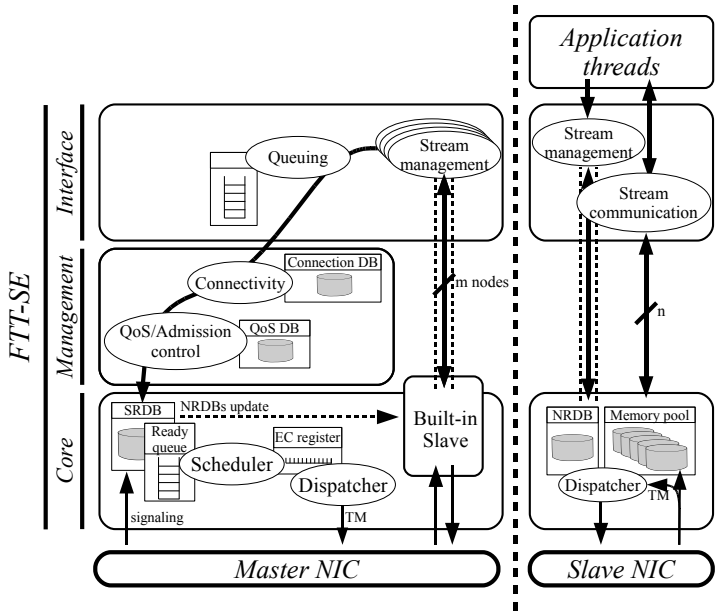
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2009

Ricardo Roberto Duarte Marau

Comunicações de tempo-real em switched Ethernet suportando gestão dinâmica de QdS

Real-time communications over switched Ethernet supporting dynamic QoS management



Universidade de Aveiro
2009

Departamento de Electrónica, Telecomunicações e Informática

Ricardo Roberto Duarte Marau

Comunicações de tempo-real em switched Ethernet suportando gestão dinâmica de QdS

Real-time communications over switched Ethernet supporting dynamic QoS management

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e
2009 Informática

Ricardo Roberto Duarte Marau

Comunicações de tempo-real em switched Ethernet suportando gestão dinâmica de QdS

Real-time communications over switched Ethernet supporting dynamic QoS management

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Informática, realizada sob a orientação científica do Prof. Doutor Luís Miguel Pinho de Almeida, Professor Associado da Faculdade de Engenharia da Universidade do Porto e co-orientação do Prof. Doutor Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Este trabalho foi apoiado por:

Ministério da Ciência e do Ensino Superior, por meio da Fundação para a Ciência e a Tecnologia, que me concedeu uma bolsa de Doutoramento no âmbito do III Quadro Comunitário de Apoio, (SFRH /BD/25261/2005).

Universidade de Aveiro, que me concedeu uma bolsa de Doutoramento de Janeiro a Dezembro de 2005 e proporcionou as condições logisticas, técnicas e humanas para a prossecução dos trabalhos realizados no âmbito desta tese.

Instituto de Engenharia Electrónica e Telemática de Aveiro, que apoiou financeiramente a minha participação em conferências internacionais para apresentação de resultados parciais obtidos no âmbito desta tese.

Projectos europeus ARTIST2 (IST-2004-004527) e ArtistDesign (ICT-NoE-214373), no âmbito dos quais se realizou parte dos trabalhos desta tese.

Dedicatória

aos meus pais
e à minha irmã.

o júri / the jury

presidente / president	**Doutor Amadeu Mortágua Velho da Maia Soares** Professor Catedrático da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)
vogais / examiners committee	**Doutor Raj Rajkumar** Full Professor da Carnegie Mellon University, Estados Unidos da América
	Doutora Maria de la Soledad Garcia Valls Associate Professor da Universidad Carlos III de Madrid
	Doutor Luís Miguel Pinho de Almeida Professor Associado da Faculdade de Engenharia da Universidade do Porto (Orientador)
	Doutor José Alberto Gouveia Fonseca Professor Associado da Universidade de Aveiro
	Doutor Paulo Bacelar Reis Pedreiras Professor Auxiliar da Universidade de Aveiro (Co-Orientador)

Agradecimentos /
Acknowledgements

A jornada que culminou, entre outras coisas, nesta dissertação começou há cinco anos em Aveiro. Durante este período percorri uma viagem de enriquecimento pessoal que me permitiu adquirir competências a vários níveis bem como conhecer e interagir com pessoas maravilhosas que me ajudaram e apoiaram em várias etapas da viagem. Como em qualquer viagem, houve momentos bons e momentos menos bons, penosos e mesmo desmotivantes. Mas quando eis chegado o momento de parar, respirar, olhar para trás e fazer o balanço, apenas me ocorre: valeu a pena. A todas as pessoas que directa ou directamente se envolveram nesta viagem, expresso o meu profundo e sincero agradecimento. Contudo, dada a sua relevância, gostaria de agradecer em particular:

I would like to thank:

a Luís de Almeida, por me ter proposto e convencido a iniciar a viagem; pela orientação exemplar a nível técnico, ciêntifico e pedagógico; pela enorme generosidade e paciência que sempre demonstrou ao longo destes cinco anos; pelas acesas discussões técnicas que se alongavam pela noite e que, muitas vezes, se misturavam com umas cervejas no Ramona.

a Paulo Pedreiras, por ter acompanhado os meus trabalhos durante estes anos como orientador, com uma competência técnico-científica irrepreensível; por tê-lo feito de uma forma bastante próxima, atenta e humana.

a ambos pelo empenho e dedicação profissional, bom como pelas condições e meios oferecidos para desenvolver o meu trabalho; pelos bons momentos de amizade, convívio e entre-ajuda, o meu sincero e sentido agradecimento. Obrigado pela inspiração! Gostaria ainda de manifestar o meu apreço à Nani e à Cristina, à Sofia, Joana, Pedro e Francisco, pela simpatia e compreensão.

to Michael González Harbour, to Daniel Sangorrín and to Julio L. Medina, from the University of Cantabria, for the support and collaboration in part of my work. Also, for having received me in Santander in a visit that although short, happened in an early stage and was very helpful to clarify much of the purpose of my work. "¡Gracias!"

to Javier Silvestre, from the Polytechnic University of Valencia (Alcoy), for its valuable collaboration and skills in Multimedia and QoS, letting the development of a case-study application that wraps much of my work, making it meaningful. For having invited me to Alcoy and provided all the means and support during my stay. "Gracias también a su familia, Mónica, Jaime y Concha que me recibieron con mucha alegría y cariño. Moltes gràcies!"

to Raj Rajkumar Karthik and to Lakshmanan from the Carnegie Mellon University, for their valuable scientific contribution to the work in this dissertation and also for having made possible and pleasant my visit to Pittsburgh. I have to remark and be thankful for the outstanding commitment and support to my research topics during my stay. This collaboration was determinant to the theoretical formulation in Chapters 4 and 5. Thanks a lot!

to Thomas Nolte, Professor at Mälardalen University, Sweeden, for its valuable collaboration in the *Servers* and *Hierarchical composition* topics that enriched and amplified the applicability scope of this work. Thanks!

a Nuno Figueiredo, que prontamente aceitou o repto de, no âmbito da sua tese de Mestrado, integrar o protocolo apresentado nesta dissertação na plataforma de controlo por ele desenvolvida, permitindo desta forma uma avaliação real de algumas propriedades do protocolo. De destacar o espírito pro-activo e empreendedor que imprimiu no trabalho, permitindo o desenvolvimento de agradáveis discussões técnicas e linhas de trabalho futuro. Obrigado!

a Isidro Calvo, a Manuel Barranco, a Iria Ayres, a Pablo Basanta, que passaram longos períodos em Aveiro e com eles trouxeram alegria e boa-disposição ao laboratório. Obrigado pela vossa amizade, companheirismo e claro... por me haber enseñado algo de español. "Gracias chicos!"

a Valter Silva, a Frederico Santos, a Rui Santos, a Margarida Urbano, a Alexandre Vieira colegas de laboratório e amigos com quem tive o prazer de desenvolver agradáveis discussões técnicas durante estes anos, bem como, discussões menos técnicas que proporcionavam agradáveis gargalhadas e momentos de descontracção. Obrigado!

a José Luís Azevedo, a José Alberto Fonseca, a Pedro Fonseca, professores durante a minha Licenciatura que tiveram um papel relevante na definição das minhas áreas de interesse em comunicações e sistemas embutidos e na consequente vontade em prosseguir os meus estudos nesse âmbito. Obrigado!

a Ana Margarida, a Alexandre Santos, a Pedro Leite, a Ana Ferreira, a Tiago Silveira, grande amigos, que sem dúvida ajudaram a vincar o início desta viagem. Obrigado malta, jamais esquecerei!

a Rodolfo Andrade, a Nuno Letão, a Alexandra Moura, pelas muitas palavras de apoio e pelas quartas-feiras loucas.

ao grupo de Salsa, que durante estes anos ajudou a descontrair e descomprimir nos momentos de maior tensão. Em particular agradeço à Luísa e ao Hugo Leite pela amizade, pelo apoio e pelas muitas salsas animadas com boa música. E sem esquecer o que a Salsa me deu, agradeço à Jeanette que soube marcar a sua presença amiga e cúmplice. Obrigado!

Finalmente, agradeço às pessoas mais importantes na minha vida: os meus pais e irmã. A base de tudo o que sou e de todo este trabalho, foram vós que a construíram.

Ricardo Marau

Palavras-chave

Tempo-real, Comunicações de tempo-real, Comunicações industriais, Sistemas embutidos, Sistemas distribuídos, Escalonamento de tempo-real, Gestão de Qualidade de Serviço, Ethernet.

Resumo

Durante a última década temos assistido a um crescente aumento na utilização de sistemas embutidos para suporte ao controlo de processos, de sistemas robóticos, de sistemas de transportes e veículos e até de sistemas domóticos e eletrodomésticos. Muitas destas aplicações são críticas em termos de segurança de pessoas e bens e requerem um alto nível de determinismo com respeito aos instantes de execução das respectivas tarefas. Além disso, a implantação destes sistemas pode estar sujeita a limitações estruturais, exigindo ou beneficiando de uma configuração distribuída, com vários subsistemas computacionais espacialmente separados. Estes subsistemas, apesar de espacialmente separados, são cooperativos e dependem de uma infraestrutura de comunicação para atingir os objectivos da aplicação e, por consequência, também as transacções efectuadas nesta infraestrutura estão sujeitas às restrições temporais definidas pela aplicação.

As aplicações que executam nestes sistemas distribuídos, chamados *networked embedded systems* (NES), podem ser altamente complexas e heterogéneas, envolvendo diferentes tipos de interacções com diferentes requisitos e propriedades. Um exemplo desta heterogeneidade é o modelo de activação da comunicação entre os subsistemas que pode ser desencadeada periodicamente de acordo com uma base de tempo global (*time-triggered*), como sejam os fluxos de sistemas de controlo distribuído, ou ainda ser desencadeada como consequência de eventos assíncronos da aplicação (*event-triggered*). Independentemente das características do tráfego ou do seu modelo de activação, é de extrema importância que a plataforma de comunicações disponibilize as garantias de cumprimento dos requisitos da aplicação ao mesmo tempo que proporciona uma integração simples dos vários tipos de tráfego.

Uma outra propriedade que está a emergir e a ganhar importância no seio dos NES é a flexibilidade. Esta propiedade é realçada pela necessidade de reduzir os custos de instalação, manutenção e operação dos sistemas. Neste sentido, o sistema é dotado da capacidade para adaptar o serviço fornecido à aplicação aos respectivos requisitos instantâneos, acompanhando a evolução do sistema e proporcionando uma melhor e mais racional utilização dos recursos disponíveis.

No entanto, maior flexibilidade operacional é igualmente sinónimo de maior complexidade derivada da necessidade de efectuar a alocação dinâmica dos recursos, acabando também por consumir recursos adicionais no sistema. A possibilidade de modificar dinâmicamente as caracteristicas do sistema também acarreta uma maior complexidade na fase de desenho e especificação. O aumento do número de graus de liberdade suportados faz aumentar o espaço de estados do sistema, dificultando a uma pre-análise. No sentido de conter o aumento de complexidade são necessários modelos que representem a dinâmica do sistema e proporcionem uma gestão optimizada e justa dos recursos com base em parâmetros de qualidade de serviço (QdS).

É nossa tese que as propriedades de flexibilidade, pontualidade e gestão dinâmica de QdS podem ser integradas numa rede switched Ethernet (SE), tirando partido do baixo custo, alta largura de banda e fácil implantação. Nesta dissertação é proposto um protocolo, *Flexible Time-Triggered communication over Switched Ethernet* (FTT-SE), que suporta as propriedades desejadas e que ultrapassa as limitações das redes SE para aplicações de tempo-real tais como a utilização de filas FIFO, a existência de poucos níveis de prioridade e a pouca capacidade de gestão individualizada dos fluxos. O protocolo baseia-se no paradigma FTT, que genericamente define a arquitectura de uma pilha protocolar sobre o acesso ao meio de uma rede partilhada, impondo desta forma determinismo temporal, juntamente com a capacidade para reconfiguração e adaptação dinâmica da rede. São ainda apresentados vários modelos de distribuição da largura de banda da rede de acordo com o nível de QdS especificado por cada serviço utilizador da rede.
Esta dissertação expõe a motivação para a criação do protocolo FTT-SE, apresenta uma descrição do mesmo, bem como a análise de algumas das suas propiedades mais relevantes. São ainda apresentados e comparados modelos de distribuição da QdS. Finalmente, são apresentados dois casos de aplicações que sustentam a validade da tese acima mencionada.

Keywords Real-time, Real-time communications, Industrial communications, Embedded systems, Networked embedded systems, Distributed systems, Real-time scheduling, Quality of Service management, Ethernet.

Abstract During the last decade we have witnessed a massive deployment of embedded systems on a wide applications range, from industrial automation to process control, avionics, cars or even robotics. Many of these applications have an inherently high level of criticality, having to perform tasks within tight temporal constraints. Additionally, the configuration of such systems is often distributed, with several computing nodes that rely on a communication infrastructure to cooperate and achieve the application global goals. Therefore, the communications are also subject to the same temporal constraints set by the application requirements.

Many applications relying on such networked embedded systems (NES) are complex and heterogeneous, comprehending different activities with different requirements and properties. For example, the communication between subsystems may follow a strict temporal synchronization with respect to a global time-base (time-triggered), like in a distributed feedback control loop, or it may be issued asynchronously upon the occurrence of events (event-triggered). Regardless of the traffic characteristics and its activation model, it is of paramount importance having a communication framework that provides seamless integration of heterogeneous traffic sources while guaranteeing the application requirements.

Another property that has been emerging as important for NES design and operation is flexibility. The need to reduce installation and operational costs, while facilitating maintenance is promoting a more rational use of the available resources at run-time, exploring the ability to tune service parameters as the system evolves.

However, such operational flexibility comes with the cost of increasing the complexity of the system to handle the dynamic resource management, which on the other hand demands the allocation of additional system resources. Moreover, the capacity to dynamically modify the system properties also causes a higher complexity when designing and specifying the system, since the operational state-space increases with the degrees of flexibility of the system.

Therefore, in order to bound this complexity appropriate operational models are needed to handle the system dynamics and carry on an efficient and fair resource management strategy based on quality of service (QoS) metrics.

This thesis states that the properties of flexibility and timeliness as needed for dynamic QoS management can be provided to switched Ethernet based systems. Switched Ethernet, although initially designed for general purpose Internet access and file transfers, is becoming widely used in NES-based applications. However, COTS switched Ethernet is insufficient regarding the needs for real-time predictability and for supporting the aforementioned properties due the use of FIFO queues too few priority levels and for stream-level management capabilities. In this dissertation we propose a protocol to overcome those limitations, namely the Flexible Time-Triggered communication over Switched Ethernet (FTT-SE). The protocol is based on the FTT paradigm that generically defines a protocol architecture suitable to enforce real-time determinism on a communication network supporting the desired flexibility properties.

This dissertation addresses the motivation for FTT-SE, describing the protocol as well as its schedulability analysis. It additionally covers the resource distribution topic, where several distribution models are proposed to manage the resource capacity among the competing services and while considering the QoS level requirements of each service. A couple of application cases are shown that support the aforementioned thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> System - A group of interacting, interrelated, or interdependent elements forming a complex whole.
>
> The American Heritage Dictionary of the English Language, Fourth Edition.

For the last decades we have witnessed the generalization of the automation concept as the mean to provide comfort assistance, economic benefits and even mission-critical aid, in a wide range of application domains. The growing deployment of embedded systems has been supporting such evolutionary process, pushing the development of large-scale embedded systems that interact with the environment as within a *system*. On the quest to realize a completely integrated system, connectivity became the key on promoting such interaction between electronic devices and the environment.

Large-scale networked embedded systems (NES) can be found in areas such as transportation, industrial, medical or communications. The deployment of NES on these domains is constrained by the natural application goal that many times poses stringent timeliness requirements, as well as by non-technical aspects such as deployment cost and time-to-market that cannot be neglected. Additionally, these systems are attaining complexity levels that make them hard to be assessed with a single holistic view. To cope with such complexity, there are methodologies that promote seamless integration between the several individual elements, such as component-based development that offers dependable modular composability and the cyber-physical systems concept that abstracts data fusion and services ubiquity.

On the process of integrating those subsystems towards the application goal, it is of utmost importance defining how they interact and the consequences of such interaction. The timeliness requirements on these interactions are frequently critical when there are activities to be done within strict temporal bounds, which if not met, may result on severe consequences

for the system correctness. Therefore, the system must be supported with mechanisms that enforce predictable and time-bounded interactions.

However, on this integration process other aspects must be considered when distributing the system. Moved by economical or locality constraints, some components such as communication networks are designed to interact with as many components as to the limit of providing the timeliness guarantees. Deploying such components on dynamic environments, i.e., systems whose requirements change at run-time, demands a whole new level of resource management, with mechanisms capable of maintaining the system predictability and provide desired levels of Quality of Service.

1.1 Network Flexibility

In their early times, NES had their application scope confined to well defined control applications, for which predictability requirements were addressed with static offline scheduling, i.e., all network activities were cautiously modeled and planed during the system design phase, based on a complete a priori knowledge of the system properties [67]. During run-time the system was then coordinated by that design-time (offline) plan. This planning procedure is the simplest and most effective approach once all activities and corresponding activation instants are known beforehand. For this reason, many safety-critical applications employ static offline scheduling.

