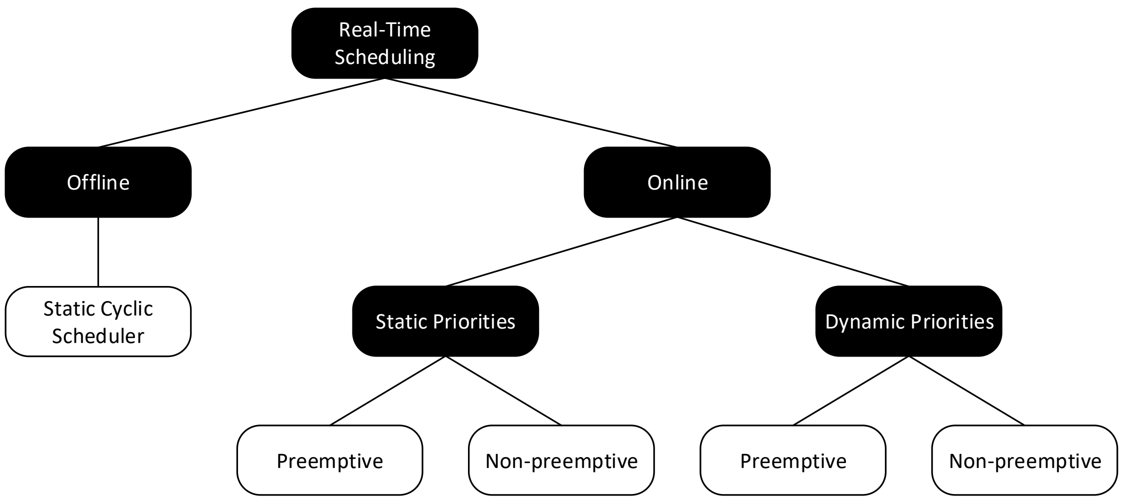


Figure 3.1: Taxonomy for real-time scheduling algorithms

The scheduling algorithms fall into two general categories: (i) online, where scheduling decisions are performed while the system is running, and (ii) offline, in which the schedule is built before starting the system. The advantage of offline techniques lies in the possibility of using complex and computational demanding algorithms to obtain highly optimized schedules. However, online changes in the task set are inherently not possible and thus, they are not adequate for dynamic systems where new tasks may be added at run-time and their requirements changed. With respect to online scheduling techniques, they are much more flexible and are able to perform scheduling decisions upon the occurrence of events that require rescheduling, *e.g.* arrival of new tasks. Online algorithms are further divided into two categories: (i) scheduling that is based on static priorities that are derived from fixed information, and (ii) scheduling based on priorities that are dynamically changing following run-time information, *e.g.* proximity to deadlines. Each of the last two categories can be identified as being preemptive or non-preemptive. In preemptive scheduling techniques, the on-going execution of a task can be suspended and interrupted in order to give way to a higher priority task. In non-preemptive scheduling, a running task executes to its completion, even if a higher priority task becomes ready for execution. The next sections will present some of the most relevant scheduling algorithms.

3.1.3.1 Scheduling techniques for periodic tasks

Periodic tasks are of great importance for many real-time systems, in particular for industrial applications which commonly employ closed-loop control techniques for the operation and management of processes and machinery. Such tasks are required to be cyclically executed at proper rates and completed within their deadlines. To that end, several techniques and algorithms can be employed by schedulers. Examples of the most well known and used techniques include Timeline Scheduling (TS), Rate Monotonic (RM), and Earliest Deadline First (EDF) [49]. These will be briefly explained next.

Timeline Scheduling

Timeline Scheduling, also known as Cyclic Executive, consists in segmenting time into equal length time intervals (time slots). The execution of each task is statically allocated to a certain time slot in such a way that the frequency and deadline requirements for all tasks in the system is respected. The association of task executions and time slots for a whole hyperperiod is stored in a scheduling table that is typically built offline, *i.e.*, before the system is started. At run-time, a dispatcher follows the table's schedule and ensures that tasks are only executed at their predetermined slots. The schedule is repeated at the start of every hyperperiod.

The main advantages of this scheduling technique is its simplicity and behaviour predictability. However, its flexibility is very limited since it is built offline and it is very sensitive to application changes. These problems are solved by priority-based algorithms, *e.g.* Rate Monotonic.

Rate Monotonic

The Rate Monotonic [50] is a simple online preemptive algorithm based on static priorities and constrained deadlines, *i.e.*, $\forall_{\tau_i} \in \Gamma : D_i = T_i$. Following RM scheduling, priorities are monotonically assigned according to their request rates. In particular, tasks with shorter periods have higher priorities. That is:

$$\forall_{\tau_i,\tau_j} \in \Gamma : T_i < T_j \Rightarrow P_i > P_j \tag{3.6}$$

Task scheduling is done online, *i.e.* at run-time. Whenever a task instance is activated or the execution of a task finishes, the scheduler selects for execution the task with the shortest period among all ready tasks. Liu and Layland [50] showed that RM is optimal among all fixed-priority assignments, *i.e.*, no other algorithm can schedule a task set that cannot be scheduled by RM.

Earliest Deadline First

Earliest Deadline First [50] is an online preemptive algorithm based on dynamic priorities and constrained deadlines. EDF assigns higher priority levels to tasks with the nearest (earliest) deadline relative to the current time instant. That is, during run-time the following relation holds:

$$\forall_{\tau_i,\tau_j} \in \Gamma_{T_a} : d_i < d_j \Rightarrow P_i > P_j \quad , \; T_a \in \mathbb{R} \tag{3.7}$$

where Γ_{T_a} is the subset of Γ comprising only the ready tasks at instant T_a, and (d_i,d_j) are the absolute deadlines at the same instant T_a for task τ_i and τ_i, respectively.

Compared to RM scheduling, EDF is able to achieve higher utilization while keeping guaranteed timeliness, and reducing the number of task preemptions. However, EDF is more complex and exhibits high run-time overheads which can be problematic for systems with low processing power.

3.1.3.2 Scheduling techniques for aperiodic tasks

Contrasting to periodic tasks, that have well defined activation patterns and are easily controlled and synchronized within the scheduling framework, aperiodic tasks are asynchronously activated by events that are typically external and not controllable by the scheduler. A challenge faced by systems which have to combine both type of tasks, such as those herein addressed, is to be able to execute aperiodic tasks within adequate response times while still fulfilling the timeliness requirements of all periodic jobs. To that end, several techniques have been devised to bound the interference of aperiodic tasks and make it more deterministic. These approaches can be grouped into two main groups: (i) techniques based on fixed-priority scheduling, and (ii), techniques based on dynamic-priority scheduling.

Techniques from the former category typically target systems where periodic tasks are scheduled according to the RM policy and all tasks are fully preemptable. These include [49]:

- ***Background scheduling***. This is the simplest method to schedule aperiodic jobs and consists in executing aperiodic tasks in a best-effor manner, *i.e.*, in background when no periodic tasks are ready for execution. Although the timeliness guarantees for periodic jobs are preserved, no guarantees for aperiodic tasks are provided, particularly for systems with high periodic loads. Therefore, background scheduling is only useful for work-sets where aperiodic activities do not have demanding timing requirements;

- ***Server-based schedulers***. Compared to background scheduling, server-based techniques are able to provide good response times and timeliness guarantees for aperiodic tasks. The main idea behind these techniques is the use of a periodic task, known as server, whose purpose is to act as an proxy for the execution of associated aperiodic tasks. A server is characterized by a period T_s and an execution time C_i, commonly referred to as server capacity or budget, just like a regular periodic task. Servers are scheduled together with periodic tasks and, when active, service associated aperiodic requests within the limit of their configured server capacity. The scheduling of aperiodic tasks within a given server may resort to different algorithms than the one used by the system for the periodic jobs. Although server-based schedulers are not optimal, they strike an excellent balance between performance, and computational and implementation complexity;

- ***Slack stealing [51].*** This algorithm provides substantial improvements on the response time of aperiodic events over server-based solutions. Slack stealing takes advantage of the fact that, for the majority of systems, there is no benefit in completing periodic tasks earlier than their deadline. Therefore, it creates passive tasks which attempt to make time for servicing aperiodic tasks by delaying the execution of periodic requests as much as possible while avoiding deadline misses. Although slack stealing based techniques provide excellent performance when compared to the background and server-based scheduling, they are exhibit high computational and implementation complexity.

There are several types of server-based, fixed-priority schedulers in the literature, with different improvements with respect to their performance. The most well-known algorithms are [49]:

- ***Polling Server (PS)*** [52]. A polling server activates every T_s, where it fully replenishes capacity and checks for pending aperiodic requests. If one or more requests are pending upon activation, the server executes them within its budget, otherwise, it becomes suspended until the next activation and all its budget is lost. Requests arriving during the suspended state must wait until the next server activation. Tasks execute only during the server capacity C_i, and are preempted if the budget is completely exhausted. Therefore, aperiodic executions are constrained and have a similar impact in the schedulability of the system as regular periodic tasks with period T_s and computation time C_s. The significant drawback of polling servers is their poor response time, since requests that arrive just after server activations have to wait for the following instances to be serviced;

- ***Deferrable Server (DS) [53].*** The deferrable server is another server-based scheduler that presents slightly better response times for aperiodic requests than the polling server. Following this algorithm, the server task is also modeled as a periodic task with period T_s and execution time C_i. Contrary to the polling server, which polls pending tasks every period and drops all its budget in the absence of requests, DS preserves its budget until the end of each period or until it is exhausted. This allows requests to use the available budget at any time during the server period. Akin to the PS, server capacity is fully replenished at the beginning of each server period. A negative aspect from the use of deferrable servers is a lower schedulability bound for the periodic task-set since aperiodic tasks may now execute at any time during the servers' period[49];

- ***Priority Exchange (PE) [52].*** The priority exchange algorithm exchanges a slight degradation of the response times provided by deferrable servers for better schedulability bounds. Here, servers are typically configured as high priority periodic tasks, and are able to exchange their budget for the the execution time of lower priority periodic tasks. As in the polling server, the server fully replenishes its budget at the beginning of each server instance and checks for pending aperiodic requests. If the server is the highest priority active task, it serves pending aperiodic requests using the available capacity, otherwise its budget is exchanged for the execution time of the periodic task which is currently ready and has the highest priority. Hence, servers' budget are not lost but preserved at the priority of the lower priority task involved in the exchange. When an aperiodic request is received, it is run with the priority currently associated with the server's budget. Note that this algorithm exhibits a significantly higher computation overhead and complexity than that of a deferrable server;

- ***Sporadic Server (SS) [54].*** The sporadic server exhibits better average response time of aperiodic tasks than those of the polling server while preserving the schedulability of the periodic task-set. Akin to the deferrable server, it preserves its capacity until it is consumed by an aperiodic request. Unlike the other server-based techniques, the sporadic server schedules the replenishment only when budget is consumed, and for an amount equal to the consumed budget. These replenishment rules, from a scheduling point of view, make the impact of SS equal to that of a normal periodic task with a period T_s and execution time C_S. However, the implementation of SS is more complex than that of DS.

An important remark, although some of the aforementioned techniques are able to improve the average response time of aperiodic tasks, it has been proven in [55] that there are no optimal algorithms that can both minimize the response time of every aperiodic request while guaranteeing the schedulability of a periodic task-set ordered according to a fixed-priority scheme. Hence, there is always a trade-off between response time and schedulability when using server-based scheduling solutions.

There are similar techniques to handle aperiodic events based on dynamic-priority scheduling, which provide better utilization bounds when compared to the fixed-priority ones. Many of these solutions are enhanced versions of the server-based algorithms such as, for example, an extension of the sporadic server called the Dynamic Sporadic Server (DSS) [56] and the Dynamic Priority Exchange Server (DPE) [56], an enhancement of the priority exchange server. The reader is referred to [49] for a more detailed overview and analysis of this matter.

3.1.4 Schedulability analysis

Schedulability analysis allow to determine *a priori* if a given task-set under a certain scheduling policy is feasible, *i.e.*, if all tasks will meet their deadlines at runtime. There are several approaches in the literature such as, for example, utilization and response time based tests [49]. Utilization-based tests are typically less computationally complex and faster when compared to other approaches, however, they can exhibit a high level of pessimism. Tests based on the analysis of response times are usually less pessimistic and can provide response time bounds for each task. However, these are more complex and not suitable for dynamic scheduling systems with low processing power.

Utilization-based tests

In general, utilization-based tests compute the processor utilization factor U, *i.e.* the fraction of processor time, for the execution of a given task set Γ and compare it to a certain upper bound. Only sets with utilization factors bellow the threshold are considered to be schedulable. This threshold depends on the task set, *i.e.* particular relations among tasks' periods and deadlines, and on the employed scheduling algorithm. Nonetheless, if a task set exhibits an utilization greater than 1.0, it cannot be scheduled by any algorithm.

A seminal work on utilization-based scheduling is the one presented by Lui and Layland in [50] for periodic task sets under the Rate Monotonic (RM) scheduling policy. The task model consists of independent periodic tasks with relative deadlines equal to their periods. Moreover, all tasks are considered to be released at the same time, which corresponds to the critical instant for all tasks under RM. Tasks can be preempted and the resulting system overhead is considered to be negligible[1]. The utilization factor U for a set Γ with n tasks is given by:

$$U = \sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) \tag{3.8}$$

According to [50], a given task set is guaranteed to have a feasible schedule under RM if the *sufficient test* expressed by Inequality 3.9 is true. Note that the right side of the inequality corresponds to the least upper bound for processor utilization.

$$U < n \times \left(2^{\frac{1}{n}} - 1 \right) \tag{3.9}$$

Response time analysis

Response time schedulability tests compute the worst-case response time for all tasks in the set and compare it with the respective deadlines. If the worst-case response time of any task in the set is bigger than the task's deadline, the set is deemed not schedulable.

An important response time test for fixed-priority preemptive systems was presented by Joseph and Pandya in [57]. According to them, the longest response time R_i of a periodic task τ_i is given by the sum of its worst computation time C_i and the amount of interference I_i that it can suffer from the higher priority tasks in the system. That is:

$$R_i = C_i + I_i \tag{3.10}$$

The maximum interference happens at the critical instant which, for a set of periodic tasks, occurs when task I_i is released with all other higher priority tasks ($hp(i)$) at the same instant. The maximum interference is computed as:

$$I_i = \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \tag{3.11}$$

[1]When required, this overhead can be accounted for in the analysis.

Combining Equation 3.10 and Equation 3.11 yields:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \tag{3.12}$$

Equation 3.12 can be solved by iterating the following equation:

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j \right) \tag{3.13}$$

where the iteration starts with $R_i^0 = 0$ and stops when $R_i^{n+1} = R_i^n$ or when R_i^n exceeds the task's deadline. The present analysis was improved by Audsley *et al.* in [58] to address the effect on non-preemption. They extended Equation 3.10 to include blocking B_i introduced by lower priority tasks executions:

$$R_i = C_i + I_i + B_i \tag{3.14}$$

The enhanced equation can be solved using the same iterative technique. The blocking factor is computed as follows:

$$B_i = \begin{cases} 0, & lep(i) = \emptyset \\ max_{j \in lep(i)}\{C_j\}, & lep(i) \neq \emptyset \end{cases} \tag{3.15}$$

An important remark, due to the presence of the blocking factor the critical instant is re-defined and now occurs when the task τ_i and all other higher priority tasks are released just after the instant in which the lower priority task that can block τ_i the longest starts its execution.

3.1.5 Hierarchical scheduling

The scheduling techniques introduced in Section 3.1.3 are able to provide good system utilization while assuring temporal correctness, however, they share a weakness: the schedulability analysis must be performed globally, *i.e.* by analyzing all applications in the system together. Although this necessity imposes no limitations when the system is *closed*, *i.e.* when all applications that can run at the same time are determined *a priori*, it does so for open systems [59], [60]. This is particularly important since many modern systems, *e.g.* automotive domain, have become extremely complex and open, often employing concurrent development techniques to design and validate applications independently. Scheduling these systems is no easy task since users may request, during run-time, the execution of real-time applications whose schedulability has not been analyzed together with the current combination of executing applications. This is aggravated by the fact that many properties of applications in the system may be unknown. To address this issue, new scheduling approaches have been researched, typically known as hierarchical schedulers or Hierarchical Scheduling Framework (HSF). The main components and structure of such framework is depicted in Figure 3.2 [61], [62].

A HSF is generally represented as a tree (hierarchy) of nodes (components), each representing an application with its own scheduler for scheduling internal workloads, *i.e.* tasks. Components model their internal real-time requirements as a single real-time requirement called the real-time interface. This interface can be modeled, for example, as the standard periodic model

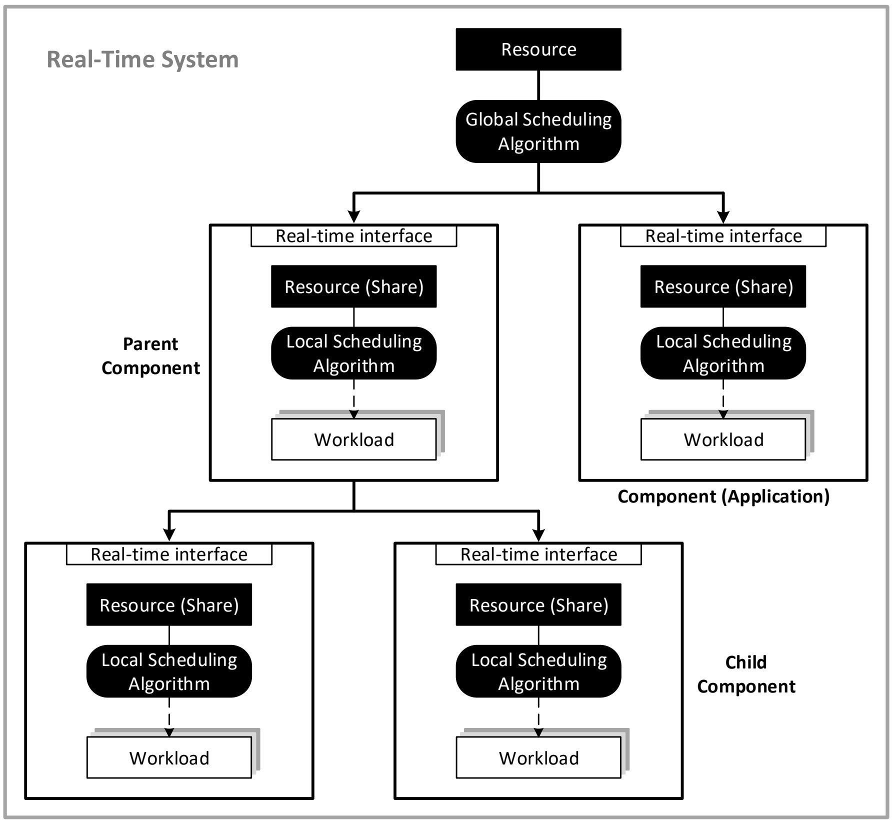


Figure 3.2: Example of a Hierarchical Scheduling Framework

(Section 3.1.2). Parent components use this interface to provide resource allocations to its child components without the need to control or even understand how child components schedule resources for their own tasks. For example, following the periodic model, a component can be treated by its parent as a single periodic task. As long as the parent component satisfies the resource requirements imposed by this task, the resource demands of the child's entire workload is also satisfied. A given parent will in turn combine all child interfaces into a single real-time interface. At system-level, a global scheduler assigns resources to all parent components at the top of the hierarchy according to a certain scheduling policy.

With this approach, complex and large real-time systems can be decomposed into several subsystems, *i.e.* components, whose timing properties can be analyzed and verified independently. These can then be assembled hierarchically such that timing properties established at the component level are also hold at the system level. According to Shin and Lee [61], [62] a hierarchical scheduling framework must fulfill the following properties:

- ***Independence***. The schedulability analysis for each component must be performed independently of other components at the same hierarchical level;
- ***Separation***. Parent and child components only interact through an interface that abstracts their internal complexity;

- ***Universality***. Any scheduling algorithm can be employed by each component;
- ***Composability***. A parent component is schedulable if and only if the timing requirements of all its childs are satisfied.

Server-based architectures are commonly employed to achieve these properties since they are an effective technique to perform controlled resource sharing. In particular, they are able to transparently provide virtual resources to HSF components that are a fraction of the capacity of the underlying hardware resources, and enforce mutual temporal isolation. Therefore, servers can easily provide suitable component scheduling interfaces.

An important HSF was presented by Shin and Lee [61], [62]. In it, they define a component C as a triplet (W, R, A), where W describes the workload, R the available resources, and A the scheduling algorithm used to schedule the resources among the workload. Then, they specified two functions: (i) $dbf_A(W, t)$, the demand bound function that quantifies the maximum possible workload W that can be submitted to the resource R, managed under the scheduling policy A, during a certain time interval t, and (ii) $sbf_R(t)$, the supply bound function that computes the minimum possible resources that R provides throughout the course of time interval t. The resource R satisfies the submitted workload W if:

$$dbf_A(W, t) \leq sbf_R, \quad \forall t \in \mathbb{R}^+ \tag{3.16}$$

Shin and Lee also define a real-time interface $\Gamma(\Pi, \Theta)$, based on the periodic model, which characterizes the periodic allocation of the resource R by workload W. Therefore, $\Gamma(\Pi, \Theta)$ represents a resource supply that periodically, every Π time units, provides Θ resource units. Therefore, the supply bound function can be defined as:

$$sbf_\Pi(t) = \begin{cases} b\Theta + max\{0,\ t - a - b\Pi\}, & t \geq \Pi - \Theta \\ 0, & t < \Pi - \Theta \end{cases} \tag{3.17}$$

where:

$$a = 2(\Pi - \Theta),\ b = \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor \tag{3.18}$$

Arvind *et al.* [63] generalized the $\Gamma = (\Pi, \Theta)$ for hierarchical frameworks defining explicit deadlines. The proposed model, known as explicit deadline periodic (EDP) model $\Omega = (\Pi, \Theta, \Delta)$, specifies a resource supply that provides every Π time units, Θ resource units during Δ time units, where $\Delta \leq \Pi$. The periodic model $\Gamma = (\Pi, \Theta)$ is defined in EDP model as (Π, Θ, Π). The supply bound function $sbf_\Omega(t)$ can be defined as:

$$sbf_\Omega(t) = \begin{cases} b\Theta + max\{0,\ t - a - b\Pi\}, & t \geq \Delta - \Theta \\ 0, & t < \Delta - \Theta \end{cases} \tag{3.19}$$

where:

$$a = (\Pi + \Delta - 2\Theta),\ b = \left\lfloor \frac{t - (\Delta - \Theta)}{\Pi} \right\rfloor \tag{3.20}$$

3.2 On real-time communications

Many real-time systems, such as those found in the modern industrial applications presented in Chapter 2, use a distributed architecture in which a set of nodes, that may be deployed in geographically separate locations, are interconnected by a communication system (network) in order to exchange information and cooperate towards a common goal. These systems, commonly known as real-time distributed systems, integrate time-constrained activities which include both task executions at nodes and the exchange of messages over the network. Therefore, the proper temporal behavior of the whole system depends not only on the timeliness of tasks executing on each processing device but also the capacity of the underlying communication system to provide the delivery of messages within specific timeliness requirements. Communication systems that are able to support such temporal requirements are known as real-time communication systems [64].

Real-time communication systems must be properly designed. If the network is overloaded, *i.e.* the demand for resources exceeds the maximum capacity, or isn't able to fulfill the Quality-of-Service (QoS) requirements of each communication stream, *e.g.* delay and drop rate, the whole system may experience a performance degradation and a partial or global system failure. [65]. The amount of network resources and the type of QoS requirements are heavily dependent on the type of the system and its applications. For example, file transfer applications prioritize bandwidth with low regard for transmission delays while closed-loop control systems demand low bandwidth but strict timing requirements, such as very low delay values and jitter [65]. The remainder of this section addresses some important issues concerning real-time communication.

3.2.1 Transaction activation paradigms

In real-time communication systems there are two different paradigms that can be used to initiate message exchanges [64]:

- ***Time-triggered.*** The time-triggered paradigm infers that a given message transmission is only triggered at a specific, predefined point in time within the communication system. In order for this to be possible, all system nodes must have a common time reference and follow a predefined scheduling table which defines the trigger points for each message. The scheduling tables can be built prior to transmissions dynamically, *i.e.* while the system is active, or offline, *i.e.* before the system startup. Scheduling tables are built by schedulers who coordinate the exchanges as to optimize the network resources and respect all timing requirements;

- ***Event-triggered.*** Following the event-triggered paradigm, message transmissions are triggered by the system nodes upon a given asynchronous event, *e.g.* the detection of movement by a sensor. In contrast to the time-triggered paradigm, the communication control is now external to the communication system. Due to the fact that these events may occur at undetermined time instants and out of the network's control it is not possible to plan ahead the instants in which communications will occur, leading to contention on the access to the resources. However, it is still possible to determine upper-bounds for the response time of messages provided that a minimum set of information is available, *e.g.* transmission time and minimum inter-arrival time between consecutive transmissions.

Due to the transmissions' scheduled nature, time-triggered communications are able to support periodic messages with low transmission latency and jitter as long as the entire system is synchronized and the scheduling of messages respected, otherwise high latency values and jitter may be induced. For example, if an imperfectly synchronized node produces a message immediately after the scheduled transmission time instant, this message will be delayed and sent in the next scheduled instant. Thus, time-triggered approaches are better suited for fully synchronized loop-control systems [66]. As event-triggered messages may be dispatched at any time, they may be handled with low latency. However, due to contention, messages may suffer interference from other ones, resulting in potentially high latency and jitter values.

In many real-time systems, both kinds of events occur naturally, *e.g.* automotive and industrial systems. Supporting a combination of both event-triggered and time-triggered communication would bring benefits for the system, such as higher flexibility and reduced costs. However, in order for both paradigms to coexist, isolation, *e.g.* temporal, between the two traffic types must be introduced to prevent the asynchronous nature of event-triggered from interfering with the time-triggered traffic performance.

3.2.2 Message scheduling

In distributed systems, nodes commonly exchange data messages through a shared medium network. Therefore, as with tasks in systems with a single processor, the access to the network must be properly scheduled in order to fulfill the timing requirements of all applications.

Message scheduling in communication networks has several similarities with task scheduling: messages can also be categorized according to timing constraints (non real-time, soft, or hard) and the activation pattern (periodic, sporadic, or aperiodic). As such, techniques employed for task scheduling, such as the periodic model scheduling algorithms, and scheduling analyses, can also be applied to the network domain. In fact, this has been extensively done in the literature, with examples including [6], [25], [67]. However, distributed real-time systems pose additional challenges. For example, resource schedulers need to keep the status of all resources update in order to accurately control their usage. However, due to the distributed nature of the system, knowledge concerning nodes and network state is not easy to gather and is not always available. Therefore, scheduling decisions typically have to be taken based on incomplete information. Additionally, data units (frames) transmitted by networks are typically non-preemptive and thus, preemptive scheduling techniques which in average provide higher schedulability cannot be used, leading to a penalty on efficiency.

3.3 Software Defined Networking Paradigm

As discussed in Chapter 2, Software Defined Networking (SDN) can be an important technology to manage networks of complex and highly dynamic applications such as those found in Industry 4.0. Therefore, the key question is *"what is SDN all about?"*. According to Nadeau [68], SDN: "functionally enables the network to be accessed by operators programmatically, allowing for automated management and orchestration techniques; application of configuration policy across multiple routers, switches, and servers; and the decoupling of the application that performs these operations from the network device's operating system."

Historically, network management was often performed manually, with configurations set up via Command-Line Interface (CLI) on a device-by-device basis. This rendered the management of networks a complex a difficult task, in particular for large and complex networks comprising many kinds of equipment, from routers and switches to firewalls and server load-balancers. To aggravate the complexity, network administrators typically had to configure individual network devices using configuration interfaces and tools that vary across vendors—and even across different products from the same vendor. Although some network management tools, e.g. CiscoWorks LAN Management Solution [69], offer a central configuration point, these still operate at the level of individual protocols and configuration interfaces. Therefore, once deployed, networks are largely static and commonly untouchable. This mode of operation has slowed innovation, increased complexity, and increased both the capital and operational costs of running a network [68], [70].

SDN is a network management architecture that emerged from the need to solve the complexity in the management of data center networks and provide the flexibility required by emerging applications such as server virtualization and cloud computing. It has been receiving considerable attention from industry and academia [71], and exhibits two particular features that are very well suited to manage a network with real-time requirements: (i) a centralized resource control, completely decoupled from the data plane, and (ii), a fine granular resource control, down to validating each single frame received in each port of a switch. The main components of a SDN architecture are depicted in Figure 3.3.

SDN network resource control is performed by a central entity named controller, that centrally performs admission control of the new traffic flows, and takes decisions about the network resource management. Being a central entity performing the resource management, the controller keeps the complete information about network topology and resource allocations. SDN controllers interact with data plane entities through the so-called southbound interface, installing rules that dictate, for example, how traffic is to be forwarded/routed throughout the network. At the data plane, network elements process traffic according to the rules instantiated by the controller. The southbound interface creates an abstraction that allows to avoid the difficulties imposed by vendor specific management technologies. The network controller is also referred in the literature as the network operating system, entity to which network applications request network services through the northbound interface. One of the common API technologies used at the northbound interface is the Representational State

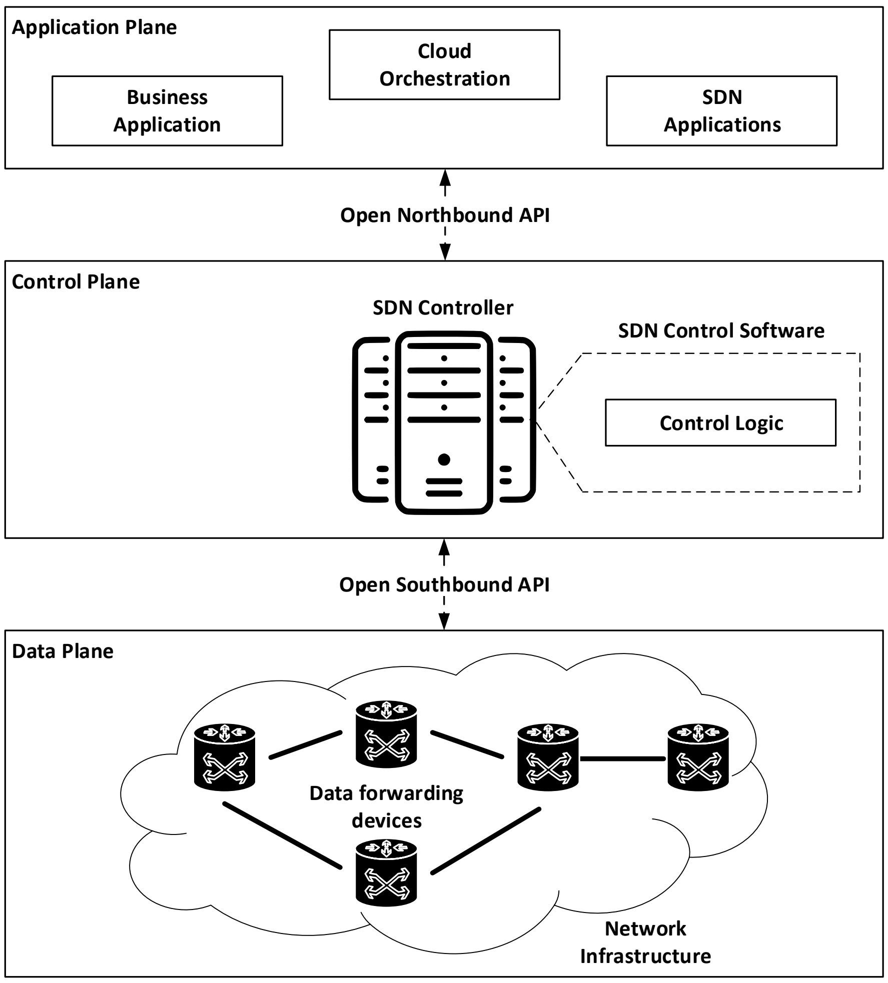


Figure 3.3: Generic SDN architecture

Transfer (REST) API that uses HTTP/HTTPS protocols to execute common operations on resources represented by Uniform Resource Identifier (URI) strings. The northbound API offers a network abstraction interface to applications and management systems at the top of the SDN stack, promoting the innovation of the network management process and new business applications [68], [71].

SDN offers just minimal QoS support with different priorities and minimum guaranteed bandwidth, which are vital for cloud-based deployments. Further improvements are currently being considered. For example, the northbound interface lacks the specification of QoS requirements that are essential for applications such as multimedia. This led the Open Networking Foundation (ONF) to standardize the Real-Time Media NBI [72]. Other proposals already aim at enhancing the support for traffic with more demanding requirements but most are just at the level of conceptual ideas, under simulated or emulated scenarios[71]. As of today, albeit offering unprecedented control granularity, SDN still lacks adequate support for network applications that impose strict deadlines on network traffic delivery, such as many

IIoT applications. Noticeably, SDN's most popular southbound signaling protocol, OpenFlow, lacks mechanisms to specify real-time requirements.

3.3.1 OpenFlow protocol

The OpenFlow switch protocol [73], standardized by the ONF, is the most widely accepted and employed southbound for SDN-based networks. An SDN/OpenFlow architecture comprises two main elements: OpenFlow-enabled controllers and OpenFlow-enabled forwarding devices. OpenFlow controllers have the same role of being the "network brain" as traditional SDN controllers, however, they implement the necessary software stacks to communicate with and configure OpenFlow data plane devices using the standardized API and services. OpenFlow data plane devices are hardware (or software) entities that are specialized in packet forwarding and implement a forwarding datapath according to the standard. The main components of an OpenFlow data plane device (OpenFlow switch) are shown in Figure 3.4 [73].

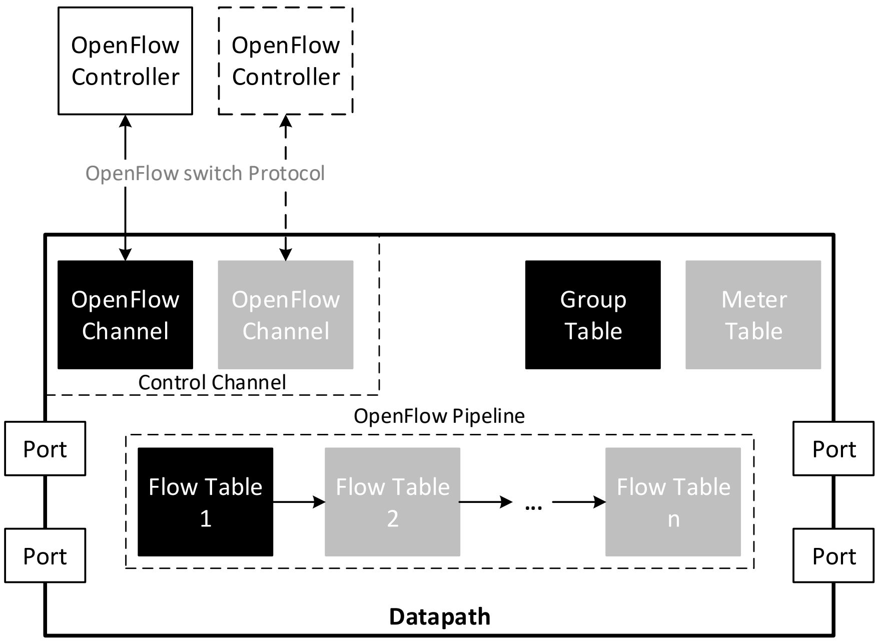


Figure 3.4: OpenFlow switch architecture

An Openflow switch consists of a forwarding pipeline comprising one or more flow tables, the *OpenFlow Pipeline*, a *group table*, and one or more *OpenFlow channels* to external controllers. An OpenFlow channel provides a reliable message delivery service based on Transport Layer Security (TLS) and implements the interface that connects each switch to an OpenFlow controller. Through this interface, a controller may configure and manage the switch, receive events from the switch, and send frames out the switch. The *Control Channel* component of the switch may support one or multiple OpenFlow channels, enabling multiple controllers to share the management of the device. A *meter table* may also be supported by the device's datapath. This table allows OpenFlow to forward traffic through programmable flow meters in order to implement simple QoS operations, such as rate-limiting. Each meter measures the rate of traffic flows assigned to it and can be configured to drop frames or increase the drop precedence of DiffServ flows upon reaching a programmable rate threshold. *OpenFlow ports* are abstractions of network interfaces that allow to receive/transmit traffic from/to the network as well as within internal processing. An OpenFlow device supports a set of OpenFlow ports which may not be identical to the set of network interfaces provided by the switch hardware. Three types of ports are defined: (i) physical, that correspond to a hardware interface of the switch, (ii) logical, which don't correspond directly to hardware interfaces and may be used to define and perform traffic processing using non-OpenFlow methods, and (iii), reserved ports, a subset of logical ports that are defined by the OpenFlow standard to enact certain processing actions such as, for example, sending a received frame to the controller. All ports have port counters, for statistic purposes such as registering the number of processed traffic frames, state, *e.g.* online or offline, and configuration parameters, *e.g.* drop all traffic. Finally, the *OpenFlow pipeline* and the *group table* implement the services that perform traffic look-ups and forwarding.

The group table comprises a set of entries (*group entries*), which enable OpenFlow to specify sets of actions (*buckets*) to facilitate complex operations such as fast-failover and link aggregation. These are not going to be addressed here in detail. The reader is referred to [73] for more information. Flow tables are the cornerstone of the OpenFlow pipeline processing. The structure of a flow table[2] is depicted in Figure 3.5 [73].

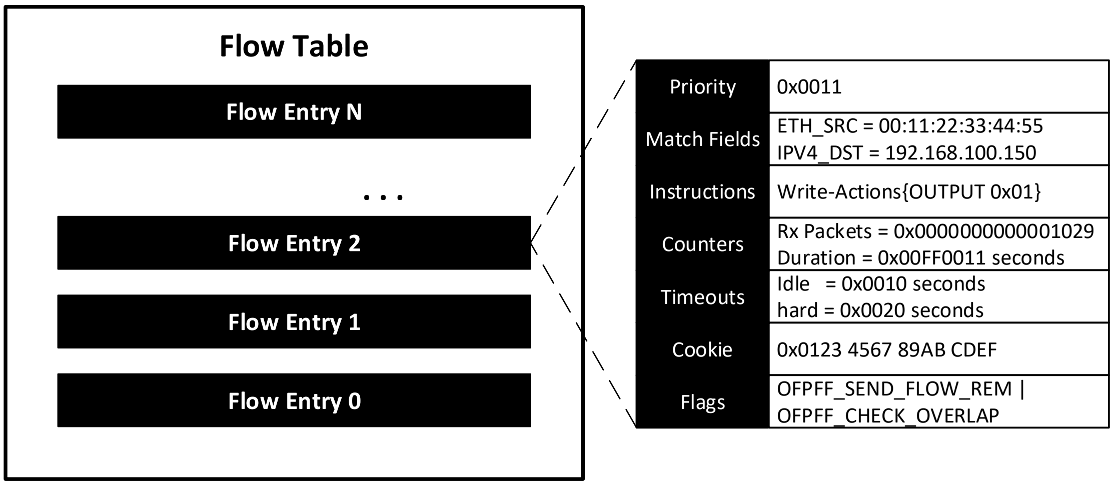


Figure 3.5: OpenFlow Flow Table

Every table in the pipeline comprises a set of *flow entries* that can be configured by an OpenFlow controller. Each flow entry contains a set of *match fields* that are used to filter frames according to several fields such as, for example, Ethernet and IPV4 addresses. When a

[2]Note that the presented figure is merely indicative, not all the possible instructions, actions, match fields, flags, and counters are illustrated

frame matches a given entry, *i.e.* when match fields and the corresponding frame fields are equal, a set of *instructions* associated with that particular entry are executed. The execution of these instructions may result in changes to a list of actions (the *action set*) associated to the frame, the modification of the frame itself, *e.g.* addition of VLAN tags, or may modify the pipeline processing, *e.g.* indicate the next table to process the frame. The order according to which frames are matched against flow entries in a particular table follows the matching precedence that is established by the priority field of each flow entry (higher value holds higher precedence level). Finally, flow entries also contain a cookie value that can be used by controllers to filter how some configuration commands affect certain entries, counters for statistic purposes, configuration flags, and timeout fields that can be used to specify the maximum liveness period for the entry.

3.3.1.1 OpenFlow pipeline processing

A simplified view of the pipeline processing is depicted in Figure 3.6 [73].

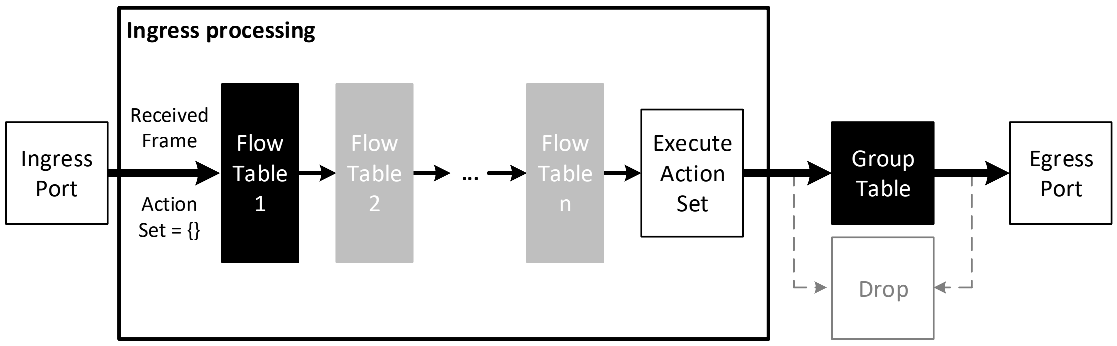


Figure 3.6: Simplified OpenFlow pipeline processing

Frames received at ingress ports are sent to the first flow table in the OpenFlow pipeline for ingress processing. The frame is then matched against the table's flow entries to select a flow entry. If there is a match, the instructions in the matched flow entry are executed. Besides modifying the frame or its action set, the performed instructions may also explicitly direct the frame to another subsequent flow table, where the same process is repeated. If the matching flow entry does not direct frames to another flow table, the current stage of pipeline processing stops and the frame's current action set is executed. From the execution of the action set a frame may either: (i) go through additional processing in the group table, (ii) be forwarded to a given OpenFlow egress port, or (iii), be dropped if no output/group action is present in the action set. The additional processing carried within the group table may as well either drop the frame or forward it to one or several OpenFlow egress ports. OpenFlow switches may optionally support an additional OpenFlow pipeline to enact processing within the context of a specific egress port. This egress pipeline operates akin to the ingress OpenFlow pipeline.

If a frame does not match a flow entry in a given flow table, an event known as *table miss* occurs. In this scenario, the frame is either dropped or, if configured, processed according to the instructions of a special entry known as *table-miss flow entry*. Table-miss flow entries can be configured by OpenFlow controllers to specify how traffic unmatched by other entries should be dealt with. These entries are similar to regular entries and support the same instructions and actions. However, table-miss flow entries wild-card all match fields and have the lower priority value (0). Controllers typically configure these entries to either drop traffic or send it to the controller, in a process named *Packet-in*. The controller may then analyze this traffic and, if it so decides, set up appropriate forwarding rules in the data plane.

3.3.1.2 OpenFlow actions

Concerning the action set, an OpenFlow switch may support the following actions [73]:

- ***Output port_no***. The Output action forwards frames to the specified OpenFlow port for egress processing;
- ***Group group_id***. Sends frame to be processed by the specified group in the group table;
- ***Drop***. There is no explicit action to represent drops, however, frames whose action sets have no Output and no Group action are dropped;
- ***Set-Queue queue_id***. Sets the egress queue id for a frame. Forwarding behavior is dictated outside the OpenFlow protocol by the services serving the specified queue;
- ***Meter meter_id***. Directs frame to the specified meter. The frame may be then dropped as result of the metering process;
- ***Push-Tag/Pop-Tag ethertype***. Push/pop a new header onto/from the frame. VLAN, MPLS, and PBB headers are specified;
- ***Set-Field field_type value***. Set the specified field to a certain value;
- ***Copy-Field src_field_type dst_field_type***. Copies data between any header fields;
- ***Change-TTL ttl***. Modifies the value of IPv4 TTL, IPv6 Hop Limit or MPLS TTL in the frame.

The *action set* associated with each frame is empty by default and may be filled with a list of *actions* by instructions of matched flow entries. In case an action set contains more than one action, these must be performed according to the following ordering [73]: (i) copy TTL inwards, (ii) apply all tag pop actions, (iii) apply MPLS, PBB, VLAN tag push actions following this order, (iv) copy TTL outwards, (v) decrement TTL, (vi) apply all set-field actions, (vii) apply all QoS actions, such as meter and set_queue, (viii) process frame according to the group action, and (ix), forward the frame to the port specified by the output action.

3.3.1.3 OpenFlow signalling messages

The OpenFlow switch protocol implements several classes of messages that allow a controller to: (i) query information regarding the capabilities and state of a given OpenFlow switch, (ii), configure the OpenFlow services of data plane devices, and (iii), receive events, e.g. flow entries expiration and port state changes, from data plane devices. The existing messages are categorized into three types: controller-to-switch, asynchronous, and symmetric.

Controller-to-switch
Controller-to-switch messages are initiated by the controller to directly manage os inspect the state of an OpenFlow switch, and may or may not require a response. For the configuration of data plane devices, the following controller-to-switch messages are defined:

- ***Switch configuration***. Can set and query switch configuration parameters, e.g. drop or reassemble IP fragments;
- ***Modify-state***. Allow the controller to add, delete, and modify flow/group entries, configure meters, and modify the behavior of OpenFlow ports and tables;
- ***Role-request***. Used by the controller to request a change in its role as master or slave controller;
- ***Asynchronous-configuration***. Allows to define which events are to be reported to the controller.

The operation state of switch devices can be monitored using the following messages:

- ***Read-State***. Messages used by the controller to collect switch information, such as current configuration, statistics and capabilities. These are typically implemented over multipart message sequences;
- ***Barrier***. To query message dependencies or to receive completed operation notifications;
- ***Features***. Used to request the identify and basic capabilities, e.g. number of flow tables, of a switch.

There is also a *Packet-out* message which the controller may use to send frames out of a specified port on the device. It is commonly used to forward frames received via *Packet-in* events. Controllers are able to group a set of messages into *Bundles* in order to enact several configurations/requests as a single transaction. All messages in a bundle must be able to successfully applied, otherwise, none of them are performed.

Asynchronous messages
Asynchronous messages are initiated by the switch without a controller soliciting them and they are used to update the controller of network events and changes to the switch state. Switches send asynchronous messages to controllers, for instance, to denote a new unregistered packet arrival or to notify a switch state change. OpenFlow asynchronous messages include:

- ***Packet-in***. Upon table miss events, switches may send the related frames to the controller via Packet-in messages;
- ***Flow-removed***. Message sent to the controller to notify a flow entry removal from a flow table;
- ***Flow-monitor***. Message sent to the controller to notify changes in a flow entry;
- ***Port-status***. Informs controllers of changes on a switch port, *e.g.* link has gone down;
- ***Role-status***. Message to notify a controller of a change in its role. For multi-controller scenarios;
- ***Controller-status***.Informs controllers of changes in OpenFlow channels. Used, for example, to notify controllers in a multi-controller scenario of the severance of a connection to a specific controller.

Symmetric messages
Symmetric messages are initiated by either the switch or the controller and sent without solicitation. These are:

- ***Hello***. Messages that are exchanged between the switch and controller upon connection startup;
- ***Echo***. Request/reply messages that are used to verify the liveness of a controller-switch connection;
- ***Error***. Messages used by the switch or the controller to notify problems to the other side of the connection. Normally used to report the failure of a request;
- ***Experimenter***. Messages that provide a standard way for OpenFlow devices to offer additional functionality that is not specified by the OpenFlow protocol;

3.3.2 Using SDN in industry

The unprecedented level of flexibility provided by SDN has raised interest on using it in industry as reported in the scientific literature, including qualitative and quantitative evaluations, as well as methods to extend its functionality and use it in real-time Ethernet networks. This section reviews some of the most relevant scientific contributions in this area.

3.3.2.1 Qualitative evaluations

Henneke *et al.* [48] and Ehrlich *et al.* [74] evaluate the use of SDN in future industrial networks and identify industrial communication requirements that are not adequately supported by SDN, yet. Both studies highlight the inability of existing southbound APIs in expressing industry-grade requirements, *e.g.* real-time timeliness requirements, as a limiting factor on enabling proper QoS provisioning and monitoring services. Ehrlich *et al.* [74] also identifies a set of ten requirements for future industrial networks (a superset of those identified in Section 2.3) and conclude that SDN already fulfills a subset of them, namely independence from underlying network technologies, support for online (re)configurations, and security-related features. Kálmán [75] overviews the most relevant characteristics of SDN and identifies possible ways to apply it in industrial Ethernet scenarios. The work refers significant advantages in network deployment, monitoring, and dynamic management brought by SDN centralized control.

3.3.2.2 Performance analysis

Thiele *et al.* [76] performed a formal analysis of the OpenFlow protocol (OFP) visualizing its deployment on a TSN Ethernet network. The work focus on the actual management flow, not on real-time traffic performance, and considers the network topology, the communication with the controller, and respective scalability limits. Nonetheless, the analysis shows that attaining latency values below 50ms is possible. Herlich *et al.* [77] highlight the possible gains in supporting arbitrary network topologies, dynamic (re)configurations, and fast fail-over by using SDN on real-time Ethernet networks. Experiments based on a virtual platform with OpenFlow deployed on Ethernet POWERLINK (EPL) networks show fail-over operations without packet losses.

3.3.2.3 Extensions and use in real-time networks

In [78], Ternon *et al.* investigate how the FTT paradigm can be instantiated on standard OpenFlow hardware making it suitable for real-time scenarios. The paper presents a new protocol, FTT-OpenFlow, that enhances the response time of sporadic real-time traffic and of non-real-time traffic. They show analytically that the proposed solution meets the requirements of an avionics scenario. However, this work targets single switch topologies and does not address its applicability to multi-hop scenarios. In [79], Nayak *et al.* exploit the logical centralization of SDN to build a global view of the network and compute routes and transmission schedules that reduce in-network queuing of time-triggered traffic. The central controller uses OpenFlow to configure the data plane devices and enforce the computed routes, and transmits the schedule to the source nodes that synchronize their transmissions according to the assigned temporal slots. Despite obtaining low latency and jitter, the proposal does not address coexistence with sporadic real-time traffic. Ahmed *et al.* [80] propose SDPROFINET, a SDN deployment over PROFINET networks. Similarly to [79], the central controller acquires network information and configures the data plane PROFINET data channels according to the desired routes. Despite the gains in network management, operational flexibility is constrained by PROFINET. In [81], Ishimori *et al.* propose a hierarchical scheduling approach, similar to that of Linux Traffic Control (TC), to overcome the limitations of the FIFO queues in OpenFlow devices. It supports HTB (Hierarchical Token Bucket), RED (Randomly Early Detection), and SFQ (Stochastic Fair Queuing). However, these policies only provide bandwidth-based traffic shaping and thus, explicit support to real-time traffic is still poor.

CHAPTER 4

Towards a real-time data plane

Contents

*A*s discussed in Chapter 2, communication networks, in particular networks based on the Ethernet technology and on the standard TCP/IP stack, are an extremely important component of present and future industrial systems. The boundaries for services that such networks must provide in the future are being pushed forward, with new applications demanding even more strict requirements in terms of flexibility, reconfiguration and timeliness guarantees. This chapter presents a survey of existing real-time Ethernet technologies and stacks them up in order to find the best candidate for a data plane that is able to fulfill the aforementioned requirements. The chapter starts by detailing switched Ethernet, and its limitations. Then, a survey on existing real-time technologies based on switched Ethernet is presented. Finally, the features of the most relevant technologies are compared, and the most promising ones explained in detail at the end of the chapter.

4.1 Switched Ethernet

Ethernet bridges, also commonly known as switches, were first standardized in 1990 as the IEEE 802.1D standard [82] in an attempt to solve the performance issues that plagued shared Ethernet networks, in particular its low level of bandwidth utilization and lack of determinism that stemmed from the use of a shared multi-access bus and the non-deterministic nature of the CSMA/CD arbitration algorithm.

Switched Ethernet networks maintain the same segmented physical architecture as hub-based Ethernet, with stations now connected to bridges via a full-duplex point-to-point link, in a star/tree layout. Unlike Ethernet hubs, bridges operate at the second layer of the Open Systems Interconnection (OSI) reference model, *i.e.* data link layer, and run a learning mechanism that allows them to only forward received frames through the link connected to the destination station, avoiding unnecessary broadcasts and bandwidth waste. Moreover, by eliminating physical, direct connections between ports, and by queuing frames at the output ports until the medium is idle, bridges define a private collision domain for each of its ports. If there is a single station per network segment connected to a bridge port, an arrangement known as micro-segmentation, no collisions can occur. Switched Ethernet networks exploit and incorporate this micro-segmentation into their topologies in order to effectively sidestep the performance issues associated with the CSMA/CD protocol [13], [22]. An example of such topologies is depicted in Figure 4.1. Bridges also overcome the limits on total segments between two hosts and support mixed speeds at its interfaces.

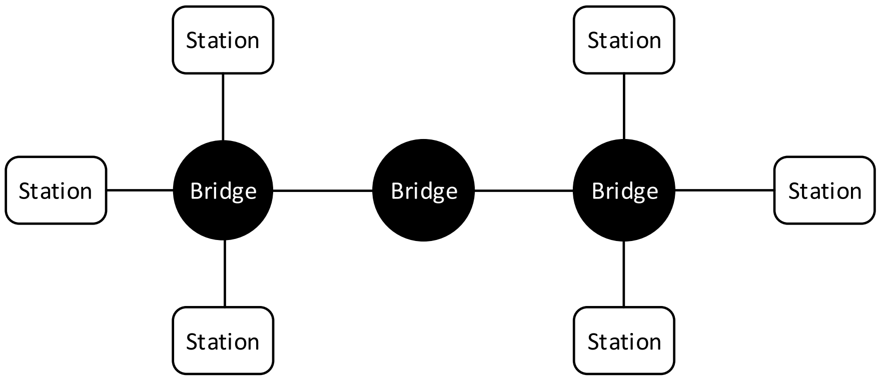


Figure 4.1: Switched Ethernet network topology example

Switched Ethernet networks keep the original Ethernet frame structure untainted. Its format, including overheads from the physical layer of the OSI model, is shown in Figure 4.2 [10].

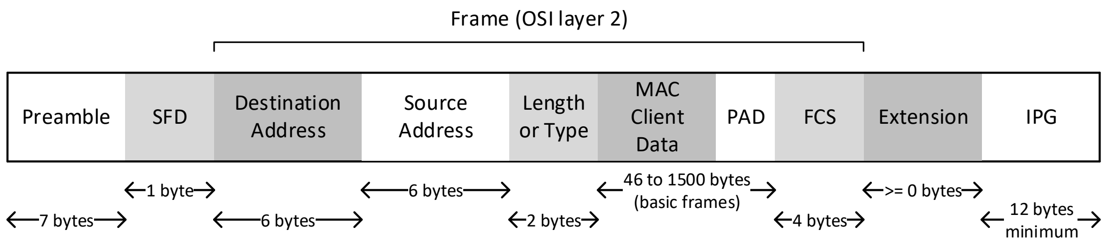


Figure 4.2: Ethernet (IEEE 802.3) frame structure

The function of each frame field is the following [10]:

- ***Preamble***: alternating sequence of '1' and '0' bits used for the synchronization of the receiver's clocks;
- ***Start Frame Delimiter (SFD)***: "10101011" sequence that marks the end of the preamble and the start of the Ethernet frame;

- ***MAC Destination Address***: address of the frame's destination network interface;
- ***MAC Source Address***: address of the frame's source network interface;
- ***Length/Type***: indicates the length of the payload in bytes, or the protocol encapsulated in the payload, *e.g.* 0x0800 for IPv4 datagrams;
- ***MAC Client Data***: contains the user data. The maximum supported size is 1500 bytes for basic frames, 1504 bytes for Q-tagged frames [23] or 1982 bytes for encapsulation services requiring extra tags [10];
- ***PAD***: contains, if required, padding to guarantee the imposed minimum frame size of 64 bytes (excludes the size of preamble and SFD fields);
- ***Frame Check Sequence (FCS)***: contains a cyclic redundancy check value to detect errors in a received MAC frame;
- ***Extension***: extension bits only required for 1000 Mbps half duplex operation modes;
- ***Inter-Packet Gap (IPG)***: idle time between consecutive frames (a minimum of 96 bits).

The first Ethernet bridges, as defined by IEEE 802.1D [82], had limited support for the segregation of traffic into classes associated to different QoS requirements and relied on in-band mechanisms present in some IEEE 802 technologies, *e.g.* IEEE Std 802.5 token-passing ring and Fiber Distributed Data Interface (FDDI), to convey priority information. To address this issue, a set of standards have been proposed to extend the original Ethernet frame structure and provide traffic differentiation. Such amendments include VLAN tagging (IEEE 802.1Q) and a priority identifier (IEEE 802.1p and IEEE Std 802.1AC)[1]. The format of a VLAN-tagged Ethernet frame is depicted in Figure 4.3 [23].
The function of each field found on the additional 802.1Q tag is:

- ***Tag Protocol Identifier (TPID)***: identifies the new framing format and has the fixed value of 0x8100. It is located at the same position as the original Length/Type field of untagged frames;
- ***Priority Code Point (PCP)***: indicate the frame's priority level, from 0 through 7;
- ***Drop eligible indicator (DEI)***: can be used separately, on in conjunction with PCP, to encode the frame's eligibility to be dropped in the presence of congestion;
- ***VLAN Identifier (VID)***: specifies the VLAN to which the frame belongs. The reserved value 0x000 indicates that the 802.1Q tag only conveys a priority level, *i.e.* frame does not belong to a specific VLAN.

[1]These amendments are now part of the IEEE 802.1Q-2014 standard

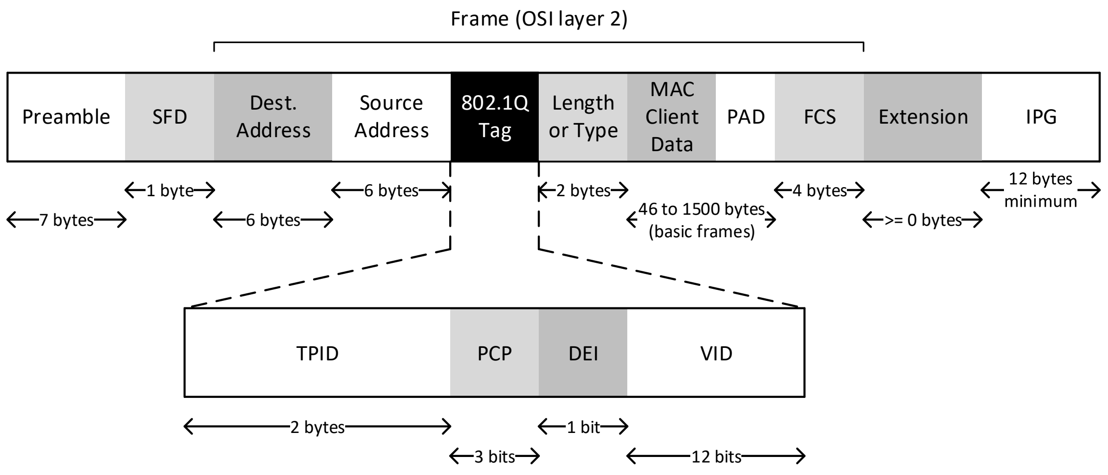


Figure 4.3: IEEE 802.1Q VLAN-tagged Ethernet frame structure

A notable remark: the PCP value does not necessarily correspond to an effective and absolute priority level throughout the network. The value conveyed by PCP is translated by the *Traffic Class Table* associated with each potential egress port into a traffic class numbered from 0 through 7, with lowest and highest dispatching priority levels respectively. Traffic class tables are configurable, and may translate priority levels to different traffic classes in different bridges along the network. Priority level 7 is usually translated into the highest priority traffic class 7. However, due to legacy and interoperability reasons [23], priority level 1 is typically translated into traffic class 0, while priority 0 is associated to traffic class 1 (see Table 4.1). This means that priority level 1 typically conveys the lowest priority.

Bridges that support the IEEE 802.1Q VLAN tagging and associated services, *e.g.* traffic segregation by priority, are referred to as VLAN bridges while traditional bridges are defined as MAC bridges [23]. Focus will be placed only on VLAN bridges for the remaining of the document since they are nowadays common and essential for Time-Sensitive Networking (TSN) networks.

Regarding the addressing space, bridges commonly support the three Ethernet address types [23]:

- ***Unicast***: a unicast transmission implies directing frames from a source station directly to the destination station. To this end, bridges observe the source address of incoming frames and send them to the output port (egress port) that is connected to the destination;

- ***Broadcast***: a broadcast transmission implies forwarding frames from a source station to all other stations simultaneously. Ethernet bridges recognize this address (0xFFFFFFFFFFFF) and forward broadcast traffic to all ports but the port from which it was received (ingress port);

- ***Multicast***: a multicast transmission implies sending frames from a source station to a group of stations. IEEE 802.3 defines a special set of addresses that can be used to define multicast groups. However, multicast protocols operate at the layer 3 of the OSI model and thus, a pure layer 2 bridge handles multicast addresses as broadcasts. There are, however, bridges that support layer 3 services and are able to properly handle multicast addresses.

In order to perform traffic forwarding, bridges perform a set of functions known as the *Forwarding Process*. An overview of its main functions and components[2] is depicted in Figure 4.4 [23].

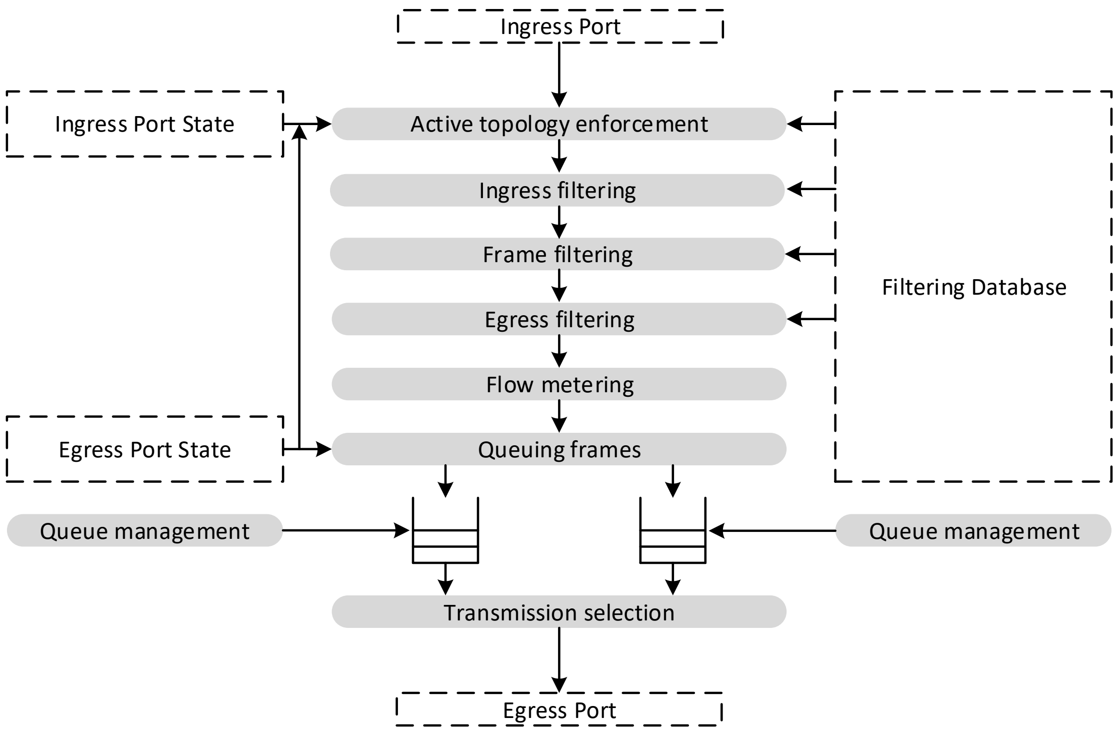


Figure 4.4: The *Forwarding Process* of Ethernet bridges

Frames are received at a given ingress port, processed by the *Forwarding Process* pipeline, and may be eventually transmitted through one or several egress ports. Although most pipeline functions rely on information conveyed by the frame's headers, *e.g.* VID and MAC addresses, some also require information regarding the state of ingress and egress ports, as well as filtering information stored in the Filtering Database (FDB). Port states, *e.g.* disabled or forwarding, dictate if a given port is forwarding traffic and/or participating in the *Learning Process*, and may be controlled administratively or by active topology management protocols such as Rapid Spanning Tree algorithm and Protocol (RSTP). The FDB contains two types of filtering information: (i) static, which is added to, modified, and removed

[2]As defined in IEEE 802.1Q-2014 [23]. Some functions were extended by separate amendments, in particular, Time-Sensitive Networking (TSN)-related enhancements (see Section 4.3.1)

from the FDB by explicit management actions performed through configuration consoles or network management protocols such as Simple Network Management Protocol (SNMP) and Network Configuration Protocol (NETCONF), and (ii), dynamic information which is built dynamically by the bridges' internal *Learning Process* [23]. The *Learning Process* observes the MAC addresses and the VLAN Identifier (VID) of ingress frames to create or update dynamic filtering information. Both static and dynamic filtering entries contain:

- ***A MAC address specification***, comprising a single or multiple individual/group MAC destination address(es);
- ***A VID specification***, containing the VID, or set of VIDs, of specific VLANs to which the entry applies;
- ***A port map***, with a control element for each egress port, specifying if frames, matching the aforementioned address and VID specifications, are to be forwarded through the port or filtered, *i.e.* discarded.

The *Forwarding Process* starts with an active topology enforcement procedure that, based on the state of ports and filtering information on the FDB, determines if a given received frame is to be submitted to the *Learning Process* or dropped in order to prevent unwanted learning of MAC addresses and path loops. Frames are afterwards inspected by an ingress filtering which, if enabled, discards frames with a VID that is not associated with the ingress port. Next, the *Forwarding Process* queries the FDB and sets the potential egress ports for each frame. Ports that aren't related with the frame's VID are removed from the set of egress ports by the following egress filtering operation. Frames may then be subjected to flow classification and metering[3] by rules based on VID, MAC addresses and priority. Flow meters may discard frames based on the DEI bit, frame size, and maximum bandwidth. After metering, each frame is queued to each of the potential egress ports. For each port, the *Forwarding Process* provides one or more First-In First-Out (FIFO)-like queues, each queue corresponding to a distinct traffic class, and a configurable Traffic Class Table that maps frames into a traffic class according to their priority level. The recommended priority to traffic class mapping is shown in Table 4.1 [23]. Up to eight traffic classes may be supported, allowing separate queues for each priority level.

Finally, each queue is associated with a transmission selection algorithm that select frames for transmission if and only if: (i) the operation of the algorithm determines that there is a frame available for transmission, and (ii), the transmission algorithm of numerically higher traffic classes, *i.e.* with higher priority, determines that there are no frames available for transmission. On each port, a *Transmission Selection Algorithm Table* assigns, for each traffic class, the algorithm that is to be used.

[3]Although flow metering is enforced after egress filtering, the applied meters operate per ingress port, not per potential egress ports.

Table 4.1: Recommended priority mapping for the number of implemented traffic classes

		Available Traffic Classes							
Traffic Category	Priority	1	2	3	4	5	6	7	8
Best Effort	0 (Default)	0	0	0	0	0	1	1	1
Background	1	0	0	0	0	0	0	0	0
Excellent Effort	2	0	0	0	1	1	2	2	2
Critical Applications	3	0	0	0	1	1	2	3	3
Video (< 100 ms latency and jitter)	4	0	1	1	2	2	3	4	4
Voice (< 10 ms latency and jitter)	5	0	1	1	2	2	3	4	5
Inter-network Control	6	0	1	2	3	3	4	5	6
Network Control	7	0	1	2	3	4	5	6	7

Three algorithms are currently defined by IEEE 802.1Q-2014 [23]:

- ***Strict priority***: determines that there is a frame available if the queue contains one or more frames;
- ***Credit-Based Shaper (CBS)***[4]: a frame is available for transmission if the queue contains one or more frames, and the shaper credit is zero or positive;
- ***Enhanced Transmission Selection (ETS)***: based on the allocation of bandwidth to traffic classes, it determines that there is a frame for transmission if: (i) the queue is not empty, (ii) the class has not surpassed the allocated bandwidth, (iii) there are no frames available for transmission for any queues running strict priority or CBS algorithms.

With respect to the way the *Forwarding Process* is triggered, two schemes are commonly employed:

- ***Store-and-forward***: in this method the bridge waits until the full frame is received before executing the *Forwarding Process*. This allows for the verification of the frame's data using the FCS field. If the data is found erroneous, the frame can be discarded, preventing error propagation in the network. However, this process induces additional end-to-end transmission latency;
- ***Cut-through***: the bridge starts the *Forwarding Process* as soon as the MAC addresses, and VID for VLAN-tagged frames, are received and analyzed. This allows for lower latency values but it is less resource-efficient since it can propagate erroneous packets through the network, wasting bandwidth for information that will be nevertheless discarded by the receiver.

4.1.1 Limitations

Although the use of Ethernet switches brought a huge improvement on global throughput, traffic isolation, and can be used to build network topologies free from the non-determinism

[4]Explained in detail in Section 4.3.1.1 due to its origin (IEEE 802.1Qav) and common association with Time-Sensitive Networking

inherent to the CSMA/CD algorithm, there are still other phenomena that negatively impact the real-time performance of switched Ethernet networks. Collisions and retransmissions, which are associated to induced forwarding delays and jitter, are effectively eliminated by switching technology. However, a full prediction of bounds for traffic transmission times is still not possible, in most part due to queuing and congestion issues at switches' output ports, particularly under heavy load conditions and multilevel topologies [22], [83]. For example, in interlinks, *i.e.* single links interconnecting two switch devices, the traffic received by multiple ingress ports of a given switch may easily exceed the transmission rate of the interlink. In this scenario, queues at the interlink egress port may overflow and frames can be dropped. Moreover, switched Ethernet has a limited and small number of supported traffic classes and respective queues: only a maximum of eight queues is standardized, from which only five or six are practically available for real-time traffic since the former, the background traffic, and the traffic related to network control and management must be segregated into distinct classes. This number is insufficient to implement efficient priority-based scheduling policies, in particular for large sets of real-time streams [84]. Also, many real-time applications, *e.g.* automotive and aeronautical, demand efficient handling and coexistence of both periodic and sporadic traffic; this requires a level of isolation, *e.g.* temporal, that switched Ethernet does not provide.

4.2 An overview of real-time Ethernet technologies

In pursuance of real-time behavior on switched Ethernet, several approaches and techniques have been proposed. Akin to shared Ethernet, dozens of solutions emerged, employing distinct approaches that range from software modifications at end nodes, master-slave protocols, and traffic shaping, to specialized switching hardware and network stacks. As such, each technology exhibits different levels of interoperability with legacy switched networks and real-time capabilities. To classify the existing solutions, several taxonomies can be used. For example, in [29], protocols are categorized following the use of Commercial-Of-The-Shelf (COTS) or non-COTS equipment, while the IEC 61784-2 [85] specification groups existing technologies into 16 communication profiles that grade real-time capabilities based on nine performance indicators such as message latency, throughput, jitter for periodic traffic, and redundancy. In [83], solutions are classified by degree of interoperability with IEEE 802.3 compliant nodes.

The work on this thesis addresses real-time networks which may include traffic from Ethernet compliant nodes and thus, the latter taxonomy was used to evaluate the different solutions and help filtering the most promising for our system. Therefore, three classification categories are considered: non interoperable, interoperable homogeneous and interoperable heterogeneous [83].

Non interoperable technologies

Technologies under the non interoperable class are deemed incompatible with standard Ethernet devices without the use of tunneling or gateways, and mainly rely on alterations to the IEEE 802.3 MAC layer and specialized hardware/software. Examples of solutions under this class include Ethernet POWERLINK [86], FTT-SE [25], and Avionics Full-DupleX switched ethernet (AFDX) [87]. Ethernet POWERLINK employs a master-slave approach, in which a single master node in the network (Managing Node) individually polls slave nodes (Controlled Nodes) for transmission. Slave nodes are completely passive and only react to the master explicit requests. Communications are organized in a Time Division Multiple Access (TDMA) fashion, with phases dedicated to time- and event-triggered communications. Standard Ethernet switches may be used, and nodes are implemented either on standard Ethernet devices using a special stack on top of the MAC layer or with special hardware for improved real-time performance. Akin to POWERLINK, Flexible Time-Triggered Switched Ethernet (FTT-SE) also follows a master-slave transmission control technique and coordinates all traffic transmissions under TDMA cycles comprising isolated temporal windows for time- and event-triggered traffic. Unlike POWERLINK, FTT-SE employs a signalling mechanism which is used by nodes to report the status of their communication queues, and the master is able to simultaneously poll all communications in the network by broadcasting a message containing the schedule for the on-going TDMA cycle. FTT-SE is based on COTS switches and components, with special software deployed at nodes. AFDX is an avionics data network technology whose communication protocols are derived from commercial Ethernet and the Internet Protocol (IP) stack. The standard physical and Media Access Control (MAC) layer of Ethernet is kept unchanged, and the network and upper layers mix standard IP stack services, e.g. User Datagram Transport (UDP), with services tailored for avionics applications. Communications between end systems are organized into Virtual Links that are associated with a specific route and bandwidth allocation. AFDX switches incorporate special filtering and policing functions, and forward traffic according to pre-configured static routes. End systems perform traffic shaping following each Virtual Link bandwidth allocation. AFDX switches also enforce Virtual Links' bandwidth reservations. Redundancy is achieved by the duplication of interfaces at the end systems, and switches.

Interoperable homogeneous technologies

In the interoperable homogeneous category, improved nodes and regular IEEE 802.3 compliant nodes can coexist and communicate, however, offered real-time guarantees only exist under the assumption that all devices follow the protocol. Solutions belonging to this category typically rely on the use of special stacks over the standard Ethernet MAC layer in order to control and shape network traffic. EtherNet/IP [26] and the RETHER protocol for switched networks [88] are examples of interoperable homogeneous technologies. EtherNet/IP uses standard Ethernet and switches, and implements the Common Industrial Protocol (CIP) [89] on top of the TCP/IP stack (OSI transport layer). CIP encompasses a suite of messages and services tailored for manufacturing automation applications, including control, safety, synchronization, and network management. Following a producer-consumer communication paradigm, it facilitates the distribution of data while providing a more efficient network bandwidth usage. Besides CIP, EtherNet/IP provides limited real-time capabilities since it uses COTS switched Ethernet and relies on the IEEE 802.1Q VLAN tagging to assign the highest priority traffic class to real-time traffic. RETHER relies on special software over the standard Ethernet MAC layer of nodes and switches, and keeps COTS Ethernet hardware untouched. Applications use the custom software to request real-time reservations which are enforced by a deadline-driven token passing protocol that prioritizes real-time transmissions over non-real-time data. Non real-time nodes get to use the remaining bandwidth after all real-time nodes have been serviced. The whole network operates using the standard Ethernet MAC protocol (CSMA/CD) when no real-time traffic is queued for transmission in any node.

Interoperable heterogeneous technologies

In this class, technologies are able to provide real-time guarantees even in the presence of standard Ethernet nodes. A common feature of these solutions is the use of specialized Ethernet switches to segregate and isolate legacy traffic from real-time traffic. Examples include PROFINET IRT [27], TT-Ethernet [28], HaRTES [29], and Time-Sensitive Networking (TSN) [30]. PROFINET IRT relies on customized Ethernet switches to build bi-phase periodic communication cycles comprising a mandatory phase for the confinement of isochronous real-time traffic (IRT phase), followed by a standard phase where non real-time communications are allowed. In order to execute the real-time schedule with high accuracy, PROFINET IRT synchronizes all devices in time with the IEEE 1588 Precision Time Protocol. Similarly to PROFINET IRT, TT-Ethernet organizes time in cyclic periods with temporal segments allocated for three distinct classes: (i) time-triggered, in which traffic is dispatched according to a pre-defined communication schedule deployed at nodes and switches, (ii) rate-constrained, which shapes event-triggered traffic according to reserved bandwidth allocations, and (iii), best-effort, for traffic that requires no timing guarantees, *e.g.* standard Ethernet traffic. HaRTES also provides cyclic periods with dedicated segments for time-triggered, even-triggered, and best effort communications. Switches poll time-triggered communications by sending a broadcast message at the beginning of each cycle, and traffic within the event-triggered slot is shaped by a configurable, hierarchical set of server-based shapers. TSN extends the standard Ethernet switches with several real-time enhancements, the most relevant being the Credit Based Shaper (CBS) algorithm and *time gates*. Both mechanisms control how traffic stored at egress queues is dispatched. CBS is aimed for use by traffic classes associated with event-triggered traffic and provides a shaping service based on bandwidth allocation. Time gates address time-triggered traffic and may be used to define schedules where queues are serviced only at specific time intervals.

4.3 Finding a flexible and efficient real-time data plane

On the quest to find the most suitable technology for the Software-Defined Networking framework proposed by this work, the aforementioned interoperable heterogeneous solutions were compared. Note that non interoperable and interoperable homogeneous are not considered due the lack of support for standard Ethernet devices.

PROFINET IRT exhibits impressive performance indicators, *e.g.* capable of 1 ms cycle times with sub-micro jitter values for time-triggered traffic [90]. However, it is not suitable for dynamic environments, *i.e.* applications where devices may be be connected/disconnected at any time and communication requirements may vary over the time, since its schedule for real-time traffic is static and has to be pre-defined offline. Likewise, TT-Ethernet does not support the online admission of new real-time streams and the modification of existing reservations. TSN does allow for dynamic reconfiguration of the system to some extent, but it is crippled by the limited number of traffic classes inherited from IEEE 802.1Q devices which prevents the implementation of efficient scheduling policies for applications with a high number of nodes and traffic streams [84]. Moreover, these solutions do not provide component-oriented design methodologies that enable an efficient system composability with safe resource sharing and virtualization. This aspect is highly relevant for the management of complex distributed real-time systems, as attested by frameworks such as AUTOSAR in the automotive domain [91], IMA in avionics [92], and IEC 61499 in industrial automation [93]. HaRTES was designed with the aforementioned limitations in mind, and features:

- Online admission control;
- Dynamic QoS management for both time- and event-triggered traffic;
- A configurable hierarchical server framework for event-triggered traffic;
- Traffic policing at ingress ports;
- Accommodation of traffic from standard Ethernet nodes as event-triggered or background traffic.

Due to the aforementioned reasons, HaRTES is considered to be the best option as enabler of the framework developed in the scope of this thesis, and is presented in detail in Section 4.3.2. TSN, due to its popularity and wide-spread interest by the industry as a real-time enabler is presented in detail in Section 4.3.1.

4.3.1 Time sensitive networking (TSN)

TSN is a set of technical standards developed by the IEEE 802.1 time-sensitive networking task group [30], previously known as audio/video bridging task group. It provides protocols and mechanisms to improve the real-time behaviour, *e.g.* guaranteed packet transport with bounded low latency and jitter, of IEEE 802 network technologies. TSN focuses on four main aspects: temporal synchronization among devices, end-to-end bounded latency and high reliability for real-time traffic streams, as well as management of network resources. An overview of the set of standards and amendments[5] related to the TSN framework is depicted in Figure 4.5.

In summary, the standards associated to TSN are:

[5]At the time of writing, the following amendments are incorporated into the IEEE 802.1Q-2014 standard: IEEE 802.1{Qav,Qat}. However, for simplicity, the community continues to refer to the amendment tags.

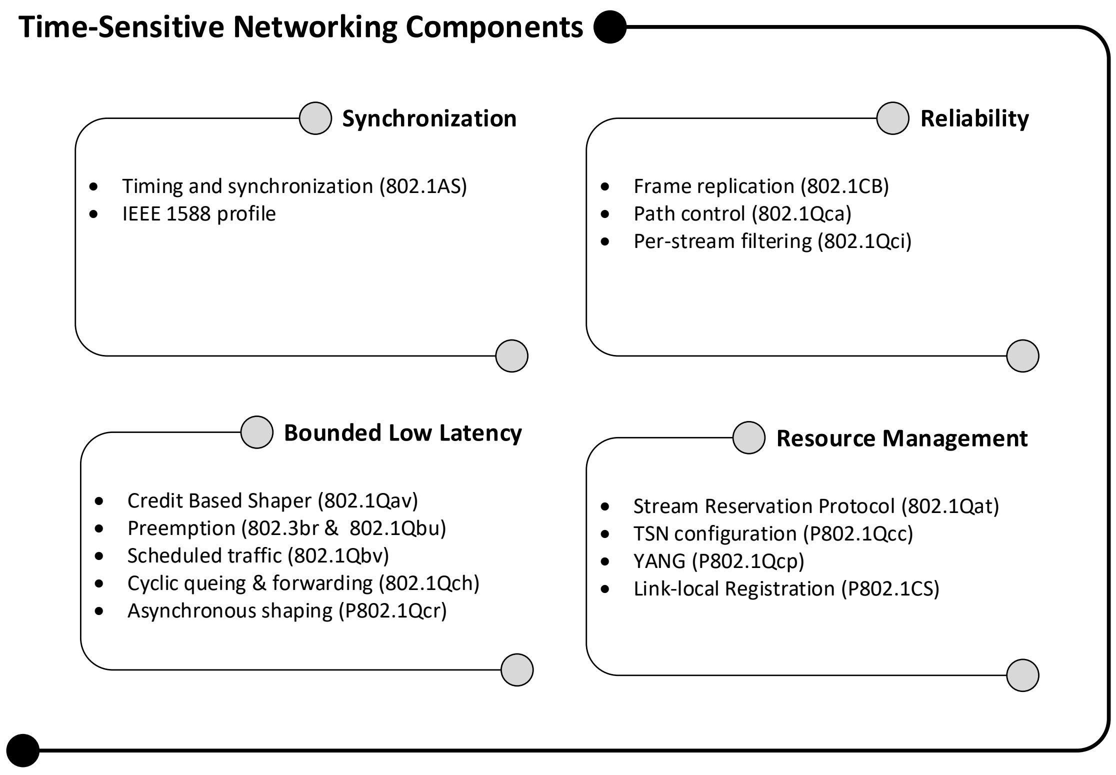

Time-Sensitive Networking Components
Synchronization
• Timing and synchronization (802.1AS)
• IEEE 1588 profile
Reliability
• Frame replication (802.1CB)
• Path control (802.1Qca)
• Per-stream filtering (802.1Qci)
Bounded Low Latency
• Credit Based Shaper (802.1Qav)
• Preemption (802.3br & 802.1Qbu)
• Scheduled traffic (802.1Qbv)
• Cyclic queing & forwarding (802.1Qch)
• Asynchronous shaping (P802.1Qcr)
Resource Management
• Stream Reservation Protocol (802.1Qat)
• TSN configuration (P802.1Qcc)
• YANG (P802.1Qcp)
• Link-local Registration (P802.1CS)

Figure 4.5: Time-Sensitive Networking (TSN) set of standards and amendments

- ***IEEE 802.1AS and IEEE P802.1AS-REV***[6] [94], [95]: specifies protocols and procedures to synchronize the local clocks of all devices across bridged and virtual bridged local area networks, with high accuracy, to ensure jitter and synchronization requirements of time-sensitive applications such as audio and video streaming;

- ***IEEE 802.1Qav*** [96]: specifies how priority values encoded in VLAN tags can be used to segregate time-critical and non-time-critical traffic into different traffic classes with distinct **QoS** requirements. It also specifies a new forwarding mechanism, the Credit Based Shaper (CBS) algorithm, to shape traffic in accordance to stream reservations;

- ***IEEE 802.1Qbu and IEEE 802.3br*** [97], [98]: define procedures and enhancements for the MAC layer and forwarding process of IEEE 802 devices to provide support for traffic preemption. Traffic classes may be classified either as *express* or *preemptable*. Frames of express classes are able to interrupt the transmission of preemptable classes' frames in a non destructive way;

- ***IEEE 802.1Qbv*** [99]: augments the forwarding process of IEEE 802 devices with the addition of a new component, termed transmission gate, to each egress traffic queue. The state of the transmission gate, *i.e. open* or *close*, determines whether or not queued

[6] At the time of writing, it was still a Project Authorization Request (PAR), *i.e.*, still on development and not published as a standard

frames can be selected for transmission. Transmission gates' state can be changed according to a well-defined schedule so as to provide a basic support for time-triggered communications;

- ***IEEE 802.1Qch*** [100]: specifies the use of stream filters and transmission gates to implement a new traffic shaping method, entitled Cyclic Queuing and Forwarding (CFQ), for time-sensitive traffic streams. In CFQ, time is divided into time intervals with fixed duration and traffic frames are queued and transmitted along a network path in a cyclic manner, *i.e.*, frames transmitted during interval i by bridge A are received by bridge B within time interval i and are transmitted onward to the next bridge/device during interval i+1. Thus, a well defined, not optimal, upper bound for latency is easily calculated, being completely described by the time intervals' length and number of hops;

- ***IEEE P802.1Qcr***[6] [101]: proposes an additional layer of shaped queues to merge traffic into the existing egress queue structure. This aims to cope with the poor fine grained traffic management for asynchronous traffic which derives from the limited number of egress queues of existing bridging standards;

- ***IEEE 802.1CB*** [102]: defines procedures and protocols to improve the reliability of traffic streams by: (i) replicating the stream's packets at the source system, (ii) splitting replicas into multiple *Member* streams, sent through disjoint paths through the network (iii) rejoining *Member* streams at relay points in the network or at the destination system while eliminating all redundant frame copies;

- ***IEEE 802.1Qca*** [103]: introduces the use of the Intermediate System to Intermediate System (IS-IS) protocol to control Ethernet networks. It surpasses common topology protocols such as Shortest Path Bridging (SPB) and offers explicit path control with bandwidth reservation and redundancy. It enables network management applications in a central Network Management System (NMS) to collect information regarding the network topology and enforce explicit forwarding paths and/or redundant paths;

- ***IEEE 802.1Qci*** [104]: introduces a list of stream filters that determine, on a per-stream basis, filtering and policing actions to be applied to frames received on a specific stream. It enables filtering frames based on arrival times, rates and bandwidth;

- ***IEEE 802.1Qat*** [105]: defines Stream Reservation Protocol (SRP), which allows network resources to be dynamically reserved for specific traffic streams requiring guaranteed QoS guarantees along a bridged local network. With SRP, end systems may disseminate through the whole network their willingness to be producers ("Talkers") or consumers ("Listeners") of specific streams. When a Talker and one or more Listeners announce for the same stream, and a network path with sufficient resources exists, network bridges configure the forwarding mechanisms proposed in IEEE 802.1Qav, e.g. CBS, to enforce the stream's QoS requirements. Then, bridges notify the associated Talker and Listeners so that communication may start;

- ***IEEE P802.1Qcc***[6] [106]: extends the capabilities of SRP with support for more streams, better description of stream characteristics, support for layer 3 streaming, and introduces three network management approaches: (i) fully centralized model, where end systems report their stream requirements directly to a central management system that then configures the whole network devices accordingly, (ii) fully decentralized, as defined by the original SRP approach, and (iii) partially centralized, where requests are sent to the edge bridge closest to the end system and then forwarded to the central management system;

- ***IEEE P802.1Qcp***[6] [107]: specifies Unified Modeling Language (UML) and Yet Another Next Generation (YANG) data models for the configuration and monitoring of bridges. They can be used combined with management protocols such as NETCONF to simplify network configuration;

- ***IEEE P802.1CS***[6] [108]: defines Link-local Registration Protocol (LRP), a link-local registration protocol that replicates databases on both ends of a point-to-point link. It serves the same purpose as Multiple Registration Protocol (MRP), *i.e.* facilitate the creation of application protocols that distribute information among network devices, while overcoming MRP's scalability and performance issues, *e.g.* performance significantly drops for databases sized over 1500 bytes (such as those found in SRP).

In TSN networks, IEEE 802.1Q VLAN bridges enhanced with the aforementioned amendments interconnect Ethernet end stations in topologies identical to those found in standard switched Ethernet networks. All devices are synchronized in time to ensure jitter and synchronization requirements of time-sensitive applications such as audio and video streaming. Streams only allow a single producer ("Talker") and one or several consumers ("Listeners"). An example of a TSN network is depicted in Figure 4.6. Two real-time traffic streams are defined [23]:

- ***Stream Reservation (SR) class A***: Profiled for audio/voice streams. Defines a *class measurement interval* of 125 μs, a maximum latency target of 2 ms over 7 hops, and a priority level of 3 (mapped to traffic class 7 when eight traffic classes are supported, see Table 4.1);

- ***Stream Reservation (SR) class B***: Profiled for video streams. Defines a *class measurement interval* of 250 μs, a maximum latency target of 50 ms over 7 hops, and a priority level of 2 (mapped to traffic class 6 when eight traffic classes are supported, see Table 4.1).