However, many real-world systems are complex and dynamic, evolving during their life-time and consequently changing their requirements and properties. In order to effectively address this scenario, the system components have to become more flexible, i.e., being able to evolve accordingly and modify their run-time performance. However, typically, the higher the flexibility degree, i.e., more configuration options, the more complex it is to integrate the system components and coordinate the configuration options. Examples of configuration options within a communication system are the physical media, network topology, transmission rate, buffering and forwarding issues or variations in the communication requirements.

During the design of the system, when to address such dynamic variability and complexity, it becomes harder or even impossible to compile a stateful knowledge of all system configuration options and provide an offline schedule plan [119]. In such cases, the use of a full static design approach becomes infeasible or at least it conducts to poor resource efficiency since the offline assumptions typically over-estimate the network utilization at run-time. A more efficient way to manage the network while providing offline predictability is deploying dynamic scheduling for use at run-time. The a priori guarantees regarding the system timeliness are in this case based on mathematical models of the resource properties evolution at run-time that define minimum requirements for each moment in time. Both approaches

require the same amount of information of the system evolution. However, with the dynamic approach, based on a more accurate resource requirements estimation, it is possible to integrate the system with less over-provisioning, thus providing better feasibility results.

NESs rely on a communication network as the central component to inter-connect the distributed nodes. Adding operational flexibility to the system is not constrained to the network, it must be coordinated comprising all related components. As an example, any data exchange is closely related to tasks that ultimately have to be scheduled in the sending and receiving nodes.

As referred above, the main motivation to improve the operational flexibility of a communication network is to enable its use within a dynamic operational scenario. A similar motivation is argued by Prasad [103] *et al.* and Lu *et al.* [80], stating that the use of static schedule plans, thus without operational flexibility, leads to inefficient resource usage and to non-graceful service degradation, i.e., the services are either fully functional or inactive, there is no solution in between. And they complement saying that it is more efficient to leave some operational decisions in terms of resource usage to run-time.

Therefore, flexibility as a means to reconfigure and/or adjust the system on-the-fly is an important property to provide efficient resource management, naturally extending to the network, the fundamental resource in a NES.

1.2 Efficient resource management

Efficient resource management of evolving systems requires the ability to react to changes. In the scope of this thesis we will distinguish two different aspects related to the flexibility properties of a resource, namely reconfiguration, which means adding and removing components and subsystems, and adaptability, which denotes the ability to adjust (semi-)autonomously the amount of resources provided to each application subsystem, according to the instantaneous requirements. This adaptability property becomes specially relevant when integrating application subsystems from different scopes that share common resources.

Such flexibility properties leverage the resource usage and performance, while providing the necessary Quality of Service (QoS) levels to each application subsystem. In one hand, it can be used in a system with variable number of users, either human or not, to minimize the QoS levels of each in order to minimize the total resource usage and accommodate a higher number of users [71, 91]. In the other hand, it can be used to maximize the resource utilization, delivering the best possible QoS level to the services [60, 104]. Additionally, systems that can self-reconfigure dynamically are able to cope with hazardous events by evolving to operational configuration

states that maintain the system integrity and dependability in general (e.g. military systems or telecommunication systems) [51, 92, 112].

This introduces the need for dynamic QoS management within a resource framework. However, to achieve it, a proper infrastructure is required that provides a resource with reconfiguration and adaptation properties, and in the other hand application-oriented models are also required in order to provide seamless integration with the system changes.

1.3 Proposition and contributions

The thesis supported by this dissertation argues that using a resource that provides reconfigurability and adaptability properties allows supporting efficient QoS management in a dynamic environment.

The Flexible Time-Triggered (FTT) paradigm has been proposed in the past to provide real-time communications in dynamic environments, having been deployed over communication protocols such as CAN and shared Ethernet. In this thesis it is proposed implementing the FTT on a switched Ethernet communication framework, leading to the Flexible Time-Triggered over Switched Ethernet (FTT-SE) protocol. We show that FTT-SE can bring substantial improvements and contributions in terms of reconfigurability and adaptability with respect to COTS switched Ethernet systems, namely:

- Support for parallel multi-path traffic forwarding in the switch (unicast and broadcast models), while maintaining the protocol requirements for predictability, with any scheduling policy.

- Handling the asynchronous traffic with a novel message signaling mechanism that improves its scheduling flexibility, particularly supporting open server-based scheduling approaches, and allows a seamless integration between asynchronous and synchronous traffic.

- System analysis including the schedulability analysis of the periodic traffic, based on a utilization bound that accounts for the impact of release jitter.

Finally, this work also presents a systematic methodology on the deployment of dynamic QoS management mechanisms over a communication resource such as the FTT-SE. Several bandwidth distribution approaches are proposed to handle specific QoS level requirements. This methodology is general and includes other existing QoS management techniques as particular cases.

Some of these contributions have been published in conference proceedings and journals and are listed below:

- Ricardo Marau, Luís Almeida, Paulo Pedreiras, and Thomas Nolte. Towards Server-based Switched Ethernet for Real-Time Communications. In *Proc. of the WiP session of the 20th Euromicro Conference on Real-Time Systems(ECRTS'08)*, 2 July 2008

- Ricardo Marau, Luís Almeida, Paulo Pedreiras, M. González Harbour, Daniel Sangorrín, and Julio M. Medina. Integration of a flexible network in a resource contracting framework. In *Proc. of the WiP session of the 13th Real-Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, 3 April 2007

- Javier Silvestre, Luís Almeida, Ricardo Marau, and Paulo Pedreiras. Dynamic QoS management for multimedia real-time transmission in industrial environments. In *Proc. of the 12th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA'07)*, September 2007

- Ricardo Marau, Luís Almeida, and Paulo Pedreiras. Enhancing real-time communication over COTS Ethernet switches. In *Proc. of 6th Int. Workshop on Factory Communication Systems (WFCS'06)*, pages 295–302, Torino, Italy, 27 June 2006. IEEE

- Javier Silvestre, Luís Almeida, Ricardo Marau, and Paulo Pedreiras. MJPEG Real-Time transmission in industrial environment using a CBR channel. In *Proc. of 16th Int. Conf. on Computer and Information Society Engineering (CISE'06)*, Venice, Italy, 24 November 2006. (also published on Enformatika Trans. on Engineering, Computing and Technology, Vol 16, Nov. 2006 ISSN 1305-5313)

- Ricardo Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and Thomas Nolte. Server-based Real-Time Communications on Switched Ethernet. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS-RTSS'08)*, 2008

- Ricardo Marau, Luís Almeida, Paulo Pedreiras, M. González Harbour, Daniel Sangorrín, and Julio M. Medina. Integration of a flexible time triggered network in the FRESCOR resource contracting framework. In *Proc of the 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'07)*, Patras, Greece, 25 September 2007. IEEE

- R. Marau, P. Pedreiras, and L. Almeida. Signaling asynchronous traffic over a Master-Slave Switched Ethernet protocol. In *Proc. on the 6th Int. Workshop on Real Time Networks (RTN'07)*, Pisa, Italy, 2 July 2007

- R. Marau, P. Pedreiras, and L. Almeida. Enhanced Ethernet Switching for Flexible Hard Real-Time Communication. In *Proc. on the 5th Int. Workshop on Real Time Networks (RTN'06)*, Dresden, Germany, July 2006

1.4 Dissertation outline

This chapter outlined the scope of this thesis and briefly addressed the need for flexibility support to deploy dynamic QoS management. To support our thesis, this dissertation is organized as follows:

Chapter 2 introduces basic terms and concepts related to real-time communication technology with emphasis on switched Ethernet networks. It also addresses the types of scheduling commonly used in networked embedded systems and techniques to assess the system feasibility with respect to timeliness.

Chapter 3 presents the Flexible Time-Triggered communication paradigm and the issues related with its implementation on a switched Ethernet network, leading to the FTT-SE protocol. This protocol is the basis for the remaining chapters and for this reason it is presented in detail. Particular attention is devoted to the communication framework, traffic and scheduling models and interface with the application layer.

Chapter 4 presents the analysis for assessing the system timeliness using the communication model of the FTT-SE protocol. It presents a sufficient schedulability test based on the communication load of each link. Beyond the proof for this test, a set of simulation-based experiments is conducted that further validate the scheduling analysis.

Chapter 5 discusses the dynamic QoS management topic within a communication framework, with emphasis to the FTT-SE protocol model. It addresses various bandwidth distribution mechanisms that consider the QoS level requirements of each service along in the resource management.

Chapter 6 includes several case-studies that address the applicability of the FTT-SE framework in different scenarios. Firstly the framework is included within a global resource management middleware. Then, the protocol is applied within a multimedia embedded system where the dynamic QoS management plays an important role. Finally, the deployment of server-based scheduling within the framework is discussed, as well as the possibility of providing an FTT-SE enabled switch, i.e., an Ethernet switch that embeds some core features of the protocol.

Chapter 7 sets the conclusion of this dissertation, discussing the contributions, validating the thesis and suggesting some lines for future research.

Chapter 2

Background

This chapter introduces basic terms and concepts related to real-time scheduling and communication used throughout this dissertation.

2.1 Real-time systems

Computer based embedded systems are already supporting a wide range of applications that we use in our everyday life, ranging from home appliances and office equipment to communication in transportation systems, robotics and process/manufacturing industries. These applications use sensing and actuating devices, forming a computer-based control architecture to interact with the environment. A control algorithm is implemented to collect sensing data and produce correct logical output through the actuator. However, many of such applications include *time* as an integral part of that algorithm and thus the outputs have to be produced within specific time windows.

These systems are called real-time systems in which the *correctness of the system behavior depends not only on the value of the computation but also on the time at which the results are produced*[120]. Typically, a time bound called *deadline* is defined, representing the instant up to which the results must be produced. Due to its inherent importance, a considerable research effort has been devoted to this issue, and thus over the last decades, several techniques have been developed to determine whether a computation completes within that deadline, in a worst-case scenario.

Depending on the particular system, the violation of deadlines may conduct to results that ultimately lead to bad or catastrophic scenarios. As an example, consider the airbag system of a car. The purpose of the airbag is to, in a collision scenario, absorb the shock of one's head inside the car. The *timing* at which the bag inflates is critical, if it goes off too early, too much gas leaves the bag before the head impact. Conversely, if it goes off too late, the airbag becomes useless. Therefore, all the procedure starting from the collision sensing until inflating the bag is critical and the intermediate steps

for processing and communication have to be time bounded.

In [68] deadlines are classified as *hard*, *firm* or *soft*. If the result is somehow valid even after the deadline passes, the deadline is classified as *soft*, otherwise it is *firm*. Whenever failing to meet a *firm* deadline leads to a catastrophe, the deadline is called *hard*. Similarly, a computer system is considered a hard real-time system if it is constrained by at least one *hard* deadline, otherwise, it is considered as *soft* real-time system.

The execution of a real-time system can be confined to a single computing unit (CPU) or comprehend several CPUs. These CPUs can either be together and coupled with shared memory (multiprocessor system), or decoupled with distributed memory (distributed system).

Distributed embedded systems (DES) represent a subset of embedded applications where the sensing, computing and actuating nodes are dispersed, motivated by either spatial or processing constraints, and connected by means of a network. This distributed approach to realize embedded applications has received a lot of attention from the research and industrial communities due to its advantages over a centralized system, mainly in terms of composability, maintainability, installation, cost reduction, fault-tolerance, among others.

2.1.1 Real-time scheduling

One of the top requirements for a real-time system is timing predictability, i.e., the ability to predict and enforce certain temporal properties of the system. Thus, a number of real-time models have been developed to capture the temporal behavior of the system.

A typical real-time system can be modeled as a set of tasks that interact and cooperate with each other and the environment, towards a common and global goal. Tasks thus represent activities handled by the computational system.

Task

A task is the computational unit in the system that implements a part of the application logic. One important characteristic is its triggering model. Depending on the activation source, tasks can either be *time-triggered* or *event-triggered*, depending whether they are activated at predefined time instants or by the occurrence of an event, i.e., a significant change in the system state. Time-triggered tasks are usually periodic with a fixed inter-arrival time. Conversely, event-triggered tasks are usually aperiodic or sporadic. Aperiodic tasks have no specific arrival pattern, i.e., can be triggered at any time. Sporadic tasks, although having no strict periodic pattern, their activation pattern is regulated by a known minimum inter-arrival.

Choosing the task model to implement a specific part of the system depends on the characteristics and requirements for each subsystem. The time-triggered model is more oriented to tasks regularly executed synchronously with others in the system, such as a periodic sampling in control systems. To handle asynchronous events such as alarms or interrupts, aperiodic or sporadic tasks provide better performance results. However, the performance of an aperiodic task cannot be guaranteed due to the lack of determinism on the release pattern. Such tasks can only be considered when the system is constructed in a way to confine their interference on the real-time functions. The sporadic traffic is one option to include asynchronous tasks with real-time properties in the system task set. The minimum inter-arrival pattern results either naturally from the triggering event or from some timing enforcement mechanism. In the worst-case they can be considered as periodic traffic with period equal to their minimum inter-arrival time.

A set of periodic tasks Γ can be formally defined as:

$$\Gamma = \{\tau_i(C_i, T_i, Ph_i, D_i, Pr_i), i = 1, ..., n\}$$

where:

- C_i is the worst-case computation time required by task τ_i, also referred as Worst-Case Execution Time (WCET);
- T_i is the period of task τ_i;
- Ph_i is the initial phase of task τ_i;
- D_i is the relative deadline of task τ_i;
- Pr_i is the priority or value of task τ_i.

The activation instant ($a_{i,k}$) and absolute deadline value ($d_{i,k}$) of the generic k^{th} instance of the periodic task τ_i can be computed as follows, with $k = 1, 2, \ldots$.

$$\begin{aligned} a_{i,k} &= Ph_i + (k-1) * T_i \\ d_{i,k} &= a_{i,k} + D_i \end{aligned}$$

The same notation is valid for sporadic tasks, except that the period (T_i) becomes the minimum inter-arrival time ($Tmit_i$) and the initial phase is not defined. In this case the earliest possible activation instant and absolute deadline instants can be computed as:

$$\begin{aligned} a_{i,k} &\geq a_{i,k-1} + Tmit_i \\ d_{i,k} &= a_{i,k} + D_i \end{aligned}$$

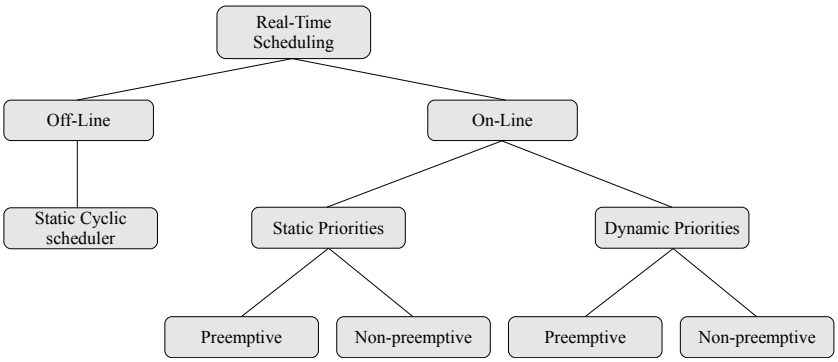


Figure 2.1: Taxonomy of real-time scheduling algorithms.

Scheduler

The tasks in the system compete to access to system resources (e.g. processor, I/O device or communication bus). The resource scheduler is a system component that decides the order in which the resource is assigned to the tasks waiting for it. The procedure of selecting the task that executes at a particular point in time is called scheduling and the set of rules that, at any time, determines the order by which tasks are executed is called a scheduling algorithm.

More accurately, a scheduling problem can be defined [37] as the composition of three sets: a set of n tasks $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, a set of m processors $P = \{P_1, P_2, ..., P_m\}$ and a set of s resources $R = \{R_1, R_2, ..., R_s\}$. Furthermore, precedence relations among tasks can be specified through a directed acyclic graph and each task can have associated timing constraints. In this context scheduling means to assign processors from P and resources from R to tasks from Γ in order to complete all tasks under the imposed constraints.

The scheduling algorithms fall into two general categories, online and offline [37] (Figure 2.1). The former ones can then use static (fixed) priority scheduling policies or dynamic priority scheduling policies. The first ones are based on fixed information that is available at pre-run-time. Conversely, the second ones perform the scheduling decisions with run-time collected information, e.g. the release instants of asynchronous tasks, hence necessarily performed online.

In the offline scheduling category, all scheduling is computed prior to execution. This approach requires a complete characterization of the system tasks in advance, not being suitable to changes to the system requirements. The scheduling result is a time-table with the tasks activations that is read cyclically at run-time, with a length typically equal to one hyper-period, i.e., the least common multiple of the periods of all tasks. This approach, demands low run-time overhead, but the memory-size requirements to hold the whole schedule plan can be very large when the task periods are relative

primes or present large asymmetries.

On the other hand, online scheduling allows executing dynamic schedulers and brings the ability to modify the system properties at run-time, e.g. adjusting to environment requests. The penalty though, is the need to execute the scheduler at run-time, which from the application point-of-view represents overhead. Furthermore, unlike the offline scheduling case, the scheduler execution must provide prompt answers within a bounded time, which strongly constrains its complexity.

Another categorizing property refers to the blocking effect that a task may cause to others when accessing a resource. In this aspect, the scheduling algorithms can be preemptive or non-preemptive. Non-preemptive algorithms execute tasks until completion, regardless of other tasks readiness and priority. In this case scheduling decisions are only required after the tasks completion instants. Preemptive algorithms can however suspend tasks during their execution, if at some instant a task with higher priority becomes ready. When executing in non-preemptive mode, a task, upon becoming ready, must wait at least for the completion of the currently running task, independently of their relative priorities. The effect of a higher priority ready task waiting for a lower priority task to release a resource is called blocking. Preemptive scheduling grants higher responsiveness to higher priority tasks, since these tasks do not suffer blocking from lower priority ones. However, in this case, scheduling events are generated more often, in all task activation instants, resulting in higher overhead.