When requested by an application at an end station, the SRP instantiates the necessary resources and configurations of TSN bridges across the network in order to guarantee the aforementioned profiles of SR class A and/or SR class B. Applications specify the properties of a stream through two parameters (TSpec): (i) the maximum size of a frame, excluding headers and framing overheads, transmitted by the producer station ("Talker"), and (ii), the maximum number of frames that the Talker may transmit within one *class measurement*

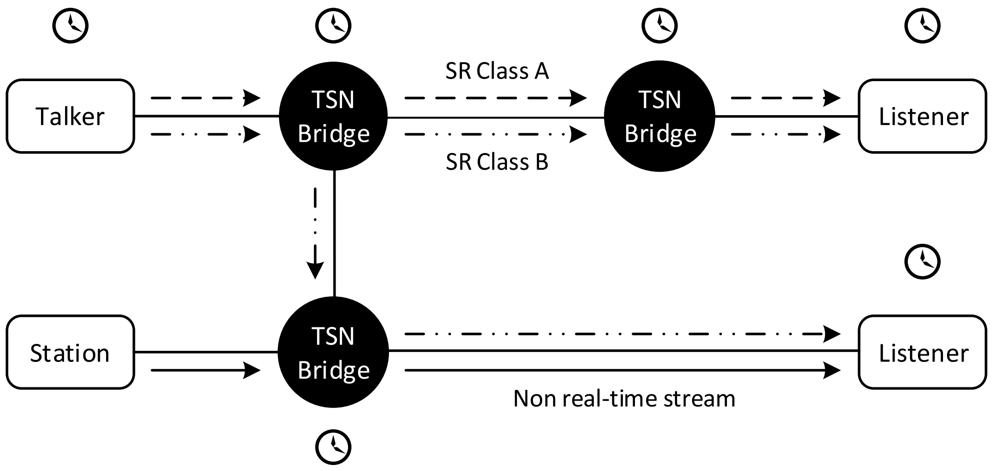


Figure 4.6: An example of a TSN network

interval. In order to provide latency and bandwidth guarantees for stream reservations, the *Transmission Selection Algorithm Table* is configured to assign the Credit-Based Shaper (CBS) algorithm to egress queues associated with SR classes.

TSN also provides a mechanism, known as *transmission gates* (or colloquially as *time-gates*), that allows the control of transmissions from each egress queue following a time schedule. However, the configuration of this mechanism is currently specified as being made through management, *e.g.* configuration consoles or network management protocols, and no association with a resource management protocol such as SRP is defined. Moreover, the only standardized use-case is supplied by the IEEE 802.1Qch [100], which suggests the use of *transmission gates* to implement a "Cyclic Queuing and Forwarding" scheme (CQF). According to the aforementioned amendment, in CQF, traffic is transmitted along a network path within fixed-length time intervals in a cyclic manner. For example, frames transmitted by bridge A during time interval i are received by bridge B. Bridge B then transmits frames received during interval i in interval i+1, and so on. This technique allows to precisely describe the latency introduced as time-sensitive frames transit the network, which is completely described by the length of time intervals and the number of hops.
A brief overview of the operation of the CBS algorithm and *transmission gates* is presented next.

4.3.1.1 Credit-Based Shaper (CBS) algorithm

CBS is an algorithm that can be used to shape the transmission of traffic from egress queues in accordance with the bandwidth negotiated for a stream reservation. The operation of each CBS instance is governed by two external parameters, *i.e.* modifiable by management, associated with each queue [23]:

- ***Port Transmission Rate (portTransmitRate)***: the current transmission rate, in bits per second, of the egress port;
- ***Idle Slope (idleSlope)***: the actual bandwidth, in bits per second, that is currently reserved for use by the queue. The sum of idle slopes of all CBS queues must not exceed 75% of the port's available bandwidth. Lower priority queues may share unused bandwidth from higher priority traffic classes.

A set of internal parameters are also used to describe the operation of CBS:

- ***Maximum Frame Size (maxFrameSize)***: the maximum size, in bits, of a frame for the concerned traffic class;
- ***Maximum Interference Size (maxInterferenceSize)***: the maximum size, in bits, of any burst traffic that can delay the transmission of a frame for the concerned traffic class. For the highest priority class, it is equal to the maximum frame size supported by the underlying port MAC. Remaining classes must account the interference owning to any higher traffic class;
- ***Credit***: the transmission credit, in bits, currently available to the queue. Credit is set to zero when positive, and no frames are queued or being transmitted;
- ***Send Slope (sendSlope)***: rate of change of *Credit*, in bits per second. Computed as:

$$sendSlope = idleSlope - portTransmitRate \tag{4.1}$$

- ***High Credit (hiCredit)***: The maximum value of *credit* that can be accumulated. Computed as:

$$hiCredit = maxInterferenceSize \times \frac{idleSlope}{portTransmitRate} \tag{4.2}$$

- ***Low Credit (loCredit)***: The minimum value of *credit* that can be accumulated. Computed as:

$$loCredit = maxFrameSize \times \frac{sendSlope}{portTransmitRate} \tag{4.3}$$

The operation of CBS can be briefly described as follows. A queued frame is selected for transmission if there is no conflicting traffic, *i.e.* no ongoing transmission and no higher priority traffic awaiting transmission, and the credit value is zero or positive. Credit decreases at the rate of *sendSlope* during the transmission, and increases back to zero at the rate of *idleSlope* once the transmission of the frame is complete. *hiCredit* and *loCredit* place a bound on the maximum burst size a class may perform after awaiting transmission due to interference from other traffic classes.

An example of the operation of the CBS algorithm is depicted in Figure 4.7 (adapted from [23]), in which three frames are queued in the same queue while conflicting traffic is being

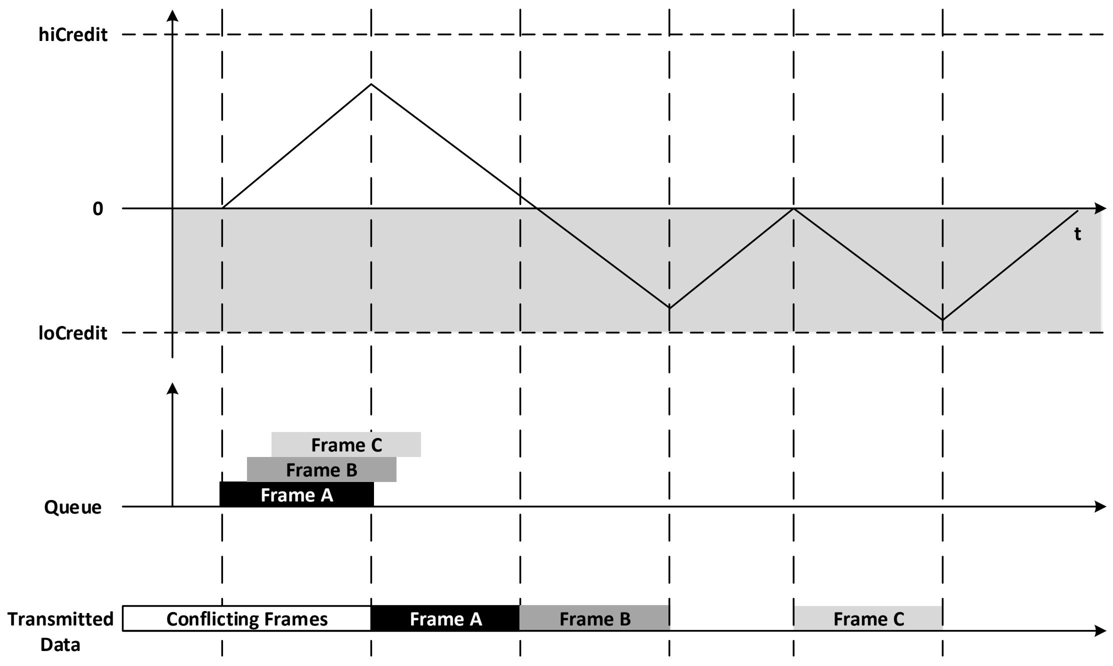


Figure 4.7: An operation example of the Credit-Based Shaper algorithm

transmitted. As there are frames in the queue awaiting for transmission, *credit* accumulates at the rate of *idleSlope*. Once all conflicting traffic has been transmitted, frame A is selected for transmission and credit decreases at the rate of *sendSlope*. When the transmission of the frame A ends, *credit* is still greater or equal to 0 and there is no conflicting traffic and thus, frame B is transmitted right away. When its transmission concludes, *credit* is negative and the transmission of the last frame is delayed until *credit* returns to zero.

Note: In order for the CBS algorithm to operate properly, all traffic classes that support the CBS algorithm must have higher priority than classes running the Strict Priority algorithm.

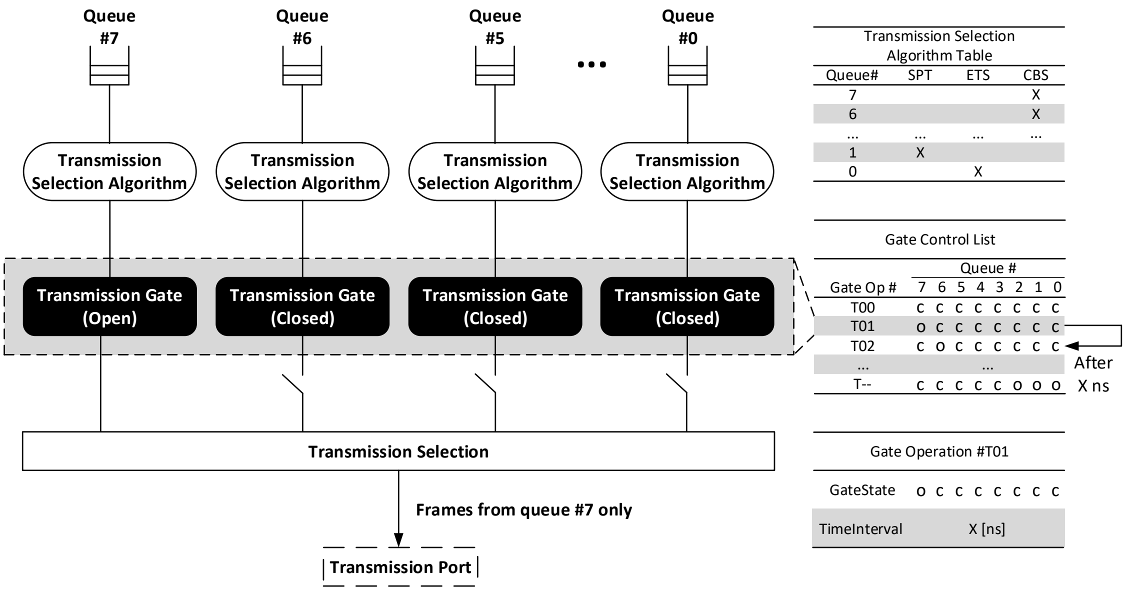
4.3.1.2 Enhancements for scheduled traffic (Transmission Gates)

TSN bridges and end stations may support the use of *transmission gates* to allow the transmission of frames according to a time schedule. A *transmission gate* is an entity that can be associated with a given traffic class queue, and is able to enable or disable the associated transmission selection algorithm, *e.g.* CBS or Strict Priority, in order to allow or prevent it from selecting frames from the concerned queue. A *transmission gate* has two possible states:

- ***Open***: queued frames are selected for transmission in accordance with the normal operation of the transmission selection algorithm;
- ***Closed***: queued frames are not selected for transmission.

Each port has an associated *Gate Control List* which contains an ordered list of *gate operations*. A *gate operation* contains the state of each *transmission gate* for a given time interval

(*TimeInterval*). Every *Gate Control List* can be configured to be executed at a given time instant (*CycleStartTime*, expressed as a PTP timescale) and repeated periodically (every *OperCycleTime* periods, in seconds). An example of a configuration using *transmission gates* is presented in Figure 4.8.


Queue #7
Queue #6
Queue #5
Queue #0
...
Transmission Selection Algorithm
Transmission Selection Algorithm
Transmission Selection Algorithm
Transmission Selection Algorithm
Transmission Gate (Open)
Transmission Gate (Closed)
Transmission Gate (Closed)
Transmission Gate (Closed)
Transmission Selection
Frames from queue #7 only
Transmission Port
After X ns

Transmission Selection Algorithm Table

Queue#	SPT	ETS	CBS
7			X
6			X
...	...	...	...
1	X		
0		X	

Gate Control List

	Queue #							
Gate Op #	7	6	5	4	3	2	1	0
T00	c	c	c	c	c	c	c	c
T01	o	c	c	c	c	c	c	c
T02	c	o	c	c	c	c	c	c
...	...							
T--	c	c	c	c	c	o	o	o

Gate Operation #T01

GateState	o	c	c	c	c	c	c	c
TimeInterval	X [ns]							

Figure 4.8: An example of the application of TSN transmission gates

Note: The state of a *transmission gate* affects the *idleSlope* of the queue's CBS algorithm. It is now zero when the gate is *Closed* and as given by Equation 4.4 when the gate is *Open*. The variable *operIdleSlope* is the configured fixed bandwidth for the stream reservation, while *GateOpenTime* is equal to the total amount of time that the gate state is *Open* during *OperCycleTime*.

$$idleSlope = operIdleSlope \times \frac{OperCycleTime}{GateOpenTime} \tag{4.4}$$

4.3.1.3 Limitations and related research

An important limitation of TSN is the low, limited number of supported traffic classes which impairs system scalability and the real-time performance, in particular for large systems. As there are only eight queues, it is not possible to segregate traffic into individual queues if the number of flows in the system grows past the available class number. Thus, interference from several flows assigned to a same queue may happen, leading to aggravated latency and jitter values. Moreover, the presence of a set of time-triggered traffic with big prime values as transmission periods can lead to huge and unmanageable *gate control lists*, a problem already faced for example in WorldFIP [6]. Thus, a lot of research has been addressed to study the performance impact related to the assignment of traffic into traffic classes and the use of time-gates, as well as to devise strategies in order to attain the necessary system schedulability.

Work to improve fault-tolerance has also been carried-out. A brief overview of existing work now follows.

In [109], Craciunas *et al.* studied the scheduling challenges that affect the real-time communication of streams under the use of transmission gates (IEEE 802.1Qbv). They identified constraints for the computation of schedules that are able to guarantee deterministic end-to-end latency and low jitter, and proposed several possible configurations. The schedulability and scalability of the proposed approach is evaluated under simulations. Ko *et al.* [110] studied the distribution of bandwidth between scheduled traffic and the remaining traffic classes in order to establish an optimal relationship that guarantees the requested QoS. Simulations based on an automotive scenario were used to evaluate the proposed distribution ratio. Pop *et al.* [111] introduced a Integer Linear Programming (ILP) formulation to assign time-triggered flows to TSN classes and synthesize the necessary gate control lists. They also proposed an optimization strategy for the routing of flows through the network. The enhanced worst-case delay analysis of [112] is used to verify the schedulability of the obtained configurations. A similar work is presented in [113], but also takes into account the QoS impact on event-triggered, lower priority real-time traffic. In [114], Álvarez *et al.* formulated the use of time and spatial redundancy for TSN networks to increase reliability, in particular against transient faults, and reduce resource consumption. They combine their time-based replication mechanism (Proactive Transmission of Replicated Frames (PTRF)) proposed in [115], which replicates frames of critical streams through the same path, with TSN's frame replica elimination mechanism (IEEE 802.1CB). The approach is able to reduce the number of redundant paths while being able to tolerate permanent and temporary faults.

4.3.2 Hard Real-Time Ethernet Switch (HaRTES)

Hard Real-Time Ethernet Switch (HaRTES) [29] is a modified Ethernet switch which provides real-time communication services with hierarchical, server-based resource sharing mechanisms. It was developed to overcome some limitations of the FTT-SE protocol, in particular the compatibility with standard Ethernet stations, and to provide advanced resource management services, *e.g.* hierarchical servers, so as to provide a higher degree of flexibility in the design of complex distributed real-time applications. More specifically it features:

- Embedded FTT master features such as admission control, Quality-of-Service (QoS) manager and traffic scheduling, amongst others;
- Support for synchronous (time-triggered) and asynchronous (event-triggered) traffic;
- Hard and soft real-time guarantees for the synchronous and asynchronous traffic, respectively;
- Hierarchical, server-based traffic scheduling for the asynchronous traffic;
- Seamless integration of standard, non-FTT compliant nodes, without jeopardizing real-time performance;

- Traffic enforcement and policing mechanisms;
- Low switching latency and jitter.

Akin to FTT-SE, HaRTES employs a master-slave technique to control network communications, with the master node implemented within the Ethernet Switch (Figure 4.9) and communications following the Flexible Time-Triggered (FTT) paradigm [116].

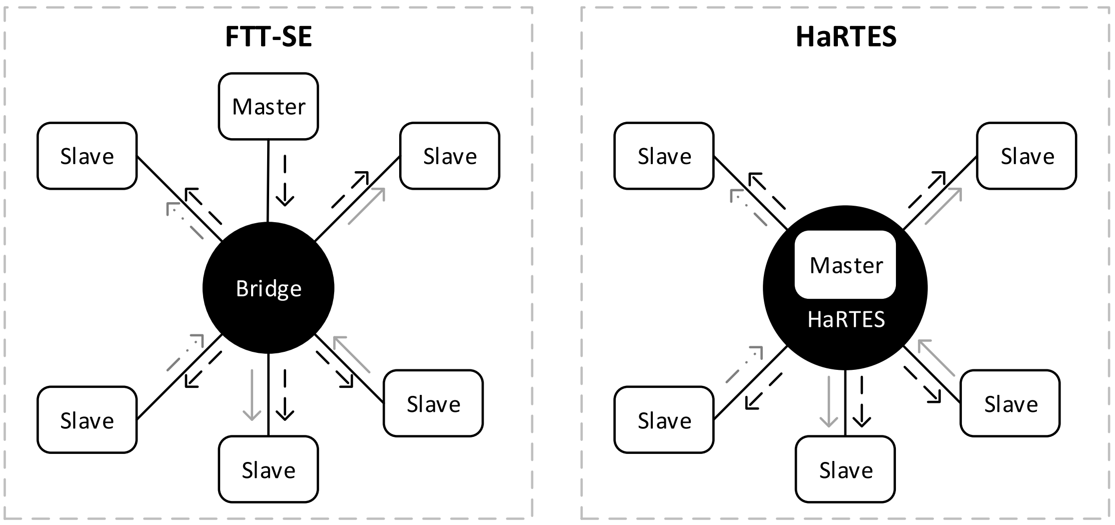


Figure 4.9: FTT-SE vs HaRTES network architecture

The integration of the master node within the Ethernet switch provides significant advantages [29]:

- ***The asynchronous traffic handling is simplified.*** Instead of being polled by the master node, asynchronous traffic is now autonomously triggered by nodes and managed by the switch in order to maintain proper temporal behaviour;
- ***Increased system integrity.*** Unauthorized communications can be blocked at the switch's input ports to prevent interference with the rest of the system;
- ***Integration of Ethernet legacy nodes.*** Nodes not compliant with the protocol can be integrated without compromising the real-time communications;
- ***Improved network synchronization.*** The Trigger Message (TM) is now transmitted with very low jitter and latency.

The master of HaRTES coordinates communications within fixed-duration time-slots called Elementary Cycles (ECs) (Figure 4.10). In order to organize communications and support synchronous and asynchronous traffic, each Elementary Cycle (EC) is divided into two windows: a synchronous window, in which synchronous transmissions are scheduled and triggered by the master, and an asynchronous window, in which asynchronous communications triggered by the nodes are managed by hierarchical server-based traffic scheduling mechanisms. This temporal isolation is enforced by the switch itself, effectively eliminating interference between the two traffic classes. The master inside the switch keeps information regarding all traffic streams and data exchanges. It also performs traffic scheduling for the time-triggered traffic, online admission control, traffic policing at ingress ports, and is able to reconfigure existing reservations without disruption of service.

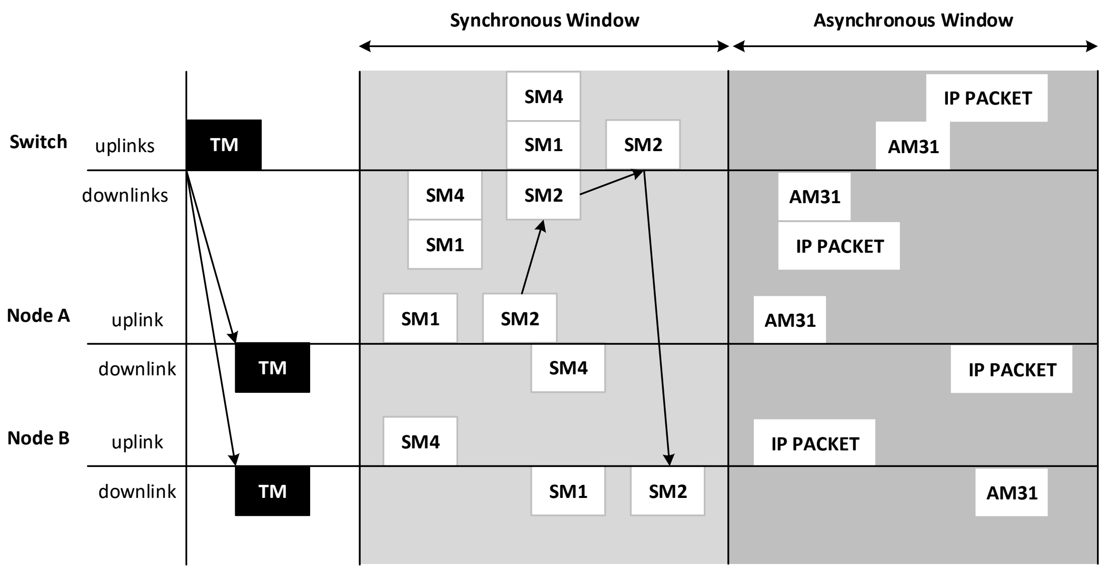


Figure 4.10: HaRTES elementary cycle overview

Synchronous transmissions in HaRTES are scheduled and triggered (polled) by its internal FTT master via special messages known as trigger messages (TMs). TMs are broadcast at the beginning of each EC, and contain the IDs of the time-triggered messages to be transmitted within the current cycle's synchronous window. FTT-compliant nodes decode received TMs and transmit immediately its scheduled message(s). The switch then forwards the received messages to the correct egress port(s). Unscheduled synchronous messages, such those from badly behaved nodes, or synchronous messages transmitted outside the synchronous window bounds are promptly discarded at ingress ports. Scheduling for time-triggered traffic is done online, on an EC basis, according to a user-defined scheduling policy such as Rate-Monotonic. The master guarantees that all messages scheduled for transmission within a given EC fit, avoiding window overruns and queue buildup. By generating the TM within the switch and transmitting it directly to the network, TM transmissions occur in a highly precise manner, with very low jitter and latency.

The asynchronous traffic is triggered by nodes without being scheduled and polled by the master. Once the asynchronous traffic is received by the switch, it is queued in dedicated memory pools and transmitted when appropriate, *i.e.* when its transmission conforms to the negotiated timing properties. In order to enforce the asynchronous traffic transmission behaviour, hierarchical, server-based mechanisms are used within HaRTES. An example of a configuration for this asynchronous framework is shown in Figure 4.11 [29].

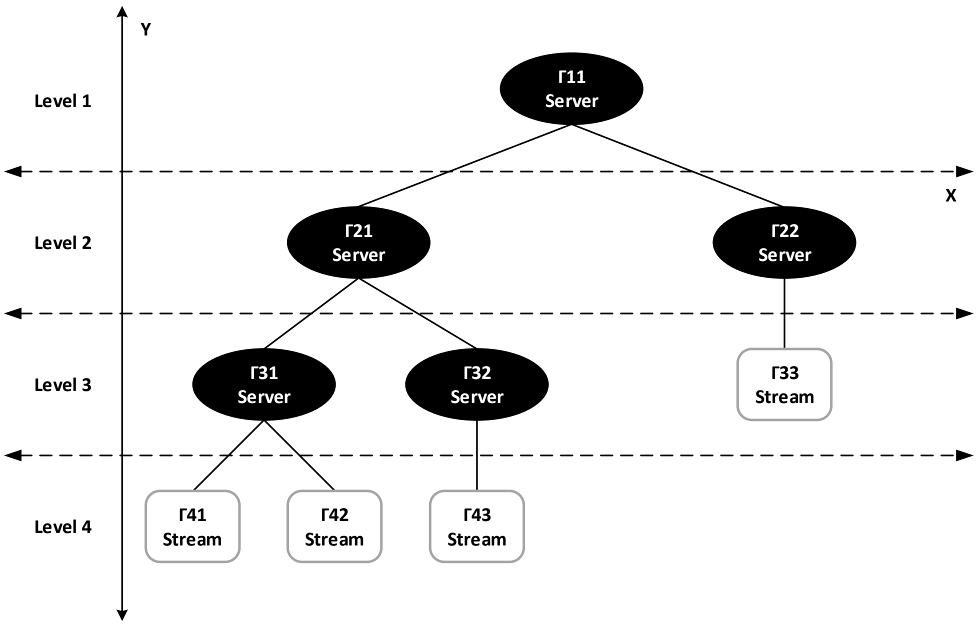


Figure 4.11: HaRTES hierarchical server-based scheduling

During the registration process, producer nodes ask for resource reservations by stating their asynchronous streams' properties, *e.g.* identifier, priority, minimum inter-arrival time, etc. Upon acceptance, the switch allocates the necessary resources, *e.g.* memory, and assigns a given server to each stream. This server is configured so to enforce the stream's temporal behaviour. Asynchronous packets are then transmitted to the network within the asynchronous window, provided that its associated servers allow it, *i.e.* servers have enough budget. These servers can be composed hierarchically by the users in order to enable a flexible management of resources within complex systems and applications. Non real-time traffic, *e.g.* from non FTT-compliant nodes, is also transmitted within the asynchronous windows, albeit in a background fashion, *i.e.* when no asynchronous traffic is queued or when all stream's servers are budget depleted [29].

4.3.2.1 HaRTES Platform Architecture

HaRTES is fully implemented in hardware, more specifically using the NetFPGA 1G (Figure 4.12) platform [117], an open source, low-cost and reconfigurable hardware platform optimized for high performance networking projects developed by the Stanford University.

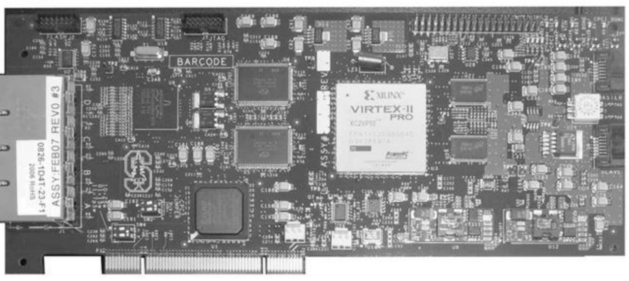


Figure 4.12: HaRTES hardware platform (NETFPGA 1G)

The NetFPGA includes all the necessary resources to design a complete, custom Ethernet switch. More specifically it provides (Figure 4.13):

- A Field Programmable Gate Array (FPGA) (FPGA);
- Four Gigabit Ethernet interfaces;
- Memory for packet/data storage (4.5MB of SRAM and 64MB of DDR2 DRAM);
- A standard Peripheral Component Interconnect (PCI) interface to interact with a host computer, *e.g.* for complex operations.

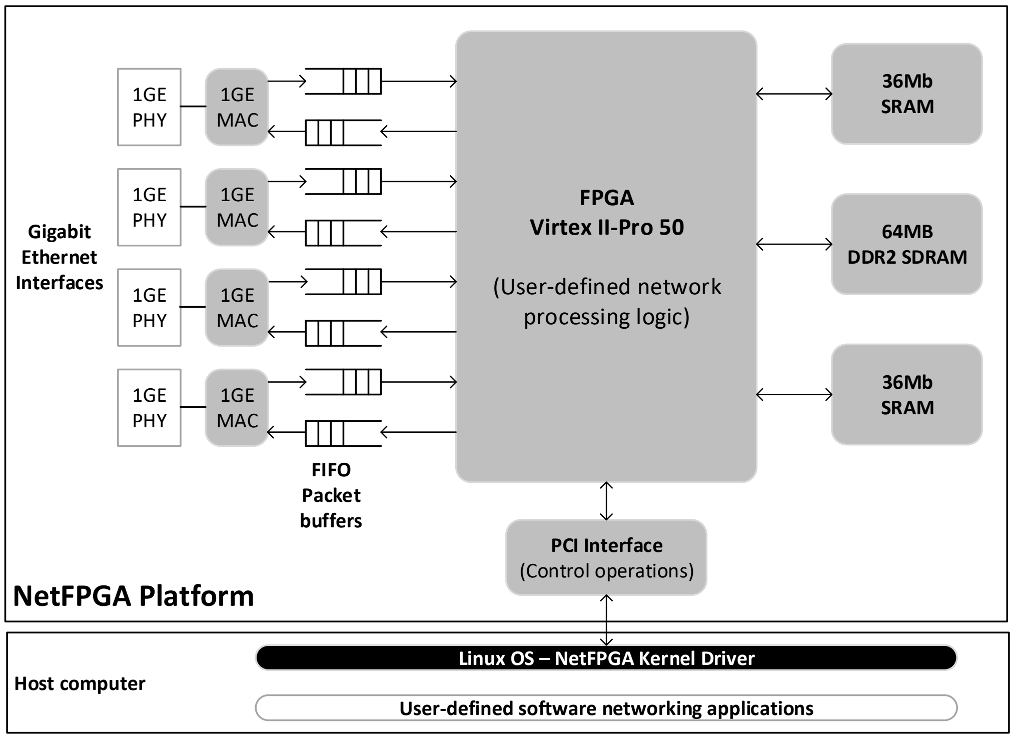


Figure 4.13: HaRTES hardware resources

By fully implementing the entire datapath and control plane in hardware, the switch exhibits low switching and processing latency values with extremely low jitter at the egress ports. An overview of HaRTES internal logic architecture is illustrated in Figure 4.14.

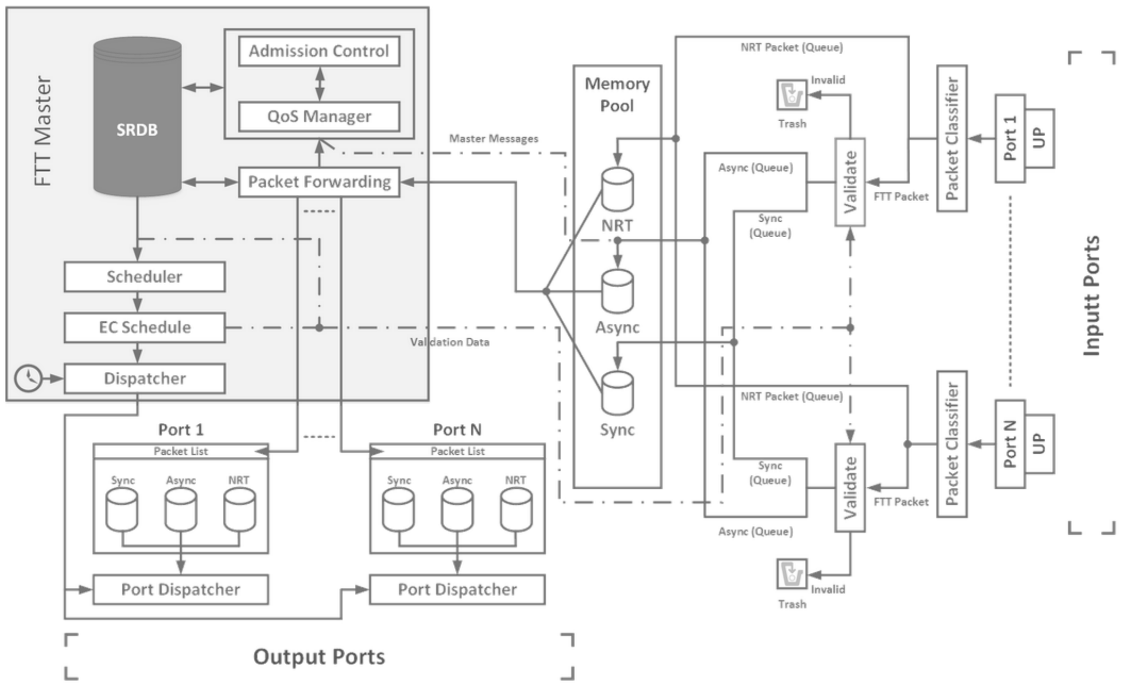


Figure 4.14: HaRTES internal architecture

The main blocks of the architecture are:

- ***The FTT master module***. It includes all the management and decision making logic, such as the admission controller, QoS manager, traffic scheduler and global dispatcher. It also includes the database with the information regarding registered traffic streams and their properties;
- ***Memory Pool***. Here is where all the received packets' data is stored. Three main memory sub-pools exist, each one associated to a given traffic category;
- ***Input ports module***. At the input ports, packet classifiers validate and either admit or trash received packets. Packets are also categorized as synchronous, asynchronous or non-real-time;
- ***Output ports module***. At the output ports separated queues for each traffic type are implemented. Moreover, the logic of the server-based scheduling for asynchronous traffic is also implemented here, one for each port.

Packets received at the input ports are classified into synchronous, asynchronous or non-real-time packets and stored in the dedicated memory pool queues. Invalid packets, *e.g.* corrupted and unauthorized, are discarded. Memory pointers to each stored packet are then generated. These contain information on each packet, *e.g.* type, size, stream ID, memory address, etc., and are used by all of the internal switch's modules. When a module has to access a given packet's content, it will look up the memory address within the pointer and read the contents directly. Each of these generated pointers is then forwarded to the FTT-Master module. Packets carrying requests, *e.g.* stream registration/removal, will be handled by the Admission Control module, while the destination of regular packets will be looked up and their pointers stored at the respective output port(s). At each port, dedicated FIFO queues for each registered synchronous/asynchronous stream and non-real time streams are instantiated. Within each EC, the Port Dispatcher module (on each port) retrieves pointers from the appropriate queues and transmits the associated packet to the network. Note that the mechanisms described for the synchronous and asynchronous transmissions rule and determine which queue is to be served at a given instant.

4.3.2.2 HaRTES Configuration API

HaRTES provides a simple API that allows applications to configure the offered real-time services. A summary of the available operations is presented in Table 4.2.

Table 4.2: HaRTES API

Category	Operation	Description
OP_CAT_SYNC	OP_REG	Register a synchronous stream.
	OP_DEREG	Deregister a synchronous stream.
	OP_MODIFY	Modify properties of a synchronous stream.
OP_CAT_ASYNC	OP_REG	Register an asynchronous stream.
	OP_DEREG	Deregister an asynchronous stream.
	OP_MODIFY	Modify properties of an asynchronous stream.
OP_CAT_SERVER	OP_SET	Enable and configure a server.
	OP_UNSET	Disable a server.

Synchronous (SYNC) and asynchronous (ASYNC) streams are specified by ID, period or minimum inter-arrival time, offset (for synchronous streams only), deadline, priority, frame length, ingress and egress ports. Asynchronous streams have an additional parameter, the server bitmap, which associates the stream with a set of servers. There is an independent set of servers per output port, and each server can be configured with a budget and replenishment period, which are typically equal to the frame length and minimum inter-arrival time of the associated stream, respectively. Table 4.3 provides a description and value range for each of the aforementioned parameters.

The structure of some HaRTES API messages is shown in Figure 4.15. Remaining messages follow a similar structure.

Table 4.3: HaRTES API parameters

Parameter	Description	Value range
ID	Stream identifier.	$]0, 32]$
Period	SYNC: periodicity of frames. ASYNC: minimum time between consecutive frames.	$]0, 2^{16}-1]$ ECs
Deadline	Frame's relative deadline.	$]0, 2^{16}-1]$ ECs
Offset	SYNC: relative phasing. ASYNC: ignored parameter.	$]0, 2^{32}-1]$ ECs
Priority	Absolute priority. Lower value equals a higher level.	$]0, 2^8 - 1]$
Frame length	Maximum length, in bytes, of a stream's OSI layer 2 frame. Includes all headers, payload and FCS.	$[68, 2^{16} - 1]$
Producers	Stream's ingress ports.	- - -
Consumers	Stream's egress ports.	- - -
Server bitmap	Bit vector to associate servers to a stream. E.g. 0x83 = "1000 0011" associates server 1, 7 and 8.	$]0, 2^8 - 1]$
Server ID	Server's ID.	$]0, 8]$
Port	Server's egress port.	$[0, 3]$
Server budget	Server's budget, in bytes.	$[0, 2^{16} - 1]$
Server period	Server's replenishment period.	$[0, 2^{32}-1]$ ECs

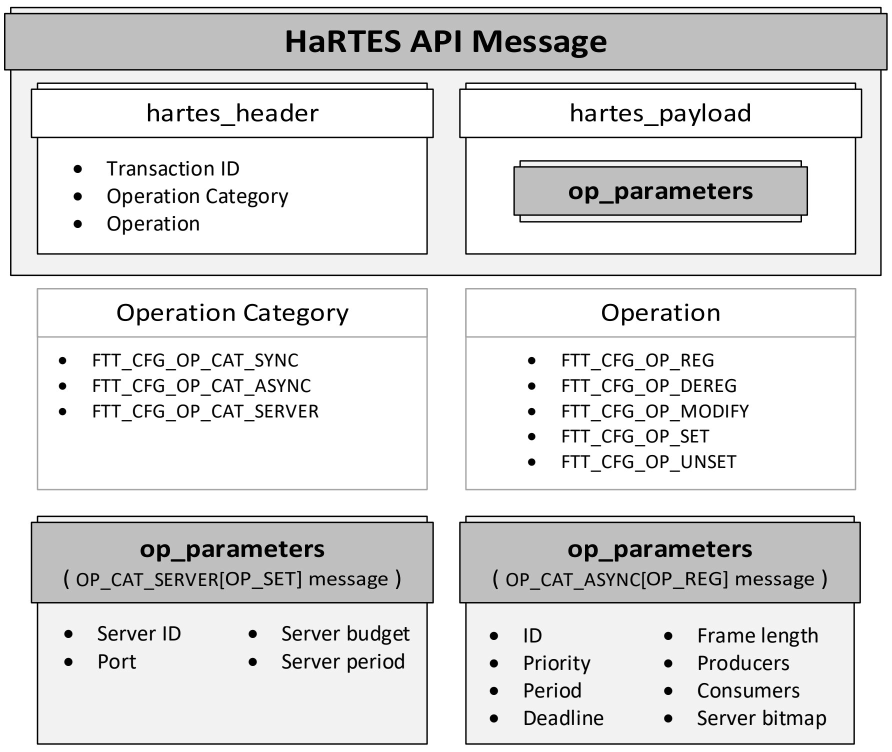


Figure 4.15: Structure of a HaRTES API message

4.3.2.3 Related research

Research has been conducted to improve HaRTES, in particular, to scale its use to big networks comprised by several interconnected HaRTES switches and increase fault-tolerance.

Ashjaei *et al.* [118] investigated the challenges of connecting multiple HaRTES switches while preserving the real-time performance of the whole network. They proposed a forwarding method, named Distributed Global Scheduling, to handle the traffic forwarding through the several hops. They also devised a response time analysis for the method and evaluated the embodied level of pessimism. In[119], Ashjaei *et al.* proposed an improved forwarding method for multi-hop HaRTES networks, called Reduced Buffering Scheme. They developed a response time analysis for the new method and compared it with the previous Distributed Global Scheduling method. They concluded that new method performs better and brings an improvement on the response time for all types of traffic. Rodriguez-Navas and Proenza [120] proposed a method to enhance HaRTES with a multicast service for synchronous messages. The proposed method takes advantage of the centralized, online scheduling service of HaRTES to reduce complexity and bandwidth utilization. Ballesteros *et al.* [121] proposed a new switch called HaRTES/PG, based on the existing HaRTES switch, in order to prevent the propagation of Byzantine node behaviours and ensure that local errors can't interfere with the global communication. In this work, they studied the possible errors that may lead to Byzantine node behaviours and global communication disturbance in HaRTES. They also presented some ideas on how to prevent the propagation of these errors in with the new switch. Gessner *et al.* [122] explored some architectural designs to improve the reliability of HaRTES-based networks. They add some level of fault tolerance through the replication of switches.

CHAPTER 5

A real-time SDN framework

Contents

*T*his chapter introduces the development of a software-defined networking framework for complex real-time systems, such as those of Industry 4.0. It starts by presenting the proposed system architecture, followed by the design of an extended control plane that is able to conduct network resource reservations for real-time traffic flows. Next, the enhancement of a real-time data plane with SDN services is discussed.

5.1 A reference architecture for a real-time SDN framework

The proposed reference architecture follows the common SDN design, in which the control plane is enacted by a logically centralized controller that is responsible for the configuration of a network comprising one or more switching devices. When combined, these devices realize the data plane. An example for the reference architecture is depicted in Figure 5.1. Note that the data plane can also implement traditional start/tree topologies found in switched Ethernet networks. Akin to standard SDN approaches, it is desirable the existence of a dedicated management network connecting the controller to each switching device on the data plane for performance and security sake. If this is not possible, in-band channels within the data network can be used. However, strong security measures must be put in place and enough bandwidth should be reserved for these management communications. Additionally, admission control and scheduling algorithms must consider interference that may arise from the presence of in-band traffic. Throughout this work, it is considered the use of a dedicated management network to connect switching devices at the data plane to the controller. Moreover, although this work focus on single-controller topologies, as the employed southbound protocol (OpenFlow) natively provides support for multiple controllers, multi-controller topologies are also possible.

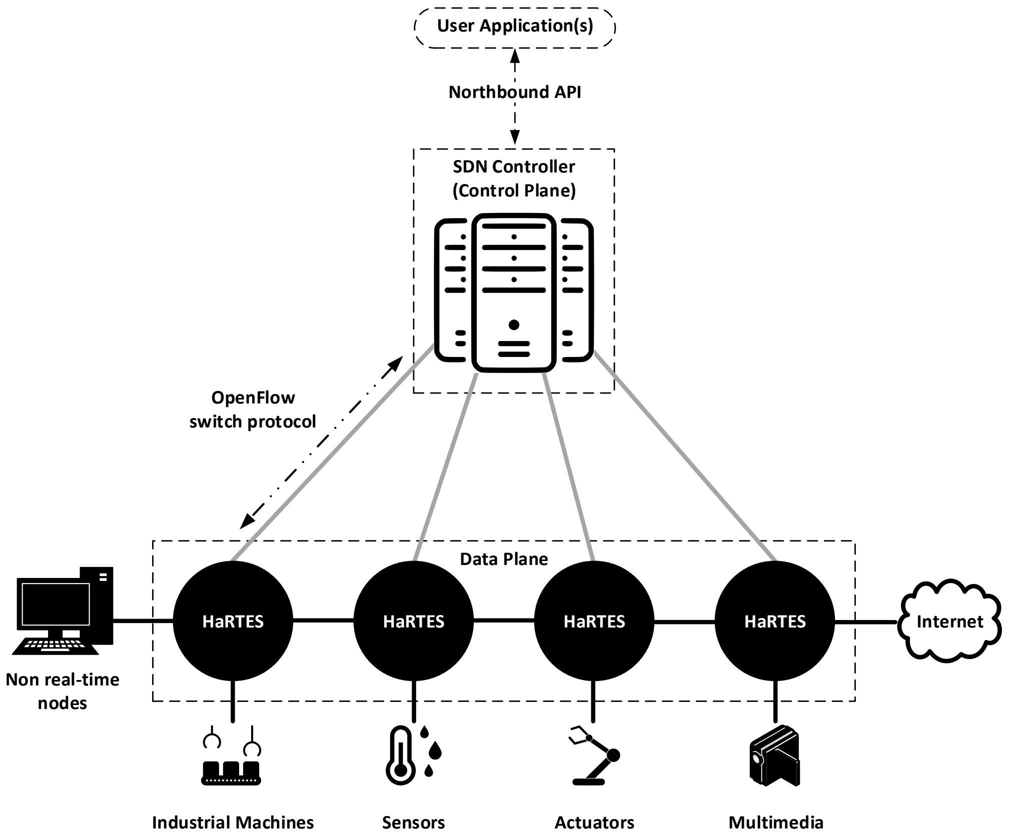


Figure 5.1: Reference architecture for the real-time SDN network

The central controller interacts with applications through northbound Application Programming Interface (API) interfaces which can be built, for example, under the ONF's *Intent NBI* paradigm [72]. Applications may request the reservation of network resources for both non real-time and real-time flows, accompanied with the necessary information for determining network end-points, *i.e.* consumers and producers, and the necessary resources, *e.g.* payload size and transmission period. The development of this northbound API is not addressed in this work, and is left as future work. For now, it is assumed that all the necessary information regarding application flows is available in a database inside the controller. Upon receiving requests, the controller runs an admission control algorithm to ensure the system has the necessary network resources and that real-time guarantees for both new and existing flows are met. If a flow is admitted the controller sets the entire data path with adequate configurations, otherwise, the request is denied. The data plane network relies on HaRTES switches as the enabling platform with real-time services. The reasoning behind this choice of HaRTES is presented in Section 4.3. The controller entity uses the southbound protocol OpenFlow to configure data plane services.

The development of the described framework was organized into three distinct workflows:

- ***Development of the admission control module.*** A traffic scheduling analysis must be executed before admitting new flows in order to provide temporal guarantees for real-time traffic. This admission control must be devised since it is not provided by standard SDN controllers;

- ***Extension of the control plane (OpenFlow protocol)***. Standard OpenFlow lacks an API and services that are adequate to manage real-time traffic and thus, it must be extended;

- ***Enhancement of HaRTES with SDN services and OpenFlow API***. Although HaRTES provides flexible real-time services, it lacks basic SDN functionality and an API that is compatible with OpenFlow. It is then necessary to enhance HaRTES with such capabilities.

These aforementioned workflows are described in detail by the following sections.

5.2 A SDN controller with real-time admission control

The SDN controller is the central piece of any software-defined network. It stands between network devices and applications and implements the intelligence to manage existing network resources and communication flows according to application requests. Although there is a myriad of SDN controllers, *e.g.* NOX, RYU, ONOS, and OpenDaylight, they are mainly designed for the management of data center resources and carrier operator networks, and do not provide the necessary services to manage networks for communications with strict timeliness requirements, *e.g.* admission control with real-time schedulability analysis. Therefore, a SDN controller application with real-time control capabilities was specifically designed for the real-time SDN framework. Figure 5.2 presents an overview of the developed controller.

The real-time controller is implemented on the RYU framework [123], an open-source framework designed for the development of network management and control applications. RYU is written in Python programming language and follows a component-based software architecture, meaning that it is possible to build applications on top of bare-bone services and add additional modules to expand functionality as required, greatly reducing complexity and computational overheads. It also supports various protocols for managing network devices, *e.g.* NETCONF, including all OpenFlow protocol versions up to version 1.5. For these reasons, it is considered an adequate development framework for the real-time controller.

The real-time controller application is able to communicate with one or more data plane switching devices using the standard OpenFlow protocol. The OpenFlow protocol API and the OpenFlow channels are all provided and managed by RYU's bare-bone services. Devices first connecting to the controller are handled by the Setup Handler at the Data Plane Manager. This handler creates an entry for the device at the Device Database and performs basic configurations to the device's OpenFlow services, *e.g.* installs Table-miss flow entries. The Device Database maintains for each device information concerning : (i) basic capabilities, such as the number of Ethernet ports, OpenFlow tables, and supported instructions, (ii) installed

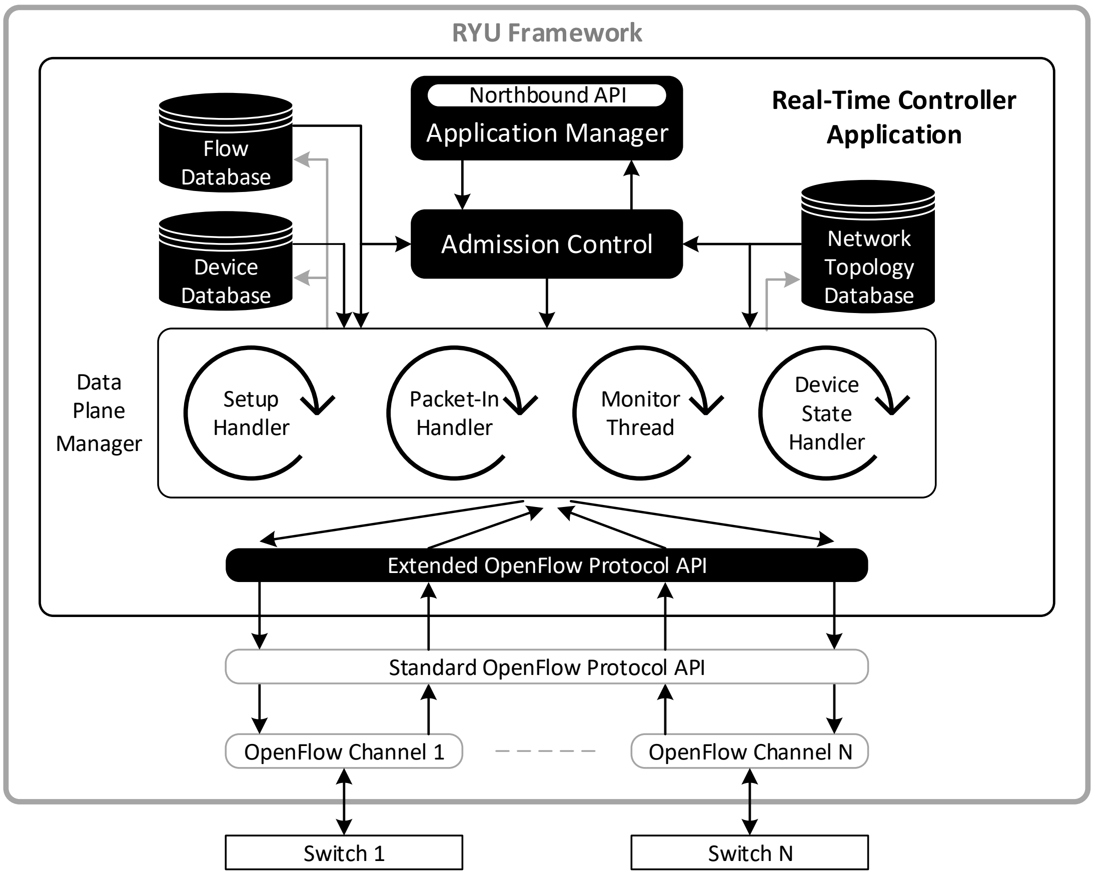


Figure 5.2: Main components of the real-time SDN controller

OpenFlow configurations, for example, the installed flow entries, (iii) statistic information retrieved from OpenFlow counters, and (iv) state information, such as the state (online or offline) of Ethernet ports. The Monitor Thread periodically polls statistics and installed OpenFlow configurations from each device, while the Device State Handler receives events sent by devices regarding state changes. Combined, they keep the Device Database up-to-date and consistent.

Applications communicate via northbound protocols with the Application Manager. As the northbound API is not yet defined, application communication requirements are deployed either offline or over a console. Application requests are then forwarded to the Admission Control unit which determines if a given traffic flow can be accepted into the network. In case of an affirmative decision, the Admission Control unit relays the request to the Data Plane Manager that then installs the adequate configurations on the data plane. If the flow is rejected, a denial notification is sent to the Application Manager and transmitted afterward to the requesting application. Extensions to the OpenFlow protocol API allow the Data Plane Manager to configure both standard OpenFlow services and real-time services on data plane switching devices. The specification and development of these extensions is presented in Section 5.3.

In order to run the admission tests, the Admission Control consults the Flow Database and the Network Topology Database to retrieve information concerning currently installed traffic flows and network topology, respectively. These databases are built and updated accordingly by the Data Plane Manager upon changes in the network and devices' state, for example, when a new switching device is connected into the network or an Ethernet port of some device is disabled. The network topology is modeled as a directed graph using the NetworkX library [124].

5.2.1 The admission control

When applications request for a new traffic flow, the communication system must first verify whether it is possible to guarantee the requested resources and constraints, *e.g.* temporal or bandwidth, without compromising the overall system, *i.e.*, the inclusion of new flows must not result in the system failing to provide the QoS required by existing communications. Requests that fail to meet this assertion are rejected, otherwise, the new flows are accepted into the system. This procedure is done at the implemented real-time controller application by the Admission Control unit.

The overall operation of the Admission Control unit is depicted in Figure 5.3.

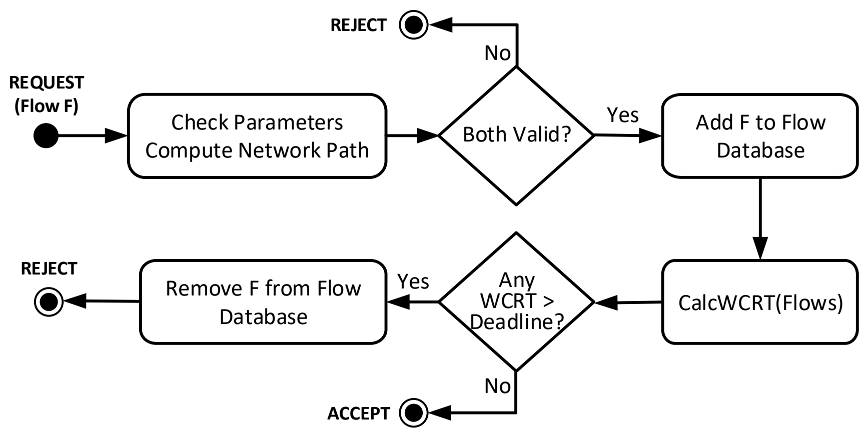


Figure 5.3: Overview of the admission control unit operation

First, it validates the request parameters, *i.e.*, checks if the request has all the necessary information and all values are within proper ranges. Then it analyzes the network topology and computes the network path for the new flow. If the request parameters are invalid, or a valid path does not exist, no further inspections are performed and the request is rejected. Finally, the Admission Control unit performs a set of algorithms to determine if the flow can be accepted. This process is executed in two main stages. First, an analysis is performed to check if the system is capable of meeting the QoS parameters for the new flow. For example, the response time for the new flow is calculated considering all the existing flows in the system, and compared to the flow's deadline. If the computed time is higher than the deadline, the requested QoS can't be guaranteed and thus, the flow is rejected. Secondly, the ability of the system to provide the QoS requirements demanded by each existing flows is evaluated as if the new flow is now part of the system. Using the previous example, the response time for every flow is computed considering the presence of the new flow and compared against their own deadlines. If the QoS can't be satisfied for all flows, the request is rejected.

In the following section, the network and traffic models, essential for the execution of any kind of QoS evaluation, are defined. The algorithms executed by the Admission Control unit for the assessment of the system's QoS conformance for traffic flows are explained in Section 5.2.1.2.

5.2.1.1 Network and traffic model

Network Model

The data plane of the proposed reference architecture (Section 5.1) comprises N_χ interconnected HaRTES switches. The model for each switch is akin to that of a store-and-forward Ethernet switch comprising a set of N_p Ethernet ports, Φ. All ports are considered to be operating at the same bit rate $\boldsymbol{\Xi}$ and in full-duplex mode. Therefore, frames can be concurrently transmitted and received. Switches impose switching delays to frames being forwarded through them. These delays comprise two components: (i) the *hardware fabric latency ($\boldsymbol{\epsilon}$)*, which corresponds to delays introduced by frame processing mechanisms, and (ii) the *store-and-forward delay ($\boldsymbol{\Upsilon}$)*, which represents the time required to receive a frame before forwarding it to the egress port. The former delay depends on the speed and architecture of switches' internal processes. An upper bound for the fabric latency can be determined, for example, experimentally. The latter delay is related to the frame size ($\boldsymbol{\Lambda}$), which includes all headers and overheads, and the switch ports' bit rate. It can be computed as follows:

$$\Upsilon = \frac{\Lambda}{\Xi} \tag{5.1}$$

All switches in the network are configured with equally sized Elementary Cycles. The length of the EC is denoted by ΔEC. The size of a synchronous and asynchronous window is denoted by ΔSW and ΔAW, respectively. The EC must have both synchronous and asynchronous windows, and these must not overrun the EC. That is, for $\Delta EC, \Delta SW, \Delta AW \in \mathbb{N}$ it follows that $(\Delta SW + \Delta AW) < \Delta EC$. The length of synchronous and asynchronous windows may differ between switches and/or ports of a given switch, however, interconnected ports must exhibit the same window configuration. Therefore, for an interconnected pair of ports A and B, it follows that: $\Delta SW_A = \Delta SW_B \ \wedge \ \Delta AW_A = \Delta AW_B$.

In summary, the set of all switches in the reference architecture is defined throughout this work as

$$X = \{\chi_i | \ \chi_i = \{\epsilon_i, \ \Delta EC, \ \Phi_i\} \quad , i = 1, 2, .., N_\chi\} \tag{5.2}$$

with the set of ports for a given switch χ_i being described as

$$\Phi_i = \{\phi_j | \ \phi_j = \{\Xi, \ \Delta SW_j, \ \Delta AW_j\} \quad , j = 1, 2, .., N_p\} \tag{5.3}$$

Every physical connection between two switches or between a node and a switch are modeled as a set of two unidirectional links, one per each direction (ingress or egress). Each link is denoted by ł . Propagation delays in links are ignored (considered to be 0). The set of all links in the network, Ł, is presented as:

$$Ł = \{ł_x, \ x = 1, 2, .., N_ł\} \tag{5.4}$$

Traffic Model

Concerning network traffic, the employed model restricts flows (streams) to unicast transmissions. Nonetheless, multicast and broadcast traffic flows can be included as a set of multiple unicast streams. Moreover, frames from a given stream cannot be preempted during transmission.

Time-triggered streams are represented following the traditional real-time periodic model. Therefore, a set of N_{ps} periodic streams, $\boldsymbol{\Gamma}$, is expressed as:

$$\Gamma = \{PS_i | \ PS_i = (\Lambda_i, \ C_i, \ T_i, \ O_i, \ D_i, \ P_i, \ Ł_i) \quad , i = 1, 2, .., N_{ps}\} \tag{5.5}$$

Each stream is defined by the transmission time C_i of a maximum sized frame, the periodicity of transmissions T_i, an initial offset O_i expressing the instant of the first transmission with respect to a reference time instant, the relative deadline D_i, the fixed stream's priority P_i, and the set of links $Ł_i$ that stream PS_i crosses through. All parameters are expressed by positive integer numbers, with the exception of O_i which can also be 0. C_i is a direct function of the maximum frame size Λ_i, which includes OSI layer 2 headers and the FCS field, and takes into account all OSI layer 1 overheads for its computation. For the considered switched Ethernet networks, C_i is computed following Equation 5.6. The start of frame (SOF) and Inter-Packet Gap (IPG) are sized 8 and 12 bytes, respectively.

$$C_i = \frac{SOF + \Lambda_i + IPG}{\Xi} \tag{5.6}$$

The model is assumed to be constrained, that is, $\forall_{i=1,2,..,N_{ps}} \ D_i \leq T_i$. Periods, offsets, and deadlines are all expressed as integer multiples of the EC duration, that is:

$$C_i, D_i, T_i \in \mathbb{N}, O_i \in \mathbb{N}_0 \mid \forall x \in \{C_i, \ D_i, \ T_i, \ O_i\} : \ x \equiv 0 \ (mod \ \Delta EC) \ , i = 1, 2, .., N_{ps} \tag{5.7}$$

The priority P_i for all streams is assigned following the fixed-priority Rate-Monotonic (RM) scheduling policy. Therefore, streams with the same period have the same priority level. Three subsets, *hp(PS_i)*, *lp(PS_i)*, and *hep(PS_i)*, identify the sets of streams with higher, lower, and higher or equal priority than a given stream PS_i, respectively. These are formally described as follows:

$$hp(PS_i) = \{PS_x | \ PS_x \neq PS_i \ \wedge \ P_x > P_i \quad , x = 1, 2, .., N_{ps}\} \tag{5.8}$$

$$lp(PS_i) = \{PS_x |\ PS_x \neq PS_i \ \wedge \ P_x < P_i \quad , x = 1, 2, .., N_{ps}\} \tag{5.9}$$

$$hep(PS_i) = \{PS_x |\ PS_x \neq PS_i \ \wedge \ P_x \geq P_i \quad , x = 1, 2, .., N_{ps}\} \tag{5.10}$$

$Ł_i$ comprises the set of $Nł_i$ links traversed by frames pertaining to stream PS_i. It is presented as follows:

$$Ł_i = \{ł_x |\ x = 1, 2, .., Nł_i\}, \quad Ł_i \subset Ł \tag{5.11}$$

The set of links traversed by a stream PS_i along its network route, from a specific link $ł_a$ until another specific link $ł_b$, is given according to the following definition:

$$Ł_{i,a,b} = \{ł_y |\ y = a, a+1, .., b\}, \quad 1 \leq a \leq b \leq Nł_i,\ Ł_{i,a,b} \subseteq Ł_i \tag{5.12}$$

The total response time for a frame of stream PS_i is denoted as RT_i and corresponds to the time lapse between the instant in which the frame becomes ready at the sender's interface and the instant in which its reception at the receiver's interface terminates. Additionally, response times can also be computed for transmissions between a specific pair of links. In this case, $RT_{i,a,b}$ corresponds to the time interval since the frame has been placed for transmission in the output queue of the egress link $ł_a$ and the instant in which the frame is completely received by the interface connected to the ingress link $ł_b$. All response times are expressed as multiples of ΔEC.

The scheduling and forwarding of time-triggered traffic is performed according to the Reduced Buffering Scheme (RBS) [4], [67]. Following RBS, traffic produced and sent through a single switch is defined as local traffic while traffic that crosses multiple switches is known as global traffic. In order to increase efficiency, all switches are synchronized in time, e.g. using IEEE 1588, and the size of the synchronous window for each link may be differentiated and selected according to foreseen local and global loads. Local traffic is managed in accordance with the normal operation of HaRTES for a single switch (recall Section 4.3.2). Global traffic is scheduled only at ports that are physically connected to producer nodes. Then, it is forwarded through interlinks belonging to the streams' routes while there is enough time within the links' synchronous windows. If an interlink does not have enough time left in the present synchronous window, the respective switch stores the received global frames into dedicated priority queues at the respective egress port. Frames stored at priority queues are forwarded in the following elementary cycle in descending order of priority. Figure 5.4 presents a scenario showing the transmission of a frame from a given stream PS_i throughout a multi-hop network.

In the supplied example, stream PS_i has node A and node B as producer and consumer, respectively. Therefore, the set of links for this stream is $Ł_i = \{ł_1, ł_2, ł_3, ł_4\}$. At the beginning of EC n, HaRTES H2 sends a Trigger Message (TM) to node A (ingress $ł_8$) polling the transmission of a PS_i frame. Node A stores the requested frame in its egress queue and transmits the frame through $ł_1$ at the beginning of the synchronous window. This frame is completely received by H2 after the store-and-forward delay (Υ) which then processes and

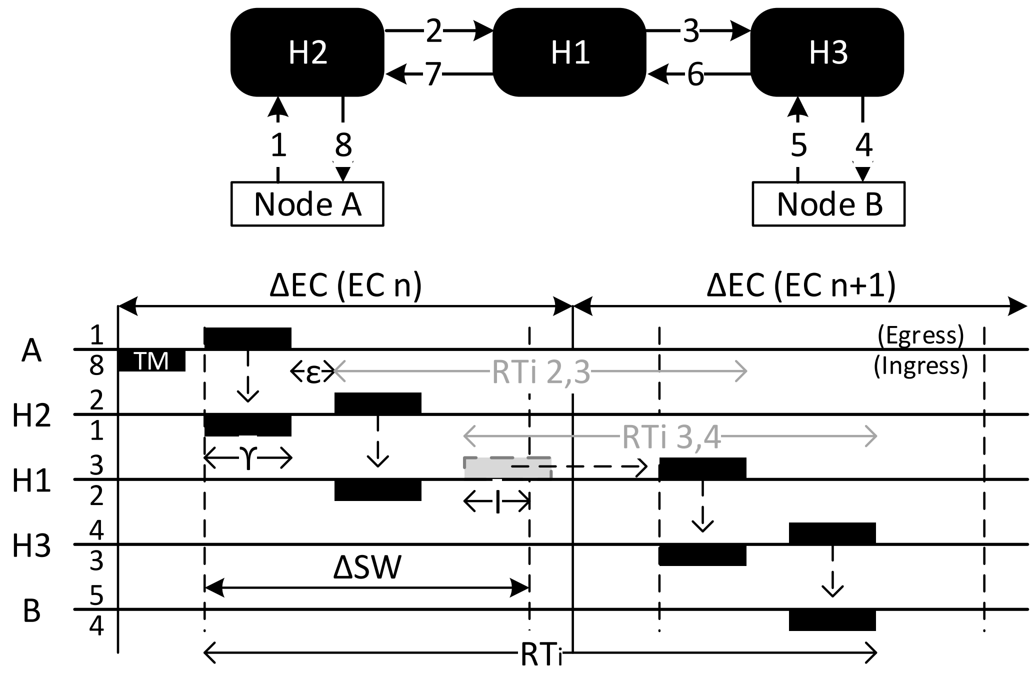


Figure 5.4: Reduced Buffering Scheme (RBS) example

stores it at the respective egress queue, i.e. egress queue of l_2. This queuing process adds a delay of ϵ to the transmission of PS_i to switch H1. The same process is repeated by H1, however, there is now no space in H1's synchronous window to transmit the received frame. Therefore, the frame is stored at the appropriate egress queue and transmitted in the following elementary cycle. This causes an idle time I at the end of the synchronous window in EC n. Finally, H3 receives the frame and forwards it to the destination node.

5.2.1.2 The schedulability analysis for synchronous traffic

This work employs the response time analysis proposed in [4], [67] to perform the admission control of time-triggered traffic. The aforementioned analysis is specifically designed for networks comprising several interconnected HaRTES switches and the RBS forwarding scheme. In short, the devised analysis calculates the response time of traffic from the source to the sink node, link-by-link, while checking whether frames are buffered in any switch along the route. Algorithm 1 [67] describes how the response time for a given stream can be calculated using the described approach.

Algorithm 1 Response time calculation for stream PS_i

```
1: $RT_i = 0$
2: $a = b = 1$
3: while $b \leq Nł_i$ do
4:     $RT_{i,a,b} = responseTimeCalc(PS_i, Ł_{i,a,b})$
5:     $RT_{i,a,b} = \lceil \frac{RT_{i,a,b}}{\Delta EC} \rceil$
6:     if $(a! = b)$ && $(RT_{i,a,b}! = RT_{i,a,(b-1)})$ then
7:         $RT_i = RT_i + RT_{i,a,(b-1)}$
8:         $a = b$
9:     else
10:        $b = b + 1$
11:    end if
12: end while
13: $RT_i = RT_i + RT_{i,a,(b-1)}$
```

The algorithm starts by setting the initial parameters, *i.e.*, the total response time RT_i is set to zero (line 1) and the link indexes a and b are set to select the first link in PS_i's link set $Ł_i$ (line 2). The main loop (line 3) computes the response time, link-by-link, until the last link ($Nł_i$) in $Ł_i$. It starts by computing the response time for PS_i frames traversing the subset $Ł_{i,a,b}$ (line 4), rounding the resulting response time up to a multiple of ΔEC, *i.e.* number of elementary cycles (line 5). The rounded response time is then compared to that obtained in the previous iteration (line 6). However, this is only done when $ł_a \neq ł_b$, that is, only when the response time has been computed over two or more different links. This is to account for: (i) the fact that frames can not be buffered at the egress link of a node due to having being scheduled by the switch, and (ii), when a frame is determined to have been buffered by a switch (line 6), the algorithm re-initializes the computation to the egress link of the buffering switch (line 8). Buffering is detected when the computed response time for a subset $Ł_{i,a,b}$ comprising two or more links differs from the result obtained from the calculation up to the next-to-last link in the same subset, *i.e.*, $Ł_{i,a,(b-1)}$ (line 6). The reasoning behind this is the fact that for all possible sets $Ł_{i,a,b}$ comprising at least two distinct links, *i.e.* $a \neq b$, the next-to-last and the last elements forcefully correspond to the ingress and egress link of the same switch. Therefore, if the response time up to the ingress link of that switch differs (in number of ECs) from the one up to its egress link, it is safe to assume that the frame will be buffered by this switch. In this case, the computed response time up to the ingress link is added to the total response time and the computation starts anew from the egress link of the buffering switch (line 7 and 8). If the frame is found not be buffered, the next link in $Ł_i$ is added to the response time calculation of the next loop iteration (line 10). The total response time is found when the main loop reaches the last link in the route of PS_i (line 13).

The response time of PS_i for a given subset $Ł_{i,a,b}$ (line 4 of Algorithm 1) is based on the classical response time analysis employing the accumulation of delays within iterations. The response time computation is described by Equation 5.13.

$$rt_{i,a,b}(x) = \frac{C_i}{\alpha_{i,a,b}} + I_{i,a,b}(x) + B_{i,a,b}(x) + SD_{i,a,b}(x) \tag{5.13}$$

The calculation considers the transmission time C_i of the PS_i frame itself as well as three types of interference that may arise from the existence of other time-triggered streams in the network: (i) $I_{i,a,b}$, the interference from messages with higher or equal priority that have links in common with PS_i, (ii) $B_{i,a,b}$, blocking from messages with lower priority that share links with PS_i, and (iii) $SD_{i,a,b}$, the switching delay for the stream across the entire route. The computation of the response time is complete when the iteration of Equation 5.13 converges, *i.e.*, $rt_{i,a,b}(x) = rt_{i,a,b}(x-1)$. The first iteration computes $rt_{i,a,b}(0) = \frac{C_i}{\alpha_{i,a,b}}$.

As transmissions of time-triggered frames are confined to the synchronous window, an inflation factor $\alpha_{i,a,b}$ is used to adequately compute the transmission time of frames within the window [6]. The calculation of this factor is described by Equation 5.14.

$$\alpha_{i,a,b} = \frac{min(\Delta SW_l - I_l)}{\Delta EC} \quad, l = a, a+1, .., b \tag{5.14}$$

To compute $\alpha_{i,a,b}$ while accounting for the worst-case scenario, the minimum sized window among all links in $Ł_{i,a,b}$ and the maximum idle time that may occur in the aforementioned window are used. The maximum idle time is calculated following Equation 5.15.

$$I_l = max(\Lambda_i, \Lambda_x), \quad \forall x \in [1, N_{ps}] \;\wedge\; PS_x \in hep(PS_i) \;\wedge\; ł_l \in Ł_x \tag{5.15}$$

The computation for the first kind of interference, $I_{i,a,b}$, is straight- forward and is given by Equation 5.16. Note that the transmission time for interfering messages C_j must also be inflated by $\alpha_{i,a,b}$.

$$I_{i,a,b} = \sum \left(\left\lceil \frac{rt_{i,a,b}(x-1)}{T_j} \right\rceil * \frac{C_j}{\alpha_{i,a,b}} \right), \forall_{\mathrm{j}} \in [1, \mathrm{N_{ps}}] \wedge PS_j \in hep(PS_i) \wedge Ł_j \cap Ł_{i,a,b} \neq \emptyset \tag{5.16}$$

The second interference type, $B_{i,a,b}$, is computed by Equation 5.17. Note that a particular stream with lower priority can only block PS_i once throughout the route.

$$B_{i,a,b} = \sum_{\substack{t=a+1 \\ a \neq b}}^{b} max\left(\frac{\Lambda_x}{\alpha_{i,a,b}}\right), \forall_x \in [1, N_{ps}] \wedge PS_x \in lp(PS_i) \wedge ł_t \in Ł_x \wedge \forall_y, a+1 \leq y \leq t, ł_y \notin Ł_x \tag{5.17}$$

Finally, the accumulated switching delay for the stream along the whole route is supplied by Equation 5.18. The computation of this last term, albeit resulting into a somewhat simple equation, has some interesting details. Authors are referred to [4] for some insight on this matter.

$$SD_{i,a,b} \sum_{\substack{t=a+1 \\ a \neq b}}^{b} max\left(\frac{\Upsilon_i, \Upsilon_x}{\alpha_{i,a,b}}\right) \quad, \forall x \in [1, N_{ps}] \;\wedge\; ł_t \in Ł_x \;\wedge\; ł_{(t-1)} \in Ł_x \tag{5.18}$$

The source code for the implementation of the presented analysis can be consulted in Appendix D.

5.3 A real-time empowered control plane

Software Defined Networking (SDN) frameworks are primarily designed to provide an abstraction of the network lower-level functionality so as to allow network administrators to dynamically configure and manage network resources in a flexible and convenient way, without having to dwell on complex protocols. The development of such frameworks didn't consider the support and management of real-time communications and, as consequence, they do not provide traffic models and APIs that are adequate for this type of communications. OpenFlow, being a southbound protocol for SDN platforms, is no exception.

OpenFlow relies on two main services to enable limited QoS support: (i) an action to forward traffic to specific output queues in the data plane, and (ii), traffic shapers that restrict traffic flows according to configurable bandwidth thresholds. Although one could use a set of output queues to, for example, implement priority-based queuing schemes, these are only configured outside the OpenFlow protocol, *e.g.* through data plane vendors' command line tools or dedicated configuration protocols. Additionally, the available shapers are only able to limit a given flow's maximum bandwidth, which is configurable by either setting the maximum number of packets per second or kilobits per second (kbps). One could use these services to provide QoS guarantees to event-triggered traffic, however, the offered guarantees would be severely constrained. The support for time-triggered traffic with standard OpenFlow is simply non-existant.

It is clear that extensions to the current OpenFlow protocol[1] are essential in order to enable adequate real-time services. To this end, two distinct approaches may be pursued: (i) modify the current OpenFlow services and API, adding new services to the protocol as needed, or (ii), keep standard OpenFlow services and API intact, while providing a non-intrusive real-time add-on with its own set of services and API.

The former approach requires that several objects and features of OpenFlow are modified. For example, flow table entries would have to be modified so as to be able to categorize traffic as time-triggered, event-triggered or non-real-time traffic. Within the same traffic category, several flows may be served by distinct real-time services, *e.g.* deferrable servers, and so, at the end of the pipeline processing real-time flows must be directed to the respective services. Moreover, each real-time flow has a set of requirements which are specified through a set of metrics, *e.g.* period and deadline, that are inexistent in OpenFlow APIs. Thus, present APIs and message structures would have to be modified and/or extended, and a database to store each real-time flow's requirements would have to be implemented. Since this approach is significantly intrusive and could hinder the extensions adoption by the standard, as well as being a hindrance for the update of our developed framework to future OpenFlow version releases, it will not be engaged. The latter approach keeps the current standard unaltered and introduces a separate real-time add-on with its own set of messages, services and API. This add-on will follow the philosophy of OpenFlow and its API and messages will reuse as much as possible common structures already defined by the standard. For example, OpenFlow

[1]At the time of writing the latest published version of the standard was v1.5.0

defines data structures for messages' headers and contents which are compatible and can be adopted by the real-time add-on. Naturally at a certain point, the structure of messages will have to diverge but the introduced changes will closely follow the mindset of OpenFlow. Next, we will take a look on the developed add-on module.

5.3.1 The OpenFlow real-time add-on

The proposed add-on architecture, and its main components, is depicted in Figure 5.5.

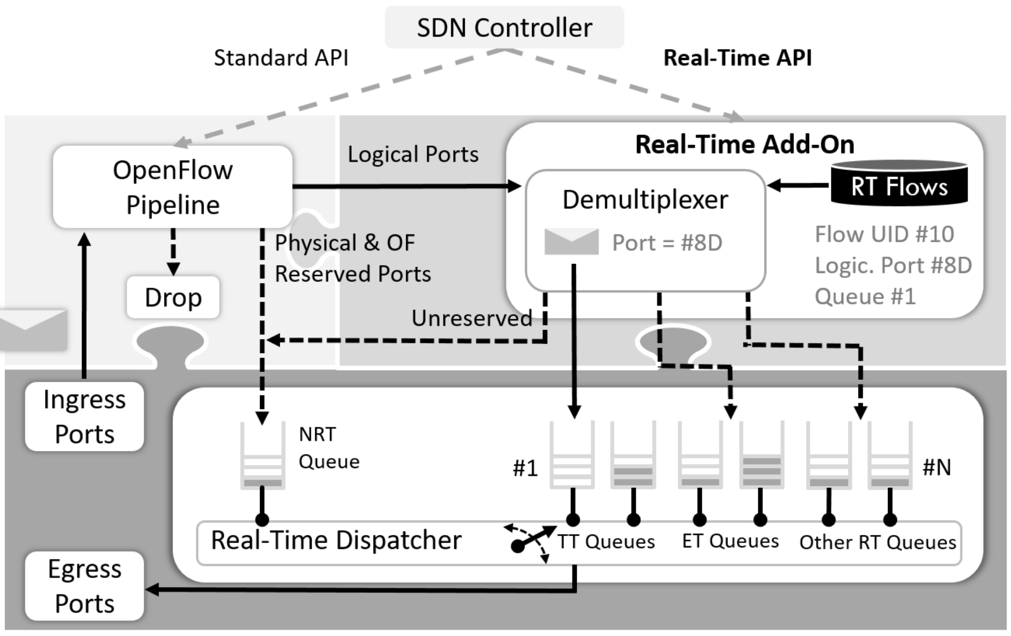


Figure 5.5: Real-time OpenFlow add-on

Two distinct domains exist: the standard OpenFlow domain and the real-time OpenFlow add-on domain. The standard domain implements the OpenFlow pipeline with all the objects and services defined by OpenFlow completely unchanged. As in the original *modus operandi*, flow, group and meter tables are configured by the controller through the standard OpenFlow switch protocol API in order to rule the pipeline processing and forward flows to the desired OpenFlow output ports. These ports may be: (i) physical, which are directly mapped to switching hardware ports, (ii) reserved, *i.e.* logical entities that are associated to specific OpenFlow-defined processing such as flooding packets to several physical ports, and (iii), logical ports that may be related to processing instructions that are defined outside the OpenFlow protocol.

Logical ports are the cornerstone of the add-on. In OpenFlow, there are a total of $2^{24} - 1$ ports that can be freely used as either physical or logical. The real-time add-on defines a subset of these ports to be used for real-time processing, where each logical port is linked to a given real-time stream and real-time service, *e.g.* time-triggered or event-triggered. A controller may query the switch regarding logical ports that are reserved and supported real-time services. Using this information, the controller can then configure the pipeline to correctly identify and direct filtered flows to the desired real-time services. Note that within the standard OpenFlow domain there is no explicit notion of real-time and non-real-time flows; flow entries, their matching rules, and pipeline processing outcomes are kept unchanged. Only the controller knows how to distinguish flows and it must configure flow table entries to properly filter and direct packets to non real-time or real-time ports.

Real-time traffic streams and their properties have to be provided so that reservations are made and the appropriate real-time services configured. This process happens at the real-time add-on domain. The controller registers real-time streams into the database of switching devices through the real-time API. Devices then translate each stream entry into reservations that are enforced by the appropriate real-time services. These services must process the received traffic from related streams and take adequate actions to enforce the negotiated real-time behaviour. How these services are implemented and operate is not defined by the add-on since it is intended to be technologically agnostic and not tailored for a particular real-time protocol. It is the responsibility of the device to deny requests whose real-time timings cannot be guaranteed by the implemented services. Nonetheless, data plane devices do must implement two distinct dispatchers: one to dispatch non real-time traffic and another for real-time traffic. The dispatching of real-time traffic should always take precedence over the non real-time.

In summary, the process that a controller must conduct in order to successfully add a real-time traffic flow into the system is:

1. ***Configure the appropriate real-time services***. The controller uses the add-on API to register the real-time stream into each data plane device across the stream's route, providing information such as the ID of the stream being registered and its real-time properties, *e.g.* type of real-time service and timing requirements (see Section 5.3.1.1). Note that the specified ID is unique and (indirectly) defines the logical port for the OpenFlow pipeline;

2. ***Configure the OpenFlow pipeline***. Using the standard OpenFlow API, the controller configures the necessary rules, *i.e.* flow entries with adequate matching rules, instructions, and actions, to redirect matching traffic to the real-time logical port that is linked to the stream registered in step 1. This step must be executed for each device across the stream's path.

In case one of the aforementioned steps fails, *e.g.* a switching device denied the request due to an error and/or unsupported request, the controller must abort the process, undo all previous configurations, and report to the application. Traffic from successfully admitted flows are processed on each device according to the flowchart of Figure 5.6.

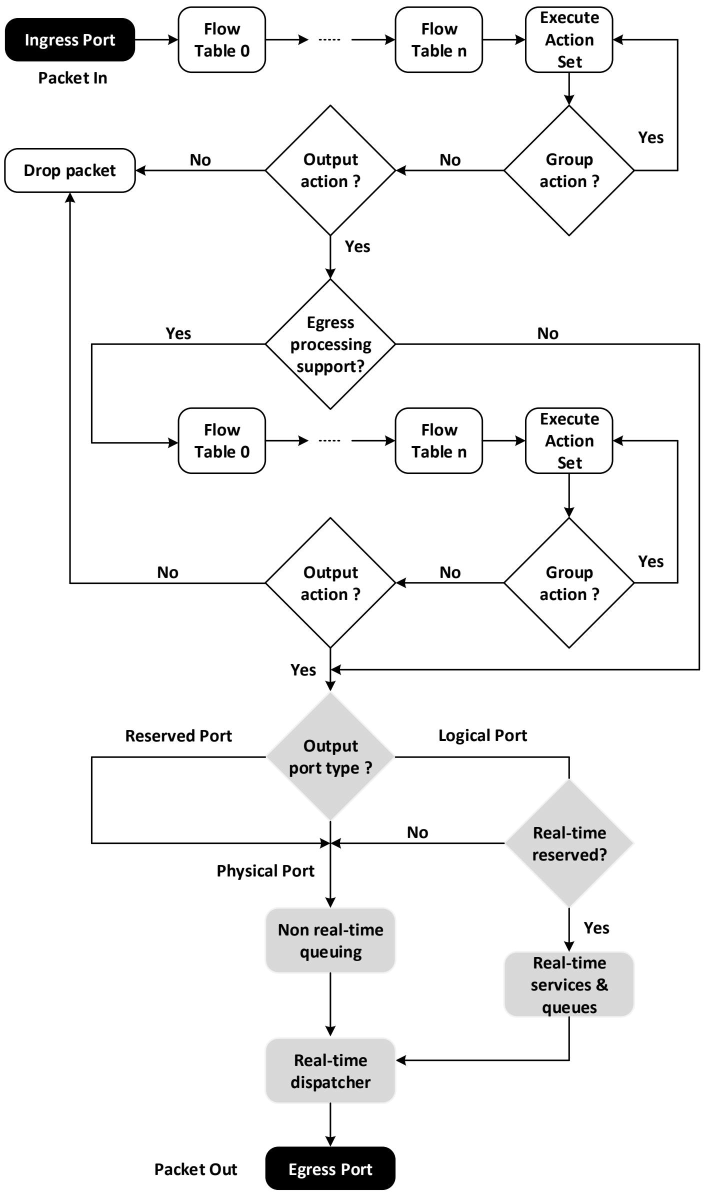


Figure 5.6: Traffic processing flowchart

Traffic received at an ingress port goes through the original OpenFlow pipeline, going through one or more flow tables until it matches a flow entry or is dropped, e.g. upon not matching any entry. At the end of the processing pipeline, traffic with an associated output action may either go to: (i) a reserved port, (ii) a physical port, (iii) a non real-time logical port, or (iv), a real-time logical port. Traffic for the first three types of ports is stored in standard non real-time queues, while traffic for real-time ports is stored in queues managed by real-time services. The real-time dispatcher then serves both non and real-time queues according to its internal algorithm. Traffic with no output action or valid ports is dropped.

The real-time API of the OpenFlow add-on is presented in the next section.

5.3.1.1 Real-time OpenFlow API

One of the most important components of the real-time OpenFlow add-on is the Real-Time API, summarized in Table 5.1, which implements the interface between controllers and real-time services. Controllers use this interface to configure and maintain real-time reservations on data plane switching devices. In short, it uses specific messages to add, remove and modify real-time stream reservations, get the list of registered real-time streams and the individual parameters of existing reservations, as well as the real-time services supported by the device.

Table 5.1: Real-time API

Message	Description	Sender
RT_ST_ADD	Add stream to the add-on database.	Controller
RT_ST_MODIFY	Modify properties of an existing stream.	Controller
RT_ST_DELETE	Removes stream from add-on database.	Controller
RT_ST_LIST_REQUEST	Get the ID of all streams of a given type.	Controller
RT_ST_LIST_REPLY	Reply to RT_ST_LIST_REQUEST.	Switch
RT_ST_PROP_REQUEST	Get properties of a given stream.	Controller
RT_ST_PROP_REPLY	Reply to RT_ST_PROP_REQUEST.	Switch
RT_ST_REMOVED	Notification of stream removal events.	Switch
RT_ST_STATS_REQUEST	Get properties and statistics of all streams.	Controller
RT_ST_STATS_REPLY	Reply to RT_ST_STATS_REQUEST.	Switch
RT_ST_CAP_REQUEST	List supported real-time services.	Controller
RT_ST_CAP_REPLY	Reply to RT_ST_CAP_REQUEST.	Switch

Real-time reservations are specified by a unique ID, type, and traditional real-time properties: period or minimum inter-arrival time, deadline, offset, priority, frame length, and a list of associated producers and consumers. Three types of streams are defined: (i) time-triggered (TT), (ii) event-triggered (ET), and (iii) other (OT). The first two types are for traffic that follows the respective activation paradigm, while the latter provides some flexibility to associate streams to other processing algorithms. The list of producers and consumers main function is to identify the device's ingress and egress ports for that particular stream. Akin to OpenFlow flow entries, each reservation can be configured with idle and hard timeouts. Finally, request messages contain a miscellaneous field which can be used to provide information outside the specification of the add-on. The objective of this field is to provide a wider range of compatibility with proprietary technologies and implementations. A miscellaneous field also exists on each entry of consumer and producer lists. Table 5.2 provides a description and value range for each of the aforementioned parameters.

A few remarks, the priority of a stream conveys an absolute priority level among all streams forwarded by the same device. It is computed by the controller and it may derive from a combination of parameters given by applications, such as criticality or a global priority, as well as from scheduling algorithms, such as Rate Monotonic (RM) or Deadline Monotonic (DM). For the time being, since the northbound API is not yet defined, the computation of this priority simply follows the RM scheduling algorithm. The frame length is also computed

Table 5.2: Real-time API parameters used to describe a real-time flow

Parameter	Description	Value range
UID	Unique stream identifier.	$]0, 2^{32} - 1]$
Type	Type of real-time stream.	TT,ET,OT
Period	TT traffic: periodicity of frames. ET traffic: minimum time between consecutive frames. OT traffic: minimum time between consecutive frames.	$]0, 2^{32}-1]$ μs
Deadline	Frame's relative deadline.	$]0, 2^{32}-1]$ μs
Offset	TT traffic: relative phasing. ET traffic: ignored parameter. OT traffic: ignored parameter.	$]0, 2^{32}-1]$ μs
Priority	Absolute priority. Higher value equals a higher priority level.	$[0, 2^{16} - 1]$
Frame length	Maximum length, in bytes, of a stream's OSI layer 2 frame. Includes all headers, payload and FCS.	$[68, 2^{16} - 1]$
Producers	List of stream's ingress ports.	- - -
Consumers	List of stream's egress ports.	- - -
Idle timeout	Maximum time interval with no received traffic before entry expires. If 0, stream is never removed.	$[0, 2^{32} - 1]$ s
Hard timeout	Time interval before stream expires. If 0, stream is never removed.	$[0, 2^{32} - 1]$ s
Miscellaneous	For purposes outside the add-on specification.	- - -

by the controller. Applications report to the controller the length of data that they produce; the controller then considers the application payload length and protocol overheads, *e.g.* from the network and transport layers, to compute the maximum frame length. If application data has to be split into several Ethernet frames, *e.g.* size greater than the maximum transmission unit (MTU), the controller must compute adequate frame length, period, and deadline values to account for the several fragments.

The implementation of all messages for the Real-Time API follows the existing OpenFlow message structure mindset, and thus, all messages start with the common OpenFlow header, and conveyed data structures are 8-byte aligned and packed with padding. To implement the message payload, two approaches are possible: (i) define new message types and data structures for the OpenFlow message type space, or (ii), use experimenter messages, defined by the standard for non-standard functionalities within the OpenFlow message type space. Again, in order to remain the standard untouched, the latter approach was taken.

Experimenter messages include:

- ***Common OpenFlow header (ofp_header)***. Conveys the protocol version, identifies the type of message, in this case an experimenter, the length of the message including this header, and a transaction ID to facilitate request/reply pairing;
- ***Experimenter field***. Uniquely identifies the entity defining the extension. Assigned by the Open Networking Foundation upon request;

- ***Experience field.*** Identifies the type of message encapsulated in the experimenter message payload. It is used to indicate the type of Real-Time API message within the payload;

- ***Payload (experimenter_data).*** Carries the Real-Time API message payload, whose contents may vary depending on the type of message.