2.1.2 Examples of scheduling policies

The seminal work by Liu and Layland on real-time scheduling [76] defines two of the most important algorithms for task scheduling on uniprocessor systems, Rate Monotonic for static-priority assignment and Earliest Deadline First for dynamic-priority systems. These algorithms are a milestone within their classes for the fact that they are optimal in the sense that they are able to produce a feasible schedule whenever any other algorithm of the same class also is.

Rate Monotonic scheduling

Rate Monotonic (RM) scheduling [76] is an online preemptive algorithm based on static priorities and with deadlines equal to periods, i.e., $\forall_{\tau_i \in \Gamma} : D_i = T_i$.

According to the RM algorithm, priorities are assigned monotonically with respect to the tasks period, i.e., the shorter the period, the greater the priority:

$$\forall \tau_i, \tau_j \in \Gamma : T_i < T_j \Rightarrow Pr_i > Pr_j \tag{2.1}$$

Task	T	C
τ_1	4	2
τ_2	6	2
τ_3	11	1

Table 2.1: Periodic task set properties.

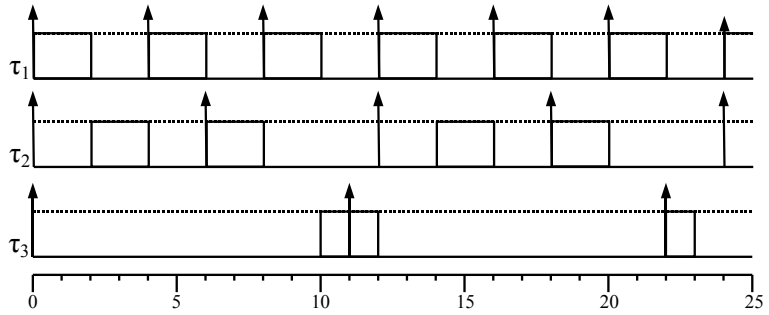


Figure 2.2: Schedule generated by RM.

At run-time, whenever a task instance is activated or the running task finishes executing, the scheduler selects the task with shorter period among the ready ones. The overall complexity of this algorithm is $O(n)$ since inserting a new task instance in an ordered queue of n elements may take up to n steps. The tasks are sorted according to their priority, so at dispatching time, the task heading the queue is the one selected.

Figure 2.2 shows a timeline of an RM schedule with the task properties stated in Table 2.1. It can be observed that task τ_1 always executes first, since it has the shortest period among all tasks, and thus the highest priority. Task τ_2 always executes before task τ_3 because it has a shorter period.

Earliest Deadline First scheduling

Earliest Deadline First (EDF) [76] algorithm is an online preemptive algorithm based on dynamic priorities with relative deadlines equal to periods, i.e., $\forall_{\tau_i \in \Gamma} : D_i = T_i$. According to the EDF algorithm, the earliest the deadline the highest the priority of the task. During run-time the following relation holds:

$$\forall_{Ta} \in \mathbb{R} \quad \forall_{\tau_i,\tau_j} \in \Gamma_{Ta} : d_i < d_j \Rightarrow Pr_i > Pr_j \tag{2.2}$$

where Γ_{Ta} is the subset of Γ comprising the ready tasks at instant Ta and (d_i, d_j) are the absolute deadlines of tasks τ_i and τ_j, at the same instant Ta.

At run-time, whenever a task instance is activated or the running task finishes executing, the scheduler selects the task with highest period among

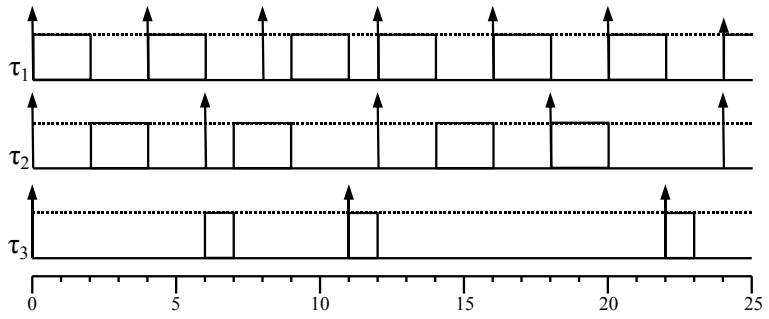


Figure 2.3: Schedule generated by EDF.

the ready ones. Since the task priorities are dynamic, it is necessary to sort the ready task queue whenever new task instances are activated. Thus, the time complexity of this algorithm is $O(n * log(n))$. It follows that EDF scheduling requires higher run-time overhead than RM scheduling, which can be problematic for systems based on low processing power CPUs, often found in some embedded distributed control applications. However, as it will be seen further on, compared to RM, the EDF algorithm is able to achieve higher utilization factors and, at the same time, reduce the number of preemptions. This results in a trade-off between run-time overhead and schedulability level, which must be evaluated case by case.

Figure 2.3 shows the EDF scheduling timeline for the same set in Table 2.1. Comparing with the timeline in Figure 2.2, built with RM, it is noticeable the variation of the priority with the distance to the deadline during run-time, for instance, at time t=6 task τ_3 has the shortest deadline and thus executes before task τ_2.

2.1.3 Schedulability analysis

To perform the schedulability analysis, there exist three different approaches: utilization-based tests, demand-based tests and response time tests. One approach may be more adequate than the others depending on specific system aspects, such as the available computational power or the time available to execute the analysis. Utilization-based tests are usually less computationally complex and faster when compared with the others, thus being more suitable to be used in dynamic scheduling systems. But on the other hand, response-time-based tests are usually less pessimistic and can provide individual response time bounds for each task.

Utilization-based

In [76] Liu and Layland propose a utilization-based test for synchronously released periodic tasks using the Rate Monotonic (RM) priority assignment.

The task model consists of independent periodic tasks with relative deadlines equal to their periods. All tasks are released at the same time called critical instant. Moreover, it is assumed that once started, the task instances execute until completion or preemption and that the operating system overhead (e.g. time required for context switching and tick handling) is small and can be ignored. However, when required, the operating system overhead can be accounted for in the analysis.

The utilization factor U of a task set composed of n tasks is defined as:

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

According to [76] there is a least upper bound for the task set utilization that guarantees the existence of a feasible schedule. The least upper bound, on the right side of Inequality 2.3, supports the following schedulability test.

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) < n \times (2^{\frac{1}{n}} - 1) \tag{2.3}$$

The authors prove that if the test succeeds, the tasks will always meet their deadlines. The lower bound given by this test tends to 69% as n approaches infinity. However, this is a sufficient test, only, and thus there are task sets that may not pass the test and yet meet all their deadlines.

Several other utilization-based analysis followed for RM scheduling. For example, Lehoczky *et al.* developed an exact analysis in [72] and an extension to fit arbitrary deadlines [73]. However, these tests are much more complex compared to Inequality 2.3. Within Inequality 2.3, a task set with n messages takes n steps, thus the computational complexity of this method is $O(n)$.

In [111], Sha *et al.* further extended Inequality 2.3 to cover *blocking-time* due to non-preemption. In this case high priority tasks can be blocked by lower priority tasks, during a time B. This blocking, also called priority inversion, occurs at most once in each task instance activation if a suitable resource access protocol is used (e.g. Priority Ceiling Protocol). For these assumptions, a set of n periodic tasks is schedulable by RM if:

$$\forall i,\ 1 \leq i \leq n,\ \sum_{j=1}^{i-1} \left(\frac{C_j}{T_j} \right) + \frac{C_i + B_i}{T_i} \leq i \times (2^{\frac{1}{i}} - 1) \tag{2.4}$$

where B_i is the time during which task τ_i is blocked by lower priority tasks (priority inversion). The task set is supposed to be ordered by decreasing priorities, i.e., $\forall i, j : i < j \Rightarrow Pr_i \geq Pr_j$.

The blocking factor B_i is defined as follows:

$$\begin{cases} B_i = 0, & Pr_i = \min_{j=1..n} \{Pr_j\} \\ B_i = \max\limits_{j \in lep(i)} \{C_j\}, & Pr_i \neq \min_{j=1..n} \{Pr_j\} \end{cases} \tag{2.5}$$

where $lep(i)$ is the set of tasks with priority less than or equal to task i.

Liu and Layland [76] also present a utilization-based test for EDF, but in this case they prove that the least upper bound is 1, i.e., full resource utilization can be achieved.

Following the same model that led Inequality 2.3:

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) < 1 \tag{2.6}$$

This is a necessary and sufficient condition (exact) that if verified assures a task set feasibility. In terms of complexity, for a task set with n tasks this takes at most n steps, thus the complexity of this method is also $O(n)$.

Demand-based

Demand-based schedulability analysis has been developed for EDF and it measures the amount of computation needed by the task set, within a time interval $t \in [t_1, t_2)$. The processor demand $h_{[t_1,t_2)}$ is given by:

$$h_{[t_1,t_2)} = \sum_{t_1 \leq r_k, d_k \leq t_2} C_k \tag{2.7}$$

where r_k and d_k are the release time and absolute deadline of all the k instances of all tasks activated at or after t_1 and with deadline at or earlier than t_2.

Assuming a synchronous release, i.e., $t_i = 0$ and $t_2 = t$, Equation 2.7 can be expressed as $h(t)$, given by:

$$h(t) = \sum_{D_i \leq t} \left(\left[1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right] \times C_i \right) \tag{2.8}$$

where D_i is the relative deadline of task i. Given this processor demand, a task set is feasible iff

$$\forall t, h(t) \leq t$$

This test needs to be checked at specific time instants, only, as proved in [37].

Response time

Response time tests evaluate the worst-case response time of all tasks in the set and compare it with the respective deadlines. These tests have been particularly useful in fixed priority systems ([111] and [31, 32]). According to Audsley *et al.* in [32], the longest response time of a periodic task τ_i, denoted as R_i, is given by the sum of its computation time (C_i) with the amount of interference that it can suffer from higher priority tasks (I_i), calculated at the critical instant.

For fixed-priority scheduled systems, the critical instant occurs when releasing task i synchronously with all other higher priority tasks, typically considered time 0. The worst-case response time is then calculated as follows:

$$R_i = C_i + I_i \tag{2.9}$$

The amount of interference due to higher priority tasks is:

$$I_i = \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.10}$$

where $hp(i)$ is the set of tasks with higher priority than task i and deadlines are shorter than or equal to periods.

Hence, combining Equations 2.9 and 2.10 results:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \tag{2.11}$$

Equation 2.11 can be solved using the following iterative procedure, where the approximation to the $(n+1)^{th}$ value happens in n steps.

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j \right) \tag{2.12}$$

The first approximation step can be initialized to $R_i^0 = C_i$ and the solution is reached when $R_i^{n+1} = R_i^n$, or the deadline is violated.

The analysis presented in [32] also includes the effect of non-preemption due to resource sharing. In this case Equation 2.9 can be used redefining the interference to include the blocking factor due to lower priority tasks, as follows:

$$I_i = B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{I_i}{T_j} \right\rceil \times C_j \tag{2.13}$$

Note that the blocking factor B_i results from the non-preemption due lower priority tasks that may be executing and holding a shared resource when task i is released. B_i is given by Equation 2.5.

Contrarily to what happens in fixed priority systems, the worst-case response time of a tasks scheduled with dynamic priorities is not necessarily obtained considering the first instance released with the synchronous release pattern. Instead, the task τ_i is found in a deadline-i busy period in which all tasks but τ_i are released synchronously and must be computed inside. This interval ends when all tasks with relative deadline shorter than or equal to that of task i finish their execution. The interval may contain several instances of task i, one of which has the worst-case response time [56]. Furthermore, Palencia and González Harbour have extended the response time analysis for EDF scheduled systems to include offsets [59, 98, 97].

2.1.4 Handling asynchronous events

Periodic tasks have known activation patterns controlled and synchronized with the scheduler framework. However, sporadic and aperiodic tasks that handle asynchronous events are activated asynchronously with respect to the scheduler. The challenge is scheduling jointly the synchronous and asynchronous tasks while preventing the asynchronous ones from jeopardizing the fulfillment of the timeliness requirements of the periodic traffic and still providing the best response times possible.

There are several approaches to handle both sporadic and aperiodic tasks that provide bounded and predictable interference, based on fixed-priority scheduling (namely RMS):

- **Background scheduling**. This is the simplest approach to handle the aperiodic activities. It confines the aperiodic tasks to be scheduled in background, i.e., when no periodic tasks are ready for execution. The periodic guarantees are preserved, however, there are no guarantees to the aperiodic tasks, which execute in a best-effort manner.

- **Polling Server (PS)** [124]. This approach uses a regular periodic task to run the aperiodic handling. At any activation instance of that task, the arrival of aperiodic events is checked which, if positive, triggers the execution of the aperiodic task within the budget delimited by the periodic polling task (server). The bandwidth is strictly reserved for the aperiodic task, i.e., aperiodic tasks execute only during the server execution. At the end of each serving instance the executing tasks are preempted and resumed in the following serving job. The aperiodic execution is thus constrained by the timeliness guaranties of the periodic task.

- **Deferrable Server (DS)** [124]. This approach, also referred to as preserving capacity PS, improves the average response time of the aperiodic requests with respect to the PS. The DS preserves the capacity (budget) of the server periodic task until the end of its period, so that aperiodic requests can be serviced at anytime, given that there is enough budget. However, deferring the server execution causes a certain schedulability penalty in the periodic task set.

- **Sporadic Server (SS)** [117]. This approach is bandwidth conservative, as opposed to the DS. It differs from the DS in the way it replenishes its capacity. Whereas the DS replenishes the server capacity periodically, the SS schedules the replenishment of the server when the budget is consumed. Hence, it preserves the reactivity of the DS without penalizing the schedulability of the periodic traffic.

- **Slack Stealing** [74]. This approach substantially improves the response time to the aperiodic events over the previous methods (SS or DS). Instead of reserving a bandwidth as of a periodic task, it creates passive tasks that collect the execution time for the aperiodic tasks by promptly stealing the time to the periodic tasks, without causing their deadlines to be missed. On the arrival of an aperiodic task the execution of periodic tasks is delayed as far as to execute the request, while avoiding deadline misses.

These approaches are however constrained by a low schedulability bound inherent to the fixed-priority scheduling. Similar techniques exist to handle the aperiodic events with dynamic-priority scheduling. Such techniques provide a better resource utilization comparing with the fixed-priority ones. An example is the Total Bandwidth Server (TBS) and the Constant Bandwidth Server (CBS). A more exhaustive overview and analysis of fixed- and dynamic-priority servers can be found in [37].

2.1.5 Hierarchical schedulers

A hierarchy of schedulers generalizes the role of CPU schedulers by allowing them to schedule threads as well as groups of threads scheduled with their own scheduler. A general, heterogeneous scheduling hierarchy is one that allows arbitrary (or nearly arbitrary) scheduling algorithms throughout the hierarchy.

This enables a better use of the system, in which in some cases one specific scheduling policy better fits a part of the system whereas other parts benefit more from using other scheduling policies. This hierarchical separation facilitates the system composability at design phase, which can be particularly useful to integrate legacy applications or subsystems.

Using server-based schedulers, the budget or share capacity provided by the server can be scheduled at the same time as the rest of the system relying on fixed- or dynamic-priority scheduling. In this way several servers can be integrated as a hierarchical composition that provides timeliness confinement domains to the whole system.

2.2 Real-time communications

Originally, computer based applications were mainly centralized using uniprocessor systems. The availability of low-cost micro-controllers serial interconnections enabled the development of "intelligent" nodes, i.e., nodes with processing and communication faculties, fostering the development of distributed architectures.

In these distributed architectures, the processing units need to exchange information, thus each one is attached or integrates a network interface unit

providing access to a suitable communication system. This type of system is loosely coupled in the sense that all information exchange is performed exclusively via the communication system using messages.

A distributed real-time system is one distributed system that integrates activities with strictly defined completion time bounds. These time-constrained activities include both tasks execution within the processors and messages exchange in the communication network that must be considered together when defining the system timeliness properties.

An example of a distributed real-time system is the ABS breaking system in a vehicle. The system prevents the wheels on a vehicle from locking while braking. There are several sensors in the wheels that sense the angular velocity of each wheel. Then, the velocity of each wheel is compared to detect the moment a wheel is about to lock, which triggers a mechanism that alleviates the braking pressure on the required wheel. The system is composed by a central computing unit and several sensors and actuators connected by a communication network. The success of the system depends on the time interval that goes from the wheel lock detection to the brake actuation. Moreover, the operating control loop relies on low jitter activities, hence the need for determinism and predictability.

Therefore, the temporal behavior of the whole distributed system depends not only on the timeliness of tasks executing on each processing device but also on the capability to provide message delivery within specific timing requirements [128, 96]. Communication systems able to support such temporal requirements are called real-time communication systems. The remainder of this section addresses some important issues concerning real-time communication.

2.2.1 Event- and Time-triggered communication

The event-triggered(ET) and time-triggered(TT) communication paradigms represent two different approaches to trigger the communications within distributed systems. While in event-triggered communication, messages are sent by the application upon the occurrence of some asynchronous event, e.g. a sensor changing value, according to the time-triggered paradigm, messages are sent only at precisely predefined time instants.

The two models can be compared as follows:

- **Predictability**. In time-triggered systems the system information is normally managed within a cycle to periodically update/poll the status of the system variables. This approach exhibits greater determinism since it is possible to compute in advance the instants at which the streams and communications will take place. On the other hand, such level of knowledge is not available for event-based systems since events may occur at arbitrary and unforeseeable time instants. Thus, the

temporal predictability of this class of systems is lower. Therefore, while TT systems allow a per job timing analysis, ET systems normally rely on worst-case analysis.

- **Resource utilization**. In real-time system, resources have to be dimensioned to address the worst possible scenario occurring during the system operation. For TT systems, the resources are issued with known activation patterns, so the amount of resources can be pre-allocated and then used at run-time. Conversely, for ET systems, since the load activation pattern is not regular, the worst-case scenario has to be considered in which all activations occur every T_{mit} (the minimum inter-arrival time). This may result in a resource utilization penalty for ET systems. On the other hand, in TT systems the committed resources are effectively used at run-time, whereas in ET systems the resources are used when actually needed, only. This may lead to a rather large reduction in average resource utilization in ET systems.