An example of the structure of the RT_ST_ADD message is shown in Figure 5.7. The implementation of the remaining messages follows a similar structure[2]. For more details, the reader is referred to Annex B that contains the C header for the implemented API.

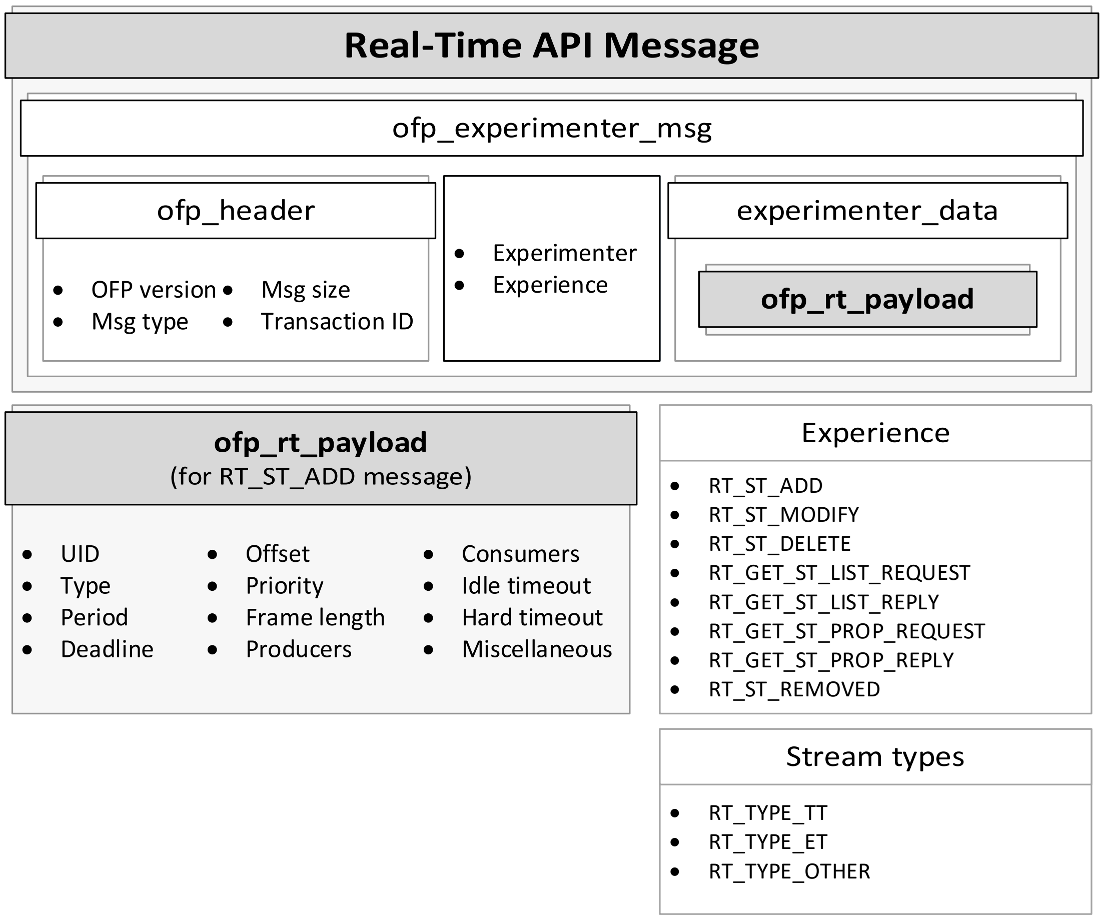


Figure 5.7: Structure of a Real-Time API message

[2]Due to their potential size, RT_ST_STATS_{REQUEST,REPLY} use OpenFlow multipart experimenter messages. Nonetheless, their structure is very similar.

5.4 A real-time Ethernet data plane

The reference architecture for the real-time SDN framework (Section 5.1) relies on HaRTES switches for the implementation of a data plane empowered with real-time services. Despite providing a suitable platform for complex applications with demanding timing and flexibility requirements, HaRTES is not designed according to the SDN paradigm, *i.e.* there is no decoupling of network control functions from the forwarding plane, and lacks the basic services required by the OpenFlow standard. In addition, HaRTES's API (see Section 4.3.2.2) is tailored for the configuration of its real-time services and therefore, is incompatible with the OpenFlow switch protocol.

Hence, in order to integrate HaRTES in the proposed framework it is necessary to: (i) remove its network control functions, *e.g.* learning forwarding tables, (ii), implement a basic OpenFlow pipeline, *e.g.* configurable flow tables, entries, and actions, and (iii), create an OpenFlow-compatible interface, which is able to communicate with controllers and translate requests from the OpenFlow domain into proper HaRTES commands. To this end, three possible architectures can be pursued:

- ***Hardware-oriented architecture***. HaRTES is currently fully implemented in hardware technology (FPGA). Therefore, a natural approach is to expand the current implementation with the required OpenFlow services and interface layer;
- ***Software-oriented architecture***. A second approach is to fully implement both OpenFlow and HaRTES services in software. Open-source, software versions for both HaRTES (HaRTES-SW) and OpenFlow-compatible virtual switches already exist. OpenFlow services and API could be extracted from such OpenFlow virtual switches, *e.g.* Open vSwitch[3], and incorporated into HaRTES-SW;
- ***Hybrid architecture***. This approach combines the flexibility of software with the performance of a hardware-based implementation. This can be achieved by implementing performance-demanding services in hardware, *e.g.* real-time services and switching, and realize complex management operations in software, *e.g.* interpretation of OpenFlow messages.

The first approach would most likely exhibit the highest performance among all approaches since both OpenFlow and HaRTES services would be fully implemented in hardware logic. However, it also exhibits the highest level of complexity due to the intricate nature of the OpenFlow protocol. For example, controllers communicate with data plane devices over a dedicated communication channel which is typically encrypted using the TLS protocol or runs directly over the TCP/IP protocol. The setup and maintenance of this channel requires the execution of several functions, such as establishing the connection using Uniform Resource Identifier (URI) addresses, monitoring the liveness of the channel with echo-reply messages, and channel reestablishment procedures. If several controllers exist, multiple control channels

[3]Homepage: https://www.openvswitch.org/

have to be instantiated and managed. A process to elect the OpenFlow master controller and remaining slave controllers must also be supported. Moreover, OpenFlow messages are built over a significantly large set of complex data structures comprising optional fields, padding, and variable-sized lists. Although these messages and conveyed operations are easily decoded using a software programming language such as C, it would require complex hardware logic. An additional drawback is the lack of flexibility to be easily modified to comply with future versions of the protocol. Due to aforementioned reasons, this approach was not executed.

The second approach exhibits the highest flexibility of all and would likely require minor changes to the current HaRTES-SW version, considerably lowering the implementation effort. However, as with most pure software-based solutions, the real-time performance of HaRTES-SW is poor and insufficient for real-time applications with strict timing requirements such as loop-controlled physical processes that are extremely common in industrial scenarios. Therefore, this approach is also discarded.

As for the third approach, two slightly different schemes were considered:

- ***Software-based OpenFlow Processing and Hardware-based Switching.*** Implement OpenFlow services, such as the flow processing pipeline and its objects, in software. The remaining data plane services, such as packet storage, dispatching and real-time services, are performed in hardware;

- ***Software-based OpenFlow Mediator and Hardware-based Forwarding Pipeline.*** Implement OpenFlow management related functions, such as maintenance of the communication channel with the OpenFlow controller and interpretation of OpenFlow messages, in software. The OpenFlow processing pipeline as well as the remaining data plane services are implemented in hardware.

Following the first proposition, OpenFlow and HaRTES services are kept completely separated. OpenFlow services would fully reside in software, taking advantage of the inherent flexibility of software programming languages and resources to execute the complex OpenFlow management and traffic control services. In this way, entities related to flow management, such as flow tables, group tables, meters and counters, as well as the dedicated communication channels to the controller, would be implemented, configured and managed in software. In contrast, the remaining data plane functions would be realized in hardware, taking advantage of the inherent extreme performance to perform operations, such as switching and dispatching, with very low latency and jitter values. Traffic management and storage entities, input and output queues, as well as the mechanisms to provide communications with real-time guarantees would be implemented, configured and managed in hardware.

The aforementioned storage, switching and real-time services are already functional and embedded in HaRTES switch. The nonexistent OpenFlow entities and algorithms could be retrieved from existing OpenFlow virtual switches. To run the software services, an external CPU or one embedded in the FPGA can be used. Since the available resources of the hardware platform used to implement HaRTES are currently limited, an external CPU, in this case a host PC, is necessary.

For the proper operation of the OpenFlow protocol, services at the host PC and the remaining data plane services in HaRTES must be able to communicate. For example, headers and specific data fields of received packets are vital for the proper operation of the OpenFlow pipeline. Therefore, a copy of each received packet must be sent to the software platform. Additionally, switching services must be instructed with the outcome of the pipeline processing to properly forward packets. In order to create a dedicated communication channel between both platforms two physical interfaces are available: (i) an Ethernet port, and (ii), a PCI interface. Either one of these can be used to instantiate the required communication channel, however, resorting to the PCI interface is preferable since, in its current implementation, the HaRTES prototype has a low number of available Ethernet ports (only 4).

Although this scheme is able to implement complex OpenFlow pipeline processing, it may suffer from performance issues, in particular, extra latency and jitter on the whole packet forwarding process due to frequent exchanges of large quantities of control data between the hardware and software platforms. Moreover, as flow matching and other pipeline operations are executed in software, an additional penalty on the forwarding performance and determinism is introduced. Although the use of a real-time operative system would ameliorate some of the aforementioned problems, experience with the HaRTES software version has shown[4] that it might still not be enough to attain latency and jitter within acceptable values for some applications.

The second scheme, presented in the next section, overcomes the aforementioned issues, and is the one employed for the enhancement of HaRTES.

5.4.1 The SDN augmented HaRTES

To overcome the inherent issues of the previously presented approach, an improved design was devised. The major difference between the two solutions is the migration of a subset of OpenFlow objects and services to the hardware platform. In particular, the whole OpenFlow pipeline is now implemented in hardware logic, while remaining services, *e.g.* that deal with requests from/to the controller, are performed by an OpenFlow Mediator (OFM) daemon, in software. The proposed system architecture is illustrated in Figure 5.8.

[4]Maximum jitter values for time-triggered traffic were hard to keep under $100\mu s$, with Xenomai OS v2.6.3 running on an Intel Core i7-4770 @ 3.4GHz, 8GB RAM DDR-3, and Intel I350 NIC @ 100 Mbps.

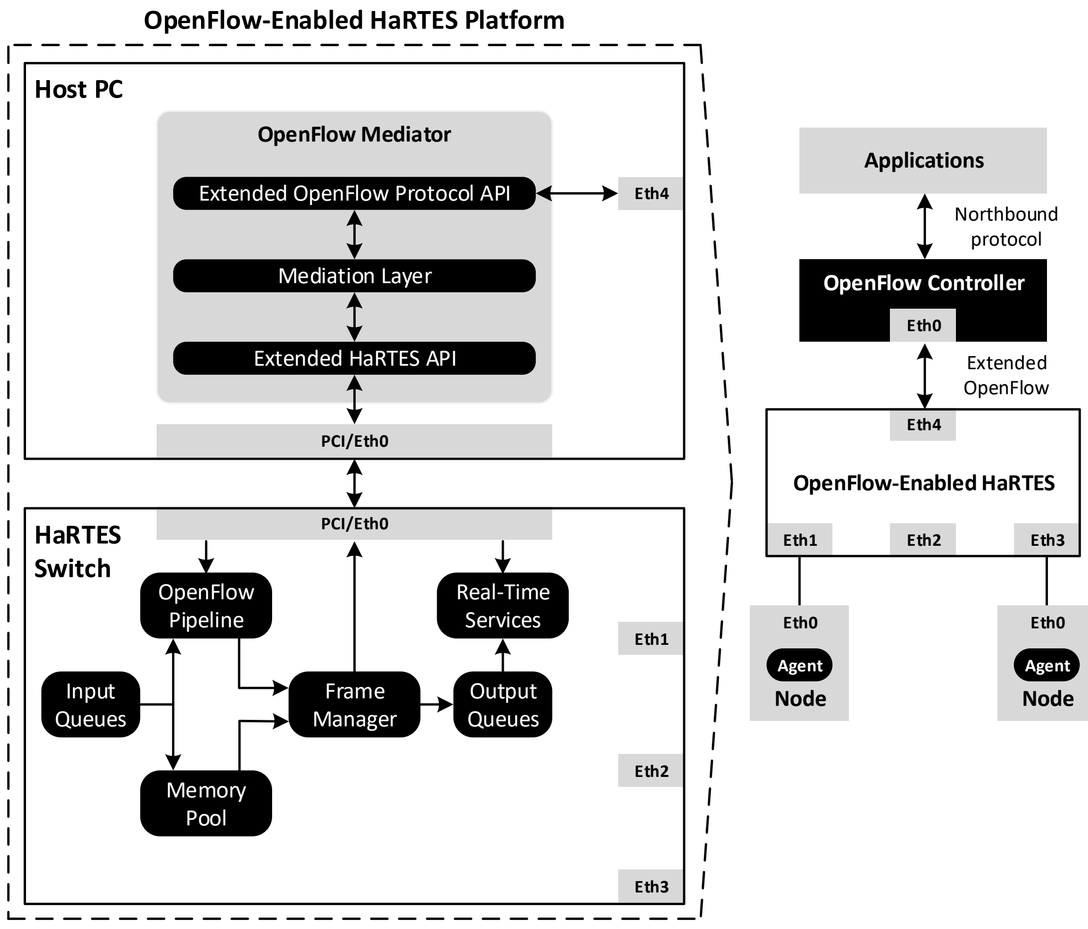


Figure 5.8: OpenFlow-enabled HaRTES platform architecture

In the enhanced HaRTES platform, traffic frames are received at HaRTES Ethernet ports and buffered by the respective input queues. Frames are retrieved from input queues in a cut-through fashion, *i.e.* while they are still being received, and stored in the central memory pool. At the same time, a copy of each frame is also sent to the OpenFlow pipeline which starts processing the frame as soon as all the required information is received, *e.g.* Ethernet and IPv4 headers. The actions, resulting from the pipeline processing, are then handled by a frame manager that populates the output queues accordingly. Output queues are managed by HaRTES's real-time services which perform the dispatching of stored frames according to the configured reservations. Non real-time traffic frames are also managed by the aforementioned services, in a best-effort way.

The OpenFlow pipeline and HaRTES's real-time services are configured by the OpenFlow mediator, which deals with the complexity of OpenFlow management functions and messages, and translates OpenFlow requests issued by controllers into suitable HaRTES commands. The OpenFlow pipeline and the OpenFlow Mediator are presented in detail in Section 5.4.1.1 and Section 5.4.1.3, respectively.

Performance wise, the implementation of the OpenFlow pipeline in hardware provides significant advantages:

- ***Faster forwarding***. Forwarding latency and jitter due to data exchange between software and hardware services is eliminated. For example, it is no longer necessary to send copies of entire received frames from the HaRTES platform to the host PC since these are now directly delivered, in hardware, to the pipeline;
- ***Increase in pipeline processing efficiency***. The true parallel computation power offered by hardware logic can be exploited in order to increase pipeline efficiency. For example, the whole pipeline, or parts of, can be cloned for each Ethernet port and executed independently. In this way, the need for queuing traffic and performing arbitration to access a single, shared pipeline, is eliminated. Additionally, parallel execution can also be exploited to match all flow table entries against frames simultaneously;
- ***Deterministic execution***. Worst-case execution times are easier to determine since hardware logic is extremely deterministic.

Unfortunately, this arrangement also exhibits some downsides, the most relevant being a higher difficulty in implementing pipelines with multiple tables and complex filters and actions. Nevertheless, the current OpenFlow protocol standard only demands a subset of all defined services, *e.g.* a single flow table with a simplified set of filters and actions. All the optional and obligatory services were identified, and can be consulted in Annex C. Therefore, focus will be placed on the implementation of mandatory components and functions. Optional ones will be gradually introduced as the prototype gets more mature. The following OpenFlow objects and services are currently implemented:

- ***Objects***: OpenFlow channels, and one ingress flow table;
- ***Ports***: Physical, logical, and the ALL, CONTROLLER, TABLE, ANY, UNSET, NORMAL, and FLOOD reserved ports;
- ***Instructions***: WRITE-ACTIONS and GOTO-TABLE;
- ***Actions***: OUTPUT and DROP.

5.4.1.1 A hardware implementation of an OpenFlow pipeline

The layout of the currently implemented OpenFlow pipeline is depicted in Figure 5.9.

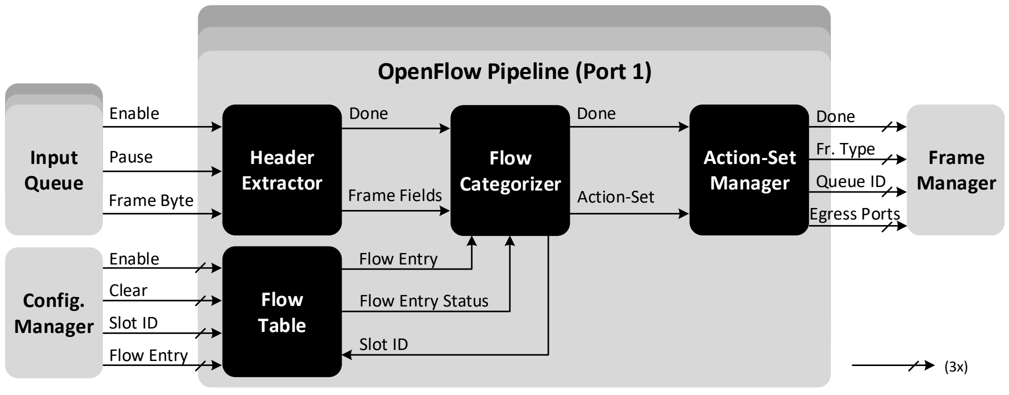


Figure 5.9: HaRTES's OpenFlow pipeline

A clone of the entire OpenFlow pipeline is assigned to each physical port in order to explore parallel execution and eliminate resource contention. Frames received at each Ethernet interface are buffered byte-by-byte by the MAC controller into the respective FIFO input queue. A Finite-State Machine (FSM), the Header Extractor (HE) unit, readily retrieves buffered bytes in a cut-through fashion, *i.e.* while the frame is still being received, and extracts the fields that are essential for flow matching. The FSM operation is paced by the Enable and Pause signals which indicate when there are valid stored bytes (Pause = '0') of an incoming frame (Enable = '1'). Once all fields are extracted, the HE unit sends a request (Done = '1') to the Flow Categorizer (FC) that then begins to match them against flow entries. Configured flow entries (Flow entry status = '1') are matched one-by-one in descending order of priority until there is a match or the table-miss flow entry, *i.e.* the last entry in the table with priority equal to 0, is reached. The ID and action-set of the matched entry is then sent to the Action-Set Manager (A-SM) unit which decodes the received information into queuing instructions, *e.g.* frame type, the egress queue ID, and target output ports, for the Frame Manager (FM) unit. Finally, the FM unit stores frames into the correct output queues. The Configuration Manager (CM) simultaneously configures and replicates flow entries on all pipelines according to commands received from the HaRTES API.

The structure of each flow entry is shown in Figure 5.10. A flow entry comprises three main segments: (i) match fields, which contain the values for the several header fields to be matched against frames, (ii) flags for the match fields, used to signal if a certain field is to be wild-carded, and (iii), the associated action-set. For the action-set, drop and output OpenFlow actions are currently implemented. Depending on the configured action, the action-set also comprises several parameters:

- ***Drop Action***: has no associated parameters;
- ***Output Action***: Port type indicates if the target port is physical, reserved, or logical. For all port types, the forwarding Ethernet ports are indicated (Egress ports field). Parameter 1 and 2 are ignored for both physical and reserved ports, and for logical ports, identify the stream type (TT, ET, NRT) and stream ID, respectively.

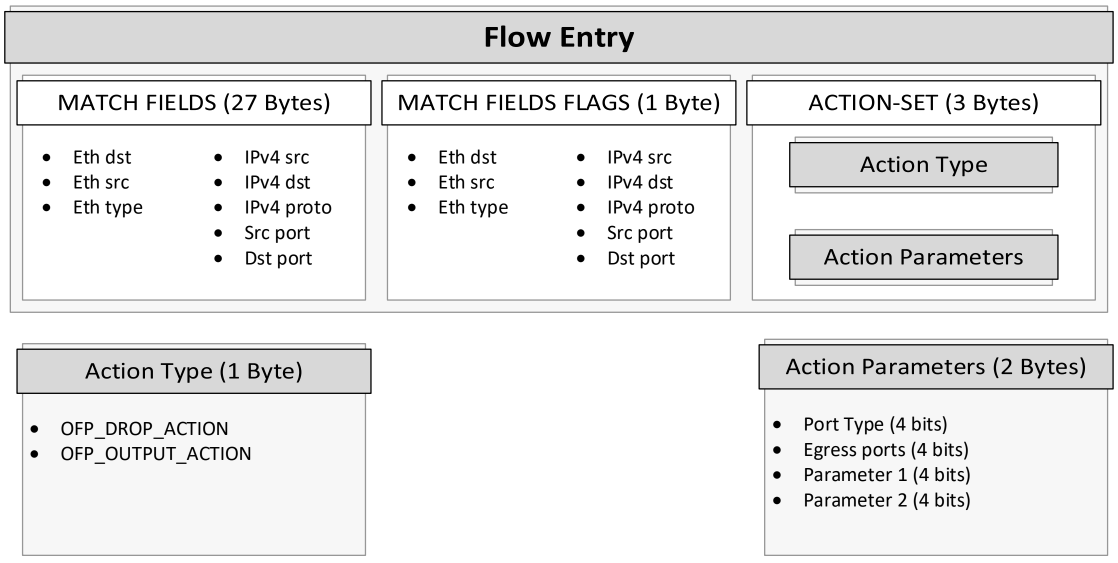


Figure 5.10: Structure of an HaRTES's flow entry

Flow entries are stored in a Flow Table which, due to the size[5] of each flow entry, is implemented using FPGA's internal memory blocks, *i.e.* dual port block RAM (BRAM). Additional tables can be deployed at the expense of more resources and a slight modification of the Flow Categorizer FSM. Table 5.3 shows the necessary logic resources[6] for a single pipeline comprising one flow table and 32 entries.

Table 5.3: Pipeline logic resources utilization

Unit	Resources								
	FSM	Counter		32x244 BRAM	Comparator				D-Type Flip-Flop
		16-bit	8-bit		48-bit	32-bit	16-bit	8-bit	
Head Extractor	1	1	2	0	0	0	0	0	218
Flow Table	0	0	0	1	0	0	0	0	277
Flow Categorizer	1	0	1	0	2	2	3	1	31
Action-Set Manager	0	0	0	0	0	0	0	0	23
Total	2	1	3	1	2	2	3	1	549

Concerning performance, the current OpenFlow pipeline processes a received frame within a time interval P_{RT} that is mainly dominated by (i) the pipeline clock speed f_{clk}, (ii) the maximum number of supported flow entries N_{FE} and (iii) the time to receive all the supported flow matching fields t_{HE}. A bound for P_{RT} can be computed following Eq. 5.19, where the rightmost term accounts for the number of clock cycles (4) related to output logic between modules (HE, FC, and A-SM) and the decoding of operations by the A-SM module. With a f_{clk} of 62.5MHz, the pipeline of the current HaRTES prototype is able to process frames at

[5]The length of flow entries could be optimized by using hashes instead of raw values for the match fields.
[6]After synthesis and before place and route in Xilinx ISE Design 10.1.

wire speed.

$$P_{RT} = t_{HE} \ + \ N_{FE} * \frac{1}{f_{clk}} \ + \ 4 * \frac{1}{f_{clk}} \tag{5.19}$$

5.4.1.2 The extension of HaRTES API

In order to be possible to configure flow entries, an extension of the HaRTES API was necessary. Table 5.4 provides a summary of the operations currently supported by the extended HaRTES API. The same approach can be used to provide additional operations as more OpenFlow objects are implemented.

Table 5.4: Extended HaRTES API

Category	Operation	Description
OP_CAT_SYNC	OP_REG	Register a synchronous stream.
	OP_DEREG	Deregister a synchronous stream.
	OP_MODIFY	Modify properties of a synchronous stream.
OP_CAT_ASYNC	OP_REG	Register an asynchronous stream.
	OP_DEREG	Deregister an asynchronous stream.
	OP_MODIFY	Modify properties of an asynchronous stream.
OP_CAT_SERVER	OP_SET	Enable and configure a server.
	OP_UNSET	Disable a server.
OP_CAT_OFP	OP_REG	Register a flow entry.
	OP_DEREG	Deregister a flow entry.
	OP_MODIFY	Modify properties of a flow entry.

The main structure for messages generated by the HaRTES API is kept unchanged. For the registration of flow entries and the modification of their properties, the entire data set of a flow entry is included as payload ("op_parameters"). Messages for the removal of a flow entry only carry the ID of the target entry. Figure 5.11 shows the structure of the message generated by the extended API upon the registration of a new flow entry.

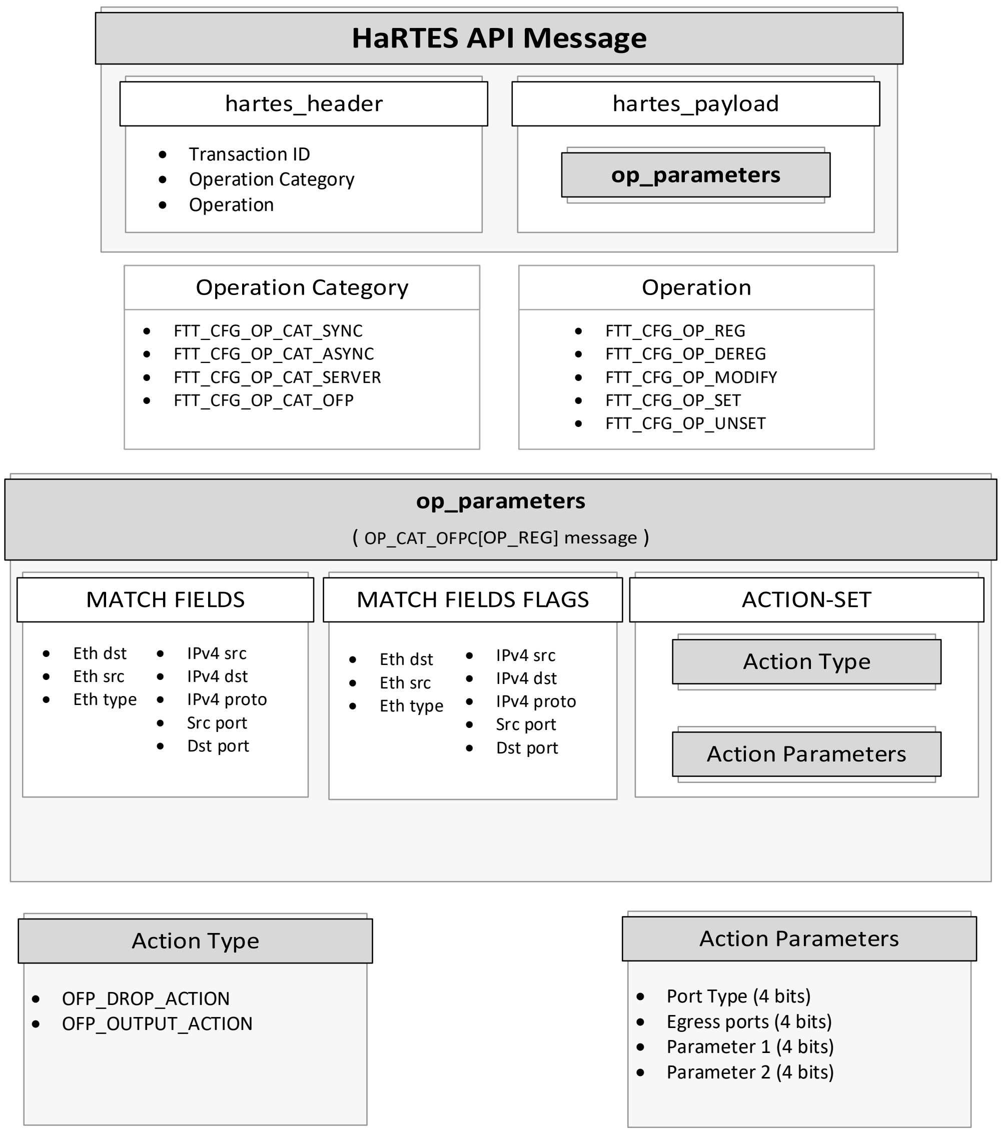


Figure 5.11: Extended HaRTES API message example

5.4.1.3 A software implementation of an OpenFlow mediator

The OpenFlow Mediator deals with the complex management aspects of the OpenFlow switch protocol and is responsible for the translation between OpenFlow and HaRTES domains. Specifically, the mediator is responsible for: (i) the establishment and maintenance of all dedicated OpenFlow channels for communication with controllers, (ii) the interpretation of OpenFlow requests and respective messages, (iii) the translation of OpenFlow requests into suitable calls of HaRTES API for the proper configuration of the OpenFlow pipeline and real-time services, and (iv), keeping databases with information regarding the system state and capabilities, *e.g.* current configurations and supported real-time logical ports. Figure 5.12 shows the logical components of the proposed mediator.

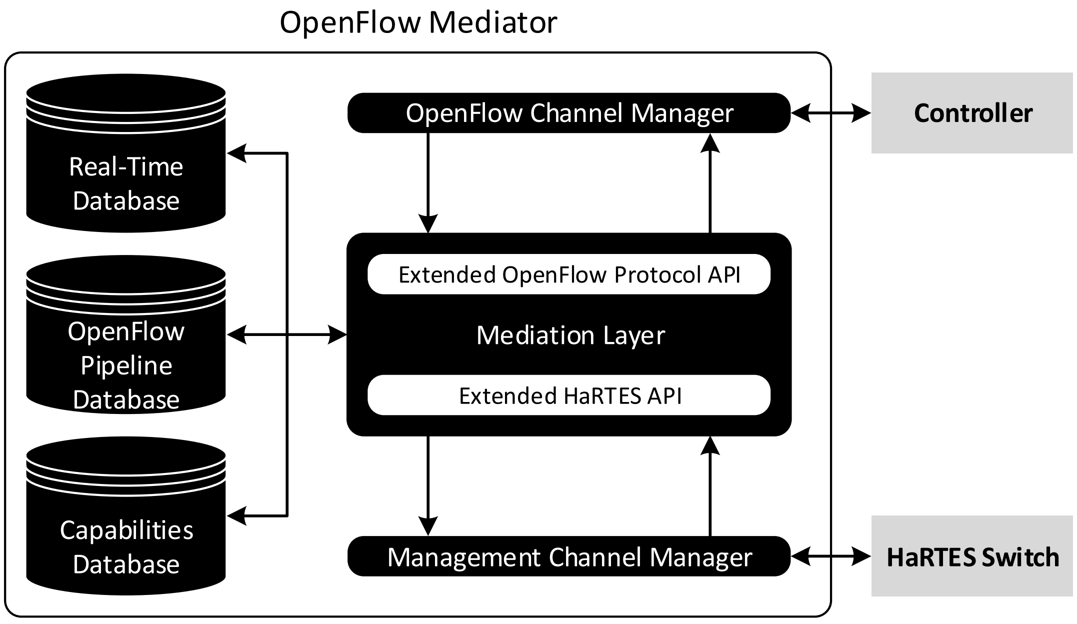


Figure 5.12: Logic architecture of the OpenFlow Mediator

Channels to OpenFlow controllers are established and managed by the OpenFlow Channel Manager unit which is able to execute all management functions defined by the standard. It forwards OpenFlow requests from controllers to the Mediation Layer unit and transmits replies or packet-in operations issued by that layer to the corresponding controller. The Mediation Layer implements the extended OpenFlow API (Section 5.3.1), decodes and interprets requests, and consults internal databases to validate requested operations and fill-in replies. Three databases are stipulated: (i) Real-Time Database, that contains all installed real-time streams and their properties, (ii) OpenFlow Pipeline Database, which keeps, for example, a synchronized image of the installed configurations at the device's pipeline, and (iii) Capabilities Database, that stores information regarding the capabilities of the existing pipeline, *e.g.* number of flow tables/entries and existing real-time logical ports. If an OpenFlow request is deemed valid, the Mediation Layer uses the extended HaRTES API (Section 5.4.1.2) to execute the necessary configurations. Finally, the Management Channel Controller mediates the transaction of requests and replies between the aforementioned layer and the HaRTES switch.

The current prototype for the presented OpenFlow Mediator daemon is based on the open-source code of the popular CPqD OpenFlow 1.3 Software Switch (ofsoftswitch13) [125], stripped of switching functionalities and enhanced with the Mediation Layer, Management Channel Controller, and databases. The standard OpenFlow API of ofsoftswitch13 is enhanced with the proposed real-time extensions.

CHAPTER 6

Validation of the real-time SDN framework

Contents

*I*n Chapter 5, the development of a cohesive real-time SDN framework was thoroughly discussed. This chapter presents the experiments that validate the correct operation of the framework, as well as a quantitative analysis of its capabilities.

6.1 The validation of the proposed framework

As discussed in Section 2.3, networks for modern industrial systems must be able to provide strict timeliness guarantees while being able to modify the properties of existing reservations and execute the admission of new communication streams or the removal of existing ones. All of this must be possible while the network is online and without causing disruption or interference to on-going communications.

The herein proposed framework is specifically designed to meet the set of requirements highlighted in Section 2.3 by exploiting the flexibility on network management provided by SDN technologies, and combining it with a capable real-time switching platform. This section presents a set of experiments whose goal is to validate the correct operation of the developed framework and provide a quantitative analysis of its capabilities, with emphasis on its real-time and reconfiguration potential. Four main aspects are evaluated: (i) the ability to support the coexistence of several types of traffic with distinct bandwidth and real-time requirements, (ii) the possibility to enact online changes to network reservations, (iii) the correctness of the implemented admission control algorithms, and (iv) the responsiveness to reconfiguration requests and scalability of the admission control unit.

Aspects (i) and (ii) are analyzed in Section 6.1.1. To that end, the presented experimental scenarios embrace the coexistence of all types of traffic (time-triggered, event-triggered, and non real-time traffic), perform online alterations on existing reservations, and incorporate misbehaving nodes and traffic not conforming to the negotiated reservations. Experiments specifically tailored to verify the correct operation of the admission control and its schedulability analyses are presented in Section 6.1.2. In these, large sets of traffic streams with real-time requirements are generated, and their response times measured and/or simulated. The obtained results are then compared with the worst-case response times from the output of the admission algorithms. Section 6.1.3 presents the experiments whose objective is to evaluate the responsiveness of the admission control unit to (re)configuration requests, as well as how its algorithms scale with network complexity. In particular, the response time of a request for the addition of a new real-time flow is observed experimentally for a set of networks with a variable number of switches and pre-installed flows. Finally, in Section 6.1.4, the whole framework is evaluated under a realistic Industry 4.0 use-case: *adaptable production.*

6.1.1 Evaluating the data plane real-time capabilities

Figure 6.1 illustrates the setup used to conduct the experiments for the real-time performance evaluation. It comprises four nodes and one OpenFlow-enabled HaRTES switch.

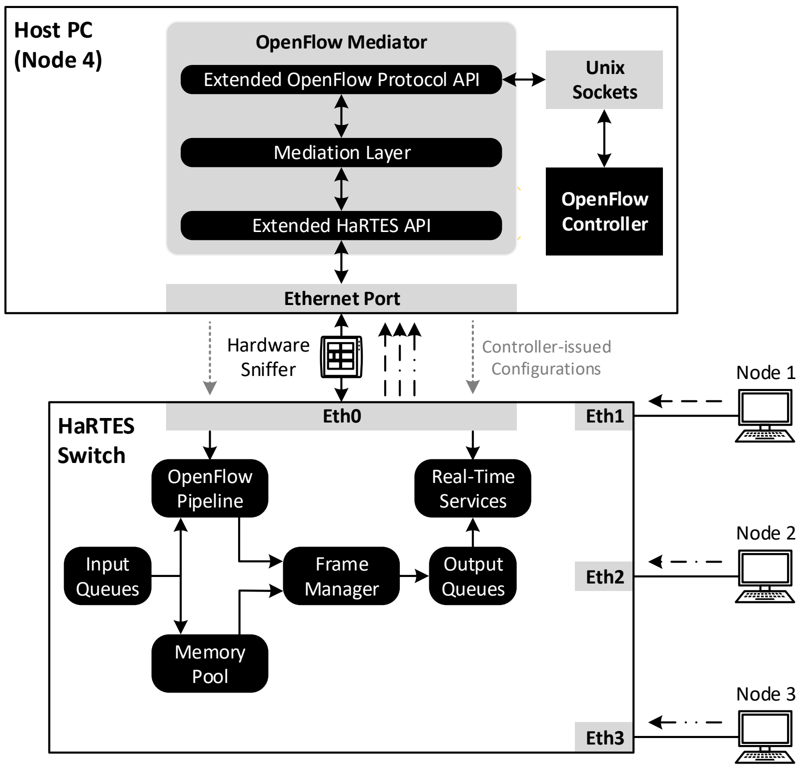


Figure 6.1: Experimental Setup

Nodes 1, 2, and 3 generate traffic flows in a mix of time-triggered, event-triggered and non-real-time traffic. Such traffic classes are representative of the diversity of requirements found in Industry 4.0 scenarios. To assess the precision of the real-time enforcement, traffic generators that provide precise patterns are used. Therefore, nodes 1 to 3 are implemented on FPGAs, providing high accuracy traffic with low jitter, when needed. All flows have node 4 as the sink node. Node 4 also executes the OpenFlow Mediator daemon, which interacts with the controller and sends configurations and reservations into the switching platform. Since the PCI controller for the HaRTES switch is currently not functional, an Ethernet port is used to connect HaRTES to the Host PC. For this reason, and due to the limited number of available ports in the switching platform, the OpenFlow controller is also instantiated in Node 4. The communication interface between controller and the mediator daemon is based on standard Unix sockets. A FPGA-based sniffer captures all traffic going to the sink node and generates time-stamps with a resolution of 1us.

Two types of experiments were performed. The first experiment (Exp1) evaluates the system capability to support time-triggered (TT) traffic, which, in common industrial applications, is extremely sensitive to latency and jitter. The second experiment (Exp2) appraises the system ability to dynamically configure new flows and reservations, while concurrently supporting distinct traffic classes with guaranteed QoS. Table 6.1 compiles the properties for each traffic flow generated in both Exp1 and Exp2 experiments.

Table 6.1: Properties of the traffic generated in the experiment scenarios

Experiment	Node	Flow Type	Payload (Bytes)	Period (μs)
Exp1	1	Time-triggered	50	1000
	2	Time-triggered	50	1000
	3	Non real-time	1500	130
Exp2	1	Time-triggered	50	1000
	2	Event-triggered	1000	130
	3	Non real-time	1500	130

Note: The payload conveys only the frame data, no headers and other overheads from the OSI layer 2 are included

In Exp1, the network is overloaded with standard non real-time (NRT) OpenFlow traffic to assess the amount of interference induced on time-triggered (TT) communications that may arise from the presence of high network loads. The NRT flow intentionally saturates the ingress link of node 4 to cause as much interference as possible. In Exp2, both time-triggered and event-triggered (ET) real-time traffic, as well as standard OpenFlow traffic (NRT), are deployed concurrently. In this experiment, NRT traffic keeps the system overloaded while the reservations for the ET traffic are dynamically reconfigured into three distinct modes, that correspond to different flow rates and, consequently, reserved bandwidth. In both experiments, the enhanced HaRTES platform implements the data plane functionality, and links are configured to 100 Mbit/s. The FTT Elementary Cycle (EC) length is set to 1 ms, with the windows configured as shown in Figure 6.2.

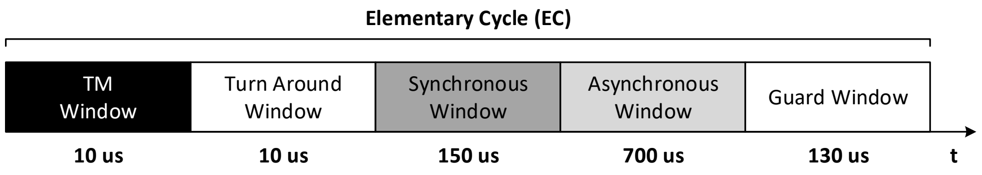


Figure 6.2: Elementary cycle parametrization

As reminder, the Elementary Cycle starts with the transmission of the Trigger Message (TM) that synchronizes HaRTES nodes and conveys the schedule of the time-triggered traffic for that cycle. It follows the turn-around window, which allows nodes to interpret the TM and prepare the scheduled time-triggered transmissions. Then, it follows the Synchronous Window, where scheduled time-triggered transmissions are carried out. Event-triggered and non real-time transmissions occur within the asynchronous window, in which event-triggered traffic is always prioritized over the non real-time traffic. Finally, the guard window inserts idle time to prevent EC overruns caused by event-triggered/non real-time frames. Additional details on the inner workings of HaRTES are available in Section 4.3.2.

A simple application in the controller manages all OpenFlow and real-time reservations. It uses both the standard and extended OpenFlow APIs to install reservations for each traffic flow. Table 6.2 lists the properties of the installed reservations for both experiments.

Table 6.2: Properties of the configured reservations

Traffic stream	Reservation	Properties	
Time-triggered	Time-triggered slot	1 slot per EC	
Non real-time	Best effort	- - -	
Event-triggered	Deferrable server	Budget (Bytes)	1018
		Period (No. ECs)	4 (Mode 1)
			2 (Mode 2)
			1 (Mode 3)

The presented reservations were set up online and after system boot, *i.e.* after all devices have been powered-up and without pre-installed reservations. The process to configure reservations follows two consecutive phases: (i) the controller accesses the standard OpenFlow API to configure the necessary flow entries so as to identify each flow and output it to the respective logical port, (ii) the controller accesses the real-time add-on API to register the real-time flows and their properties, indirectly instantiating the adequate reservations. The controller also accesses the real-time API to modify the properties of existing reservations. Table 6.3 and Table 6.4 present the requests sent through the standard OpenFlow and the real-time APIs, respectively.

Table 6.3: Requests to the OpenFlow standard API

Standard OpenFlow API					
Flow	Operation	Match Fields			Action
		Eth Source	Eth Type	IP Source	
TT (1)	Add Flow Entry	Node 1 MAC	TT EtherType	* * *	Output LOGICAL TT_1
TT (2)		Node 2 MAC	TT EtherType	* * *	Output LOGICAL TT_2
ET		Node 2 MAC	IP Protocol	Node 2 IP	Output LOGICAL ET_1
NRT		Node 3 MAC	* * *	* * *	Output PHYSICAL 0

* * * (Don't care)

Table 6.4: Requests to the OpenFlow add-on real-time API

Extended OpenFlow Real-Time API						
Flow	Operation	Stream Type	UID	C (Bytes)	T (No. ECs)	Consumers
TT (1)	Reg. stream	Time-triggered	TT_1	68	1	Eth0
TT (2)		Time-triggered	TT_2	68	1	Eth0
ET		Event-triggered	ET_1	1018	4 (Mode 1)	Eth0
	Mod. stream	Event-triggered	ET_1	1018	2 (Mode 2)	Eth0
		Event-triggered	ET_1	1018	1 (Mode 1)	Eth0
NRT	- - -	- - -	- - -	- - -	- - -	- - -

The packets inter-arrival times and associated jitter for each traffic class are chosen as metrics to evaluate the system performance. Figure 6.3 and Figure 6.4 show the inter-arrival timings, for received packets of each traffic class, on Exp1 and Exp2 respectively. Table 6.5 presents a statistical breakdown of these values.