- **Temporal composition**. This property assures that when different subsystems are integrated within the same system, the temporal behavior of each part is unaffected. On event-triggered subsystems, that integration necessarily disturbs the individual temporal behavior of each part. Events from different subsystems ultimately may happen simultaneously and interfere. The integration of time-triggered systems allows avoiding this scenario by introducing temporal offsets that prevent simultaneous activations.

- **Alarm shower**. Alarm showers are normally rare and thus difficult to predict. An alarm shower is typically triggered by a system failure that triggers a large burst of asynchronous events in consequence of that failure. This scenario may lead to overload in ET systems, while in TT systems the load is kept constant.

Given these features, it is normally considered that time-triggered approaches are better suited to control systems [99], to cater the demands for periodic messages with low latency and jitter, related to the periodic sampling and actuation. On the other hand, event-triggered approaches are more adequate to promptly handle asynchronous messages with low latency. In an overload period of asynchronous requests, systems can be designed to include the so called graceful degradation, by allowing the failure of some less important activities while striving to execute the tasks considered as more important to the system. Another alternative is to switch to a TT approach to handle the asynchronous events overload. This allows maintaining the system load within a predefined bound.

Despite their differences, many applications include both periodic and asynchronous activities and thus benefit from a joint support for both event

and time-triggered traffic (e.g. automotive systems [77]) and thus, a combination of both paradigms in order to share their advantages is desirable. An important aspect is that temporal isolation of both types of traffic must be enforced or, otherwise the asynchronous nature of event-triggered traffic would corrupt the properties of the time-triggered traffic. This isolation is achieved by allocating bandwidth exclusively to each type of traffic.

A typical implementation makes use of communication slots called elementary cycles, or micro-cycles (e.g. [105]), containing two consecutive phases dedicated to each type of traffic. The communication timeline becomes, then, an alternate sequence of time-triggered and event-triggered phases. The maximum duration of each phase can be tailored to suit the needs of a particular application. If each type of traffic is forced to remain within the respective phase then temporal isolation is guaranteed. This concept is used, for example, in the WorldFIP [23], Foundation Fieldbus-H1 [23] and FlexRay [33] fieldbuses, as well as in FTT-CAN [28] and FTT-Ethernet [101].

2.2.2 Message scheduling

Distributed systems are normally built over a shared medium network through which the nodes exchange data messages. Similarly to any shared resource that integrates a real-time system, its use is subject to time constraints that must be met. Therefore, as with tasks in microprocessors, the access to the communication network must also be properly scheduled. Referring to the real-time properties presented in Section 2.1, other similarities can be found between message scheduling in communication networks and task scheduling in microprocessors; messages can also be categorized according to the timeliness requirements (soft, hard and best-effort) and the activation pattern (periodic, sporadic or aperiodic). This allows the use of some results obtained for processor scheduling (e.g. [127] and [47]). Additionally, also the real-time scheduling paradigms (offline, online with fixed or dynamic priorities assignment) are present in real-time message scheduling [26].

However, there are additional challenges posed by the constraints inherent to a distributed architecture. One characteristic of a resource scheduler is that it needs to keep the resource status updated to accurately control its usage. However, due to the distributed nature of the system, complete knowledge about the system state, including network and nodes, is hard to gather or even unavailable, and thus scheduling decisions may have to be taken based on incomplete information [111]. This is particularly important when the access to the network is distributed in which case it is necessary to assure the system status consistency when taking communication scheduling decisions. The lack of complete information about the system state or the substantial overhead required to get such information consistently, leads to scheduling techniques that do not keep their optimality, comparing to the

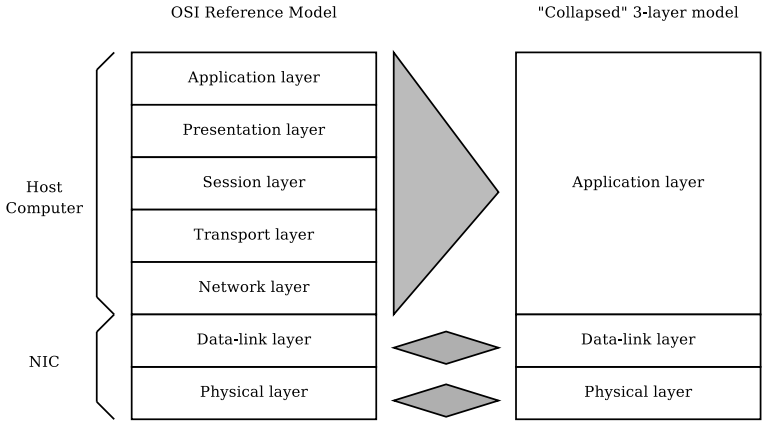


Figure 2.4: The ISO/OSI reference model.

homologous microprocessor ones [81].

Another major issue is the organization of the communication in packets that cannot be preempted during transmission. Systems that use single packet messages are thus non-preemptive. The penalty here is efficiency because preemptive systems are known to provide a higher level of schedulability when compared to non-preemptive ones. A partial solution to this problem consists on limiting the maximum packet size. This way the penalty incurred with a lower priority packet blocking the transmission of a higher priority one is reduced. Long messages can be broken in packets that are scheduled sequentially and then preemption can be considered between packets. The counterpart is an increased overhead, both on the network and on the system nodes that have to fragment and reassemble the messages.

Many different communication protocols and hardware are required to enable communication in real-time networks. Similarly to general purpose networks, the communication stack is organized in layers that carry out specific operations. The application is ultimately given a set of available services that ease and allow seamless integration of the system nodes.

Figure 2.4 illustrates the architecture of the ISO Reference Model for Open Systems Interconnection [137]. The *physical layer* is responsible for the whole transmission of raw data on the used medium. The *data-link layer* is responsible for transmitting correctly the data frames and assure any error handling related to the transmission. The *network layer* handles the setup and maintenance of network wide connections, handling the nodes addressing and message routing. The *transport layer* handles the end-to-end communication. Above the *transport layer* the communication is fully abstracted and independent of the underlying network.

Figure 2.4 also presents a "collapsed" OSI-based architecture where the upper 5 layers are simplified and merged into a single application layer. This

simplification is frequently found when implementing real-time communication infrastructures. The OSI Reference model was developed for generic communication systems and the reason for such simplification is the need to provide a lightweight protocol stack for low-end devices with real-time requirements. In fact the full OSI reference model becomes too expensive in terms of both CPU power and memory, as well as network bandwidth for such resource-constrained devices.

When it comes to design a protocol stack with real-time performance properties, one of the strategies to improve that performance is providing distinct handling paths (queues and computing priority) for real-time and non-real-time traffic (e.g. [114]).

The Medium Access Control (MAC) protocol within the *data-link layer* determines to a large extent the degree of timing predictability of the network technology. Common MAC protocols used in real-time communications networks can be classified in random access protocols, fixed-assignment protocols and demand-assignment protocols. Examples of random access protocols are:

- CSMA/CD (Carrier Sense Multiple Access / Collision Detection),
- CSMA/CR (Carrier Sense Multiple Access / Collision Resolution),
- CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance).

Examples of fixed-assignment protocols are:

- TDMA (Time Division Multiple Access),
- FTDMA (Flexible TDMA).

Examples of demand-assignment protocols are:

- distributed solutions relying on tokens,
- centralized solutions by the usage of masters.

A comprehensive study and classification of access protocols used in real-time communication over multiple-access networks can be found in [81]. The MAC protocols are described as consisting of two processes: access arbitration and transmission control. The access arbitration process determines when the node is allowed to access the communication channel to send messages, whereas the transmission control process determines for how long a node is able to use the medium to send messages. It also provides examples of protocols relying either in access arbitration or transmission control.

2.2.3 Co-operation models

As referred in the beginning of this section, in a distributed system multiple autonomous processing units cooperate towards a common objective. There are different ways and models to realize such cooperation. These models define who initiates a transaction as well as the number of partners involved in the process.

Two well-known co-operation models are the **producer-consumer** and **client-server** [133].

The **producer-consumer** model associates to each message stream a unique logical value that identifies the respective exchanges over the network. Messages are generated and received based only on these logical handles, without any explicit reference to the source nor destination nodes of the messages. The producers do not need information regarding the nodes consuming a specific message and vice versa.

The producer-consumer co-operation model inherently supports one-to-one and one-to-many communication, without incurring in spatial data consistency problems, since the same data message is used to update all the local images in all the nodes in the network that need such data, the consumer. The only requirement for the underlying network is the support for atomic broadcast or multicast.

On the other hand, this model does not solve the problem of temporal consistency that occurs when several producers interfere and cause transmission delays, ultimately leading to outdated values. The **producer-distributor-consumer** (PDC) model [126] provides the necessary synchronization between the producers, by adding a coordination layer to the producer-consumer model. A new network component is introduced to coordinate the transmissions and minimize conflicts that lead to transmission delays. This coordination is usually realized in one node, called master, that aggregates the temporal requirements of the network and enforces a suitable schedule that grants the right to transmit to each producer.

Another approach is the **client-server** model. According to this model the nodes holding information to share are servers that serve the clients upon request. The client nodes issue requests to the servers holding the desired data and then, on turn, reply with the demanded data. This communication model is by nature one-to-one and may lead to both spatial and temporal data inconsistency problems when used to support one-to-many or many-to-one communication. The server addresses the requests sequentially and when issued to transmit, at the same time, to different clients, it is inevitable the temporal inconsistency resulting from the serialization process. Additionally, it might occur that in this process the value changes and so the clients receive inconsistent results, i.e., spacial inconsistency. Another related issue refers to the asynchronous requests to the server. Inside the servers, the requests take some time to be processed and replied. When multiple requests have to

be processed, the serving time varies and creates dependency on the requests arrival pattern, which may be non-deterministic [133].

2.3 Real-Time and Switched Ethernet

Switched Ethernet is currently the most used technology for wired computer communications in many domains including those that require real-time capabilities, such as Industrial Automation or Distributed Multimedia. Despite the absence of collisions, providing real-time communication services is still not a trivial task due to the use of First-Come-First-Served (FCFS) policies, the existence of too few priority levels and limitations in the per stream management.

This section generically describes the SE technology, with special highlight to the issues related with providing real-time guarantees.

2.3.1 Switched Ethernet

Ethernet was originally developed, about 30 years ago, as an experimental technology for computer-networking and it is now supported by a collection of IEEE standards defining the physical layer, and the the data link layer.

Ethernet is continuously evolving as new technological demands urge. Regarding to the transmission speed, it evolved from the original 2.94 Mbps to 10 Mbps [3, 6], then 100 Mbps [5] and more recently 1 Gbps [7] and 10Gbps [4]. As to the physical medium and network topology, Ethernet evolved from a bus topology based on coaxial cable to a star topology [6], firstly with repeating hubs and since the mid-'90s with switching hubs.

Along with this evolution two fundamental properties of Ethernet were kept unchanged, whenever a shared medium was needed:

- a single collision domain, i.e., frames are broadcast on the physical medium so all network interface cards (NIC) can receive them, and
- the arbitration mechanism, which is a Carrier Sense Multiple Access with Collision Detection (CSMA/CD), using Binary Exponential Backoff and retry (BEB).

To comply with the CSMA/CD protocol, each NIC must hold the frames transmission until the bus becomes idle (no frame being transmitted). Only then it starts the transmission. However, all nodes follow a similar algorithm and thus collisions are prone to occur, specially after a message transmission. On a collision event all transmissions are aborted triggering a special recovery procedure. This procedure includes transmitting a jam sequence to notify the other nodes and enforce consistency in the collision detection, wait for a random period and only then retry transmitting the message. This procedure

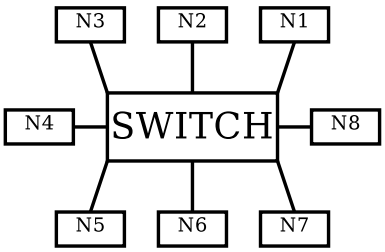


Figure 2.5: Switch micro-segmentation.

Preamble (7 octets)	SOF (1 octet)	Destination Addr. (6 octets)	Source Addr. (6 octets)	EtherType (2 octets)	Data (46..1500 octets)	FCS (4 octets)

Figure 2.6: Ethernet II frame.

is repeated several times until the message transmission succeeds. In each of the first 10 retries the width within which the random waiting period is chosen is doubled (BEB). For the following retries that width is is constant and after the 16th retry, the packet is dropped. This procedure is also known as truncated exponential backoff.

The use of a single broadcast domain and the CSMA/CD protocol creates a bottleneck to the network throughput efficiency. The higher the network load, the higher the probability for collisions and re-transmissions, casing thrashing. The solution for this problem came in the early-'90s with the technological evolution in circuit-switching that allowed using switching hubs, commonly known as switches.

A switch defines multiple collision domains, one per network segment and provides packet-switching technology. If there is a single node in each segment, collisions never occur, which grants a better network throughput since thrashing is avoided, this is called micro-segmentation (Figure 2.5). Additionally, most nowadays switches are able to process and forward multiple packets simultaneously, i.e., the switch delivers multiple-forwarding paths in parallel, which significantly improves the network throughput. Another positive aspect is that end-nodes no longer receive packets that are not addressed to them.

Ethernet frame

The MAC framing in Ethernet includes the *preamble*, meant for clock synchronization, followed by the *start of frame*(SOF) delimiter, see Figure 2.6. These are hardware requirements and their management is confined to the NICs, hence not software manageable. In addition to the frame definition, the IEEE802.3 Standard specifies a minimum interval of time that must be observed by a device connected to the Ethernet bus, between the end of a frame transmission and the start of another one. This time-slot (96 bit)

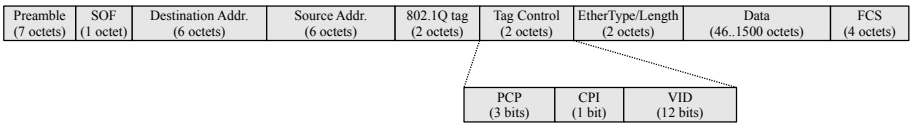


Figure 2.7: Ethernet IEEE 802.3ac frame.

between transmitted frames is referred to as the *minimum inter-frame gap*.

There are several types of Ethernet frames, from which Ethernet II (DIX v2.0)[3] frame (Figure 2.6) is the most widely used and the base for the other standard types and formats that although different, can coexist on the same physical medium.

The MAC header includes the packet destination and source addresses that denote respectively the recipient(s) and the sender of the message and the *EtherType* field (2 octets) that holds a sub-protocol identifier. For example, an *EtherType* value of 0x0800 means that the packet contains an IPv4 datagram, or a value of 0x0806 an ARP frame. The data itself is placed in the *Data* field, which can contain between 0 and 1500 bytes. However, there is a minimum frame size of 64 octets and so padding may be necessary for a minimum data of 46 octets. Finally, the Ethernet frame ends with a frame check sequence (FCS), meant for error detection.

As the industry-developed Ethernet II passed the IEEE standardization process, the *EtherType* field was modified to hold the length field in the new 802.3 standard [6], which additionally includes the IEEE 802.2 LLC header, in the first two bytes of the data segment. The two versions may however coexist considering that *EtherType* values are greater than 0x0600, which is a value larger than the longest frame size. In practice, both formats are in wide use, with the original Ethernet framing being the most common in Ethernet local area networks, since it is used by many deeply deployed protocols and applications.

Another framing format is the Novell's "raw" 802.3 frame, that discards the LLC header. Although, not compliant with the IEEE 802.3 standard that defines the LLC header at the start of header the *Data* field, this frame is meant to support IPX networks that use datagrams starting with "FF", which does not occur with the LLC header. Hence, physical compatibility is ensured.

LANs are increasingly used by new service classes, some of them with requirements quite different from the traditional ones. For instance interactive voice and video are increasingly used. To address the original lack of support for this class of services, a set of standards have been proposed to extend the framing formats, providing traffic differentiation. Such enhancements include VLAN tagging (IEEE 802.1Q [20]) and a priority identifier (IEEE 802.1p [12]) for quality of service (QoS) purposes. The IEEE 802.3ac [13] describes the extension to the basic frame as illustrated in Figure 2.7. In

this extension, the before called *EtherType* is set with a fixed IEEE802.1Q tag type (0x8100) that identifies the extended format. The new framing adds 4 octets preceding the *Data* field, namely the Tag Control and the new *EtherType/Length*. In the former, the three most significant bits of the first octet (PCP-Priority Code Point) indicate the frame priority level from 0 (lowest) to 7 (highest), the following bit (CFI-Canonical Format Indicator) is used for compatibility issues with Token-ring networks, indicating if the MAC address is in the canonical format. The remaining 12-bit field (VID-VLAN Identifier) specify the VLAN to which the frame belongs, a value of '0' indicates no VLAN.

Queuing model

Considering a switched Ethernet scenario with a micro-segmented topology, i.e., one NIC connection per switch port, and full-duplex links, the medium will always be free for transmission and thus the original medium access protocol (MAC) of Ethernet is not used. Any contention in output ports is sorted out using queues inside the switch that allow serializing transmissions.

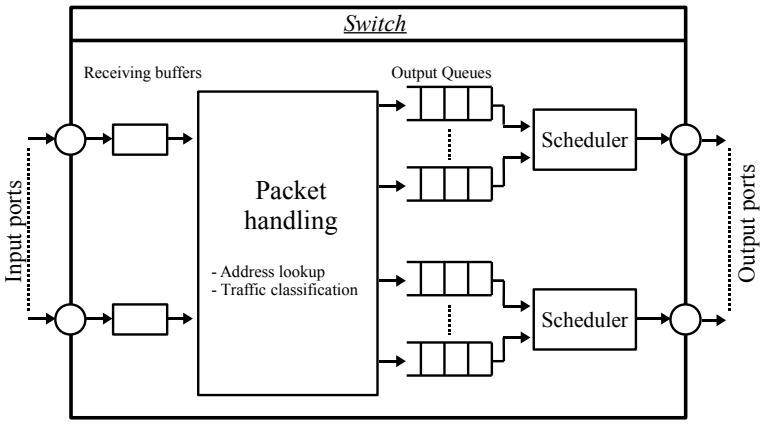


Figure 2.8: Typical switch internal architecture.