Table 6.5: Summary of observed timing figures

Exp.	Flow		Inter-arrival (μs)			Jitter (μs)	
			Min	Max	Mean	Relative	Absolute
Exp1	TT (1)		983	1002	992	18	19
	TT (2)		983	1007	992	22	24
	NRT		123	385	165	262	262
Exp2	TT		985	999	992	14	14
	ET	(Mode 1)	3954	3983	3968	29	29
		(Mode 2)	1951	1991	1984	33	40
		(Mode 3)	962	999	992	31	37
	NRT		123	384	160	261	261

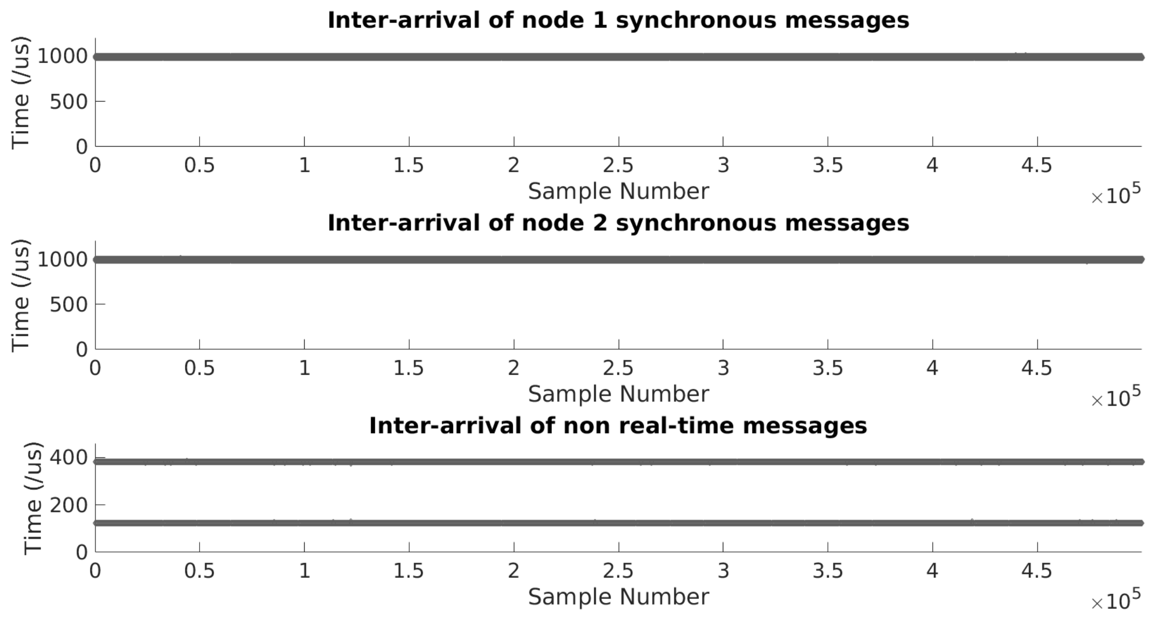


Figure 6.3: Inter-arrival timings for traffic in experiment 1

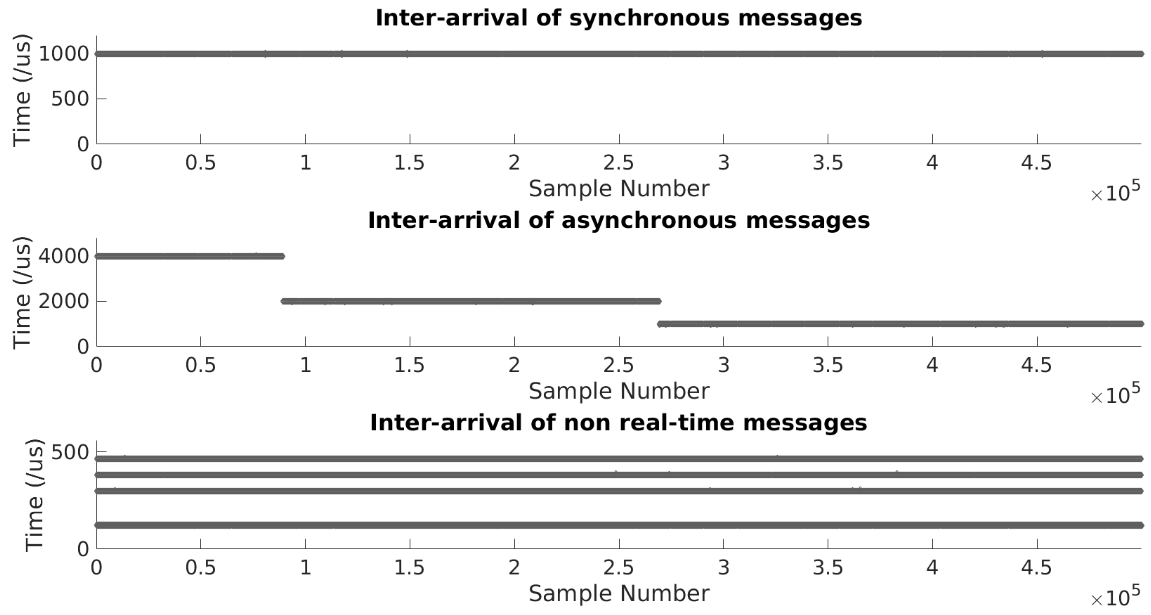


Figure 6.4: Inter-arrival timings for traffic in experiment 2

In both experiments, the time-triggered traffic is forwarded with very low jitter despite the flooding of non real-time traffic. This shows that the temporal isolation provided by HaRTES's real-time dispatcher is effective and completely eliminates interference between time-triggered and other types of traffic. The jitter for time-triggered traffic is slightly higher in Exp1 since packets from both time-triggered flows contend with each other and no particular ordering is enforced within the synchronous window. This interference corresponds to the transmission time of a time-triggered frame which accounts for approximately 7 μs.

Exp2 shows that the effective bandwidth used by the event-triggered traffic is constrained by the associated reservation, which can be dynamically modified without service disruption. There are three different modes, in which the server assigned to the event-triggered traffic is configured with replenishment periods of 4 ms, 2 ms, and 1ms (Table 6.2). As the budget capacity of the server allows only one frame transmission, the inter-arrival values correspond to the server period, as expected. Due to the absence of contention, the jitter is very low (up to 40us). This was also expected since event-triggered queues are kept full and HaRTES prioritizes this type of traffic over non real-time traffic. If the event-triggered queues were not kept full, one could expect to see interference arising from the blocking of one non real-time message, which would account for a maximum of 123 μs.

Finally, in both experiments the non real-time traffic is dispatched in a best effort way, occupying the remaining bandwidth. Its minimum inter-arrival time corresponds to the transmission time of one non real-time frame, approximately 123 μs. The maximum value observed can be explained considering that each EC conveys one event-triggered frame and six non real-time messages, the last of which overruns 121 μs of the guarding window. The non real-time frame that follows is blocked until after the transmission of the event-triggered frame in the next asynchronous window, yielding the observed 385 μs.

6.1.2 Evaluating the control plane schedulability analysis

An important service at the control plane is the admission control and its algorithms. If the employed algorithms and underlying analysis would not be flawless the admission control could make decisions based on misleading information, placing the timeliness performance of the whole network in jeopardy. Therefore, experiments based on simulations[1] and on a laboratory prototype were performed to assess the validity of the outputs resulting from the schedulability analysis.

In this experiment, all switches' links operate at the rate of 100 Mbps and the hardware fabric latency ϵ is considered to be equal to 3 μs[2]. Although the Reduced Buffering Scheme (RBS) method supports different window sizes for each link, for the sake of simplicity, the synchronous window in all links in the network are equal. The configured EC is depicted in Figure 6.5.

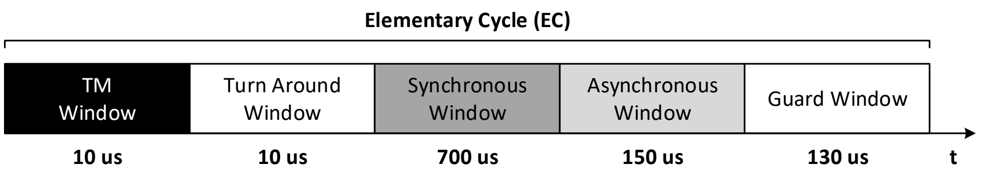


Figure 6.5: Elementary cycle parametrization

[1]Setup and simulation results reused from [4]
[2]According to extensive measurements performed in [29]

As setup, a multi-hop network comprising three HaRTES prototype switches and three nodes for the generation of time-triggered traffic was deployed. Figure 6.6 shows the implemented experimental setup. A set of thirty time-triggered streams was generated, mixing different pairs of source-sink nodes and with message periods uniformly generated within the interval $[5, 25]$ ECs. Each stream produces a single frame with a transmission time of $C_i = 123\mu s$. The priority of each stream is set according to the Rate Monotonic policy. Table 6.6 shows the parameters for each generated stream, including the period T_i and priority P_i.

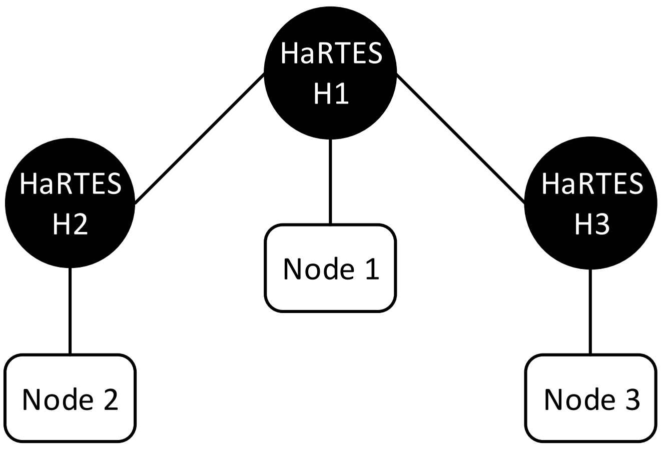


Figure 6.6: Time-triggered scheduling: experimental setup

Table 6.6: Parameters of time-triggered traffic for the prototype experiment

ID	T_i	P_i	Source	Sink	ID	T_i	P_i	Source	Sink
m1	20	6	3	1	m16	18	5	3	2
m2	20	6	1	3	m17	15	4	2	3
m3	25	7	3	1	m18	15	4	3	2
m4	15	4	3	2	m19	20	6	3	2
m5	10	2	3	2	m20	10	2	2	3
m6	15	4	3	2	m21	18	5	3	2
m7	10	2	2	1	m22	25	7	3	2
m8	20	6	3	2	m23	18	5	3	2
m9	12	3	2	3	m24	5	1	3	1
m10	5	1	3	1	m25	15	4	2	3
m11	15	4	2	1	m26	15	4	2	3
m12	10	2	3	1	m27	18	5	1	2
m13	15	4	3	2	m28	10	2	1	2
m14	18	5	3	2	m29	10	2	2	1
m15	25	7	3	1	m30	10	2	3	2

The response time of each stream was measured over the course of 60,000[3] ECs, and the minimum ($minRT$), average ($avgRT$), and maximum ($maxRT$) response time values determined. Note that all values are rounded up to be comparable to those from the analysis outputs. Therefore, it may happen that some of the values appear to be "equal". The obtained results are illustrated in Figure 6.7. The output values ($calcRT$) from the analysis algorithms that are currently implemented in the framework are also shown.

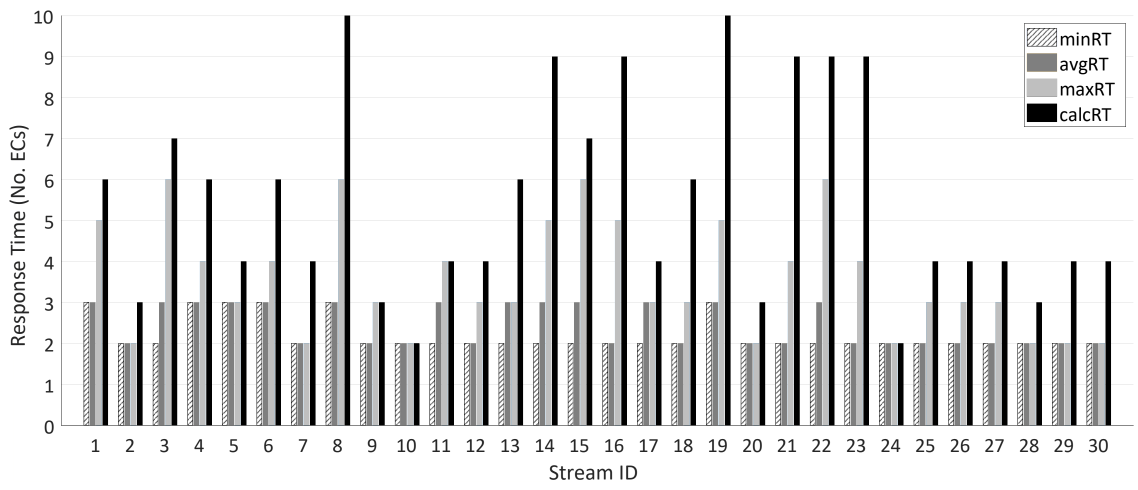


Figure 6.7: Time-triggered scheduling: experimental results

The obtained results show that all the measured response time values are either equal or lower than those predicted by the analysis algorithms. Moreover, it is also noticeable a degree of pessimism in the computed response time values, in particular, for messages with lower priority levels. For example, while the difference between the computed and the measured values for streams with priority 1 and 2 is at most 1 EC, values for streams with priority levels of 6 and 7 differ from 2 up to 5 ECs. The sources of this pessimism are discussed in [4], and can be summarized as: (i) the existence of phasing between streams, which is not taken into consideration by the analysis, and (ii), pessimism induced by the idle calculation which always considers the worst-case scenario, *i.e.* the smallest synchronous window and the largest frame size.

6.1.3 Evaluating the admission control responsiveness and scalability

An important aspect of the control plane is its ability to respond to new (re)configuration requests within time intervals that are compatible with a given application. For example, the desired time for the reconfiguration of a small industrial production system falls within a seconds range [46]. Therefore, this experiment aims to evaluate the responsiveness of the implemented admission control unit under diverse levels of network complexity. To that end, the time required to perform the admission control analysis (T_{sched}) and to configure all

[3]Significantly more than the size of the hyperperiod

devices (T_{conf}) upon a request for a single new flow was measured for networks with different sizes and number of existing flows.

The deployed setup, depicted in Figure 6.8, consists on a network of multiple HaRTES switches (N_H =1, 2, 4, 8, 16) connected to a RT SDN controller. To assess the load effect on the schedulability analysis algorithm, a set of random flows (N_S =0, 10, 50, or 100 flows) is pre-installed in the system. To capture the worst-case scenario where all devices must be properly configured, all switches are interconnected following a line topology and the new flow traverses the entire network. Additionally, as the analysis stops computing as soon as a deadline miss for a given flow is detected, long enough deadlines for all flows were considered. This ensures that the algorithm always computes the response time for all flows without stopping in the middle and thus, the worst-case execution time for the analysis is captured.

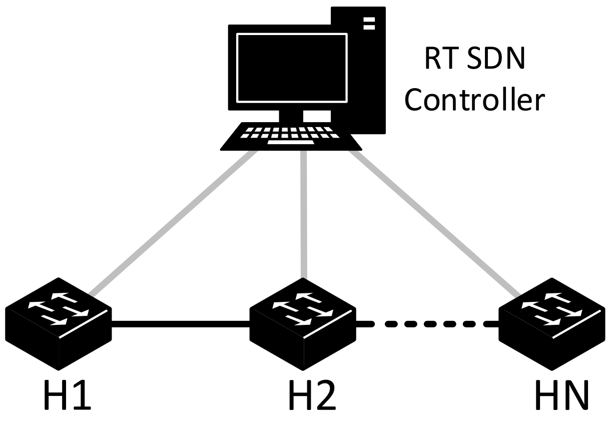


Figure 6.8: Network setup for the experiment

Figure 6.9 shows the observed execution times, with the shades/intervals representing the measured minimum-maximum range. Each data point was obtained by repeating the associated conditions 1000 times.

As expected, T_{sched} grows significantly with the number of flows since the algorithm complexity is $O(N_S * (N_H + 1))$ [4]. In contrast, T_{conf} is independent of the number of flows and directly proportional to the number of switches that must be configured. Finally, even for a reasonably sized network comprising 16 switches and 100 flows, the framework is able to respond to reconfiguration requests in less than $170ms$, which is significantly less than the reported reconfiguration times for small industrial production systems, *i.e.* seconds range [46].

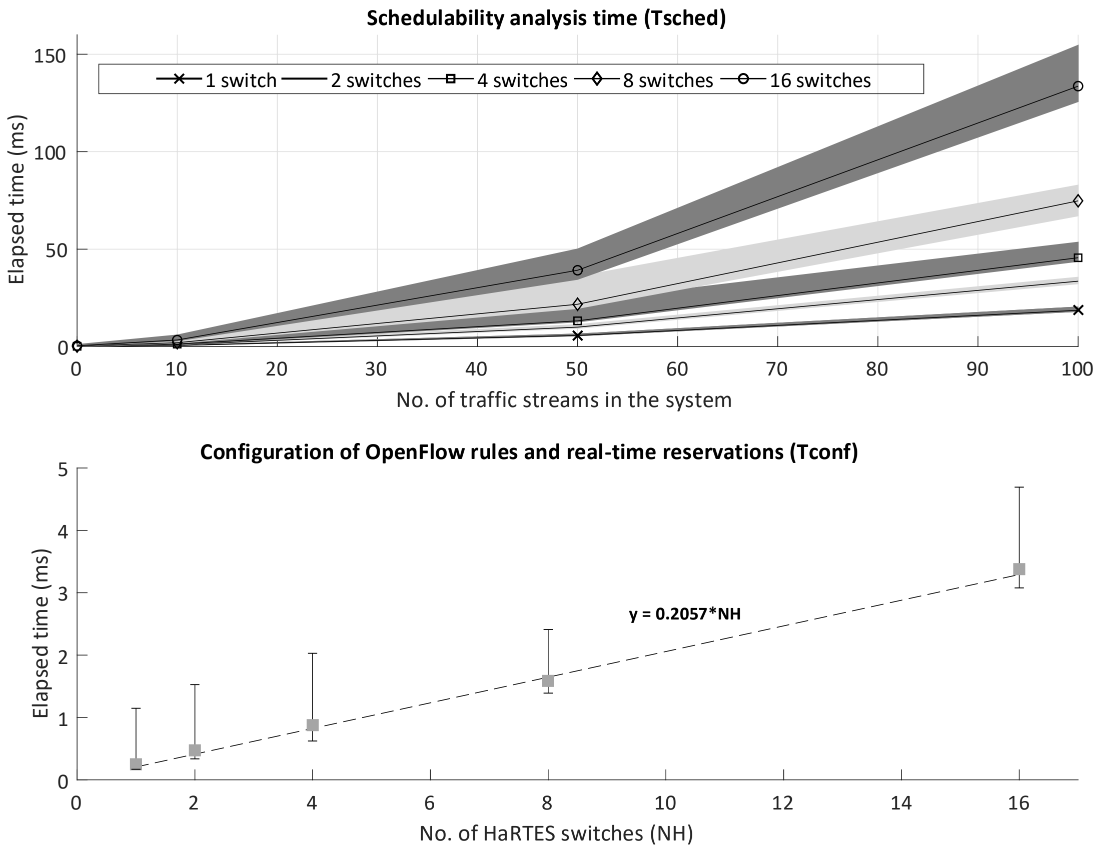


Figure 6.9: Time-triggered scheduling: experimental results

6.1.4 Evaluating the framework under a realistic Industry 4.0 scenario

This experiment aims to evaluate the traffic timeliness capabilities of the framework under a representative Industry 4.0 application scenario. In particular, it emulates the network of a smart distributed robotic cell for adaptable production. The setup is adapted from [126] considering the limited number of Ethernet ports (4) of HaRTES's hardware platforms. Moreover, the original period of flows is also significantly reduced to increase network load.

The setup, shown in Figure 6.10 contains a RT SDN controller connected to three HaRTES switches (H1, H2, H3) via dedicated links plus five application nodes (N1-N5). All links operate at 100 Mbit/s and the EC is configured as depicted in Figure 6.11. The ECs of the three switches are synchronized via an explicit signal applied to specific digital ports.

The coordinator (N3) controls robots 1 and 2 (N1 and N4, respectively) by transmitting a set of periodic commands according to data received from each robot and from a cluster of sensors (nodes N2 and N5). Table 6.7 shows the parameters of the flows used in the experiment, where T is the period or minimum inter-arrival time, PL the payload, and O the initial offset (TT traffic only).

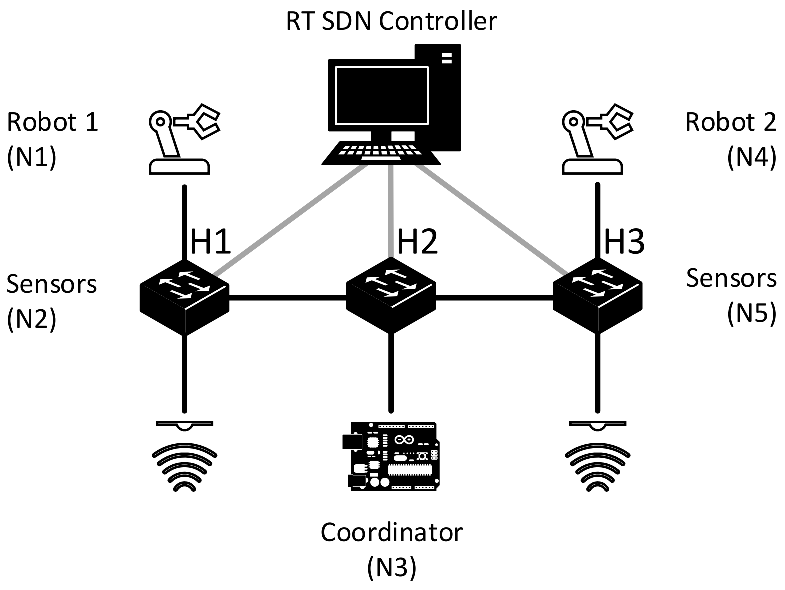


Figure 6.10: Experimental setup of the smart distributed robotic cell

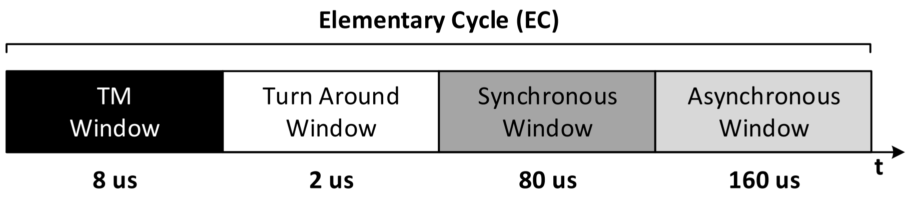


Figure 6.11: Configured elementary cycle structure

Flows 1 to 8 convey control data (TT traffic) while flows 9 to 18 carry monitoring data and alarm events (ET traffic). Finally, flows 19 to 24 represent non-real-time traffic, particularly statistics and production logs. All real-time flows are scheduled following the Rate-Monotonic scheduling algorithm, *i.e.* flows with lower periods are assigned to higher priority levels.

All nodes are implemented in FPGAs for high precision traffic generation. A hardware sniffer, namely a Hilscher netANALYZER NANL-C500-RE, captures packets in multiple links and timestamps them with nanosecond resolution. In particular, it measures the time between the first bit of a frame in the source node's link and the corresponding first bit in the sink node's link. We add the frame transmission time to captured times in order to obtain the total response time. Possible delays in nodes' interfaces is not accounted.

The application exhibits two distinct operational modes: *mode A* and *mode B*. In *mode A*, only robot 1 is operating, while in *mode B* both robot 1 and 2 operate concurrently. Therefore, during *mode A* only nodes N1, N2, and N3 communicate, while in *mode B* all

Table 6.7: Setup communication parameters (adapted from [126])

Flow ID	Source	Sink	Type	T (μs)	PL (Bytes)	O (ECs)
1-3	N3	N1	TT	500	80	0
4-6	N3	N4	TT	500	80	1
7	N2	N3	TT	250	160	0
8	N5	N3	TT	250	160	0
9,10	N1	N3	ET	1000	390	—
11,12	N4	N3	ET	1000	390	—
13	N2	N3	ET	1000	390	—
14	N5	N3	ET	1000	390	—
15,16	N2	N1	ET	250	46	—
17,18	N5	N4	ET	250	46	—
19	N1	N3	NRT	1000	750	—
20	N4	N3	NRT	1000	750	—
21	N2	N1	NRT	1000	750	—
22	N5	N4	NRT	1000	750	—
23	N3	N1	NRT	500	750	—
24	N3	N4	NRT	500	750	—

nodes communicate. The change between modes occurs online and without service disruption. Flow reservations are (re)configured upon a mode change.

Table 6.8 presents the observed message response times and jitter between consecutive frames of the same flow, as well as the worst-case response time (R) computed by the SDN controller.

Table 6.8: Real-time traffic performance

Flow ID	Response time (μs)			Jitter (μs)	R^1
	Min.	Mean	Max.		
		Mode A			
1-3	31.1-31.2	31.2-31.3	31.4-32.5	0.9-1.0	1
7	50.3	50.5	50.6	0.9	1
9-10	105.6-106.4	197.5-219.9	305.1-338.9	269.5-338.0	3
13	105.7	166.1	409.4	476.9	3
15-16	14.4-14.5	49.8-52.9	149.8-149.9	179.1-179.9	1
		Mode B			
1-6	32.1-32.2	32.2-32.3	32.4-32.5	0.8-1.0	2
7	50.3	50.5	50.7	0.9	1
8	65.0	65.9	66.6	1.0	1
9-10	105.7-106.4	202.6-232.9	340.6-390.6	258.2-322.7	4
11	105.7	258.2	475.1	496.0	4
12	104.6	341.7	461.1	393.4	4
13-14	103.6-105.6	209.9-211.0	621.0-654.3	690.3-821.6	4
15-18	14.3-14.5	51.1-54.2	189.3-199.3	306.0-329.1	1

500 000 samples per flow. EC is 250μs long.

[1] Estimated worst-case response time (No. of ECs).

The results show that, in both modes, control (TT) traffic is forwarded with latency values below $70\mu s$ (within 1 EC) and jitter at or below $1\mu s$, fulfilling the most stringent requirements (motion control applications in Table) 2.2. This is a consequence of proper EC configuration and the tight synchronization among nodes. An interesting detail can be observed with flow 8 in *mode B*, which suffers interference from flow 7 in the last link, increasing its response time. An important remark, as HaRTES follows a synchronous approach with the resolution of ECs and without an explicit control of the order of transmissions within the Synchronous Window, the TT message set must be defined with care in order to achieve very low jitter. This can be achieved, for example, by adjusting the periods and/or offsets of TT flows, a common practice employed in the construction of TT schedules. Regarding ET traffic, the results show that all frames are delivered within their minimum inter-arrival time, *i.e.* implicit deadlines. The observed jitter is relatively high for this type of traffic due to the possibility of frames being randomly blocked by ongoing transmission in switches' egress links. Finally, the measured maximum response times are within the bounds produced by the controller analysis.

Concerning mode reconfigurations, the mean and maximum time to change from *mode A* to *mode B*, *i.e.* adding robot 2 reservations, is $7.78ms$ and $9.37ms$, respectively, while from *mode B* back to *mode A*, *i.e.* deleting robot 2 reservations, is $2.98ms$ and $3.73ms$. These values were obtained by sampling 1000 mode change cycles.

6.2 On the fulfillment of Industry 4.0 network requirements

The experimental results reported in the previous sections indicate that the conceptualized SDN framework can fulfill all the Industry 4.0 network requirements identified in Sec. 2.3. With the HaRTES-based data plane, the framework is able to support the coexistence of multiple applications with distinct QoS requirements while ensuring full guaranteed timeliness (requirement R1) with relatively low pessimism (R2). HaRTES's traffic confinement and policing services ensure that communications are restricted to the negotiated reservations, maintaining timeliness guarantees even in the presence of misbehaving applications. Moreover, the flexibility of HaRTES enables the dynamic management of real-time flows by the RT SDN controller (R3).

Additionally, the enhanced SDN controller leverages the monitoring capabilities of the extended OpenFlow protocol to to keep track of existing flows, current link states, and flow statistics, providing a powerful monitoring and diagnostics framework (R4). Finally, the whole network can be set up from a single entity, *i.e.* the SDN controller, using a single management protocol, *i.e.* OpenFlow, using the proposed OpenFlow real-time extensions. The controller can also be easily extended with additional network management protocols, *e.g.* SNMP, to configure possible non-OpenFlow devices. This eases network management and allows reducing (or avoiding) the dependence on multiple vendor-specific management tools (R5).

CHAPTER 7

TSN and SDN in the context of Industry 4.0

Contents

*P*reviously, a novel SDN framework comprising real-time extensions to the OpenFlow protocol, a SDN controller with admission control capabilities for real-time traffic, and a flexible real-time data plane based on HaRTES switches, was introduced and successfully validated experimentally. Concurrently to the development of this framework, new proposals and technologies have been introduced by the scientific community that also address the flexibility, heterogeneity, and management of real-time Ethernet networks. On the one hand, TSN has been subject of continuous enhancement, with new real-time services and new features extending its (re)configurability. On the other hand, extensions and methods to use SDN in real-time networks have been subject of research (recall Section 3.3.2).

This chapter aims to discuss the most important aspects of the most relevant solutions and evaluate their suitability in the context of Industry 4.0. First, TSN as a whole and SDN-related solutions, including the herein developed framework, are evaluated and compared under a set of criteria based on the requirements identified in Section 2.3. Then, as the developed real-time

OpenFlow extensions employ generic parameters and allow distinct data plane technologies to be used, possible ways to integrate TSN devices into the framework are discussed. This chapter concludes with the highlight of the limitations that such arrangement would bring when compared to the use of a HaRTES-based data plane.

7.1 Evaluating TSN and SDN

To evaluate TSN and the different SDN-based solutions, the following criteria were used:

- **Real-time performance:** compliance with latency and jitter figures of real-time traffic;
- **Overhead:** bandwidth consumed and/or wasted by the protocols;
- **Mutual isolation:** support to heterogeneous traffic types without mutual interference;
- **Granularity of QoS control:** diversity and parametrization of allowed QoS policies;
- **Traffic management architecture:** supported logical management architectures and how it affects resource management efficiency;
- **Flexibility:** ability to create and modify reservations promptly.

For sake of conciseness, the SDN/OpenFlow extensions reviewed in Section 3.3.2.3 are labeled as: FTT-Openflow [78]; TSSDN [79]; SDPROFINET [80]; and SDN-HSF [81]. The herein proposed real-time SDN framework is labeled as OpenFlow-RT.

7.1.1 Real-time performance

TSN provides services specifically tailored for traffic with Real-Time (RT) requirements, namely CBS and Transmission Gates. The former service provides a shaping service which allows gross bandwidth reservations suitable for real-time Event Triggered (ET) streams. However, the limited number of traffic classes (up to 6, as the maximum number is 8 and two are reserved for background and management), flat server structure and hard to analyze (at least for real-time purposes) server type, constraint the response-time and jitter of this kind of traffic. Transmission Gates are specifically designed for periodic traffic, creating contention-free time-based transmission slots that suit well Time-Triggered (TT) communications, allowing low latency and jitter.

OpenFlow (OF) was not designed for real-time systems, not distinguishing RT traffic from Non Real-Time (NRT) one, and there is no explicit support to real-time activation modes (TT, ET). Time issues are only mentioned to allow the application of synchronized updates on a given set of OF switches. In addition, priorities are supported on output queues, which usually are available in a limited number, thus constraining the support of scheduling policies for realistic cases. As such, the real-time performance of OF is poor.

FTT-OpenFlow is an implementation of FTT-SE [25] on OpenFlow, which preserves the original periodic traffic management of FTT-SE while improving the handling of sporadic real-time traffic by modifying the signaling mechanism associated to these messages. It thus allows an efficient handling of both TT and ET RT traffic.

TSSDN and SDPROFINET bring support to TT traffic on SDN, allowing low figures of jitter and latency. In the former case this is achieved by synchronizing end nodes to avoid contention among TT packets. Interference between TT and other traffic is handled via priorities, eventually combined with frame preemption mechanisms. In the latter case, SDN is used to manage PROFINET switches, thus inheriting the real-time attributes of this protocol. In both cases there is good support to TT traffic, but there is no explicit handling for RT ET traffic. Moreover, TSSDN relies solely on controlling the transmission instants at end nodes, without depending on bridge-level scheduling services. As such, the schedulability level for TT traffic is reduced, as the protocol does not allow any kind of overlapping in flow paths.

In turn, SDN-HSF does not explicitly addresses RT traffic, but the enhancements on queuing disciplines brings significant improvements on traffic isolation and bandwidth control, with a positive impact on ET RT traffic.

Finally, Openflow-RT supports explicitly both ET and TT traffic by means of dynamic explicit scheduling in the former case, and hierarchical servers in the second one. This allows low latency and jitter for both kinds of traffic.

7.1.2 Overhead

Attaining low jitter in communication systems supporting event-triggered traffic is complex and, usually, impacts negatively on bandwidth utilization efficiency. This is particularly noticeable when there is joint support for TT traffic, as this kind of traffic is often associated with very strict jitter requirements.

TSN introduces a frame preemption mechanism that allows to interrupt the transmission of less important and/or jitter tolerant messages (classified as preemptable) in favor of other messages more jitter sensitive (classified as express). There is a minimum size before preemption can occur, which implies wasted bandwidth in each transmission slot for the case TT traffic is managed by Transmission Gates. Moreover, preemption also adds overhead, as control bytes must be added to the packet segments. TSSDN behaves similarly to TSN, as the use of frame preemption is allowed and it comprises the notion of slots for TT messages.

On the other hand, SDPROFINET, FTT-OpenFlow and OpenFlow RT use dedicated windows for TT traffic, blocking eventual ET transmissions that could otherwise overrun TT transmission windows. This represents network idle-time that translates to overhead. In general the impact of the inserted idle-time is moderate, as it is inserted once per window, not once per message.

Control messages are another potentially relevant source of overhead. In this regard, FTT-OpenFlow and OpenFlow RT are penalized by the need to disseminate periodically elementary cycle's transmission schedule. The impact is inversely proportional to the elementary cycle duration, starting to be relevant for cycles below $1ms$.

OF and SDN-HSF don't have relevant overheads as they don't implement the functionalities above described.

7.1.3 Mutual isolation

TSN provides a set of mechanisms that allow some degree of traffic isolation. There are distinct traffic classes that can be associated with different forwarding mechanisms, which include the selection algorithms (prioritized transmissions, shapers) and the Transmission Gates. Moreover, the availability of filtering and policing also impacts positively on isolation by allowing to bound the interference of misbehaving streams. Although this set of features is interesting from the traffic isolation point of view, the performance of TSN in this regard is impaired by the limited number of priorities, which are associated with traffic classes. As mentioned above, in practical terms only 6 classes can be used, so per-stream confinement and isolation is far from reach.

SDPROFINET is based on PROFINET and, as such, segregates traffic in NRT, RT and IRT. The IRT traffic comprises exclusive transmission slots for isochronous traffic streams. Therefore, IRT streams are completely isolated form each other and from the other classes. However, for RT traffic, the isolation is not so strong, as the mechanisms are based only on communication stack adaptations (IP partially abandoned and direct use of Layer 2/OSI services). Moreover, RT ET traffic is not explicitly supported.

TSSDN only segregates RT TT traffic from the remaining one. No other mechanisms, except references to the use of traffic prioritization, are provided. So, this protocol is quite limited in this respect.

FTT-OpenFlow and OpenFlow RT are based on the FTT paradigm. As such, they isolate the transmission of TT, ET and NRT traffic, which have exclusive transmission windows. Moreover, in both these protocols RT ET traffic is managed by hierarchical servers, which allows a fine grain stream composition (from individual streams to subsystems and systems) with bounded and predictable interference. In the particular case of OpenFlow RT, the HaRTES-based bridges implement traffic policing and have separated memory areas for the different traffic types, assuring the robustness of the isolation mechanisms.

Support of isolation on OF and SDN-HSF is poor, as there is no notion of traffic types. The simple use of priorities and improved queuing management policies are not enough to attain an acceptable level of performance in this regard.

7.1.4 Granularity of QoS control

In what concerns QoS granularity for real-time systems, TSN performance is modest. On the one hand, the TSN set of standards lacks support to some attributes commonly used in real-time systems (*e.g.* activation paradigm, precedence constraints, offsets), and the existing QoS attributes are of limited usefulness in what regards real-time applications. For example, CBS parameters are specified as bandwidth, and reservations are issued based on number of frames per time interval and maximum latency, only. Moreover, QoS is in practice specified per class, not per stream because, as mentioned above, the number of priorities/classes is low (up to 6, in practice).

OF also does not support the set of attributes commonly used in real-time systems. QoS specification is limited to bandwidth limitations and priorities. As such, its performance in this regard is poor.

TSSDN is also rather limited. The only specific reference to QoS parameterization of real-time traffic is the specification of the periodicity for TT streams, which is assumed to be expressed as an integral multiple of a base-period that corresponds to a minimum system-wide transmission period that can be supported. No other attributes or traffic types are explicitly supported. There are references also to the use of priorities to favor the transmission of TT traffic when a time slot cannot be found.

SDPROFINET uses formal specifications based on the behavioral type concept, to identify interdependencies between different devices. The authors propose using regular expression based specification mechanisms to capture a sequence chart of communication messages. Reference [80] is does not go deep enough to allow a well supported evaluation of the protocol in this regard, but the approach should allow capturing the essential attributes.

FTT-OpenFlow and OpenFlow RT are based on the FTT paradigm and support the full set of attributes commonly used in real-time systems. Streams are individually associated with attributes such as period, deadline,offsets and activation mode (TT/ET). These attributes as, in practical terms, unconstrained, *e.g.* there are no limits to the number of priorities. Moreover, for the RT ET it is possible to specify QoS at diverse levels, thanks to the presence of hierarchical servers. As such, these protocols excel in this aspect.

SDN-HSF is quite limited in what concerns QoS granularity. The improvements over SDN are restricted to queue management and the implemented disciplines (HTB, RED and SFQ) only provide bandwidth-based traffic shaping. As such, despite improving performance, explicit support to real-time traffic QoS is still poor.

7.1.5 Traffic Management Architecture

TSN inherited AVB's fully distributed model based on the Stream Reservation Protocol. Traffic reservation requests are propagated along the network and each device (end nodes and bridges) decide on the admissibility of each request. It should be noted, however, that SRP only permits to manage stream reservations associated with CBS. Centralized architectures are known to enable more efficient and responsive resource management, supported by a broader knowledge on existing resources and requirements. TSN traffic management was recently augmented with a centralized management option, where reservations are directed to a Centralized Network Configuration entity that decides about the admissibility of reservations and then, when appropriate, uses remote management protocols (*e.g.*, SNMP, NETCONF) to configure bridges in accordance with the requirements. As such, TSN excels in this aspect, by allowing both distributed (with limitations) and centralized management architectures, that have distinct advantages and disadvantages.

SDN prescribes a logically centralized architecture, where the controller is the sole entity responsible for configuring the data-plane switches. As mentioned above, this architecture is arguably more efficient in what concerns the management of network resources, but detractors also point weaknesses, *e.g.* in what concerns scalability. The enhancements brought by SDN-HSF and TSSDN to SDN do not impact significantly on the traffic management architecture of SDN, thus sharing essentially the same properties.

SDPROFINET allows the existence of diverse domain controllers, but prescribes that copies of controllers shall reside on a remote control center to allow topology changes and other modifications to be carried out centrally. The objective is to allow network stabilization and instantiate modifications consistently, eliminating the possibility of instability and transients during updates.

FTT-OpenFlow and OpenFlow RT also have strongly centralized architecture, inherited both from the base FTT architecture and SDN, which are both centralized. In fact, the more elemental functionalities of these protocols depend on the presence of a (logically) centralized entity, which performs activities that go well beyond handling reconfigurations, *e.g.* periodic traffic scheduling. Thus, these protocols depend almost entirely on the permanent availability of the controller.

7.1.6 Flexibility

TSN provides mechanisms to configure all the services directly and indirectly related with stream forwarding, *e.g.* set CBS parameters, configure the Transmission Gates' schedule, issue or remove a stream reservation, *etc.* Despite that, there are some limitations. For example, modifications to stream attributes are not directly supported, and must be instantiated as a tear down followed by a creation. This procedure requires multiple message exchanges and involves several timeout mechanisms, limiting the responsiveness to system adaptations, particularly for the case of distributed architectures, as messages have to be propagated through the entire path. More importantly, TSN only provides the basic mechanisms to parametrize the configuration of network devices, not assisting the applications in managing the allocated QoS. As such, QoS management is entirely left to the application side, which is a severe limitation in terms of flexibility and adaptability. Finally, TSN does not allow the creation of application-specific protocols. The application can only choose which protocols to use (from a predefined set) and configure them.

SDN is, in some sense, in the opposite side of TSN. Due to its "programmatic" approach, the set of protocols that can be deployed is virtually unlimited, being possible to design applications taking standard protocols eventually complemented with custom ones, specifically tailored for individual applications. The centralized architecture also potentiates quick modifications to the system configuration, so the manipulation of stream attributes and system configuration changes can be carried with low delay. As TSN, SDN but does not assist applications in QoS management.

TSSDN, SDPROFINET, and SDN-HSF are SDN-based and thus, share the characteristics of OF in what regards flexibility and adaptability. TSSDN and SDPROFINET may impose a penalization in latency upon system changes due to the computational complexity of the methdos used to derive the schedules and routes. As TSN and OF, these protocols do not provide QoS management, which must also be performed by applications.

As for the case of TSN and OF, the centralized architecture of FTT-OpenFlow and OpenFlow RT enables fast reconfigurations. These protocols allow to create, delete, and modify message streams without service disruption. A unique feature of FTT-OpenFlow and OpenFlow RT that stems from its FTT roots is the native QoS management provided by the network. These protocols provide admission control capabilities along with a QoS manager to which applications may send requests specifying acceptable levels of QoS, *e.g.* in the form of acceptable ranges or bounds for periodicity or deadline. When resources are insufficient, the QoS manager interacts with the admission control to try to find feasible configurations. Therefore, these protocols provide a much better support to flexibility and adaptability than the other ones.

7.1.7 Overall evaluation

Table 7.1 summarizes the results of the above discussion, mapping the performance of the protocols in each criteria in a qualitative scale that ranges from 1 (Worse) to 5 (Better).

Table 7.1: Evaluation of SDN, SDN extensions, and TSN with respect to performance, QoS, and flexibility

Criteria		TSN	OpenFlow	FTT--OpenFlow	TSSDN	SDPROFINET	SDN-HSF	OpenFlow--RT
RT Performance	TT	5	1	5	4	5	1	5
	ET	3	1	5	1	3	3	5
Overhead		4	5	3	4	4	5	3
Mut. Isolation		4	1	5	2	3	1	5
QoS Granularity		3	1	5	2	4	2	5
Manag. Arch.		5	3	3	3	4	3	3
Flexibility		3	4	5	4	4	4	5

From 1 (Worse) to 5 (Better)

There are some interesting conclusions that can be withdrawn. TSN performs well or very well in all criteria. The limitations it exhibits result essentially from backward compatibility issues. The limited number of priorities is particularly relevant as it constrains, in a fundamental way, several aspects of the protocol performance (*e.g.* traffic isolation, event-based messages). The overall complexity that results from the combination of an huge number of protocols, several of them not designed for real-time applications, turns the resulting system hard to analyze and to prove correct.

SDN takes a radically different approach, promising an unprecedented degree of flexibility thanks to its "programmability". However, it was designed for data centers and lacks expressiveness to handle real-time scenarios, therefore its overall real-time performance is very poor.

The potentialities of SDN have been recognized and thus several contributions eventually appeared, having all in common the objective of enriching SDN with real-time services, while preserving its essential attributes. Some of the approaches are simpler but still very limited in terms of real-time performance (SDN-HSF). Others go one step ahead, using dedicated hardware and/or modifications to the communication stack and global management, improving significantly the real-time performance of SDN (TSSDN and SDPROFINET). Finally, FTT-OpenFlow and OpenFlow RT exploit the QoS management and flexibility, characteristic of the FTT paradigm, to enhance OF with efficient real-time and QoS management services.

Summarizing, the qualitative evaluation herein presented clearly shows that TSN performs well, but it is far from perfect, having inherent performance limitations. On the other hand, it also shows that SDN can be augmented, in different ways, to support effective and efficiently applications with real-time requirements, thus being a promising alternative to TSN. Both FTT-OpenFlow and OpenFlow-RT present significant advantages in terms of real-time performance and flexibility over the other SDN-based proposals, however, OpenFlow-RT is superior due to the inherent advantages of HaRTES over FTT-SE such as, for example, the support of nodes not compliant to the protocol and the policing mechanisms within each switch for both time-triggered and event-triggered traffic.