Typical commercial switches are based on IEEE 802.1D-2004 [19] that implements FIFO queues [1] at the output ports, as shown in Figure 2.8. Hence, a packet arriving at the queue is blocked for the time needed to transmit the traffic already held in the queue. Additionally, switches supporting priorities implement several queues on each output port, one per priority level. The traffic is dispatched starting from the queue with highest priority, with no preemption at the packet level. The standard defines up to eight priority levels, however, most available switches implement just two or three.

Another characteristic of the implementation of the queues in commercial switches is that they share a common memory pool [102], without any cap

[1] A First-In-First-Out queue implements an FCFS policy

per port. This is another source of potential interference since a burst of low-priority messages forwarded to the same port can overflow the memory pool, possibly causing high-priority messages to be discarded upon queuing [102].

Real-time applications require the knowledge of the latency that a generic packet takes traversing the switch. The overall switch delay comprises the switching latency and the queuing delay. The switching latency (ε) corresponds to the minimum forwarding time, when no congestion is observed at the queues. Assuming a cut-through switch and no traffic congestion, the switching latency (ε) is constant and independent of the packet size. Instead, for a store-and-forward switch, the size of each packet (transmission time) must be considered as part of the switching latency since a packet must be fully received before being transmitted in the output ports. Regardless of the switching type, this latency is predictable and easily accounted for. However, the queuing delay depends on the traffic previously issued to the switch, which is a fraction of the arriving traffic pattern and forwarding directions. Therefore, the transmission delay analysis must be well supported with adequate models for traffic generation and queuing.

Traffic forwarding

An Ethernet a network bridge that processes and routes packets at the *Data link* layer (layer 2) of the OSI model. The switch connects network segments in a star topology and isolates the respective different collision domains by interpreting and handling atomically.

The switch, as a layer 2 networking device, examines the incoming packets individually and forwards them to the appropriate segment(s) according to their destination MAC address. For this purpose the switch maintains a forwarding table that matches the MAC addresses of the end-nodes and the network segment to which are connected [2]. This table can be statically set through the switch configuration interface or it can be dynamically updated, on every incoming packet, where the corresponding entry in the forwarding table is updated using the sender MAC address. In this case, this procedure is repeated on every incoming packet, creating or refreshing the forwarding table that relates MACs and ports. An additional aging counter is provided to each entry that allows removing that entry once a given period of inactivity elapses.

The packet forwarding can be done in either of two ways: *store-and-forward* or *cut-through*. In *store-and-forward* the switch first receives the full packet into a memory buffer and only then forwards it to the correct port(s). This technique allows discarding erroneous packets or fragments, thus avoiding error propagation in the network. *Cut-through* switches start

[2] For convenience, we will also refer to network segments as switch ports.

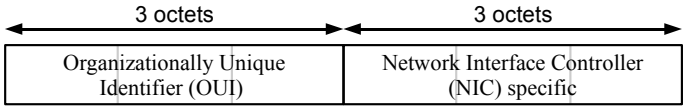


Figure 2.9: IEEE 802 MAC address (MAC-48).

forwarding the packets as soon as the destination address is received and interpreted, resulting in lower switching latencies and more determinism since the latency becomes independent of the size of packet being forwarded.

When more than one message contends for the same output port, one of them is promptly forward while the others are queued in memory. Those messages are arranged in a FIFO queue per output port.

There are three forwarding address types, unicast, multicast and broadcast. In case of a broadcast transmission, the switch forwards the packet to all other active ports with an active link. On a unicast packet transmission, the switch first examines the forwarding table for the corresponding MAC address and forwards the packet to the matching port. However, if there is no entry with that MAC address, the switch sends it to all ports, i.e., forwards it as with a broadcast address. This is normally referred to as flooding. Concerning the forwarding of multicast traffic, until recently, a multicast address was treated as a broadcast, being forwarded to all active output ports. Currently, the so-called layer 3 (or 2+) switches follow the IGMP protocol [40] that enables a true multicast forwarding.

The unique assignment of a MAC address to each network interface card (NIC) allows addressing each device within a LAN. Each manufactured NIC is assigned a MAC address that in Ethernet is 6-octets long (MAC-48), as illustrated in Figure 2.9. In order to regulate such assignment, the identifier is partially managed by the IEEE that issues an Organizationally Unique Identifier (OUI) to the network manufacturers, defining the three most significant octets of the MAC address, whereas the remaining three octets are assigned by the manufacturers.

This universal identifier directly supports the unicast transmissions as it provides an individual address within a network segment. Additionally, the IEEE defines special group or multicast addresses that allow the same packet being received by several stations. The least significant bit of the first octet of a MAC address distinguishes individual addresses from group addresses. If set to '1' it refers to a group address, either multicast or broadcast. A packet sent with a multicast destination address is received by all stations in a LAN that have been configured to receive from that address, whereas a packet sent with a broadcast address (all-ones, "FF:FF:FF:FF:FF:FF") is received by all stations within the LAN. As for the multicast addresses, the Internet Assigned Number Authority (IANA) reserves an OUI that it uses for mapping IP multicast addresses to MAC addresses. The result is a range

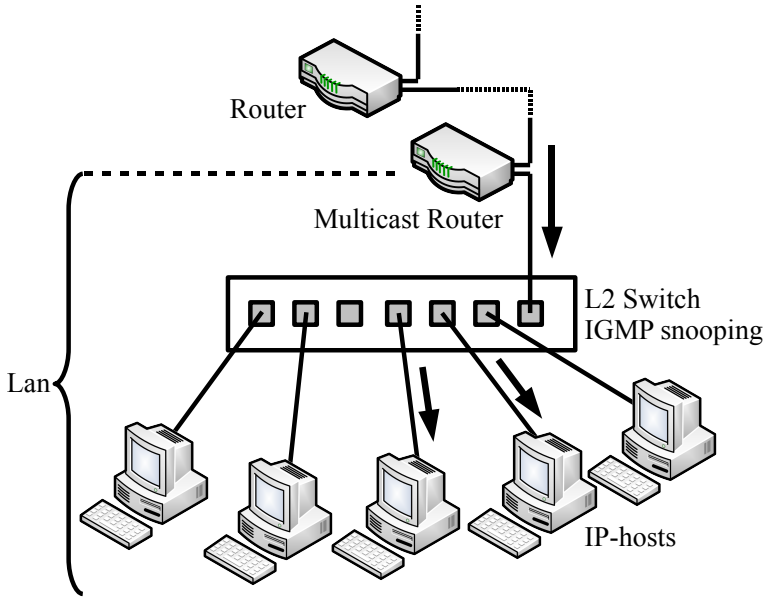


Figure 2.10: IGMP basic architecture.

of MAC addresses with OUI="01-00-5E" that is reserved to address groups of NICs.

IGMP snooping for multicast

The Internet Group Management Protocol (IGMP) [40] is a layer 3 communication protocol to manage membership associations when streaming to IP multicast groups. This protocol integrates IP-hosts and adjacent multicast routers (layer 3) that perform the multicast group membership and so forward an outbound transmission to the specific network(s). This allows setting a multicast path traversing several networks.

To set a multicast routing path, the IGMP protocol v3 includes two basic commands, the *Group Membership Query* and the *Group Membership Report*. Within each network (Figure 2.10), there is one multicast router, the querier, that periodically issues query messages to determine the active members of a group. In reply the IP-hosts send membership reports. These query and response messages allow the multicast routers to track the membership requirements on each interface and thus forward the traffic accordingly. The IGMP Host is responsible for maintaining a list of multicast groups for which it requires membership, reply to the membership queries or spontaneously trigger a membership report when it requires membership of a new multicast group.

For each IP-multicast membership group within a network, there is a multicast-MAC associated that is managed by the querier. On the arrival

of multicast datagrams to that network, the datagrams are routed with the associated multicast-MAC. Once a host reports a positive membership it knows which multicast-MAC to listen to.

However, the network topology may include cascading switches, which are layer 2 devices, thus not included in the IGMP protocol. On the ingress of a packet with a multicast address, the switch floods the ports to reach the designated IP-hosts. Although not an issue for the multicast transmission, this is not a true multicast and causes transmissions in segments that do not need such packets thus wasting bandwidth. To deliver a true multicast, there are layer 2+ switches capable of snooping the layer 3 IGMP protocol exchange (IGMP snooping [44]) between the IP-hosts and the multicast router. These switch devices use the multicast information in the IGMP membership report messages to build their multicast forwarding tables. When the switch listens to an IGMP report, meaning that a multicast route was created, the switch associates the port for the end-host to the multicast-MAC address in matter. Conversely, when it listens to an IGMP negative report it removes the host from the table.

2.3.2 Real-Time protocols over SE

In the quest for enforcing real-time behavior in switched Ethernet several techniques have been proposed, most of which rely on adding a transmission control protocol that allows controlling the traffic submitted to the switch. This approach allows using standard Ethernet hardware in the low-end nodes but introduces a arising from the additional computational complexity associated to such protocols, specially when fine-grained resolution for the transmission instants is needed.

Several approaches of this kind for switched Ethernet are classified and briefly addressed bellow.

Master/Slave

One of the simplest ways to enforce real-time predictability in a shared network or a distributed system consists of using a master-slave approach. This approach uses a special node, the *master*, that controls the network access of all other nodes, the *slaves*. The problem of enforcing timeliness is reduced to a centralized scheduling problem in the *master*. The *master* enforces its scheduling via control messages sent to the slaves granting thus network access at the right instants. Such mechanism introduces a considerable amount of protocol overhead caused by the *master* control messages that precede each transmission from a slave. The protocol becomes more or less bandwidth inefficient depending on the ratio between the sizes of the slave messages and the respective control messages. Nevertheless, this strategy is easy to deploy and provides relatively precise timeliness. It has been adopted in protocols

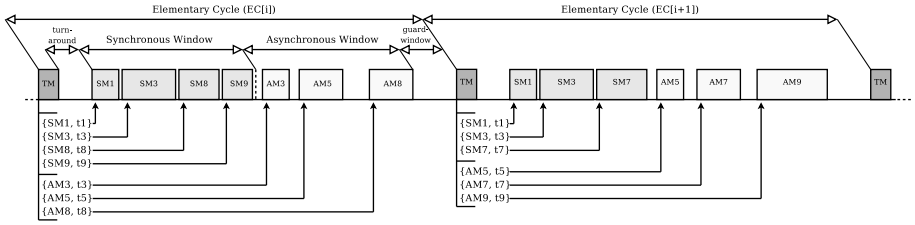


Figure 2.11: The EC structure in the original FTT-Ethernet.

developed for shared Ethernet but which worked seamlessly over switches, e.g. the ETHERNET Powerlink [22] and the FTT-Ethernet [101]. However, these protocols are based on a broadcast traffic model and are not able to exploit the parallel forwarding paths that a switch provides. This fact leads to a very strong bandwidth utilization penalty, being thus rather inefficient.

FTT-Ethernet is based on the FTT communication framework paradigm, initially implemented over the Controller Area Network (CAN) [28]. This framework is based on a master/multi-slave transmission control approach in which the communication requirements and traffic scheduling are centralized in a master node. This approach facilitates the online changes of the communication requirements and scheduling parameters, thus granting a high level of operational flexibility.

Within the FTT framework, the bus time is divided in equally sized time slots called elementary cycles (ECs). Each slot is further decomposed in two windows, synchronous and asynchronous, that address traffic with different properties, respectively, the periodic time-triggered traffic and the sporadic/aperiodic traffic (Figure 2.11). The expression *time-triggered* when associated to the periodic traffic implies the existence of a common synchronization pace, which in this case is set by the master with the cyclic framework. There is a strict temporal isolation between both windows to avoid mutual interference between both types of traffic.

The traffic is scheduled in a per cycle basis, i.e., message scheduling is done for a full EC at a time. The master identifies the messages that get scheduled in each cycle as well as their transmission instants with a trigger message (TM) sent at the beginning of each cycle. As depicted in Figure 2.11, the transmission of every message is then issued by the slave at the proper time within the right window.

Due to the global knowledge of the network requirements and the centralized control of the traffic schedule, the protocol supports arbitrary scheduling policies (e.g., rate-monotonic or earliest deadline first) and online modifications to the communication attributes with RT guarantees given that an online admission control algorithm is provided in the master node.

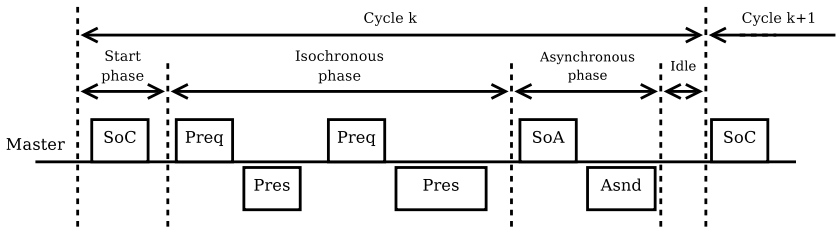


Figure 2.12: ETHERNET Powerlink cycle structure.

ETHERNET Powerlink (EPL) [22] is a master-slave protocol that provides deterministic real-time communication for standard Ethernet. The protocol was initially designed for shared Ethernet, providing a very high precision, under $1\mu s$. When deployed over a switch-based network, the precision is reduced due to extra and variable latencies introduced by the switches. The protocol also supports both periodic (isochronous) and aperiodic (asynchronous) traffic in two consecutive but isolated phases in a cyclic framework.

A master (Managing Node) controls the communications by assigning time slots for the slaves to transmit. The slaves are passive bus stations, reacting to the master explicit requests.

The communication scheduling is conducted by the master that issues time-critical data exchanges in isochronous phases within consecutive cycles. The overall cycle duration depends on the amount of isochronous data to exchange in the system. Figure 2.12 illustrates one such cycle. Each cycle is divided in four distinct phases: start, isochronous, asynchronous and idle. In the *Start Phase* the master sends a synchronization message (SoC - start of cycle) that notifies the nodes of a new starting cycle. In the *Isochronous Phase* the master triggers the periodic data transactions sending a poll request (Preq) for each. The slaves reply with the Pres (poll response) that is broadcasted to all nodes in a producer-distributor-consumer model. Isochronous transmissions are repeatedly issued every cycles, however, the transmission slots may be multiplexed in time, issuing a message every n cycles. In the *Asynchronous Phase* the master grants one node the right to transmit a message. It sends the SoA (Start of Asynchronous) message that addresses that node, which then replies with the respective asynchronous message. After this phase, the master introduces some idle time before the following cycle, to enforce a precise cycle start with low jitter.

The asynchronous transmission mechanism is very limited and cannot be used to handle time-critical communications. Unlike the isochronous traffic, for which the master is exactly aware of the messages to be polled at each time, for the asynchronous traffic, each slave has to notify the master. This is performed piggybacking a status signal onto the isochronous messages. Asynchronous-only nodes are also supported by the protocol, in which case

the nodes are polled by the managing node, in a best-effort manner. EPL also supports TCP/IP traffic, tunneled in the asynchronous phase.

TDMA

Another well-known technique to achieve temporal predictability on a shared communications network is to assign strict temporal slots to the nodes in a cyclic fashion. This is known as Time-Division Multiple Access and requires a global synchronization of the nodes when accessing the network. The temporal division is statically assigned so that all nodes can know their time slot and use it properly. Hence, it is a way to control contention in the output ports queues. The TDMA protocol is widely used in safety-critical applications for its simplicity. However, it requires precise clock synchronization in all the distributed nodes and it is not suited to allow dynamic changes in the message set, as the requirements are fully distributed. On the other hand, this protocol efficiently uses the bandwidth as there are no control messages introducing overheads, beyond those for the nodes synchronization and it is possible to exploit the parallel forwarding paths provided by the switch. In some cases, special switches are built that are also provided with the nodes transmission schedule so that the forwarding paths can be established promptly, without needing to interpret the arriving packets. This, however, does not work in COTS Ethernet switches. Examples of these approaches include TT-Ethernet and PROFINET-IRT.

EDF scheduled switching

was proposed by Hoang *et al.* [62] [61] to support a mix of real-time (RT) and non-real-time (standard IP) traffic coexisting in a switch-based Ethernet network. The RT traffic is scheduled according to the Earliest Deadline First policy and its timeliness is guaranteed by means of adequate online admission control.

The proposed system architecture requires the addition of a real-time layer (RT-l) on network components, either to the end nodes or to the switch. The RT-l is responsible for establishing real-time connections, performing admission control, providing time synchronization, and finally managing the message transmission and reception of both real-time and non-real-time traffic classes.

The switch RT channel management layer provides time synchronization by transmitting periodically a time reference message. Moreover, this layer also takes part in the admission control process, both by assessing the internal state of the switch, and consequently its ability to fulfill the timeliness requirements of the real-time message streams, as well as by acting as a broker between the nodes requesting RT channels and the targets of such requests. Finally, this layer also disseminates the internal switch state,

namely in what concerns the queues status, to allow flow-control of non-real-time traffic on the end nodes.

Real-time communication is carried out within real-time channels, a point-to-point logical connection with reserved bandwidth. Whenever a node needs to send real-time data, it issues a request to the switch, indicating both the source and destination addresses (both MAC and IP), and the period, transmission time and deadline of the message. Upon reception of such a request, the switch performs the first part of the admission control mechanism, which consists in evaluating the feasibility of the communication between the source node and the switch (uplink) and between the switch and the target node (downlink). If the switch finds the request feasible, forwards the request to the destination node. The target node analyzes the request and informs the switch about its will on accepting or not the real-time connection. The switch, then, forwards this answer to the originator node. If the RT channel is accepted, it is assigned with a system wide channel ID that univocally identifies the connection.

The real-time layer comprises two distinct queues, one for the real-time traffic, and the other for the non-real-time traffic. The former is a priority queue, where messages are kept sorted by distance to their deadlines. The non-real-time queue holds the messages in a FIFO queue. Therefore, real-time messages are transmitted according to their deadlines, while non-real-time messages are transmitted according to their arrival instant.

The feasibility analysis proposed [61] is derived from EDF task scheduling analysis, but with adaptations to account for some system specifics, such as including the overheads due to control messages and the impact of non-preemptive message transmission.

In the scope of that work, deadlines are defined on an end-to-end basis. Since the traffic is transmitted in two separate steps (uplink and downlink), the analysis must assure that the total delay induced by these steps together does not exceed the total end-to-end deadline. For a given real-time message stream i, if d_{iu} is the deadline for the uplink and d_{id} the deadline for the downlink, then the end-to-end deadline d_{iee} must be at least as large as the sum of the previous two: $d_{iu} + d_{id} \leq d_{iee}$. In [62], the authors assume the end-to-end deadline equal to the period of the respective message stream, and a symmetric partitioning of that deadline between the uplink and the downlink. An improvement is presented in [61], where the authors propose an asymmetric deadline partition scheme. Although more complex, this method allows a higher efficiency in bandwidth utilization, because a larger fraction of the deadline can be assigned to more loaded links, thus increasing the overall schedulability level.

EtheReal

[132] is another protocol that was proposed to achieve real-time behavior on switched Ethernet networks. In this approach, the protocol is supported by services implemented on the switch, only, without any changes in the operating system and network layers of end nodes. The switch services are accessible to the end nodes by means of user-level libraries.

EtheReal has been designed to support both real-time and non-real-time traffic via two distinct classes. The Real-Time Variable Bit Rate service class (RT-VBR) is meant to support real-time applications. These services use reserved bandwidth and try to minimize the packet delay and packet delay variation (jitter). Applications must provide the desired traffic characteristics during the connection set-up, namely average traffic rate and maximum burst length. If these parameters are violated at run-time, the real-time guarantees do not hold, and packets may be lost. The second service class is Best-Effort (BE), and it was developed specifically to support existing non-real-time applications like telnet, http, etc., without requiring any modification. No guarantees are provided for this type of traffic.

Real-time services in EtheReal are connection-oriented, which means that applications have to follow a connection setup protocol before being able to send data to the real-time channels. The connection setup procedure is started by sending a reservation request to a user-level process called Real-Time Communication Daemon (RTCD), running on the same host. This daemon is responsible for the set-up and tear down of all connections in which the host node is engaged in. The reservation request for RT connections contains the respective Quality of Service (QoS) requirements, namely average traffic rate and maximum burst length.

Upon reception of a connection set-up request, the RTCD contacts the neighbor EtheReal switch that evaluates whether it has enough resources to meet the QoS requirements of the new RT connection without jeopardizing the existing ones, namely switch fabrics bandwidth, CPU bandwidth for packet scheduling and data buffers for packet queuing. If it has such resources and if the destination node is directly attached to the same switch, it positively acknowledges the request. If the destination node is in another segment, i.e., connected to another switch, the switch that received the request forwards it to the next switch in the path. A successful connection is achieved if and only if all the switches in the path between the source and the target node have enough resources to accommodate the new RT connection. If one switch has not enough resources, it sends back a reject message, which is propagated down to the requester node. This procedure serves to notify the requester application about the result of the operation, as well as to let the intermediate EtheReal switches to de-allocate the resources associated with that connection request.

The EtheReal architecture employs traffic shaping and policing, within

both hosts and switches. The traffic shaping is performed to smooth the inter-packet arrival time, generating a constant rate flow of traffic. Traffic policing is used to ensure that the declared QoS parameters are met during run-time. Those functions are also implemented on the switches to ensure that an ill-behaved node, either due to malfunction or malicious software, does not harm the other connections on the network.

With respect to the packet scheduling inside the switch, the EtheReal architecture employs a cyclic round-robin scheduling algorithm. All real-time connections are served within a predefined cycle. A part of that cycle is also reserved to best-effort traffic, to avoid starvation and subsequent time-outs on the upper layer protocols.

Applications access the real-time services by means of a Real-Time Data Transmission/Reception library (RTTR), which provides services for connection set-up and tear down and data transmission and reception, beyond other internal functions already referred to, such as traffic shaping and policing.

Another interesting feature of this protocol is its scalability and high recovery capability, when compared with standard switches. For example, the spanning tree protocol (IEEE 802.1D) is used in networks of standard switches to allow redundant paths and automatic reconfiguration upon a link/switch failure. However, such reconfiguration may take several tens of seconds with the network down, typically around 30 seconds, which is intolerable for most real-time applications. On the other hand, the authors claim that EtheReal networks may recover nearly three orders of magnitude faster, within $32ms$ [131].

Traffic shaping

As opposed to transmission control, that strictly enforces the transmission times for each message on the network, this technique follows an approach based on limiting burst. Therefore, the probability of long priority inversions in the queues is kept low.

This is a fully distributed approach technique. In each network station, an interface layer called traffic smoother is placed just on top of the Ethernet driver. This layer handles and controls the message transmission rate in each node to limit bursts. Real-Time traffic is assumed to be well-behaved, i.e., periodically triggered at some control process rate, thus without bursts and so handed to the Ethernet driver right away, bypassing the smoother.

Conversely, the non-real-time (NRT) traffic can be bursty and has to go through the traffic smoother that forces an average transmission rate and a maximum burst length. This way, at the network level, the interference on the RT traffic due to NRT traffic is kept inside a bound [78].

One major drawback of this approach is that the smoothers are all distributed and it is difficult to adapt them online. In fact, adaptive techniques such as proposed in [70] and [41] only work in shared Ethernet. When using

switches, nodes have no information on the load in the output ports for which they send and thus cannot adapt the level of smoothing required at each instant. With switches, such adaptation must be done inside the switches themselves.Moreover, this approach lacks explicit support for time-triggered traffic.

AFDX (Avionics Full DupleX switched Ethernet) [15, 16, 17] is a network communication specification, initially defined to cope with the constraints set by the avionic industry to deploy a secure, reliable and deterministic network integrating a number of system modules, already in place but using independent networks, namely ARINC 429 buses [30]. AFDX defines communication channels (VL - Virtual Link) that establish logical unidirectional connections between one source and one or more destinations, thus able to abstract an ARINC 429 style network and support legacy modules. Each VL is statically defined and characterized by a Bandwidth Allocation Gap (BAG), i.e., a minimum delay between consecutive frames and by a minimum and maximum frame lengths (s_{min} and s_{max}). The enforcement of this characteristics is conducted in the emitting nodes of each VL by means of traffic shapers that provide a deterministic communication behavior that suits the T-SPEC model of network calculus (described in the following section). Then, in offline the necessary communication guarantees are provided using the network calculus that although pessimistic, easily provides deterministic upper-bounds to each communication flow. Additionally, dedicated AFDX switches enforce filtering, policing and forwarding functions based on the VLs, which are statically defined. AFDX is thus not able to modify the network requirements online and so not able to meet flexibility.

2.3.3 Schedulability analysis

Despite the absence of collisions on a switched Ethernet network, providing real-time communication services is still not a trivial task due to congestion in the output port queues. The switch aggregates the traffic from different input ports in its output ports. For that purpose, it commonly uses FIFO queues that generate substantial jitter in the outgoing pattern, degrading the system real-time performance. On this regard, real-time protocols demand for techniques to estimate the system performance. For packet switching networks and particularly for switched Ethernet those analysis should provide bounds to the queuing delay as well as to the depth of the queues in the switch.

In the remainder of this section we present two typical analysis techniques that provide real-time guarantees in worst-case scenarios for real-time communications over switched Ethernet namely Network Calculus (NC) and Response Time Analysis (RTA).

Network Calculus (NC)

Network Calculus, initially introduced by Cruz [45, 46] and later complemented by Boudec and Thiran [34], describes an algebra for analyzing performance in computer networks. It uses the Min-plus and Max-plus algebra to assess the impact of network communication constraints such as the limited links capacity, traffic congestion in the switch, interference from background traffic and traffic shaping policies enforced in the nodes. The algebra relies on a traffic characterization model that defines streams by their burstiness peak (σ) and their long-term rate (ρ), called T-SPEC.

In the real-time community, some solutions based on the NC emerged to target hard-real-time applications over SE. Loeser and Haertig [78, 79] use a fine grained traffic shaping in the nodes to guarantee that the injected traffic conforms with the T-SPEC model parameters used within the NC, so that all analysis follows naturally from that algebra. For a generic real-time job i, modeled with a W_i budget and an arrival period of T_i, the T-SPEC arrival pattern is denoted by: $\sigma_i = W_i$,$\rho_i = W_i/T_i$. Once all traffic is characterized with (σ_i, ρ_i) i=1...N, the queuing delay for a c bit/s server comes as follows [45]:

For fixed priorities:

$$D_{i\ max} = \frac{\sum_{j=1}^{i} \sigma_j + max_{i+1 \leq j \leq N}(W_j)}{c - \sum_{j=1}^{i-1} \rho_j}$$

For FCFS policy:

$$D_{i\ max} = \frac{\sum_{j=1}^{N} \sigma_j + max_{i+1 \leq j \leq N}(W_j)}{c - \sum_{j=1}^{N-1} \rho_j}$$

Once the traffic is fully characterized according to T-SPEC the timeliness analysis of a stream traversing the network is straight-forward. However, this technique only applies when the application streams conform with the T-SPEC model and it is not trivial to deduce the T-SPEC for certain traffic scenarios that are highly jittered.

Response time analysis (RTA)

Section 2.1.3 described several scheduling analysis techniques for computing tasks. The response time analysis provides relatively accurate results on whether a task set is schedulable or not. Section 2.2.2 then presented ways to map a traffic model in order to use the analysis developed in this context. For example, Koubâa and Song [115] conduct a response-time-based analysis for the traffic traversing an Ethernet switch. Their work is only focused on the switch queues. The impact of the nodes queues is modeled as release jitter affecting the arrival pattern of the packets to the switch queues. However,

nothing is said concerning how to determine this release jitter, nor how to handle different streams coming from the same node with a scheduling policy different than the one in the switch. Therefore, it only addresses the impact of the FIFO queuing policy of the switch.

The queuing delay (Rt_i) that affects a job of stream i with respect to its periodic activation, is thus upper-bounded through the usual iterative process for the case of a non-preemptive fixed priority queue:

$$I_i^{n+1} = \max_{i \leq j \leq N}(C_j) + \sum_{j=1}^{i-1} \left(\left\lfloor \frac{I_i^n + J_j}{T_j} \right\rfloor + 1 \right) \times C_j$$

$$Rt_i = C_i + I_i + J_i$$

where J_i stands for the release jitter that stream i might suffer when arriving at the queue, which is the maximum deviation from its periodic activation, and I_i stands for the maximum interference. Notice that we also assume that Rt_i should be no greater than T_i.

For an FIFO queue, the queuing delay is obtained in the following way, if the release is strictly periodic:

$$I_i = \sum_{j=1, j \neq i}^{N} C_j$$

$$Rt_i = C_i + I_i$$

If the streams suffer release jitter the analysis is rather more complex as we will see further on.

Xing Fan *et al.* conduct a wider covering analysis that includes both, the queues at the sending nodes and the queues in the switch. Both queues are analyzed independently and the overall end-to-end delay is considered adding up both stages. The first stage analysis (node queue) is easily achieved estimating the interference caused by the streams within the same node. As for the second stage, the analysis becomes more complex in which it is necessary to consider all the streams that target an output link in the switch. In this case the analysis is performed over the load submitted to the switch queue.

The cumulative workload $W_k(t_1, t_2)$ of a set of real-time streams $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ originating from source node k, is given by the sum of the traffic volume of messages released by the real-time streams during the time interval $[t_1, t_2), \forall i, t_1 \leq t_2$:

$$W_k(t_1, t_2) = \sum_{i=1}^{n} \max \left(\left(\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor + 1 \right) \times C_i, 0 \right)$$

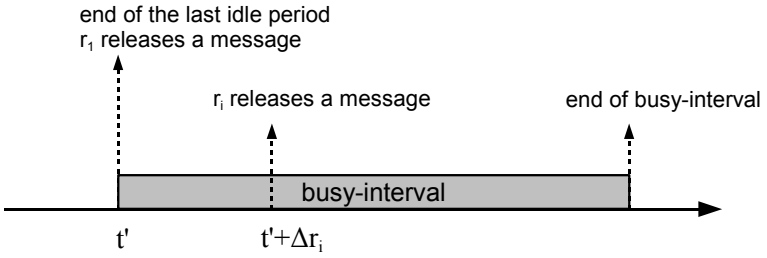


Figure 2.13: Arbitrary release in k's busy-interval.

For the switch queue analysis, given any message release pattern, without loss of generality, assume that τ_i releases messages at $t = t' + \Delta r_i, \Delta r_i \geq 0$, as illustrated in Figure 2.13. The commutative workload for the incoming link from node k during $[t', t)$ is derived as:

$$W_k(t', t) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{t - (t' + \Delta r_i)}{T_i} \right\rfloor\right) \times C_i$$

This estimates the busy-interval for node k and so the real-time guarantees can be assessed from it.

NC versus RTA

The major difference between the two system analysis techniques, network calculus and response time analysis, resides in the communication model on which each rely. The NC uses the T-SPEC model that defines a long-term average transmission rate and a maximum burst that better suits the event-based traffic, whereas, the RTA evaluates the system feasibility by evaluating the interference that occurs on a periodic activation basis.

The T-SPEC model, despite its limitations, is very general and allows modeling large range of real traffic scenarios. However, in some cases a more precise but more limited model can be used such as in control-oriented or multimedia applications in which a periodic activation model is natural. A relationship method to map the periodic model onto the NC model is proposed in [115]. Using the pseudo-periodic model (periodic with release jitter), the authors conduct a work for fixed priorities scheduling inside the switch in which two queuing delay analysis are compared, the NC algebra and the classical RTA. They conclude by saying that RTA analysis is less pessimistic than the use of NC with the penalty of being more computationally demanding.

2.4 Conclusion

Ethernet is the most popular technology for LANs today. Due to its low cost, high throughput, high availability and easy deployment, among other features, Ethernet is wide spread in multiple application domains, even those for which it was not initially designed. An example is industry automation in which Ethernet is used in a variety of applications, including those that impose critical real-time constraints. The CSMA/CD arbitration protocol used in shared Ethernet, which conflicts with the requirements for timing, was bypassed introducing switched Ethernet that additionally provides full-duplex communication and allows multiple parallel forwarding paths. Despite the absence of collisions, it is still necessary to control the traffic admission in the switch queues. Several real-time protocols emerged for this purpose in order to obtain the timeliness guarantees.

This chapter presented a brief overview of real-time systems modeling with focus on real-time communications. It introduced some background concerning real-time operating models, scheduling and the necessary analysis techniques that support timeliness guarantees. Specifically concerning the real-time communications deployment over switched Ethernet, this chapter described a communication model suited to characterize the switch and covered some paradigmatic techniques to control the traffic admission to prevent overflow in the switch queues, while supporting the designation of worst-case delay bounds.

Chapter 3

The FTT-SE protocol

This chapter presents the Flexible Time Triggered protocol over Switched Ethernet (FTT-SE). It describes the overall system architecture, including the mechanisms that support it, and presents a formulation for the global traffic scheduling problem.

The chapter includes a section with some details of an hardware implementation and the distributed middleware abstraction.

It, finally, presents a set of simulations and experimental results conducted to illustrate and validate the advantages of using the FTT-SE.

3.1 Introduction

As mentioned in Section 1.1, there is a trend towards the development of flexible protocols as a mean to efficiently address dynamic and evolving environments. The protocol hereby proposed for switched Ethernet is based on the Flexible Time-Triggered communication paradigm to enforce such operational flexibility. At the switching level, it globally coordinates the traffic submitted at each instant, avoiding potential queue overflow problems and bypassing the traffic scheduling performed by the switch allowing thus, any desired policy. The FTT-SE (FTT over Switched Ethernet), follows the same paradigm that has already been used over shared Ethernet to overcome the non-determinism of its MAC, namely the FTT-Ethernet protocol [101].

Therefore, it is proposed an adaptation of that protocol to micro-segmented networks, i.e., based on switches and with only one station connected to each port. As common to all FTT implementations [100], the main advantages of this protocol are the global traffic coordination in a common timeline, the possibility for fast and atomic online updates to the set of streams, the capability to support wide ranges of streams periods, the support of arbitrary traffic scheduling policy and the capability of handling periodic, aperiodic and non-real-time traffic with temporal isolation.

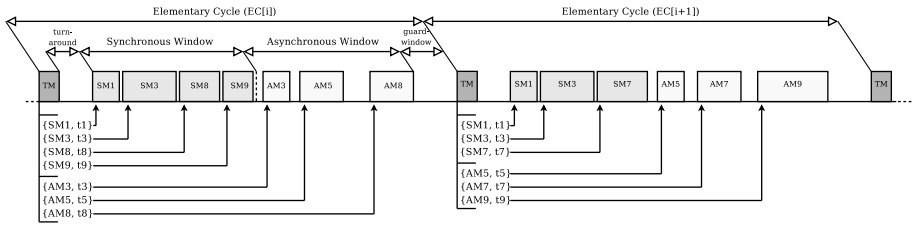


Figure 3.1: The EC structure in the original FTT-Ethernet.

3.2 FTT-SE: An enhancement of FTT-Ethernet

As discussed in Section 2.3.2, there are several ways to achieve real-time communication over switched Ethernet. However, some of them are based on non-standard hardware, a solution that conflicts with some of the key arguments supporting the use of Ethernet in real-time applications (cost, availability, compatibility with general purpose LANs). Therefore, we focus on COTS-based solutions, only, but still aiming at a high level of traffic control toward more predictable timing behavior. Particularly, we propose adapting FTT-Ethernet, originally developed to operate over shared Ethernet, to achieve tighter traffic control than with existing solutions based on COTS switches, e.g. Ethernet/IP or traffic shaping. The use of the FTT architecture brings other important benefits such as the support for arbitrary traffic scheduling policies, priority levels beyond the eight levels specified in IEEE 802.1D, offsets among streams, online admission control and bandwidth management and, finally, prevention of memory overflows inside the switches. On the other hand, FTT-Ethernet is still a master/slave protocol and, as such, introduces an additional overhead caused by master polls. However, the FTT architecture employs an improved technique, called master/multi-slave, according to which the master addresses several slaves with a single poll, considerably alleviating the protocol overhead.

Brief review of FTT-Ethernet

FTT protocols [100] organize the communication in fixed duration slots called Elementary Cycles (ECs), which are triggered with a master message called Trigger Message (TM), containing the periodic schedule for that EC. The periodic messages, called synchronous, are synchronized with the periodic traffic scheduler. The protocol also supports aperiodic traffic, called asynchronous, which is managed in the background, in the time left within the EC, after the periodic traffic (Figure 3.1).