7.2 TSN as data plane enabling technology

As discussed in Section 4.3.1, the development of TSN arises from the need for support of communications conveying time-sensitive data in IEEE 802.1 LAN/Metropolitan Area Network (MAN) networks. Therefore, TSN closely follows the network architectures and protocols defined by IEEE 802 standards. In particular, TSN networks rely on learning mechanisms and a set of protocols, *e.g.* Spanning Tree algorithm and Protocol (STP) and Multiple Spanning Tree algorithm and Protocol (MSTP), that perform vital functions such as building and maintenance of logical network topologies and traffic forwarding routes. The majority of these protocols operate in a distributed manner, *i.e.* devices autonomously exchange information with neighbours and/or nodes across network(s), in order to build the necessary configurations for each device. This operation mode contrasts with the centralized paradigm and control plane decoupling of the SDN framework. The following sections will present a discussion regarding the support for standard OpenFlow services and for the real-time extensions herein proposed on data plane networks comprising TSN devices. The discussion focuses on currently published TSN services, and proposals that are in a late stage of the standardization process, *i.e.* draft published by the work group[1] or approved for publication as standard.

[1]Such as PAR P802.1Qcc and P802.1Qcp [127].

7.2.1 Network Architecture

TSN networks are composed by computational nodes that host applications, called end stations, and nodes that control the forwarding of frames throughout the network, known as bridges. Although IEEE 802.1 (see Section 4.1) defines two bridge types, *i.e.* IEEE 802.1D MAC bridges [82] and IEEE 802.1Q VLAN bridges [23], TSN resource reservation protocols, *e.g.* SRP, and associated QoS services are only available in VLAN bridges. If a TSN traffic stream was to be forwarded by a MAC bridge at any point in the network, real-time performance could be severely compromised. Therefore, only networks that are composed by these bridges alone are considered. Moreover, the considered VLAN bridges support the maximum number of forwarding queues as foreseen by the standard (8 queues) and implement the following amendments to IEEE 802.1Q-2014 [23]:

- ***IEEE 802.1Qbv***: Enhancements for scheduled traffic. Provides time gates and associated control lists that allow the transmission of each queue to be scheduled relative to a known time scale;

- ***IEEE 802.1Qci***: Per-stream filtering and policing. Provides stream filters and gates that allow a bridge to perform filtering and service class selection for data stream's frames, in sync with a cyclic time schedule.

In the considered network, all end stations must adhere to the reservation protocols and transmission patterns for the negotiated time-sensitive streams as defined in IEEE 802.1Q-2014. End stations and VLAN bridges are synchronized in time using the protocols and mechanisms defined by the IEEE 802.1AS-2011 standard [94].

TSN supports the same network topologies as HaRTES and thus, is easily incorporated in the reference architecture. The significant difference between the two architectures is the temporal synchronization of all real-time nodes in the network by the IEEE 802.1AS-2011 protocol. Figure 7.1 represents an example for the framework's reference architecture employing a TSN-enabled data plane.

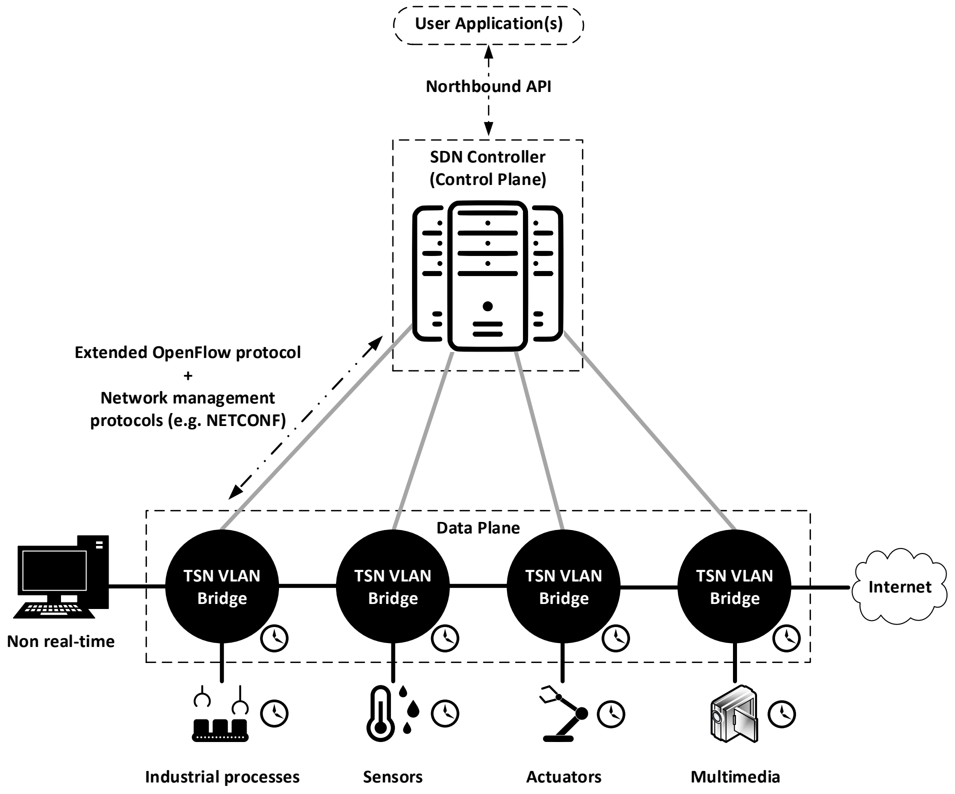


Figure 7.1: Framework reference architecture with TSN-enabled data plane

7.2.2 Supporting standard OpenFlow services

The most important characteristic of technologies that follow the SDN paradigm is the decoupling of the control plane, *i.e.*, the "intelligence" that decides how traffic is to be processed and forwarded to, from the data plane, which processes and forwards traffic according to the rules dictated by the control plane. In OpenFlow, control plane functions are performed by a central entity, composed by one or more physical nodes, which makes all the rules related to traffic forwarding and installs them into every switching device on the data plane in the network using a management protocol: the OpenFlow switch protocol [73]. In contrast, TSN VLAN bridges host both control and data plane functions. Each bridge relies on a set of protocols, *e.g.* STP and MSTP, to communicate with other bridges in the network and exchange data that allows it to build, and maintain, loop-free active topologies. Moreover, bridges autonomously build traffic forwarding rules for their own data path by analyzing the source address of frames received in ingress ports that belong to active topologies. Thus, in order to support standard OpenFlow services on VLAN bridges one must first: (i) strip bridges of the control plane related to traffic processing, and (ii) centralize the control plane for all bridges in a single logical node.

To strip the control plane from TSN devices, all protocols that directly or indirectly configure the forwarding paths and traffic processing rules must be disabled for every bridge in the network. For example, active topology management protocols such as RSTP and MSTP, manage the participation of a given port in the Learning Process. This process in turn, creates Dynamic Filtering Entries that dictate how traffic is forwarded to each egress port (recall from Section 4.1). Therefore, protocols belonging to this category must be disabled administratively, for example, by setting the Port State of every bridge port to "Disabled" (i.e. the port does not participate in active topology management functions). The same applies to protocols that autonomously manage the association of ports to VLANs, *e.g.* Multiple VLAN Registration Protocol (MVRP), and create Dynamic Filtering Entries for VLAN forwarding, *e.g.* ISIS Shortest Path Bridging (SPB). Another family of protocols that must be disabled is that which allows devices and applications to request the reservation of resources throughout the network and the association of traffic flows to those reservations. Examples include the Stream Reservation Protocol (SRP), and applications belonging to the MRP family, such as Multiple Stream Registration Protocol (MSRP) and Multiple MAC Registration Protocol (MMRP). Once all protocols have been identified, these can either be disabled administratively, through device configuration consoles on a per bridge basis, or remotely by the SDN controller using network management protocols such as Common Management Information Protocol (CMIP) or SNMP. However, due to the large number of existing protocols and device vendors, it is not trivial to ascertain if all the required protocols are possible to be disabled in all devices.

Assuming that it is possible to effectively decouple the control plane from the data plane, all control plane functions must now be moved to the SDN controller. Naturally, TSN devices do not support the OpenFlow switch protocol and thus, an alternative way to centrally configure devices is required. At the time of writing, TSN does not provide a native protocol to manage its QoS services, however, the controller may exploit existing remote management protocols such as SNMP, CMIP, and NETCONF, to centrally configure network resources. In particular, to adopt TSN devices in the proposed real-time SDN framework two approaches could be pursued:

1. ***Implement the OpenFlow Mediator daemon in each TSN bridge***. In this approach, the controller directly uses the extended OpenFlow protocol to request configurations of both standard OpenFlow and real-time services. The mediator translates requests into, for example, suitable SNMP commands to configure the necessary services;
2. ***For each bridge, execute an instance of the OpenFlow Mediator daemon in the controller***. Akin to the first approach, mediator daemons translate OpenFlow requests into adequate configuration commands. However, a daemon instance for each TSN bridge is now run within the controller.

As vendors typically have a tight and closed control of each device firmware/operative system, the former method may not be possible. Nonetheless, the developed extensions to the control plane are still valid for both approaches.

TSN bridges do not implement the standard OpenFlow pipeline and associated services. However, some existing entities and services can be leveraged so as to provide an OpenFlow-esque operation:

- ***Filtering Database (FDB)***. This database stores, amongst other type of information, configurable Static Filtering Entries. The FDB and its entries can be leveraged to act as an OpenFlow-like pipeline comprising a single Flow Table, the FDB, with several Flow Entries, the static entries;

- ***Static Filtering Entries***. With respect to traffic forwarding control, static entries are similar to Flow Entries, albeit exhibiting significantly limited filtering rules and actions;

- ***Flow Classification and Metering***. TSN bridges may support ingress metering, in which a subset of traffic frames may be identified and subjected to a given meter which can discard frames on the basis of parameters such as frame size and inter-frame time. This mechanism, and/or Credit-Based Shapers, can be used to implement a service akin to OpenFlow meters;

- ***Traffic Class Tables and Transmission Selection Algorithm Tables***. Each bridge port has an associated Traffic Class Table that maps frames to traffic classes using the frame's priority. Traffic classes are associated to individual egress queues which may be served by different dispatching services. The association of dispatching algorithms to each queue can be performed through the configuration of the port's Transmission Selection Algorithm Table. These tables may be configured and combined so as certain traffic flows are forwarded to different classes and dispatching methods, akin to OpenFlow logical ports;

- ***Port Counters***. These objects are similar to counters specified by OpenFlow, and can provide the number of bytes and frames processed by a given ingress/egress port.

With these services, one could create a simple pipeline with rules to identify traffic flows, and drop or forward their frames to the desired egress queues. This is comparable to configuring Flow Entries comprising only "Drop" or "Output" actions with physical and logical ports. Identified flows could be applied to metering, which is comparable to the OpenFlow action "Meter". However, this virtual pipeline is dramatically limited when compared to OpenFlow's mandatory services. For example, although Static Filtering Entries allow one to identify flows, the supported filters only target frames' destination MAC address and VLAN ID, and do not provide support for other OpenFlow mandatory filtering fields such as IPv4 addresses, EtherType, and TCP/UDP ports. Moreover, group tables are nonexistent and can't be easily modeled by existing TSN entities. Finally, the support for packet-in/packet-out operations is difficult to evaluate, since the features provided by management interfaces are vendor-specific.

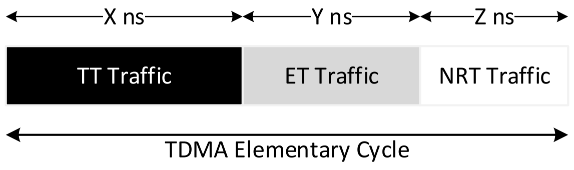
Despite the aforementioned constraints, it seems possible to provide limited *SDN-like* services over TSN bridges. Next, we take a look at the offered QoS services and evaluate "if" and "how" could they be leveraged by the devised framework in order to provide the envisioned real-time services.

7.2.3 Supporting real-time traffic

TSN bridges provide several services that can be used and combined to enforce certain levels of QoS, *e.g.* traffic classification, metering, and transmission selection algorithms. However, there are (presently) two services specifically tailored for traffic with real-time requirements: the Credit-Based Shaper (CBS) selection algorithm (Section 4.3.1.1) and Transmission Gates (Section 4.3.1.2) [23]. The former provides a shaping service which grossly enforces bandwidth reservations for streams that need low latency levels such as video and audio streaming. The latter is specifically designed for periodic traffic, and allows the confinement of traffic within specific time intervals. Due to their intrinsic properties, CBS is better suited for event-triggered traffic while transmission gates are adequate for time-triggered communications.

One of the important requirements to support both event- and time-triggered traffic in a shared communication network is the existence of isolation between these two traffic types. Failing to do so typically results into high jitter values, in particular for time-triggered communications. Therefore, a first step is to ensure that interference between the two traffic types is nonexistent, or at least greatly reduced and within controlled bounds. To that end, TSN specifies procedures for the MAC layer to allow certain traffic classes to preempt on-going transmissions (IEEE 802.1Qbu [97] and IEEE 802.3br [98]). However, despite providing true non-destructive preemption, on-going transmissions can't be instantaneously preempted and fragments from preempted frames must have a minimum size. Although one can use this mechanism to greatly reduce interference and contain it within well-defined bounds, there is another possible approach in which interference can be completely eliminated: implement a TDMA scheme with temporal windows, similar to that used by HaRTES. Figure 7.2 depicts such scheme and the most important services that are necessary for its implementation.

Similarly to HaRTES EC, the proposed TDMA scheme implements three temporal windows, each one assigned to a specific type of traffic streams: time-triggered, event-triggered, and non real-time. Windows confine the transmission of the associated traffic type within their bounds. The first step to implement such scheme is to reserve traffic classes, and their respective egress queues, for a particular type of streams. In the example of Figure 7.2, traffic classes 4 up to 7 are reserved for time-triggered streams, while classes 1 up to 3 and class 0 are assigned to event-triggered and non real-time streams, respectively. Traffic is assigned to a certain traffic class using the standard TSN mechanisms, *i.e.* Traffic Class Table and frame's VLAN Identifier. The next step is to enforce the temporal windows. To that end, the Gate Control List and its entries are configured in such a way that transmission gates only allow traffic to be selected for transmission from queues comprising traffic associated to the current temporal window. In the provided example, gate operations 0 up to 3 enforce the time-triggered window by disabling transmissions from the event-triggered and non real-time queues. The same reasoning is used for the remaining windows. Finally, adequate transmission selection algorithms must be configured for each egress queue. Time-triggered queues are served by the Strict Priority Transmission algorithm (SPT) while event-triggered queues are assigned to the CBS algorithm. Non real-time uses SPT and occupies the remaining bandwidth as



Gate Control List

Gate Op #	7	6	5	4	3	2	1	0	
	Queue #								
T00	o	o	o	o	c	c	c	c	X ns
T01	o	o	o	o	c	c	c	c	
T02	o	o	o	o	c	c	c	c	
T03	o	o	o	o	c	c	c	c	
T04	c	c	c	c	o	o	o	c	Y ns
T05	c	c	c	c	o	o	o	c	
T06	c	c	c	c	o	o	o	c	
T07	c	c	c	c	c	c	c	o	Z ns

Assigned Queues by Traffic Type

Queue#	TT	ET	NRT
7	X		
6	X		
5	X		
4	X		
3		X	
2		X	
1		X	
0			X

Transmission Selection Algorithm Table

Queue#	SPT	ETS	CBS
7	X		
6	X		
5	X		
4	X		
3			X
2			X
1			X
0	X		

Figure 7.2: TDMA for real-time communications in TSN

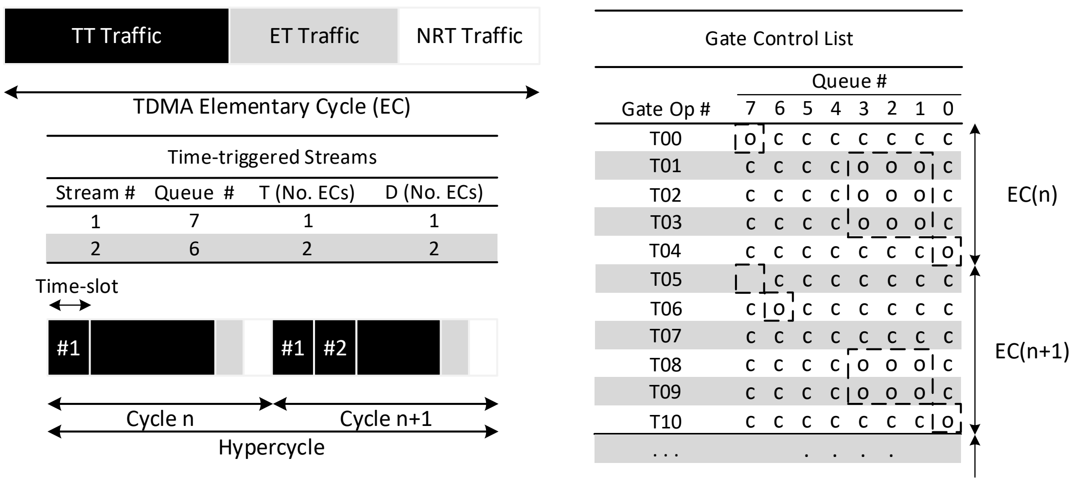
background traffic.

After deploying the previously discussed TDMA scheme, the next question is how one schedules traffic and assigns the reserved queues to streams pertaining to the same traffic category. This is addressed next.

7.2.3.1 Incorporating time-triggered traffic

With the extremely limited number of queues provided by TSN, an important matter is how can such number be optimally exploited and at the same time provide timeliness communications. Taking the previous example (Figure 7.2) as case-study, the issue is how to distribute the four available queues among a set of time-triggered streams. If the number of streams is equal or inferior to the number of available queues, the solution is plain obvious: assign a dedicated queue for each stream. However, this solution suffers from extremely poor scalability. If the number of streams is higher than the number of available queues, which can be easily attainable in industrial scenarios, queue sharing must be executed. For this, there are multiple possible schemes: (i) a single queue for all streams, which could result in a significant amount of interference and worst-case delays in particular for large stream sets, (ii) share queues for streams with equal period, this could lower some of the interference, in particular if streams have offset periods, (iii) mix streams according to common period multiples, building sets that minimize the number of stream frames in a given queue within the same time interval. All of the aforementioned schemes either have scalability or interference issues, or both. A better approach is to evolve TSN egress queues from FIFO to content-addressed queues and allow a real-time scheduler/transmission selection algorithm to retrieve frames from a given queue according to a schedule. However, current standards explicitly mandate that all implemented queues preserve frame arrival order, *i.e.* operate like a FIFO [23].

Another issue is how to schedule the transmissions of queued time-triggered traffic. Assuming that in future TSN versions it would be possible to have a dedicated queue for each stream, or a method to store frames into the existing queues according to a schedule, the time-triggered window could be further divided into several time-slots, each controlled by a given transmission gate. Frames from queued streams are now transmitted only within assigned time-slots. To control transmissions in time-slots within the window, the Gate Control List would have to be configured in order to implement a slot schedule map. Figure 7.3 shows an example for two time-slots.



Time-triggered Streams

Stream #	Queue #	T (No. ECs)	D (No. ECs)
1	7	1	1
2	6	2	2

Gate Control List

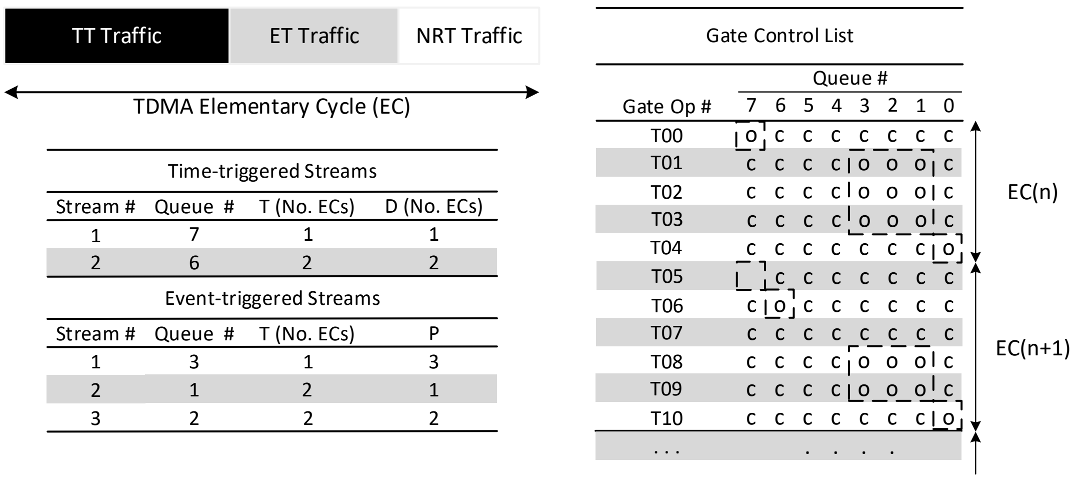
	Queue #							
Gate Op #	7	6	5	4	3	2	1	0
T00	o	c	c	c	c	c	c	c
T01	c	c	c	c	o	o	o	c
T02	c	c	c	c	o	o	o	c
T03	c	c	c	c	o	o	o	c
T04	c	c	c	c	c	c	c	o
T05		c	c	c	c	c	c	c
T06	c	o	c	c	c	c	c	c
T07	c	c	c	c	c	c	c	c
T08	c	c	c	c	o	o	o	c
T09	c	c	c	c	o	o	o	c
T10	c	c	c	c	c	c	c	o
...			.	.	.	.		

Figure 7.3: Time-triggered scheduling in TSN

In short, for each time-slot activation a gate operation entry, enabling the respective transmission gate, is required. Since time-triggered streams may have different periods, the Gate Control List must be populated with enough gate operation entries to cover all activations within an entire hypercycle, *i.e.* until the least common multiple (LCM) for all existing stream periods is reached. Although the hypercycle is relatively short for the presented example (LCM = 2), it can rapidly grow to huge values, especially in the presence of high prime period values. For example, consider a small set with only three streams and periods of 1, 9, and 101 ECs. The LCM for this set is 909, and thus, the Gate Control List would have to spawn over 909 cycles, each cycle requiring a bare minimum of 2 gate control entries, one for the ET and NRT window, and one entry per time-slot activation. This can rapidly lead to a list with overwhelming size and impossible to be effectively managed by the TSN bridge. A possible solution for this, suggested for WorldFIP in [6], is to dynamically upload segments of the entire table as needed. TSN does support the dynamic update of the Gate Control List, however, the time required to concurrently update such tables in all ports needs to be evaluated and the impact on the real-time dispatching determined.

7.2.3.2 Incorporating event-triggered traffic

Event-triggered traffic suffers the same queue-related issues that were identified for time-triggered traffic. Although one could use the solutions proposed therein for queue assignment, sharing queues among several streams induces significantly negative drawbacks. As each queue is now managed by a CBS shaper, the configured bandwidth limits apply to the whole queue. Thus, streams assigned to a given queue will compete for the same bandwidth, leading to possible starvation for some streams. Moreover, the ability to constrain the maximum bandwidth used by each stream is lost, since CBS must be configured for the stream with the highest bandwidth requirement. In this case, a stream with a lower negotiated bandwidth could just use more. Therefore, for event-triggered traffic, a dedicated queue for each stream is vital. Figure 7.4 depicts an example of such configuration comprising three event-triggered streams. Besides the clear limitation on the support for several event-triggered streams, TSN lacks the support to build hierarchical relationships between streams, an important feature to tackle with highly complex systems.



Time-triggered Streams

Stream #	Queue #	T (No. ECs)	D (No. ECs)
1	7	1	1
2	6	2	2

Event-triggered Streams

Stream #	Queue #	T (No. ECs)	P
1	3	1	3
2	1	2	1
3	2	2	2

Gate Control List

	Queue #							
Gate Op #	7	6	5	4	3	2	1	0
T00	o	c	c	c	c	c	c	c
T01	c	c	c	c	o	o	o	c
T02	c	c	c	c	o	o	o	c
T03	c	c	c	c	o	o	o	c
T04	c	c	c	c	c	c	c	o
T05		c	c	c	c	c	c	c
T06	c	o	c	c	c	c	c	c
T07	c	c	c	c	c	c	c	c
T08	c	c	c	c	o	o	o	c
T09	c	c	c	c	o	o	o	c
T10	c	c	c	c	c	c	c	o
...			.	.	.	.		

Figure 7.4: Event-triggered scheduling in TSN

7.2.4 Data plane comparative analysis: TSN vs HaRTES

Over the course of the last sections, TSN has been evaluated as a possible real-time data plane enabler for the proposed SDN framework. One of the main derived conclusions is that TSN, as in the present form, may be able to provide basic SDN services provided that device vendors grant a way to disable, either through configuration consoles or by management protocols like SNMP and NETCONF, all the protocols that intervene in the configuration of forwarding paths, traffic processing rules, and enact the reservation of network resources. Examples of identified protocols include RSTP, the MRP family, and SRP. Nonetheless, the OpenFlow services provided by TSN would be severely limited, lacking mandatory services such as Group Tables and with no support for a large number of filtering rules, as discussed in Section 7.2.2.

Additionally, the real-time services provided by TSN exhibit several limitations, the most relevant being linked to the small number of traffic classes and lack of adequate support for scheduling algorithms. Notwithstanding, it was shown the possibility of enacting temporal isolation between different traffic categories and support, with severe limitations, both time- and event-triggered real-time streams. Therefore, HaRTES continues to be considered the best choice as the data plane enabler switching platform for the herein devised real-time SDN framework. A breakdown of the currently supported services in HaRTES vs those that would be possible with TSN is presented in Table 7.2. Note that HaRTES is still being enhanced and will support even more OpenFlow services in the future.

Table 7.2: Data plane enablers : HaRTES vs TSN

	Tables		Ports	Instructions	Match Fields	RT Traffic Flows	
	Flow	Group				TT	ET
TSN	1	✗	Physical Logical (few) FLOOD NORMAL	Write-Actions	ETH_DST VLAN_VID	6 (max)*	6 (max)*
HaRTES	1	✗	Physical Logical (all) FLOOD NORMAL CONTROLLER TABLE IN_PORT	Write-Actions Goto-Table	ETH_DST ETH_SRC ETH_TYPE IPV4_DST IPV4_SRC IP_PROTO TCP_DST UDP_DST TCP_SRC UDP_SRC	32**	8**

* Can either have: [6 TT + 1 ET + 1 NRT] or [1 TT + 6 ET + 1 NRT] flows.
** Not a hard limit. Easily scaled at the cost of more FPGA resources.

CHAPTER 8

The *Finale*

Contents

8.1 Conclusions

The new trend in Industry towards Smart Production/Industry 4.0 poses new requirements for flexibility of traffic management together with the usual strict timeliness requirements. Software-Defined Networking can provide the needed flexibility but lacks appropriate support for timeliness. This issue is addressed in this work that argues:

"*The flexibility, timeliness, management, and heterogeneity requirements of emerging cyber-physical production systems can be satisfied by a framework that leverages the inherent flexibility of SDN technologies to control a network comprising switching platforms with composable and dynamically configurable real-time services*".

Herein, a novel real-time framework that extends the SDN/OpenFlow architecture with real-time communication services, and is able to support event-triggered and time-triggered real-time traffic with guaranteed Quality of Service (QoS) is presented. Three main contributions were made:

- **A set of real-time extensions to OpenFlow**, the Real-Time OpenFlow Add-On (RTOF), that accommodates the specification of real-time flows and the (re)configuration of real-time reservations while keeping compatibility with the OpenFlow standard. The RTOF is agnostic to underlying data plane technologies by using generic real-time attributes.

- **An OpenFlow-enabled real-time Ethernet data plane technology (Enhanced HaRTES)**. HaRTES switches are now augmented with a hardware OpenFlow pipeline, which currently supports most of the mandatory OpenFlow services, and a mediation layer to communicate with SDN controllers using the RTOF extensions.

- **A SDN controller for real-time applications**. This controller enacts admission control operations to guarantee the timeliness of network communications. It employs a real-time schedulability analysis for multi-hop HaRTES networks to predict the temporal behavior of the network upon the (possible) admission or reconfiguration of flows.

The capabilities of the developed framework were evaluated through experiments carried out on a real-world prototype. Four main vectors were analyzed: (i) the ability to perform (re)configurations online and admit new streams without causing disruption or interference to existing traffic, (ii) the timeliness behavior of the system in the presence of diverse classes of real-time and non real-time traffic, (iii) the accuracy of the output results from the developed schedulability analysis, and (iv) the responsiveness to requests and scalability of the admission control unit. The obtained results confirm that the framework is able to fulfill all the network requirements identified in Section 2.3. In particular, the framework is able to:

- **Support simultaneous applications with heterogeneous QoS requirements (R1)**. Experiments based on a realistic Industry 4.0 scenario (adaptable production) show the coexistence of time-/event-triggered and non real-time flows, with all timeliness guarantees being met. The observed latency and jitter values, which were under/at $70\mu s$ and $1\mu s$, respectively, show the capability of the framework to fulfill the most strict requirements imposed by motion control applications.

- **Precisely meet applications' QoS requirements (R2)**. Thanks to the traffic policing and confinement mechanisms provided by the framework's data plane technology (HaRTES), communications are efficiently restricted to the negotiated network reservations. Moreover, the analysis performed by the developed RT SDN controller exhibits a relatively low level of pessimism, as experimentally observed.

- **Support dynamic message and reservation sets (R3).** By leveraging the flexibility of the enhanced data plane and the real-time extensions to the control plane, the framework is able to perform changes to message and reservation sets online, without service disruption. Moreover, even for reasonably sized networks comprising 16 switches and 100 flows, single reconfiguration requests are evaluated and performed under $170ms$. This value is significantly less than the desired reconfiguration time of small production systems, which falls in the range of a few seconds [46].

- **Enable real-time network monitoring and diagnostics (R4).** With the services provided by the OpenFlow protocol and its extensions herein introduced, the framework is able to keep track of existing flows, current link states, and flow statistics, providing a powerful monitoring and diagnostics platform.

- **Provide a consistent set of open management tools (R5).** The centralized management of the whole network using a single protocol, *i.e.* OpenFlow and proposed real-time extensions, combined with the possibility to easily integrate additional network management protocols, *e.g.* SNMP, for possible non-OpenFlow devices, eases network management and reduces the dependence on multiple vendor-specific tools.

Finally, a qualitative study evaluated the herein proposed framework and the most relevant proposals in the literature targeting the extension of SDN and its use on real-time networks. The study concludes that this thesis's proposal provides significant advantages over the other solutions (including TSN) regarding operational flexibility and real-time performance.

8.2 Future research

An important line of future work is the investigation of northbound APIs and possible real-time extensions. At the time of writing, and to best of my knowledge, the existing northbound APIs focus on business-related applications such as, for example, network virtualization and lack adequate models/data structures to specify the requirements of real-time applications. Although the ONF has standardized the Real-Time Media NBI [72], a northbound API for multimedia applications, it mainly uses parameters such as bandwidth and other properties which are unsuitable for hard real-time applications.

Another important line of work is the integration of heterogeneous communication technologies into the data plane. In particular, the use of wireless segments and 5G technology are of great importance in the industrial landscape [40], [45]. Besides the challenges posed by the MAC layer in order to provide timeless and reliability guarantees, wireless technologies also exhibit limitations on flexible management and real-time monitoring [40]. Thus, the herein enhanced control plane could be of significant value to overcome the aforementioned management and monitoring short-comings. In fact, a first step towards the use of the herein developed OpenFlow extensions was carried by Paulo Ribeiro *et al.* [128].

Finally, as the vision for the future industrial landscape entails a massive presence of machine-to-machine (M2M) communications, the integration of enabling M2M protocols, such as OPC-UA and MQTT, is a natural, following step. The SDN framework could be exploited by these protocols to, for example, dynamically and autonomously configure channels for M2M communication with the necessary real-time guarantees.

References

[1] L. Silva, P. Pedreiras, P. Fonseca, and L. Almeida, “On the adequacy of SDN and TSN for industry 4.0”, in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, May 2019. DOI: `10.1109/isorc.2019.00017`. [Online]. Available: `https://doi.org/10.1109%2Fisorc.2019.00017`.

[2] L. Silva, P. Goncalves, R. Marau, P. Pedreiras, and L. Almeida, “Extending OpenFlow with flexible time-triggered real-time communication services”, in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, Sep. 2017. DOI: `10.1109/etfa.2017.8247595`. [Online]. Available: `https://doi.org/10.1109%2Fetfa.2017.8247595`.

[3] L. Silva, P. Goncalves, R. Marau, and P. Pedreiras, “Extending OpenFlow with industrial grade communication services”, in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, IEEE, May 2017. DOI: `10.1109/wfcs.2017.7991965`. [Online]. Available: `https://doi.org/10.1109%2Fwfcs.2017.7991965`.

[4] M. Ashjaei, L. Silva, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte, “Improved message forwarding for multi-hop HaRTES real-time ethernet networks”, *Journal of Signal Processing Systems*, vol. 84, no. 1, pp. 47–67, May 2015. DOI: `10.1007/s11265-015-1010-8`. [Online]. Available: `https://doi.org/10.1007%5C%2Fs11265-015-1010-8`.

[5] L. Silva, P. Pedreiras, M. Ashjaei, M. Behnam, T. Nolte, L. Almeida, and R. J. Bril, “Demonstrating the multi-hop capabilities of the hartes real-time ethernet switch”, in *RTSS@Work 2014 Open Demo Session of RealTime Systems*, 2014.

[6] L. Almeida, “Flexibility and timeliness in fieldbus-based real-time systems”, PhD thesis, Universidade de Aveiro, 1999.

[7] B. Galloway and G. P. Hancke, “Introduction to industrial control networks”, *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 860–880, 2013. DOI: `10.1109/surv.2012.071812.00124`. [Online]. Available: `https://doi.org/10.1109%2Fsurv.2012.071812.00124`.

[8] J.-P. Thomesse, “Fieldbus technology in industrial automation”, *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1073–1101, Jun. 2005, ISSN: 0018-9219. DOI: `10.1109/JPROC.2005.849724`.

[9] *IEEE standard for local area networks: Token ring access method and physical layer specifications.* DOI: `10.1109/ieeestd.1989.108547`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.1989.108547`.

[10] *IEEE standard for ethernet.* DOI: `10.1109/ieeestd.2016.7428776`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2016.7428776`.

[11] *Ansi/isa–50.1–1982 (r1992)- compatibility of analog signals for electronic industrial process instruments*, 1982. [Online]. Available: `https://standards.globalspec.com/std/335354/isa-50-1`.

[12] R. Zurawski, *Industrial communication technology handbook.* CRC Press, 2017.

[13] E. Jasperneite and P. Neumann, “Switched ethernet for factory communication”, in *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.01TH8597)*, IEEE. DOI: `10.1109/etfa.2001.996370`. [Online]. Available: `https://doi.org/10.1109%2Fetfa.2001.996370`.

[14] T. Sauter, “The three generations of field-level networks—evolution and compatibility issues”, *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3585–3595, Nov. 2010. DOI: `10.1109/tie.2010.2062473`. [Online]. Available: `https://doi.org/10.1109%2Ftie.2010.2062473`.

[15] C. Cseh and J. Jasperneite, “Emerging data transfer technologies for factory communication”, in *IECON ’98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.98CH36200)*, IEEE. DOI: `10.1109/iecon.1998.724048`. [Online]. Available: `https://doi.org/10.1109%2Fiecon.1998.724048`.

[16] A. F. T. Committee *et al.*, "Traffic management specification version 4.0", in *ATM Forum contribution*, 1995, 95–0013R10.

[17] J. Dwyer, "Why general motors' manufacturing automation protocol is here to stay", *Automation*, vol. 21, no. 5, pp. 19–21, 1985.

[18] N. Collins, "Boeing architecture and top (technical and office protocol)", in *Proc. Int. Conf. Networking: A Large Organization Perspective*, 1986, pp. 49–54.

[19] M. Gault and J. Lobert, "Contribution for the fieldbus standard", *Presentation to IEC/TC65/SC65C/WG6*, 1985.

[20] H. Frazier, "The 802.3z gigabit ethernet standard", *IEEE Network*, vol. 12, no. 3, pp. 6–7, 1998. DOI: `10.1109/65.690946`. [Online]. Available: `https://doi.org/10.1109%2F65.690946`.

[21] J. Guillaud, M. R. Pokam, and G. Michel, "Information superhighway enters the manufacturing world", in *Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Aug. 1995, pp. 210–215. DOI: `10.1109/HPCS.1995.662031`.

[22] L. L. Bello, M. Lorefice, O. Mirabella, and S. Oliveri, "Performances analysis of ethernet networks in the process control", in *ISIE'2000. Proceedings of the 2000 IEEE International Symposium on Industrial Electronics (Cat. No.00TH8543)*, vol. 2, Dec. 2000, 655–660 vol.2. DOI: `10.1109/ISIE.2000.930375`.

[23] "Ieee standard for local and metropolitan area networks–bridges and bridged networks", *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec. 2014. DOI: `10.1109/IEEESTD.2014.6991462`.

[24] Hirschmann Network Systems, "Real Time Services (QoS) In Ethernet Based Industrial Automation Networks", Tech. Rep., 1999.

[25] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS ethernet switches", in *2006 IEEE International Workshop on Factory Communication Systems*, IEEE, 2006. DOI: `10.1109/wfcs.2006.1704170`. [Online]. Available: `https://doi.org/10.1109%2Fwfcs.2006.1704170`.

[26] *EtherNet/IP – CIP on Ethernet Technology*, 2016. [Online]. Available: `https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00138R6_Tech-Series-EtherNetIP.pdf`.

[27] *PROFINET - the leading Industrial Ethernet Standard.* [Online]. Available: `https://www.profibus.com/technology/profinet/` (visited on 09/01/2018).

[28] TTTech), *Time-Triggered Ethernet – A Powerful Network Solution for Multiple Purpose.* [Online]. Available: `https://www.tttech.com/fileadmin/content/general/secure/TTEthernet/TTTech_TTEthernet_Technical-Whitepaper.pdf` (visited on 09/01/2018).

[29] R. Santos, "Enhanced ethernet switching technology for adaptive hard real-time applications", PhD thesis, Universidade de Aveiro, 2007.

[30] IEEE 802.1 Working Group, *Time-sensitive networking task group.* [Online]. Available: `http://www.ieee802.org/1/pages/tsn.html` (visited on 11/22/2017).

[31] J. Morse, *The world market for industrial ethernet components*, 2011. [Online]. Available: `http://www.iebmedia.com/index.php?id=8595&parentid=74&themeid=255&showdetail=true&bb=true` (visited on 08/20/2018).

[32] T. Carlsson, *Industrial ethernet is now bigger than fieldbuses*, 2018. [Online]. Available: `https://www.anybus.com/about-us/news/2018/02/16/industrial-ethernet-is-now-bigger-than-fieldbuses` (visited on 08/20/2018).

[33] M. Beck, *Ethernet in the First Mile: The IEEE 802.3ah EFM Standard*, ser. McGraw-Hill professional engineering: Communications engineering. McGraw-Hill Education, 2005, ISBN: 9780071469913.

[34] *Plattform Industrie 4.0.* [Online]. Available: `https://www.plattform-i40.de/` (visited on 08/20/2018).

[35] M. Blanchet, T. Rinn, G. Von Thaden, and G. De Thieulloy, "Industry 4.0: The new industrial revolution - how europe will succeed", *Hg. v. Roland Berger Strategy Consultants GmbH. München*,

2014. [Online]. Available: https://www.rolandberger.com/publications/publication_pdf/roland_berger_tab_industry_4_0_20140403.pdf (visited on 08/20/2018).

[36] R. C. Schläpfer, M. Koch, and P. Merkhofer, "Industry 4.0 challenges and solutions for the digital transformation and use of exponential technologies", *Deloitte, Zurique*, 2015. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf (visited on 09/20/2018).

[37] E. S. Jacek Walendowski Henning Kroll, "Industry 4.0, advanced materials (nanotechnology)", *Regional Innovation Monitor Plus 2016*, 2016. [Online]. Available: https://ec.europa.eu/growth/tools-databases/regional-innovation-monitor/sites/default/files/report/RIM%5C%20Plus_Industry%5C%204.0%5C%2C%5C%20Advanced%5C%20Materials%5C%20%5C%28Nanotechnology%5C%29_Thematic%5C%20paper.pdf (visited on 09/20/2018).

[38] ITU-T, "Recommendation itu-t y.2060: Overview of the internet of things", *SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS*, 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559 (visited on 09/20/2018).

[39] V. Koch, S. Kuge, R. Geissbauer, and S. Schrauf, "Industry 4.0: Opportunities and challenges of the industrial internet", *Strategy & PwC*, 2014. [Online]. Available: https://www.pwc.nl/en/assets/documents/pwc-industrie-4-0.pdf (visited on 09/22/2018).

[40] J.-S. Bedo, E. Calvanese, S. Castellvi, T. Cherif, V. Frascolla, W. Haerick, I. Korthals, O. Lazaro, E. Sutedjo, L. Usatorre, and M. Wollschlaeger, *White paper on factories of the future vertical sector*, 2015. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Factories-of-the-Future-Vertical-Sector.pdf.

[41] *Status report: Reference architecture model industrie 4.0 (rami4.0)*, 2015. [Online]. Available: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/5305_Publikation_GMA_Status_Report_ZVEI_Reference_Architecture_Model.pdf.

[42] *Working paper: Aspects of the research roadmap in application scenarios*, 2016. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf?__blob=publicationFile&v=10.

[43] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0", *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[44] *Discussion paper: Network-based communication for industrie 4.0*, 2016. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/network-based-communication-for-i40.pdf?__blob=publicationFile&v=6.

[45] *Etsi ts 122 261 v15.5.0 - 5g : Service requirements for next generation new services and markets*, 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/122200_122299/122261/15.05.00_60/ts_122261v150500p.pdf.

[46] W. Lepuschitz, "Self-Reconfigurable Manufacturing Control based on Ontology-Driven Automation Agents", PhD thesis, Technische Universität Wien, 2018. [Online]. Available: http://repositum.tuwien.ac.at/obvutwhs/content/titleinfo/2582212?lang=en.

[47] T. Bangemann, M. Riedl, M. Thron, and C. Diedrich, "Integration of Classical Components Into Industrial Cyber–Physical Systems", *Proceedings of the IEEE*, vol. 104, no. 5, pp. 947–959, May 2016, ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2510981.

[48] D. Henneke, L. Wisniewski, and J. Jasperneite, "Analysis of realizing a future industrial network by means of Software-Defined Networking (SDN)", in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, May 2016, pp. 1–4. DOI: 10.1109/WFCS.2016.7496525.

[49] G. C. Buttazzo, *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications*. Springer US, 2011. DOI: 10.1007/978-1-4614-0676-1. [Online]. Available: https://doi.org/10.1007%2F978-1-4614-0676-1.

[50] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973, ISSN: 0004-5411. DOI: 10.1145/321738.321743. [Online]. Available: http://doi.acm.org/10.1145/321738.321743.

[51] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems", in *[1992] Proceedings Real-Time Systems Symposium*, Dec. 1992, pp. 110–123. DOI: 10.1109/REAL.1992.242671.

[52] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments", *IEEE Trans. Computers*, vol. 44, pp. 73–91, 1987.

[53] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments", *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 73–91, Jan. 1995, ISSN: 0018-9340. DOI: 10.1109/12.368008.

[54] J. L. Brinkley Sprunt Lui Sha, "Aperiodic task scheduling for real-time systems", AAI9107570, PhD thesis, Pittsburgh, PA, USA, 1990.

[55] T.-S. Tia, J. W.-S. Liu, and M. Shankar, "Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems", *Real-Time Syst.*, vol. 10, no. 1, pp. 23–43, Jan. 1996, ISSN: 0922-6443. DOI: 10.1007/BF00357882. [Online]. Available: http://dx.doi.org/10.1007/BF00357882.

[56] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems", *Real-Time Systems*, vol. 10, pp. 179–210, Mar. 1996. DOI: 10.1007/BF00360340.

[57] M. Joseph and P. Pandya, "Finding response times in a real-time system", *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986. DOI: 10.1093/comjnl/29.5.390. eprint: /oup/backfile/content_public/journal/comjnl/29/5/10.1093/comjnl/29.5.390/2/290390.pdf. [Online]. Available: http://dx.doi.org/10.1093/comjnl/29.5.390.

[58] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling", *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, Sep. 1993, ISSN: 0268-6961. DOI: 10.1049/sej.1993.0034.

[59] Z. Deng and J. W. .-.-. Liu, "Scheduling real-time applications in an open environment", in *Proceedings Real-Time Systems Symposium*, Dec. 1997, pp. 308–319. DOI: 10.1109/REAL.1997.641292.

[60] M. Holenderski, R. J. Bril, and J. J. Lukkien, "An efficient hierarchical scheduling framework for the automotive domain", in *Real-Time Systems, Architecture, Scheduling, and Application*, InTech, 2012.

[61] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model", *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 30:1–30:39, May 2008, ISSN: 1539-9087. DOI: 10.1145/1347375.1347383. [Online]. Available: http://doi.acm.org/10.1145/1347375.1347383.