The traffic scheduling activity is carried out online and centrally in the master and the traffic schedules are disseminated by means of the TM. Since the traffic scheduling is local to one node, it is easy to enforce any kind of scheduling policy as well as perform atomic changes in the communica-

tion requirements. This last feature allows for online stream admission and removal under guaranteed timeliness as well as online bandwidth management. Similarly to the freedom with respect to the traffic scheduling policy, the specific bandwidth management scheme can also be any. These features are the kernel of the FTT paradigm and are the justification for the *flexible* attribute.

The EC schedules are built considering a broadcast transmission model as common in shared segments. Moreover, in order to bypass the non-deterministic CSMA/CD arbitration of Ethernet, the EC schedules also contain an offset from the start of the EC for the transmission of each message that grants a collision-free medium access. The slaves transmit the scheduled messages with the specified offsets and for an efficient use of the bandwidth such offsets must be respected with a good precision.

3.2.1 FTT-SE for micro-segmented networks

FTT-Ethernet can be seamlessly deployed over shared or switched Ethernet networks but it uses a full broadcast communication model. Important efficiency gains may be achieved by tailoring the FTT-Ethernet protocol to take advantage of the distinctive features of micro-segmented topologies, namely the absence of collisions and the existence of parallel transmission paths.

The inherent absence of collisions that results from the existence of private collision domains for each port leads to a noteworthy simplification of the protocol implementation in the slave nodes, which no longer need to enforce a collision-free medium access. Messages are transmitted immediately after decoding the TM, with the switch taking care of the serialization within each EC. Consequently the contents of the TM itself is also simplified, since the specification of the transmission instants is no longer needed.

On the other hand it becomes possible to take full advantage of multiple transmission paths by abandoning the pure broadcast architecture of FTT-Ethernet as long as we provide the FTT Master with information about the nature of the data exchanges, i.e., unicast, multicast and broadcast, and which end nodes are involved. With this information the master can compute which messages follow disjoint paths, i.e., non-overlapping source and destination nodes, and thus build schedules that exploit this parallelism, increasing the aggregated throughput.

This new feature corresponds to move from the broadcast-based producer/consumer cooperation model in FTT-Ethernet to a multicast publisher/subscriber scheme in FTT-SE. The master keeps a data structure with the currently existing groups of publisher/subscribers, with the identification of the respective streams and the associated physical addresses and ports. Specific calls issued by the publishers and subscribers allow creating groups and binding nodes to groups. Two different cases must be considered according to the type of switches uses. For non-multicast switches only unicast

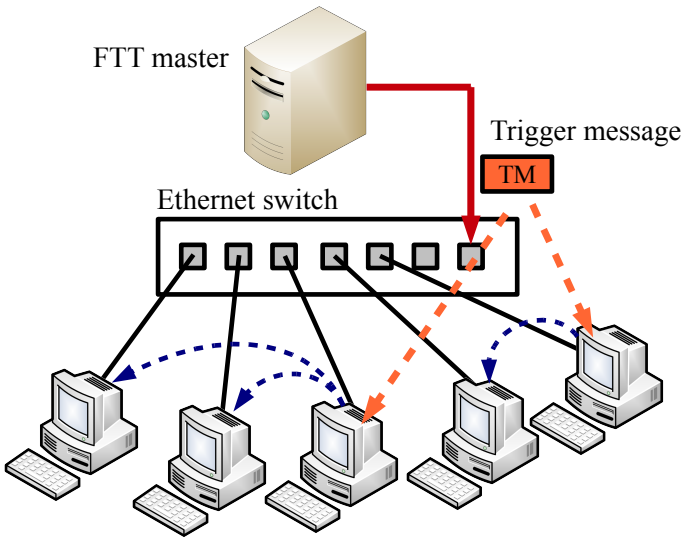


Figure 3.2: FTT-SE system architecture.

and broadcast streams can be considered. For true multicast switches the standard Internet Group Multicast Protocol (IGMP, RFC 2236) is used to setup up multicast groups. The binding process for subscribers uses IGMP messages sent explicitly to the FTT Master, which are also snooped by the switch, allowing both master and switch to build coherent forwarding tables. The master must be correctly configured to the type of switch being used.

Figure 3.2 shows the communication system architecture, with the FTT Master attached to one switch port and scheduling the transmission of the remaining stations.

The previous FTT-Ethernet protocol manages two distinct traffic classes, the periodic traffic and the aperiodic traffic. They have different scheduling models, thus being managed separately. However, in both classes the transmission is issued only upon a dispatching order that polls the messages cycle by cycle. For the periodic traffic, polling is natural since the master knows when to transmit as well as the bus status. However, the master does not have such knowledge concerning the asynchronous traffic, since it is triggered by local events in the slave nodes. Thus, asynchronous messages are blindly polled by the master, which may result or not on a message transmission, depending on whether there is a pending request.

In the case of FTT-SE, the management of both types of traffic is substantially simplified and enhanced in several aspects.

Concerning the periodic traffic, all slaves answer the master polls in the TMs transmitting as soon as possible the scheduled traffic, i.e., after the turn-around time. The precise transmission instants inside the synchronous window are not known since they depend on the speed of the slaves answer-

ing the poll. However, the scheduled periodic traffic will be transmitted in a burst, or nearly, with practically no wasted intervals between consecutive messages. This was not the case in FTT-Ethernet in which the specification of the transmission offset could lead to such wasted intervals due to limitations in timing resolution of the operative system in the slaves.

Concerning the aperiodic traffic, its management is also substantially improved with respect to FTT-Ethernet due to a new signaling mechanism.

3.2.2 Handling aperiodic transmissions in FTT-SE

The asynchronous management is a problem for a variety of protocols seeking low communication jitter and latency in switched Ethernet. Unconstrained aperiodic communication may generate bursts that fill in the output queues, leading to long priority inversions in typical FIFO queues and possibly to queues overflow and consequent packet losses. One way to improve this situation is constraining the transmission of aperiodic traffic in the nodes using traffic shaping or smoothing. This way, transmission instants are not constrained but the amount of traffic generated within a given time window is bounded. Alternatively, a more robust and timely accurate but less efficient mechanism, is the one used originally in FTT-Ethernet, based on polling. In this case, the transmission instants are adequately planned by the global scheduler. However, this approach is not very efficient given the possible long latency periods to serve the aperiodic requests, which result from the periodic polling and also the bus bandwidth that is wasted by unsuccessful polling attempts.

The polling mechanism encompasses two different phases. In the first one, called *signaling phase*, the master inquires the nodes about the existence of asynchronous traffic ready to be transmitted. In the second one, called *transmission phase*, the master polls the transmission of that traffic in adequate instants in time. These two phases can either be separately implemented, where the *transmission phase* comes as consequence of the *signaling phase* in which case the *transmission phase* only occurs if the *signaling phase* indicated the presence of pending aperiodic traffic, or merged in a single periodic poll that leads to unused bandwidth when no messages are pending at the poll instant.

Note that, whether the two phases are jointly or separately implemented, in both there is an intrinsic compromise between responsiveness and overhead; a higher responsiveness requires polling at higher rates, but polling at higher rates potentially implies a higher bandwidth overhead.

To improve the aperiodic traffic management, we propose exploring the full duplex features of common Ethernet switches to implement a new signaling mechanism in master/slave switched Ethernet protocols that does not suffer from the referred responsiveness versus overhead compromise and may dramatically improve the QoS given to the asynchronous traffic.

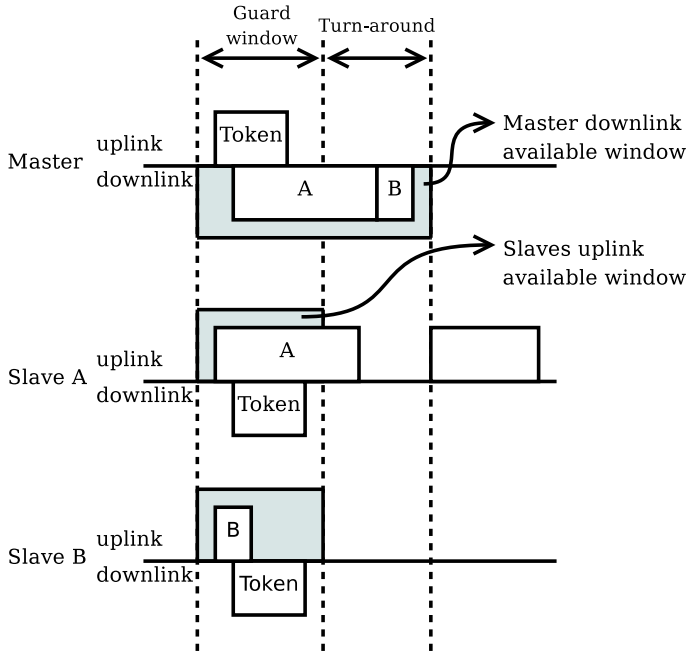


Figure 3.3: Polling token.

The out-of-band signaling mechanism

Switches with full-duplex capabilities provide the possibility of simultaneous transmissions in opposite directions in each link without interference. This feature is used to efficiently deploy a signaling path between the slaves and the master in a master/slave architecture that does not collide with the common polled transmissions.

Another property intrinsic to a master/slave architecture is the overhead that comes in two forms, one due to the need to transmit the *master poll* and another caused by the time spent by the slaves decoding and preparing the reply to the poll, which is normally called the *turnaround time*. This interval of time, which mediates between the poll reception and the beginning of the reply transmission, depends on the slave implementation technology. For standard PC-based solutions and network interface cards (NICs) can reach $200\mu s$ [83], even using RT kernels and stacks. Reducing this value to a few tens of microseconds is possible with special hardware support, only.

Yet another common property of master/slave architectures is the strict slaves synchronization on the master tokens. This behavior is typically achieved reserving a window (guard window) for transmitting the token with no interfering traffic nor software blocking activities in the nodes, thus avoiding jitter.

In a full-duplex switch, the master download link, i.e., the link that

transmits from the switch to the master, is not included in the poll token forwarding path. Therefore, messages transmitted in unicast to the master do not interfere with the actual token message transmission (see Figure 3.3). The slave nodes may safely report to the master node the internal state of their asynchronous queues during the guarding and the turn-around windows, provided that these messages finish transmission within the turn-around window.

Regarding the computing requirements within the slaves, there is also no interference whatsoever on the regular protocol messages and the slaves turn-around time is also not affected, since the report messages are submitted to the NIC during the transmission of the poll message by the master. The NIC seamlessly handles both transmission and reception with no interference. Then, when it comes to receive and decode the TM, the processor is no longer in use by the transmitting handler, thus no interference. Figure 3.3 shows one situation in which two nodes send their status messages (A and B) to the master. Both messages are completely received within the time interval defined by the poll message transmission plus the turn-around windows, thus not interfering with the normal protocol operation.

Thus, the mechanism herein proposed relies on this transmission scheme to handle the backward signaling information containing the asynchronous queuing status in the slaves. It can be applied to any master/slave full-duplex switched Ethernet protocol, as long as the slaves are able to synchronize with the transmission instant of the following master polls.

Signaling of asynchronous traffic in FTT-SE

Within the FTT-SE protocol, the master token is the Trigger Message (TM), transmitted periodically, polling the slaves for messages in that cycle, both synchronous and asynchronous. There is one TM heading each *Elementary Cycle* (EC), the interval that defines the system periodic granularity. The TM is followed by the turn-around time, the Synchronous and the Asynchronous windows. Figure 3.4 sketches the EC structure with an example of the signaling scheme herein proposed.

The signaling channel uses the reverse path of the periodic TM, virtually avoiding interference with the normal protocol operation. The slaves are thus required to synchronize with the previous TM in order to trigger the signaling message properly, i.e., under the following TM from the master (Figure 3.4).

Each signaling message includes information regarding the asynchronous queues status of the associated node and shall not use more than the minimum payload required for an Ethernet frame, for the sake of fitting as many messages within the signaling window as possible. The limited size of the signaling window may constrain the number of nodes, consider that each node can send one signaling message per EC.

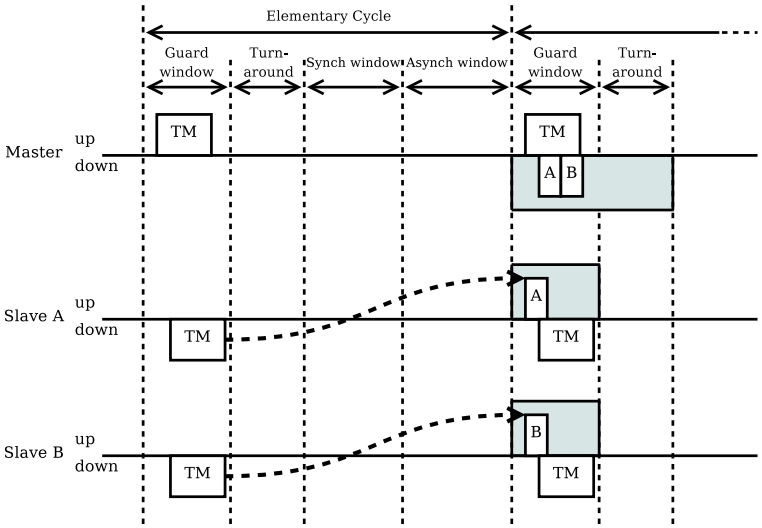


Figure 3.4: FTT-SE signaling proposal.

The turn-around time plays a significant role in this scheme. In FTT-SE this parameter is configurable and must be properly tuned according to the node's performance. Typically it must be set to a value not lower than the worst-case turn-around time among all the nodes present in the system.

The following equation allows computing the maximum number of nodes ($MaxN$) with respect to a system s with a given turn-around time, Tr:

$$MaxN(s) = \frac{Tr(s) + TM\_size(s)}{SIG\_size(s)}$$

where $TM\_size(s)$ stands for the trigger message transmission time and $SIG\_size(s)$ for the signaling message transmission time. For a Fast Ethernet network (100 Mbps) $TM\_size(s)$ equals $24\mu s$ and $SIG\_size(s)$ takes the minimum Ethernet frame transmission time, $6.72\mu s$. Assuming also a system based on standard PCs with regular NICs and using an RT kernel and stack. In such case, the turn-around time (Tr) can be bounded to $200\mu s$ [83], leading a maximum number of signaling messages, and thus of nodes, of $MaxN = 33$.

For applications requiring more nodes or exhibiting lower turn-around times (specialized hardware) two approaches can be used to avoid this scalability constraint. One is to reduce the rate at which nodes issue the signaling messages e.g., only once every 'n' cycles. This approach has a negative impact on the signaling latency but allows supporting an arbitrary number of nodes. Another possibility would be to enlarge the signaling window in the master downlink beyond the turn-around window, i.e., allow the signaling messages to use the synchronous exclusive window. In many cases this approach is feasible since most of the periodic traffic is unicast or multicast,

thus not being forward to the master, leaving the downlink of the master node partially unused. In this case, this extra traffic in the master downlink must also be considered by the synchronous messages scheduler.

3.3 The scheduling model

The scheduler specifies the order with which the messages are transmitted in the network, once activated. The master node handles such operation taking into account the individual priorities of each message that may either be static or dynamic as in EDF scheduling. We have seen that the scheduling plan is enforced by means of the TM that polls the messages in the slaves.

Such model supports strict priority scheduling but at a coarse time scale, only, with a resolution of ECs. Priority inversion may occur within the EC since the master does not control the specific instants at which the messages are transmitted by the slaves and queued in the native FCFS queues within the switch output ports.

3.3.1 The periodic traffic scheduling model

The periodic scheduling model considers N_s periodic streams (SM_i) that are stored in a structure called Synchronous Requirements Table (SRT) as shown in 3.1.

$$SRT = \left\{ SM_i : SM_i = (C_i, D_i, T_i, O_i, Pr_i, S_i, \{R_i^1..R_i^{k_i}\}), i = 1..N_s \right\} \quad (3.1)$$

Each stream is characterized by the total transmission time of each instance C_i, the relative deadline D_i, the stream period T_i, the offset O_i. Pr_i is an optional parameter that allows associating explicitly a priority to the message. Both D_i, T_i and O_i are expressed as integer numbers of ECs. Then, S_i is the sender node and $\{R_i^1..R_i^{k_i}\}$ is the set of k_i receivers for stream i. The calculation of C_i deserves a special note because the protocol automatically fragments large messages in a sequence of packets that are scheduled sequentially and individually by the master. This is particularly useful to transmit regularly large amounts of data such as video frames.

The set of streams in the SRT is scheduled by the FTT Master according to any online policy, implementing a single queue of ready periodic messages. This queue is used to build the EC-schedule that will be encoded in the TM and broadcast through the switch. The periodic scheduling though confines the traffic to the synchronous window, with duration of LSW, within the EC.

The reception of the TM causes all nodes that are senders in that EC to feed the scheduled streams to the switch through a set of M upload links l_j^u. When these streams arrive at the switch they are conveyed, with latency ε, to

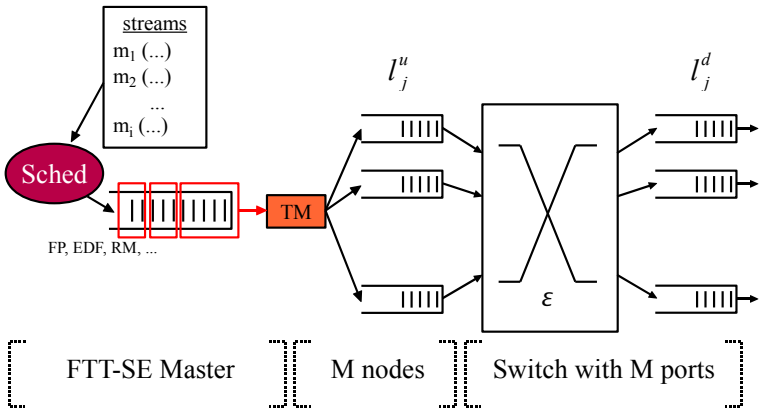


Figure 3.5: The scheduling model with FTT-SE.

the respective output ports and queued for transmission in the M download links l_j^d (Figure 3.5). The transmission of each packet is non-preemptive, as usual, but the transmission of long messages, i.e., those with multiple packets per instance, can be preempted between packets.