[62] ——, "Periodic resource model for compositional real-time guarantees", in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, Dec. 2003, pp. 2–13. DOI: 10.1109/REAL.2003.1253249.

[63] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using edp resource models", in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, Dec. 2007, pp. 129–138. DOI: 10.1109/RTSS.2007.36.

[64] H. Kopetz, *Real-Time Systems*. Springer US, 2011. DOI: 10.1007/978-1-4419-8237-7. [Online]. Available: https://doi.org/10.1007%2F978-1-4419-8237-7.

[65] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[66] P. Pedreiras, "Supporting flexible real-time communication on distributed systems", PhD thesis, Universidade de Aveiro, 2003.

[67] M. Ashjaei, "Real-time communication over switched ethernet with resource reservation", PhD thesis, Malardalen University, Nov. 2016. [Online]. Available: http://www.es.mdh.se/publications/4564-.

[68] T. N. D. and K. Gray, *SDN: Software Defined Networks*, 1st. O'Reilly Media, Inc., 2013, ISBN: 1449342302, 9781449342302.

[69] *Ciscoworks lan management solution (lms) version 4.0.* [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/ciscoworks-lan-management-solution-3-2-earlier/white_paper_c11-542881.html.

[70] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks", *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014, ISSN: 0146-4833. DOI: 10.1145/2602204.2602219. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602219.

[71] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[72] "Onf tr-523: Intent nbi – definition and principles", *Open Networking Foundation*, 2016. [Online]. Available: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/TR-523_Intent_Definition_Principles.pdf (visited on 01/12/2018).

[73] "Openflow switch specification version 1.5.0 (protocol version 0x06)", *Open Networking Foundation*, pp. 1–277, Dec. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf (visited on 07/12/2018).

[74] M. Ehrlich et al., "Software-Defined Networking as an Enabler for Future Industrial Network Management", in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2018, pp. 1109–1112. DOI: 10.1109/ETFA.2018.8502561.

[75] G. Kálmán, "Applicability of Software Defined Networking in industrial Ethernet", in *Proceedings of the 22nd Telecommunications Forum Telfor (TELFOR)*, Nov. 2014, pp. 340–343. DOI: 10.1109/TELFOR.2014.7034420.

[76] D. Thiele and R. Ernst, "Formal analysis based evaluation of software defined networking for time-sensitive Ethernet", in *2016 Design, Automation Test in Europe Conf. (DATE)*, Mar. 2016, pp. 31–36.

[77] M. Herlich, J. L. Du, F. Schörghofer, and P. Dorfinger, "Proof-of-concept for a software-defined real-time Ethernet", in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–4. DOI: 10.1109/ETFA.2016.7733605.

[78] C. Ternon, J. Goossens, and J.-M. Dricot, "FTT-OpenFlow, on the Way Towards Real-time SDN", *SIGBED Rev.*, vol. 13, no. 4, pp. 49–54, Nov. 2016, ISSN: 1551-3688. DOI: 10.1145/3015037.3015045.

[79] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive Software-defined Network (TSSDN) for Real-time Applications", in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16, Brest, France: ACM, 2016, pp. 193–202, ISBN: 978-1-4503-4787-7. DOI: 10.1145/2997465.2997487.

[80] K. Ahmed, J. O. Blech, M. A. Gregory, and H. Schmidt, "Software defined networking for communication and control of cyber-physical systems", in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2015, pp. 803–808. DOI: 10.1109/ICPADS.2015.107.

[81] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking", in *2013 Second European Workshop on Software Defined Networks*, Oct. 2013, pp. 81–86. DOI: 10.1109/EWSDN.2013.20.

[82] "Ieee standard for local and metropolitan area networks: Media access control (mac) bridges", *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pp. 1–277, Jun. 2004. DOI: 10.1109/IEEESTD.2004.94569.

[83] J.-D. Decotignie, "Ethernet-based real-time and industrial communications", *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102–1117, Jun. 2005. DOI: 10.1109/jproc.2005.849721. [Online]. Available: https://doi.org/10.1109%2Fjproc.2005.849721.

[84] D. I. Katcher, S. S. Sathaye, and J. K. Strosnider, "Fixed priority scheduling with limited priority levels", *IEEE Transactions on Computers*, vol. 44, no. 9, pp. 1140–1144, 1995, ISSN: 0018-9340. DOI: 10.1109/12.464392.

[85] *Industrial communication networks - Profiles - Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3.* [Online]. Available: `https://webstore.iec.ch/publication/5879#additionalinfo`.

[86] *IEEE 61158-2017 - IEEE Standard for Industrial Hard Real-Time Communication*, 2017. DOI: `10.1109/IEEESTD.2017.8024204`.

[87] *ARINC 664 P7 - Aircraft Data Network Part 7 - Avionics Full-Duplex Switched Ethernet*, 2005.

[88] C. Venkatramani and T.-c. Chiueh, "Design and implementation of a real-time switch for segmented ethernets", in *Proceedings 1997 International Conference on Network Protocols*, Oct. 1997, pp. 152–161. DOI: `10.1109/ICNP.1997.643709`.

[89] *The Common Industrial Protocol (CI) and the Family of CIP Network*, 2016. [Online]. Available: `https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R1_Common-Industrial_Protocol_and_Family_of_CIP_Networks.pdf`.

[90] PROFIBUS International), *PROFINET - Real Time Communication.* [Online]. Available: `http://www.profibus.org.pl/index.php?option=com_docman&task=doc_view&gid=28` (visited on 09/11/2018).

[91] *AUTOSAR.* [Online]. Available: `https://www.autosar.org/` (visited on 09/06/2018).

[92] J. Littlefield-Lawwill and R. Viswanathan, "Advancing open standards in integrated modular avionics: An industry analysis", in *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, Oct. 2007, 2.B.1-1-2.B.1-14. DOI: `10.1109/DASC.2007.4391848`.

[93] *IEC 61499.* [Online]. Available: `http://www.iec61499.de/` (visited on 09/06/2018).

[94] "Ieee standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks", *IEEE Std 802.1AS-2011*, pp. 1–292, Mar. 2011. DOI: `10.1109/IEEESTD.2011.5741898`.

[95] *IEEE draft standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications.* [Online]. Available: `https://1.ieee802.org/tsn/802-1as-rev/`.

[96] *IEEE standard for local and metropolitan area networks - virtual bridged local area network - amendment 12: Forwarding and queuing enhancements for time-sensitive streams.* DOI: `10.1109/ieeestd.2009.5375704`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2009.5375704`.

[97] *IEEE standard for local and metropolitan area networks - bridges and bridged networks - amendment 26: Frame preemption.* DOI: `10.1109/ieeestd.2016.7553415`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2016.7553415`.

[98] *IEEE standard for ethernet amendment 5: Specification and management parameters for interspersing express traffic.* DOI: `10.1109/ieeestd.2016.7592835`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2016.7592835`.

[99] *IEEE standard for local and metropolitan area networks - bridges and bridged networks - amendment 25: Enhancements for scheduled traffic.* DOI: `10.1109/ieeestd.2016.7440741`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2016.7440741`.

[100] *IEEE standard for local and metropolitan area networks - bridges and bridged networks - amendment 29: Cyclic queuing and forwarding.* DOI: `10.1109/ieeestd.2017.7961303`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2017.7961303`.

[101] *IEEE p802.1qcr – bridges and bridged networks amendment: Asynchronous traffic shaping.* [Online]. Available: `https://1.ieee802.org/tsn/802-1qcr/`.

[102] *IEEE standard for local and metropolitan area networks–frame replication and elimination for reliability.* DOI: `10.1109/ieeestd.2017.8091139`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2017.8091139`.

[103] *IEEE standard for local and metropolitan area networks– bridges and bridged networks - amendment 24: Path control and reservation.* DOI: `10.1109/ieeestd.2016.7434544`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2016.7434544`.

[104] *IEEE standard for local and metropolitan area networks–bridges and bridged networks–amendment 28: Per-stream filtering and policing.* DOI: `10.1109/ieeestd.2017.8064221`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2017.8064221`.

[105] *IEEE standard for local and metropolitan area networks—virtual bridged local area networks amendment 14: Stream reservation protocol (SRP).* DOI: `10.1109/ieeestd.2010.5594972`. [Online]. Available: `https://doi.org/10.1109%2Fieeestd.2010.5594972`.

[106] *IEEE p802.1qcc – stream reservation protocol (srp) enhancements and performance improvements.* [Online]. Available: `https://1.ieee802.org/tsn/802-1qcc/`.

[107] *IEEE p802.1qcp – bridges and bridged networks amendment: Yang data model.* [Online]. Available: `https://1.ieee802.org/tsn/802-1qcp/`.

[108] *IEEE p802.1cs – link-local registration protocol.* [Online]. Available: `https://1.ieee802.org/tsn/802-1cs/`.

[109] S. S. Craciunas, R. S. Oliver, M. Chmelık, and W. Steiner, "Scheduling real-time communication in ieee 802.1qbv time sensitive networks", in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16, Brest, France: ACM, 2016, pp. 183–192, ISBN: 978-1-4503-4787-7. DOI: `10.1145/2997465.2997470`. [Online]. Available: `http://doi.acm.org/10.1145/2997465.2997470`.

[110] J. Ko, J. Lee, C. Park, and S. Park, "Research on optimal bandwidth allocation for the scheduled traffic in ieee 802.1 avb", in *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Nov. 2015, pp. 31–35. DOI: `10.1109/ICVES.2015.7396889`.

[111] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks", *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 86–94, 2016, ISSN: 2398-3396. DOI: `10.1049/iet-cps.2016.0021`.

[112] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus", *IEEE Access*, vol. 6, pp. 41 803–41 815, 2018, ISSN: 2169-3536. DOI: `10.1109/ACCESS.2018.2858767`.

[113] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications", in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Jun. 2018, pp. 1–4. DOI: `10.1109/WFCS.2018.8402374`.

[114] I. Álvarez, J. Proenza, and M. Barranco, "Mixing time and spatial redundancy over time sensitive networking", in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Jun. 2018, pp. 63–64. DOI: `10.1109/DSN-W.2018.00031`.

[115] I. Álvarez, J. Proenza, M. Barranco, and M. Knezic, "Towards a time redundancy mechanism for critical frames in time-sensitive networking", in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2017, pp. 1–4. DOI: `10.1109/ETFA.2017.8247721`.

[116] P. Pedreiras and A. Luis, "The flexible time-triggered (FTT) paradigm: An approach to QoS management in distributed real-time systems", in *Proceedings International Parallel and Distributed Processing Symposium*, IEEE Comput. Soc. DOI: `10.1109/ipdps.2003.1213243`. [Online]. Available: `https://doi.org/10.1109%2Fipdps.2003.1213243`.

[117] NetFPGA ORG, *Netfpga home page.* [Online]. Available: `http://netfpga.org/site/#/` (visited on 12/06/2017).

[118] M. Ashjaei, P. Pedreiras, M. Behnam, R. J. Bril, L. Almeida, and T. Nolte, "Response time analysis of multi-hop HaRTES ethernet switch networks", in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, IEEE, May 2014. DOI: `10.1109/wfcs.2014.6837579`. [Online]. Available: `https://doi.org/10.1109%2Fwfcs.2014.6837579`.

[119] M. Ashjaei, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte, "Reduced buffering solution for multi-hop HaRTES switched ethernet networks", in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, Aug. 2014. DOI: `10.1109/rtcsa.2014.6910504`. [Online]. Available: `https://doi.org/10.1109%2Frtcsa.2014.6910504`.

[120] G. Rodriguez-Navas and J. Proenza, “A proposal for flexible, real-time and consistent multicast in FTT/HaRTES switched ethernet”, in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, Sep. 2013. DOI: 10.1109/etfa.2013.6648153. [Online]. Available: https://doi.org/10.1109%2Fetfa.2013.6648153.

[121] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, “Towards preventing error propagation in a real-time ethernet switch”, in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, Sep. 2013. DOI: 10.1109/etfa.2013.6648140. [Online]. Available: https://doi.org/10.1109%2Fetfa.2013.6648140.

[122] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, “Towards a flexible time-triggered replicated star for ethernet”, in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, Sep. 2013. DOI: 10.1109/etfa.2013.6648137. [Online]. Available: https://doi.org/10.1109%2Fetfa.2013.6648137.

[123] *Ryu sdn framework*. [Online]. Available: https://osrg.github.io/ryu/ (visited on 10/13/2018).

[124] *Networkx : Software for complex networks*. [Online]. Available: https://networkx.github.io/ (visited on 10/16/2018).

[125] *Openflow software switch 1.3*. [Online]. Available: http://cpqd.github.io/ofsoftswitch13/ (visited on 10/13/2018).

[126] C. Liu, F. Li, G. Chen, and X. Huang, “TTEthernet Transmission in Software-Defined Distributed Robot Intelligent Control System”, *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–13, Jul. 2018. DOI: 10.1155/2018/8589343.

[127] J. Farkas, *Introduction to IEEE 802.1 : Focus on the time-sensitive networking task group*. [Online]. Available: http://www.ieee802.org/1/files/public/docs2017/tsn-farkas-intro-0517-v01.pdf (visited on 11/22/2017).

[128] P. A. Ribeiro, L. Duoba, R. Prior, S. Crisostomo, and L. Almeida, “Real-Time Wireless Data Plane for Real-Time-Enabled SDN”, in *2019 IEEE World Conference on Factory Communication Systems (WFCS)*, May 2019, pp. 1–4. DOI: 10.1109/WFCS.2019.8757951.

APPENDIX A

List of publications and communications

A.1 Core publications

1. L. Silva, P. Pedreiras, P. Fonseca, *et al.*, “On the adequacy of SDN and TSN for industry 4.0”, in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, May 2019. DOI: `10.1109/isorc.2019.00017`. [Online]. Available: `https://doi.org/10.1109%2Fisorc.2019.00017`
2. L. Silva, P. Goncalves, R. Marau, *et al.*, “Extending OpenFlow with flexible time-triggered real-time communication services”, in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, Sep. 2017. DOI: `10.1109/etfa.2017.8247595`. [Online]. Available: `https://doi.org/10.1109%2Fetfa.2017.8247595`
3. L. Silva, P. Goncalves, R. Marau, *et al.*, “Extending OpenFlow with industrial grade communication services”, in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, IEEE, May 2017. DOI: `10.1109/wfcs.2017.7991965`. [Online]. Available: `https://doi.org/10.1109%2Fwfcs.2017.7991965`
4. M. Ashjaei, L. Silva, M. Behnam, *et al.*, “Improved message forwarding for multi-hop HaRTES real-time ethernet networks”, *Journal of Signal Processing Systems*, vol. 84, no. 1, pp. 47–67, May 2015. DOI: `10.1007/s11265-015-1010-8`. [Online]. Available: `https://doi.org/10.1007%5C%2Fs11265-015-1010-8`
5. L. Silva, P. Pedreiras, M. Ashjaei, *et al.*, “Demonstrating the multi-hop capabilities of the hartes real-time ethernet switch”, in *RTSS@Work 2014 Open Demo Session of RealTime Systems*, 2014

A.2 Other publications

1. J. Rufino, L. Silva, B. Fernandes, *et al.*, “Overhead of v2x secured messages: An analysis”, in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, IEEE, Apr. 2019. DOI: `10.1109/vtcspring.2019.8746479`. [Online]. Available: `https://doi.org/10.1109%2Fvtcspring.2019.8746479`
2. J. Rufino, L. Silva, B. Fernandes, *et al.*, “Empowering vulnerable road users in c-ITS”, in *2018 IEEE Globecom Workshops (GC Wkshps)*, IEEE, Dec. 2018. DOI: `10.1109/`

glocomw.2018.8644266. [Online]. Available: https://doi.org/10.1109%2Fglocomw.2018.8644266

3. J. Ferreira, M. Alam, B. Fernandes, *et al.*, “Cooperative sensing for improved traffic efficiency: The highway field trial”, *Computer Networks*, vol. 143, pp. 82–97, Oct. 2018. DOI: 10.1016/j.comnet.2018.07.006. [Online]. Available: https://doi.org/10.1016%2Fj.comnet.2018.07.006

4. D. Duarte, L. Silva, B. Fernandes, *et al.*, “Implementation of security services for vehicular communications”, in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Nature, 2017, pp. 79–90. DOI: 10.1007/978-3-319-51207-5_8. [Online]. Available: https://doi.org/10.1007%2F978-3-319-51207-5_8

5. L. Silva, P. Pedreiras, L. Almeida, *et al.*, “Combining spatial and temporal dynamic scheduling techniques on wireless vehicular communications”, in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, Institute of Electrical and Electronics Engineers (IEEE), May 2016. DOI: 10.1109/wfcs.2016.7496532. [Online]. Available: https://doi.org/10.1109%2Fwfcs.2016.7496532

6. L. Silva, P. Pedreiras, M. Alam, *et al.*, “STDMA-based scheduling algorithm for infrastructured vehicular networks”, in *Intelligent Transportation Systems*, Springer Nature, 2016, pp. 81–105. DOI: 10.1007/978-3-319-28183-4_4. [Online]. Available: https://doi.org/10.1007%2F978-3-319-28183-4_4

7. J. Blancou, J. Almeida, B. Fernandes, *et al.*, “eCall++: An enhanced emergency call system for improved road safety”, in *2016 IEEE Vehicular Networking Conference (VNC)*, Institute of Electrical and Electronics Engineers (IEEE), Dec. 2016. DOI: 10.1109/vnc.2016.7835964. [Online]. Available: https://doi.org/10.1109%2Fvnc.2016.7835964

8. M. Alam, B. Fernandes, L. Silva, *et al.*, “Implementation and analysis of traffic safety protocols based on ETSI standard”, in *2015 IEEE Vehicular Networking Conference (VNC)*, Institute of Electrical and Electronics Engineers (IEEE), Dec. 2015. DOI: 10.1109/vnc.2015.7385561. [Online]. Available: https://doi.org/10.1109%2Fvnc.2015.7385561

9. L. Almeida, Z. Iqbal, P. Pedreiras, *et al.*, “Developments in flexible time-triggered switched ethernet”, in *Workshop on Real-Time Ethernet (RATE 2013) in conjunction with RTSS 2013*, 2013

10. P. Silva, L. Silva, R. Marau, *et al.*, “Demonstrating real-time reconfiguration of video sensing service-oriented applications”, in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, Institute of Electrical and Electronics Engineers (IEEE), May 2012. DOI: 10.1109/dcoss.2012.59. [Online]. Available: https://doi.org/10.1109%2Fdcoss.2012.59

11. M. de Sousa, L. Silva, R. Marau, *et al.*, “A real-time resource manager for linux-based distributed systems”, in *Proceedings of the WIP Session of the 32nd IEEE Real-Time Systems Symposium (RTSS'2011)*, 2011. [Online]. Available: http://hdl.handle.net/10216/94788

12. P. Silva, L. Silva, R. Marau, *et al.*, “Demonstrating real-time reconfiguration in service oriented distributed systems”, in *32nd IEEE Real-Time Systems Symposium (RTSS'2011@Work Session)*, 2011

13. L. Silva, A. Oliveira, P. Pedreiras, *et al.*, “Ligação de Alto Desempenho entre FPGAs para Switch Ethernet FTT”, in *VII Jornadas sobre Sistemas Reconfiguráveis*, 2011

APPENDIX B

Code sample for the Real-Time OpenFlow Add-On API

```c
#ifndef OPENFLOW_PRIVATE_EXT_H_
#define OPENFLOW_PRIVATE_EXT_H_

#ifdef __KERNEL__
#include <asm/byteorder.h>
#endif

#include "openflow/openflow.h"
#include "openflow-ext.h"

#define PRIVATE_VENDOR_ID    0x00acde48
#define MAX_STREAM_NODES     10   /* Max no producer/consumer nodes per stream */
#define MAX_MP_STAT_STREAMS 5    /* Max no streams per MP reply fragment */

enum rt_openflow_type
{
    RT_ST_ADD = 0,              /* Add a real-time stream to the database */
    RT_ST_MODIFY,               /* Modify properties of an existing stream */
    RT_ST_DELETE,               /* Remove a real-time stream from the database */
    RT_ST_LIST_REQUEST,         /* Get the ID of all streams of a given type */
    RT_ST_LIST_REPLY,           /* Switch reply to RT_ST_LIST_REQUEST */
    RT_ST_PROP_REQUEST,         /* Get properties of a given real-time stream */
    RT_ST_PROP_REPLY,           /* Switch reply to RT_ST_PROP_REQUEST */
    RT_ST_REMOVED,              /* Switch notification of stream removal events */
    RT_ST_STATS_REQUEST,        /* Get properties and statistics of all streams */
    RT_ST_STATS_REPLY           /* Switch reply to RT_ST_STATS_REQUEST */
};

enum rt_openflow_stream_type
{
    RT_TYPE_TT = 1,             /* Time-triggered stream type */
    RT_TYPE_ET,                 /* Event-triggered stream type */
    RT_TYPE_OTHER,              /* Other stream types */
};
#define RT_LIST_ALL 0x00

struct rtof_stream_mod_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID,
                                            subtype = RT_ST_{ADD,MODIFY,DELETE} */

    uint32_t uid;               /* Unique stream identifier */
    uint32_t period;            /* For RT_TYPE_TT: periodicity of messages [us]
```

```
                                    For RT_TYPE_ET: minimum inter-transmission time
                                    between consecutive messages [us]*/

    uint32_t deadline;          /* Message deadline. [us] */
    uint32_t offset;            /* Relative phasing. Ignored for RT_TYPE_ET/OTHER
                                    traffic. [us] */

    uint32_t idle_timeout;      /* Stream's idle timeout before it and its associated
                                    reservations are removed [s]. If 0, stream is
                                    never removed.*/
    uint32_t hard_timeout;      /* Maximum duration before stream and its associated
                                    reservations are removed [s]. If 0, stream is never
                                    removed.*/

    uint32_t tx_time;           /* [DEPRECATED] */

    uint16_t priority;          /* Stream's priority (higher value,higher priority)*/
    uint16_t frame_length;      /* Maximum frame's length [Bytes]*/

    uint16_t n_producers;       /* Producers list size (num elements) */
    uint16_t n_consumers;       /* Consumers list size (num elements) */
    uint8_t type;               /* One of RT_TYPE_* */
    uint8_t pad[3];             /* Align to 64 bits */

    uint8_t producers[0];       /* List of stream producers */
    uint8_t consumers[0];       /* List of stream consumers */
    uint8_t misc[0];            /* Miscellaneous field */
};
OFP_ASSERT(sizeof(struct rtof_stream_mod_msg) == 56);

struct rtof_stream_prop_req_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID,
                                             subtype = RT_GET_ST_PROP_REQUEST */

    uint32_t stream_uid;                /* Unique stream identifier */
    uint8_t pad[4];                     /* Align to 64 bits */
};
OFP_ASSERT(sizeof(struct rtof_stream_prop_req_msg) == 24);

struct rtof_stream_prop_reply_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID,
                                             subtype = RT_GET_ST_PROP_REPLY */
    uint32_t uid;
    uint32_t period;

    uint32_t deadline;
    uint32_t offset;

    uint32_t idle_timeout;
    uint32_t hard_timeout;

    uint32_t tx_time;           /* [DEPRECATED] */
    uint16_t priority;
    uint16_t frame_length;

    uint16_t n_producers;
    uint16_t n_consumers;
    uint8_t stream_type;
    uint8_t pad[3];

    uint8_t producers[0];
    uint8_t consumers[0];
    uint8_t misc[0];
};
OFP_ASSERT(sizeof(struct rtof_stream_prop_reply_msg) == 56);

struct rtof_stream_list_req_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID,
                                            subtype = RT_GET_ST_LIST_REQUEST */

    uint8_t streams_type;       /* Type of requested streams. One of RT_TYPE_*.
                                   Can also be RT_LIST_ALL for requests for a list
                                   with all streams, independently of their type */
    uint8_t pad[7];             /* Align to 64 bits */
};
OFP_ASSERT(sizeof(struct rtof_stream_list_req_msg) == 24);

struct rtof_stream_list_reply_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID
                                            subtype = RT_GET_ST_LIST_REPLY */

    uint16_t list_size;                  /* Number of stream UIDs in list */
    uint8_t streams_type;                /* Type of streams contained in list.
                                            One of RT_TYPE_*. Can also be RT_LIST_ALL
                                            for lists with all streams,
                                            independently of their type */
    uint8_t pad[5];                      /* Align to 64 bits */

    uint32_t uid_list[0];                 /* List of stream UIDs */
};
OFP_ASSERT(sizeof(struct rtof_stream_list_reply_msg) == 24);

struct rtof_stream_removed_msg
{
    struct ofp_extension_header  header; /* vendor = PRIVATE_VENDOR_ID,
                                            subtype = RT_ST_REMOVED */

    uint32_t    stream_uid;                 /* UID of the removed real-time stream */
    uint8_t     reason;                     /* Reason for removal (OFPRR_IDLE_TIMEOUT,
                                               OFPRR_HARD_TIMEOUT, OFPRR_DELETE,
                                               OFPRR_GROUP_DELETE) */
    uint8_t     pad[3];                     /* Align to 64 bits */
};


struct rtof_stream_stats
{
    uint64_t packet_count;      /* Number of packets in stream counter*/

    uint64_t byte_count;        /* Number of bytes in stream counter */

    uint32_t duration_sec;      /* Time stream has been alive [s] */
    uint32_t duration_nsec;     /* Time stream has benn alive [ns]*/

    uint32_t uid;
    uint32_t tx_time;           /* [DEPRECATED] */

    uint32_t period;
    uint32_t deadline;

    uint32_t offset;
    uint32_t idle_timeout;

    uint32_t hard_timeout;
    uint16_t priority;
    uint16_t frame_length;

    uint16_t n_producers;
    uint16_t n_consumers;
    uint8_t type;
    uint8_t pad[3];

    uint8_t producers[0];
    uint8_t consumers[0];
    uint8_t misc[0];
};
OFP_ASSERT(sizeof(struct rtof_stream_mod_msg) == 56);

struct rtof_stream_stats_mp_reply_msg
{
    struct ofp_multipart_reply  header;
    struct ofp_experimenter_multipart_header
            exp_header;         /* experimenter = PRIVATE_VENDOR_ID,
                                   type= RT_ST_STATS_REPLY*/

    uint16_t list_size;         /* Number of elements in list */
    uint8_t  pad[6];            /* Align to 64 bits */

    uint8_t stats_list[0];      /* List of rtof_stream_stats */
};
OFP_ASSERT(sizeof(struct rtof_stream_stats_mp_reply_msg) == 32);

#endif
```

./code/private–ext.h

APPENDIX C

Mandatory and optional OpenFlow components/functions

Table C.1: OpenFlow Objects

Object	Description	Mandatory	Min. N^{o}
OpenFlow Channel	Communication channel with one controller.	✓	1
Ingress Flow Table	Table for ingress frames (1st stage).	✓	1
Egress Flow Table	Table for the 2nd stage (after output/group actions).	✗	- - -
Group Table	Group buckets. For processing groups of flows.	✓	1
Meter Table	Contains entries for QoS shapers.	✗	- - -

Table C.2: OpenFlow Ports

OpenFlow Ports	Description	Mandatory	Min. N^{o}
Physical ports	Hardware interfaces.	✓	- - -
Logical ports	Software interfaces for non-OpenFlow processing.	✗	- - -
ALL port	Reserved. Clones frames for all ports but frame's ingress.	✓	- - -
CONTROLLER port	Reserved. For packet-in/packet-out operations.	✓	- - -
TABLE port	Reserved. For packet-out, sends frame to 1st flow table.	✓	- - -
IN_PORT port	Reserved. Sends frame through its ingress port.	✓	- - -
ANY port	Reserved. Special value for multi-addressed requests.	✓	- - -
UNSET port	Reserved. Indicates an output port not set in action-set.	✓	- - -
LOCAL port	Reserved. Switch internal networking stack.	✗	- - -
NORMAL port	Reserved. Normal forwarding with non-OFP methods.	✗	- - -
FLOOD port	Reserved. Flooding with non-OFP methods.	✗	- - -

Table C.3: Flow Instructions

Instruction	Description	Mandatory
Apply-Actions	Apply specified actions immediately without changes to action-set.	✗
Clear-Actions	Clear all actions in the action-set immediately.	✗
Write-Actions	Merge specified actions into the current frame's action-set.	✓
Write-Metadata	Write the metadata value into the metadata field.	✗
Stat-Trigger	Send event to controller if flow statistics cross the threshold.	✗
Goto-Table	Send frame to a certain table in the pipeline.	✓

Table C.4: Flow Match Fields

Field	Description	Mandatory
OXM_OF_IN_PORT	Ingress port, physical or logical.	✓
*ACTSET_OUTPUT	Egress port from action set.	✓
*ETH_DST	Ethernet destination address. Can use arbitrary bitmask.	✓
*ETH_SRC	Ethernet source address. Can use arbitrary bitmask.	✓
*ETH_TYPE	Ethernet type.	✓
*IP_PROTO	IPv4 or IPv6 protocol number.	✓
*IPV4_SRC	IPv4 source address. Can use arbitrary bitmask.	✓
*IPV4_DST	IPv4 destination address. Can use arbitrary bitmask.	✓
*IPV6_SRC	SIPv6 source address. Can use arbitrary bitmask.	✓
*IPV6_DST	IPv6 destination address. Can use arbitrary bitmask.	✓
*TCP_SRC	TCP source port.	✓
*TCP_DST	TCP destination port.	✓
*UDP_SRC	UDP source port.	✓
*UDP_DST	UDP destination port.	✓
*METADATA	Table metadata.	✗
*TUNNEL_ID	Metadata associated with a logical port.	✗
*PACKET_TYPE	Canonical header type of outermost header.	✗
*VLAN_VID	VLAN ID from 802.1Q tag.	✗
*VLAN_PCP	VLAN PCP from 802.1Q tag.	✗
*IP_DSCP	Diff Serv Code Point (DSCP).	✗
*IP_ECN	ECN bits of the IP header.	✗
*TCP_FLAGS	TCP flags.	✗
*SCTP_SRC	SCTP source port.	✗
*SCTP_DST	SCTP destination port.	✗
*ICMPV4_TYPE	ICMP type.	✗
*ICMPV4_CODE	ICMP code.	✗
*ARP_OP	ARP opcode.	✗
*ARP_SPA	Source IPv4 address in ARP payload.	✗
*ARP_TPA	Target IPv4 address in ARP payload.	✗
*ARP_SHA	Source Ethernet address in ARP payload.	✗
*ARP_THA	Target Ethernet address in ARP payload.	✗
*IPV6_FLABEL	IPv6 flow label.	✗
*ICMPV6_TYPE	ICMPv6 type.	✗
*ICMPV6_CODE	ICMPv6 code.	✗
*IPV6_ND_TARGET	Target address in an IPv6 Neighbor Discover (ND) message.	✗
*IPV6_ND_SLL	Source link-layer address in IPv6 ND message.	✗
*IPV6_ND_TLL	Target link-layer address in IPv6 ND message.	✗
*MPLS_LABEL	Label in the first MPLS shim header.	✗
*MPLS_TC	Traffic class in the first MPLS shim header.	✗
*MPLS_BOS	Bottom of stack bit in the first MPLS shim header.	✗
*PBB_ISID	I-SID in the first PBB service instance tag.	✗
*IPV6_EXTHDR	IPv6 extension header.	✗
*PBB_UCA	UCA field in the first PBB service instance tag.	✗

Table C.5: Flow Group Types

Group Type	Description	Mandatory
Indirect	Execute the single, defined bucket in this group.	✓
All	Execute all buckets in the group.	✓
Select	Execute the selected bucket in the group.	✗
Fast Failover	Execute first live bucket.	✗

Table C.6: Flow Actions

Action	Description	Mandatory
Output	Forward frame to specified OpenFlow port.	✓
Group	Process frame through the specified group.	✓
Drop	Frames without output/group actions are dropped.	✓
Set-Queue	Set queue id for a frame.	✗
Meter	Direct frame to specified meter.	✗
Push-Tag	Push tags, e.g. VLAN, MPLS, into frame.	✗
Pop-Tag	Pop tags, e.g. VLAN, MPLS, from frame.	✗
Set-Field	Modify values of respective fields in frame.	✗
Copy-Field	Copy data between any fields in frame.	✗
Change-TTL	Modify values of IPv4/MPLS TTL, IPv6 hop limit in frame.	✗

Table C.7: Flow Meter Bands

Band	Description	Mandatory
Drop	Drop frame if threshold reached.	✗
DSCP Remark	Increase drop precedente of DSCP field if threshold reached.	✗

Table C.8: Flow Table Counters (per table)

Counter	Description	Mandatory	Min. N^o
Reference Count	No. of times table was referenced by goto-table action.	✓	1
Packet Lookups	No. of frames processed by flow table.	✗	1
Packet Matches	No. of frames that matched entries of flow table.	✗	1

Table C.9: Flow Entry Counters (per entry)

Counter	Description	Mandatory	Min. N^o
Rx Packets	No. of frames that matched entry.	✗	- - -
Rx Bytes	No. of total bytes that matched entry.	✗	- - -
Duration (s)	Amount of time entry has been installed.	✓	1
Duration (ns)	Amount of time entry has been installed.	✗	- - -

Table C.10: Port Counters (per port)

Counter	Description	Mandatory	Min. N^o
Rx Packets	No. of frames received by port.	✓	1
Tx Packets	No. of frames transmitted through port.	✓	1
Rx Bytes	No. of total bytes received by port.	✗	- - -
Tx Bytes	No. of total bytes transmitted through port.	✗	- - -
Rx Drops	No. of frames dropped at reception by port.	✗	- - -
Tx Drops	No. of frames dropped at transmission by port.	✗	- - -
Rx Errors	No. of errors while receiving frames.	✗	- - -
Tx Errors	No. of errors while transmitting frames.	✗	- - -
Rx Align Errors	No. of alignment errors while receiving frames.	✗	- - -
Rx Overrun Errors	No. of overrun errors while receiving frames.	✗	- - -
Rx CRC Errors	No. of CRC errors in received frames.	✗	- - -
Collision	No. of collisions at port.	✗	- - -
Duration (s)	Amount of time port has been alive.	✓	1
Duration (ns)	Amount of time port has been alive.	✗	- - -

Table C.11: Queue Counters (per queue)

Counter	Description	Mandatory	Min. N°
Tx Packets	No. of frames processed by queue.	✓	1
Tx Bytes	No. of total bytes processed by queue.	✗	- - -
Tx Overrun Errors	No. of overruns in transmissions processed by queue.	✗	- - -
Duration (s)	Amount of time queue has been alive.	✓	1
Duration (ns)	Amount of time queue has been alive.	✗	- - -

Table C.12: Group Counters (per group)

Counter	Description	Mandatory	Min. N°
Reference Count	No. of group references by group actions.	✗	- - -
Packet Count	No. of frames processed by group.	✗	- - -
Byte Count	No. of total bytes processed by group.	✗	- - -
Duration (s)	Amount of time group has been installed.	✓	1
Duration (ns)	Amount of time group has been installed.	✗	- - -

Table C.13: Group Bucket Counters (per bucket)

Counter	Description	Mandatory	Min. N°
Packet Count	No. of frames processed by group bucket.	✗	- - -
Byte Count	No. of total bytes processed by group bucket.	✗	- - -

Table C.14: Meter Counters (per meter)

Counter	Description	Mandatory	Min. N°
Flow Count	No. of flows references by meter actions.	✗	- - -
Input Packet Count	No. of frames processed by meter.	✗	- - -
Input Byte Count	No. of total bytes processed by meter.	✗	- - -
Duration (s)	Amount of time meter has been installed.	✓	1
Duration (ns)	Amount of time meter has been installed.	✗	- - -

Table C.15: Meter Band Counters (per meter band)

Counter	Description	Mandatory	Min. N°
Packet Count	No. of frames that reached band threshold.	✗	- - -
Byte Count	No. of total bytes that reached band threshold.	✗	- - -

APPENDIX D

Code sample for the schedulability analysis of time-triggered traffic

```python
import rtof.api
import rtof.topology
import rtof.srdb
import rtof.utils
import rtof.devdb
import math

class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class AlphaError(Error):
    def __init__(self, message):
        self.message = message

class InvalidParamError(Error):
    """Exception raised for errors in function's parameters
    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """
    def __init__(self, expression, message):
        self.message = expression
        self.message = message

def WorstResponseTimeCalc(srdb, topology, stream):

    # Validate stream's producers/consumers
    if (stream.consumers is False) or (stream.producers is False):
        print('!!! (Admission Control) : Invalid producers/consumers')
        raise InvalidParamError("aaa", "bbb")

    # Get stream's path link set
    prod = stream.producers[0]
    cons = stream.consumers[0]

    # Get network path for stream
    prod_id = topology.getDevNodeID(prod.id, prod.port)
    cons_id = topology.getDevNodeID(cons.id, cons.port)
    path = topology.getPath(prod_id, cons_id)
    if path is None:
        print('!!! (Admission Control) : No path found for requested real-time stream
    uid {}'.format(stream.uid))
```

```
        return

    # Retrieve link set for this stream
    links = topology.pathToLinkList(path)

    # Calculate worst-case response time for messages of this stream
    if stream.type == rtof.api.RTOF_ST_TYPES.RT_TYPE_TT:
        wrt = _calcTimeTriggeredWrt(srdb, topology, stream, links)
    elif stream.type == rtof.api.RTOF_ST_TYPES.RT_TYPE_ET:
        wrt = _calcEventTriggeredWrt(srdb, topology. stream, links)
    else:
        print('!!! (Admission Control) Failed to calculate wrt for stream uid {} >
    Type {:5s} not supported <'.format(stream.uid, stream.type))
        return

    return wrt

# Calculates the worst response time for messages of stream uid
def _calcTimeTriggeredWrt(srdb, topology, stream, link_set):

    rt = 0
    a = b = 0
    rt_a_b_prev = 0

    # Get essential network related information for the analysis
    # Get all TT streams in system
    all_set = srdb.getSameTypeStreamList(rtof.api.RTOF_ST_TYPES.RT_TYPE_TT)

    # Get TT streams with higher or equal priority than stream
    hep_set = srdb.getHepStreamList(stream.priority, rtof.api.RTOF_ST_TYPES.RT_TYPE_TT
    )

    # Get TT streams with lower priority streams
    lp_set = srdb.getLpStreamList(stream.priority, rtof.api.RTOF_ST_TYPES.RT_TYPE_TT)

    while b + 1 <= len(link_set):
        links = link_set[a:b + 1]
        rt_a_b = _ResponseTimeCalc(stream, links, hep_set, lp_set, all_set)
        rt_a_b = math.ceil(rt_a_b/rtof.devdb.NETWORK_EC_US)
        if (a != b) and (rt_a_b != rt_a_b_prev):
            rt += rt_a_b_prev
            a = b
        else:
            b += 1
        rt_a_b_prev = rt_a_b

    rt += rt_a_b_prev
    return rt

def _ResponseTimeCalc(stream, links, hep_set, lp_set, all_set):

    if stream.type == rtof.api.RTOF_ST_TYPES.RT_TYPE_TT:
            alpha = _calcInflationFactor(stream, links, hep_set, 'TTW')
    elif stream.type == rtof.api.RTOF_ST_TYPES.RT_TYPE_ET:
            alpha = _calcInflationFactor(stream, links, hep_set, 'ETW')
    else:
        # What to do?
        alpha = 1
    # Calculate first response time iteration iteration rt(0)
    speed = links[0].speed # We only support networks where all links are at the same
    speed
    c_i = rtof.utils.calcTxTimeMsgUs(stream.data_length, speed)
    prev_rt = inflated_ci = c_i / alpha

    rt = 0
    while (True):
        # Calculate further response time iterations until they converge ( rt(n-1) ==
    rt(n) )
        interference  = _calcTTInterference(links, hep_set, alpha, prev_rt)
        blocking      = _calcTTBlocking(links, lp_set, alpha)
        swd           = _calcTTSwd(all_set, stream, links, alpha)

        rt = inflated_ci + interference + blocking + swd

        if rt == prev_rt:
            break
        else:
            prev_rt = rt

    return rt

def _calcTTInterference(stream_links, hep_set, inflation_fact, prev_rt):

    interference_us = 0
    # For each stream in the higher or equal priority set 'hep_set', which has links
    in common with 'stream_links',
    # compute its contribute to the interference
    for st in hep_set:
        st_link_list = list(st.link_set.values())

        if rtof.utils.isListIntersectionNull(stream_links, st_link_list) is False:
            # NOTE: stream's period in RTOF API is defined as [us] while HARTES'
    streams' periods are usually expressed in multiples of ECs
            c_j = rtof.utils.calcTxTimeMsgUs(st.data_length, stream_links[0].speed) #
    We only support networks where all links are at the same speed
            interference_us += math.ceil(prev_rt/st.period)*(c_j/inflation_fact)

    return interference_us

def _calcTTBlocking(stream_links, lp_set, inflation_fact):
    blocking = 0

    # Blocking does not happen at stream's input link and when a==b
    if len(stream_links) == 1:
        # Iteration checking only the input link -> No blocking exists
        pass

    else:
        # Discard input link and check blocking for the remaining links
        links = stream_links[1:]
        prev_links = []
        for l in links:
            max_msg_length = 0
            for st in lp_set:
                # Check if link is in stream's link set
                if rtof.utils.isLinkInLinkSet(l, st.link_set) is False:
                    continue
```

```python

                # Check if previous analysed links are in stream's link set (msg can only block at a certain link and not at the following ones)
                st_link_list = list(st.link_set.values())
                if rtof.utils.isListIntersectionNull(prev_links, st_link_list) is False:
                    continue

                # Check if stream's data length is larger than the current maximum
                if st.data_length > max_msg_length:
                    max_msg_length = st.data_length


            if max_msg_length > 0:
                blocking += rtof.utils.calcTxTimeMsgUs(max_msg_length, stream_links[0].speed)/inflation_fact# We only support networks where all links are at the same speed

            # Store analysed link in analysed links
            prev_links.append(l)

    return blocking

def _calcTTSwd(streams_set, stream, stream_links, inflation_fact):
    swd = 0

    # Blocking does not happen at stream's input link and when a==b
    if len(stream_links) == 1:
        # Iteration checking only the input link -> No blocking exists
        pass

    else:
        # Discard input link and check switching delay for the remaining links
        links = stream_links[1:]
        prev_link = rtof.topology.Link(id=-1) #Dummy link that no stream has for sure

        for l in links:
            max_msg_length = stream.data_length
            for st in streams_set:
                # Check if link is in stream's link set
                if rtof.utils.isLinkInLinkSet(l, st.link_set) is False:
                    continue

                # Check if previous link is in stream's link set
                if rtof.utils.isLinkInLinkSet(prev_link, st.link_set) is False:
                    continue

                # Check if stream's data length is larger than the current maximum
                if st.data_length > max_msg_length:
                    max_msg_length = st.data_length

            swd += rtof.devdb.SWITCHING_FABRIC_DELAY_US + rtof.utils.calcTxTimeMsgUs(max_msg_length, stream_links[0].speed)/inflation_fact

            # Store analysed link in analysed links
            prev_link = l

    return swd
```

```python
def _calcInflationFactor(stream, links, hep_set, window):

    ec_us = 0
    min_lw_id_us = 0xffffffff

    # For each link that stream goes through, get the minimum TT window length and maximum message size of interfering messages
    for l in links:
        max_msg_len = stream.data_length
        # Get EC and window length from link. Interlinks have same properties on each port so we need to check only one
        if l.src is not None:
            lw_us = rtof.devdb.hartes_sw_prop[l.src.device.dp.id][l.src.port_no][window]
            ec_us = rtof.devdb.hartes_sw_prop[l.src.device.dp.id][l.src.port_no]['EC']

        else:
            lw_us = rtof.devdb.hartes_sw_prop[l.dst.device.dp.id][l.dst.port_no][window]
            ec_us = rtof.devdb.hartes_sw_prop[l.dst.device.dp.id][l.dst.port_no]['EC']

        # Get maximum sized message among the highest and same priority tt messages that share this link with the new stream
        for st in hep_set:

            if st.link_set.get(l.id, None) is not None:
                if st.data_length > max_msg_len:
                    max_msg_len = st.data_length

        lw_id_us = lw_us-rtof.utils.calcTxTimeMsgUs(max_msg_len, l.speed)

        if lw_id_us < min_lw_id_us:
            min_lw_id_us = lw_id_us

    alpha = min_lw_id_us/ec_us

    if alpha <= 0:
        raise AlphaError("Inflation factor calculation returned invalid value ({})".format(alpha))

    return alpha

def _calcEventTriggeredWrt(srdb, topology, stream, link_set):
    pass
```

./code/ac.py