Scheduling over this model becomes a two-fold problem. In one hand, it is necessary to build the EC-schedules so that the transmission of the periodic messages fits within the synchronous window of that EC, while in the other hand, it is important to to have schedulability bounds adequate to the scheduling policy.

The former issue is covered in the remainder of this chapter, including scheduling multiple links simultaneously, while the schedulability bounds are addressed in the following chapter.

3.3.2 Building EC-schedules

One issue that needs to be addressed in order to carry out the traffic scheduling online is how to build the EC-schedules considering the multiple queues associated to the M upload and download links.

The first aspect to note is that most of the current switches are full-duplex, which permits overlapping transmissions in the download and upload links. Additionally, switches are many times capable of performing cut-through forwarding, from the uplinks to the downlinks with a small latency shift, the switch latency ε.

The second aspect is the initial constraint that limits the communication activity to the synchronous window, whose maximum duration is LSW, meaning that no link can be used more than $LSW - \varepsilon$ (Figure 3.6). The turnaround time tr is the time needed by the stations to decode the TM and start their own synchronous transmissions.

A third aspect is that the transmissions in the downlinks are causal with

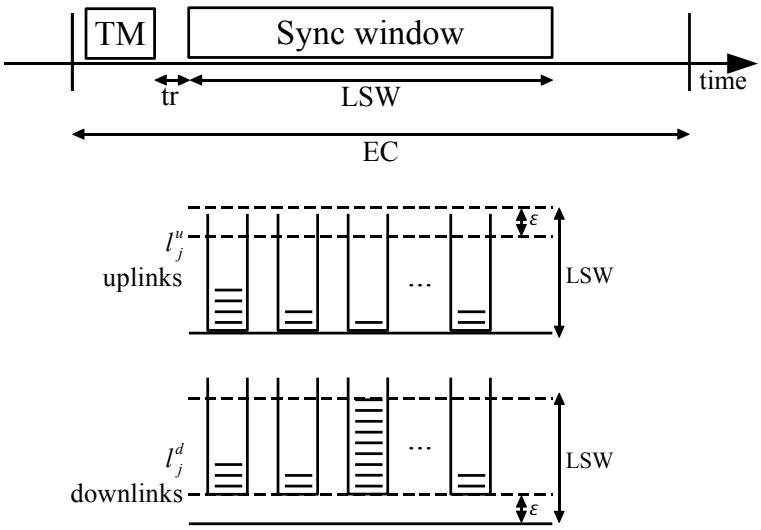


Figure 3.6: Constraining the synchronous traffic to the synchronous window.

respect to those in the uplinks and thus must always occur after them, at least an amount of time ε. This means that, if a set of packets arrive close to the end of the synchronous window at a given queue, their transmission might extend beyond the end of that window, even if the total load in that downlink takes less than LSW. Therefore, when constraining the traffic in the downlinks to LSW it is necessary not only to account for the transmission duration but also for the respective transmission instants.

This leads to the case of a modified bin packing problem, where each link is represents a bin that accumulates the transmission times of the respective streams, while constructing of the EC-schedule. For the uplinks, considering that the nodes are able to transmit sequentially, immediately after tr, the EC-schedule filling procedure is only required to check the load in each bin until the $LSW - \varepsilon$ threshold. As for the downlinks, as referred above, the finishing instant of the latest transmission (f) must be considered and checked not to override the transmitting window (Figure 3.6) set to (LSW). To determine such instant, it is necessary to keep track of the transmission instants of the previous packets sent in that queue, detect whether there is an overlap and add the respective transmission time (Figure 3.7). The approach followed to determine the instant f considers the traffic submitted to the switch in priority order in each uplink, which is easy to enforce in the slaves. Knowing this order, it is straightforward to detect overlapping transmissions in the downlinks and act properly. Note that the specific order with which the overlapping transmissions are serialized is not relevant since only the end of the transmissions of each group of messages matters.

Given these constraints, the EC scheduling plan is built probing each message in the ready queue, one by one, until any of the limits is overridden,

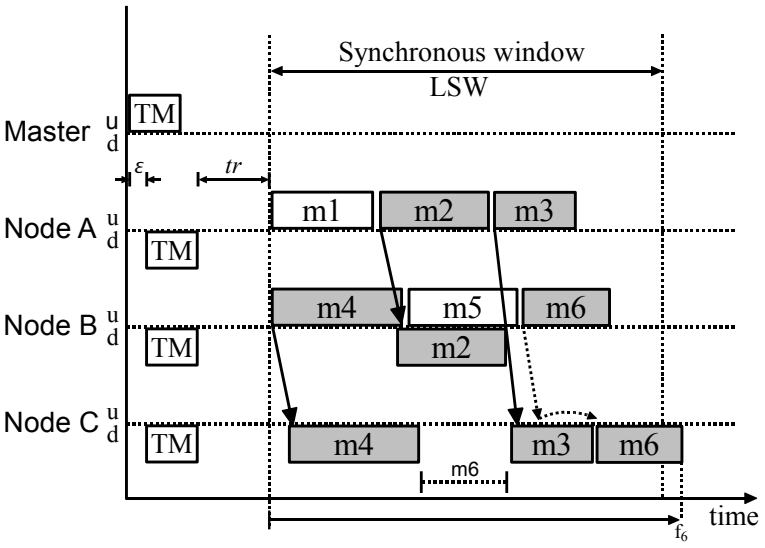


Figure 3.7: Causality constraint in the downlinks.

in the uplinks or downlinks. If including a message overrides a link boundary, that message is held back in the queue and the specific link is closed, i.e., no more messages are assigned to it in this EC. The EC-schedule building proceeds, checking the remaining messages in the ready queue until no more message can be scheduled. This procedure copes with multiple flows with different forwarding paths being handled simultaneously. The condition that stops the EC-schedule construction in each EC is given in 3.2, where the top condition concerns the uplinks and the one below concerns the downlinks. Basically, the scheduler includes in the EC-schedule all ready messages that is possible without violating condition 3.2.

$$\begin{cases} \max\limits_{j} \left(\sum\limits_{SM_i \in l_j^u} C_i \right) \leq LSW - \varepsilon \\ \max\limits_{j} \left(\max\limits_{SM_i \in l_j^d} (f_i) \right) \leq LSW \end{cases} \tag{3.2}$$

Another characteristic that emerges from this scheduling approach is that the memory requirements for any switch port μ_j^p and for any node μ_j^n (at the network driver level), are known and bounded as defined in bytes by 3.3, where r stands for the links transmission rate in bits per second, considered equal for all.

$$\max_{j=1..M} (\mu_j^n, \mu_j^p) < (LSW - \varepsilon) * r/8 \tag{3.3}$$

Note that the scheduling model is such that the synchronous load submitted to the switch in each EC always fits inside the respective synchronous

window. Thus there will be no synchronous messages left in the switch queues at the end of each EC.

3.3.3 The aperiodic traffic scheduling model

The aperiodic scheduling model is similar to the periodic one with similar parameters having similar meaning. Equation 3.4 shows the model of an aperiodic stream (AM_i), stored in a structure called Asynchronous Requirements Table (ART), holding N_a streams. Opposed to the periodic model, there are no offsets and instead of a period there is a minimum inter-arrival time $Tmit_i$, also expressed as an integer number of ECs. Large aperiodic messages are similarly fragmented and scheduled in a packet basis.

$$ART = \left\{ AM_i : AM_i = (C_i, Tmit_i, Pr_i, S_i, \{R_i^1..R_i^{k_i}\}), i = 1..N_a \right\} \quad (3.4)$$

The set of streams in the ART is scheduled by the FTT Master according to any online policy, possibly using periodic servers, implementing a single queue, also similarly to the periodic traffic. The difference to the periodic scheduling is that messages are no longer activated implicitly, as a function of time, but upon an explicit application request. The master receives those requests through the signaling mechanism described in Section 3.2.2. Once scheduled, the aperiodic messages are similarly polled via the TM in each EC and then transmitted in the respective window.

Non-real-time (NRT) messages are handled as a special case of aperiodic messages, being handled with the lowest priority and having the deadline set to infinity, in order to avoid the generation of deadline violation exceptions. Thus NRT messages are transmitted in background with respect to the real-time messages, both periodic and aperiodic. One particular aspect to note is that the sender and the receivers of these messages are not known ahead of time. However, the respective MAC addresses are sent to the master in the associated signaling message, which allows it to carry out the respective poll, later on, when such traffic is scheduled.

Note, again, that the aperiodic and NRT traffic is scheduled in a given EC only if it can be transmitted in that EC. This means that, by the end of the EC, there will be no traffic queued in the switch. This is very important, being the feature that effectively allows bypassing the traffic scheduling carried out by the switch in its internal queues, on a sparse time-base, with EC resolution.

3.3.4 Bounding the aperiodic service latency

The $Tmit$ parameter sets the maximum transmission rate with which a given AM may arrive at the switch and, in conjunction with the message length, bounds the message maximum bandwidth utilization ($C/Tmit$). However,

the sporadic messages frequently have average activation rates significantly lower than the maximum one. The aperiodic signaling scheme presented in Section 3.2.2 yields for a better use of the bandwidth, which is only used upon explicit requests, allowing other messages to reclaim the slack wasted by messages activated at lower rates. However, RT traffic is typically admitted considering the worst-case scenario and so the reclaimed extra bandwidth does not result in better schedulability levels for the RT asynchronous traffic. The advantage of this signaling mechanism for the RT asynchronous traffic is the reduction of the average response time, specially for the lower priority traffic that is promptly scheduled whenever there is slack. Furthermore, the reclaimed bandwidth may also be used by the background non-RT traffic, improving its average throughput significantly.

The response time of an asynchronous message (AM_Rt), defined as the time lapse between the instant in which the application generates the message, i.e., the message becomes ready at the sender interface, and the instant in which its transmission finishes, depends on the signaling latency ($Lsig$), on the scheduling latency ($Lsch_i$) and on the message size (3.5). $Lsig$ is the time required for the aperiodic transmission request to arrive at the master and be included in the scheduling structures and $Lsch_i$ is the delay induced by the traffic scheduler until the message is actually included in a TM (integer number of ECs) and $Lpoll$ is the time for the respective producer to answer to the poll in that TM ($Lpoll < 1\text{EC}$).

$$AM\_Rt = Lsig + Lsch_i + Lpoll \tag{3.5}$$

While the scheduling delay depends on the particular scheduling algorithm and current traffic load, the signaling latency is independent of the particular traffic configuration and can be bounded approximately between 1 and 2 ECs (Figure 3.8b), corresponding to the situations in which the AM message is queued just before or after the transmission instant of the signaling message. Notice that 1 EC is always needed after the signaling message arrives at the master so that it is inserted in the scheduling structures.

To compare the impact of the signaling mechanism in the AMs response time with respect to the common periodic polling mechanism, Figure 3.8 sketches a timeline with the worst-case situation for the response time of an asynchronous message (AM_Rt) in the two cases. In (a) a periodic polling mechanism is used in which the master periodically schedules the AM in a blind way, while (b) represents the proposed aperiodic signaling mechanism. In both scenarios the asynchronous message is registered with a minimum inter-transmission time of 3 ECs.

When activations are periodically triggered (a), the slave may ultimately have an AM transmission request right after the last poll, leading to a response time (AM_Rt) as given by Equation 3.6, where δ is an infinitesimal

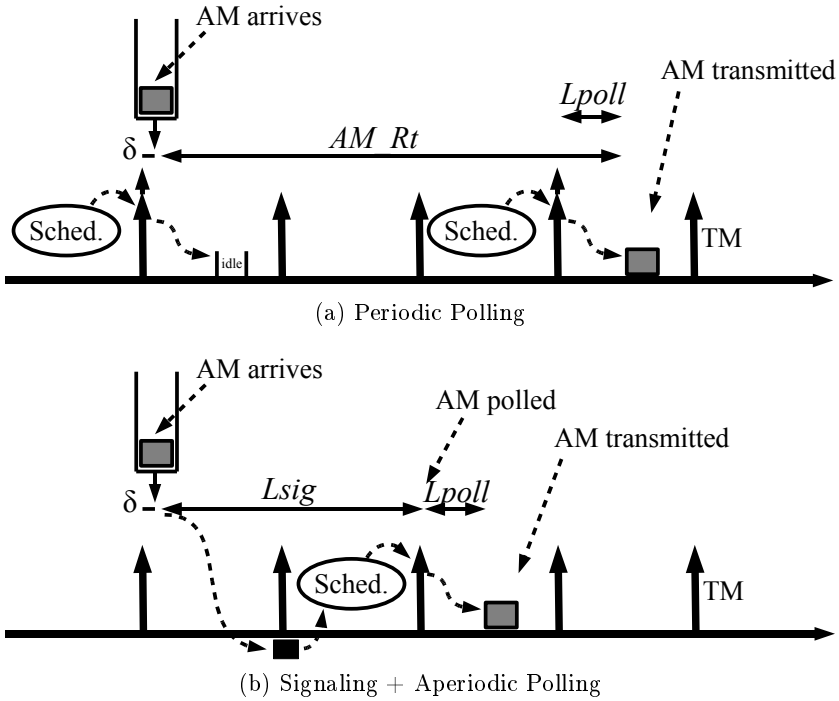


Figure 3.8: Response time to Asynchronous messages with FTT-SE (case with $Lsch_i = 0$).

and LEC is the EC length.

$$AM\_Rt = Tmit - \delta + Lsch_i + Lpoll = 3 \times LEC - \delta + Lsch_i + Lpoll \quad (3.6)$$

In this scenario the response time varies linearly with the message $Tmit$. However, when the proposed signaling scheme is applied (b), once an AM transmission is requested it produces a signal that is transported in the following EC and then handled by the master scheduler. The worst-case delay is, in this case, independent of the message $Tmit$ and given by Equation 3.7

$$AM\_Rt = 2 \times LEC - \delta + Lsch_i + Lpoll \quad (3.7)$$

The message activation signal takes at most (worst-case) 1 EC to reach the master. Afterward, the request is made eligible for scheduling and possibly dispatched in the following EC if space permits ($Lsch_i = 0$).

The best-case response time activation delay occurs in (a) when the message is queued right before its poll, as given by Equation 3.8, and in (b) when it is queued before the signaling message transmission, resulting in 3.9.

$$AM\_Rt = \delta + Lsch_i + Lpoll \quad (3.8)$$

scenario	Worst	Best	Average
(a)	$Tmit$	0	$Tmit/2$
(b)	$2 \times LEC$	$1 \times LEC$	$1.5 \times LEC$

Table 3.1: AM Response times with uniform arrival (case with $Lsch_i + Lpoll = 0$).

$$AM\_Rt = 1 \times LEC + \delta \tag{3.9}$$

If the asynchronous messages are randomly triggered by the application with uniform distribution, the average response time will vary uniformly between the best and the worst-cases.

Table 3.1 outlines the response times for the two scenarios in the worst, best and average cases. The proposed signaling scheme (b) induces a better asynchronous responsiveness for values of $Tmit$ greater than 2 ECs. The benefit for messages registered with longer $Tmit$ is obvious.

3.4 Implementation details

The FTT-SE protocol integrates a communication framework that provides a set of services possibly directly to the application. These services are part of a specific middleware that addresses distribution and system startup reconfiguration.

3.4.1 Middleware abstraction

In a distributed system the middleware provides the means to abstract the complexity of bringing together several services running on different nodes, towards an integrated computational platform.

The FTT-SE middleware provides the abstraction for synchronization mechanisms as well as a logical abstraction that associates services and the nodes, guarantees a consistent resource allocation and provides network load accounting, which yields an admission control for new sessions, specially important in dynamic and open environments. Basically the idea with this layer is to bring forth a common application interface available at all nodes, handling the communication details and providing seamless design composition.

Such abstraction layer is a valuable tool for an easier and more efficient application development and deployment. Figure 3.9 depicts a layered view of the protocol structure, including the management, interface and core layers. The protocol lies over a full-duplex *Switched Ethernet* network. Internally, the *FTT-SE core* layer represents the protocol basic communication

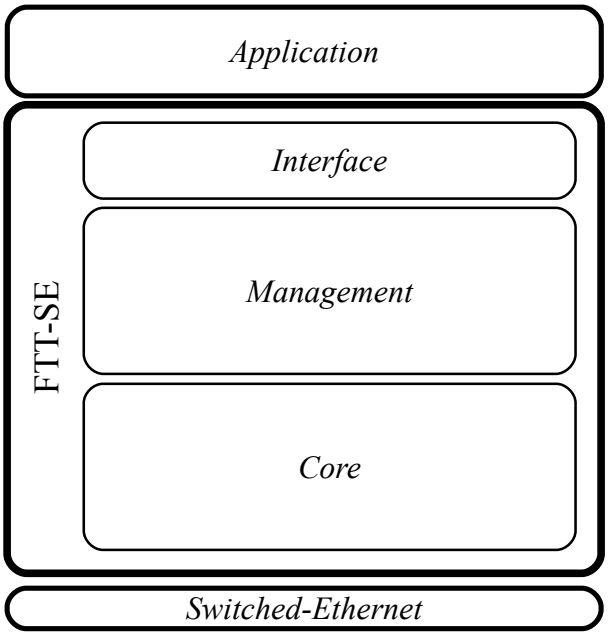


Figure 3.9: FTT-SE internal layering.

mechanisms, namely the transmission control and the traffic scheduling. The *FTT-SE management* performs a high level session control, establishing the connections registered for each stream. The idea is to avoid associating the application and the physical nodes when designing the application and decoupling the stream and the endpoint threads, which can register as producer or consumer of a given stream, independently of their location in the distributed network. Still in this layer, a QoS management module regulates online the amount of network resource assigned to each stream based on the rules set by the application. The adjustments are controlled by a QoS manager that distributes the network capacity among the streams, based on the needs and importance of each. The QoS management is covered further in Chapter 5. Finally, the *FTT-SE interface* layer provides the set of communication and management services to the application e.g., send, receive, bind, as well as resource reservation and de-registration that allow using the FTT-SE protocol.

The FTT-SE implementation is modular, allowing the coexistence of both master and slave components in the same node. This aspect has practical relevance since it allows saving on the hardware platform cost, specially when comparing to a standard implementation over switched Ethernet that does not require a master node. Note that, in such case, the master functionality can be performed by an existing application node, without requiring an extra node.

Figure 3.10 provides a detailed overview of the fundamental software

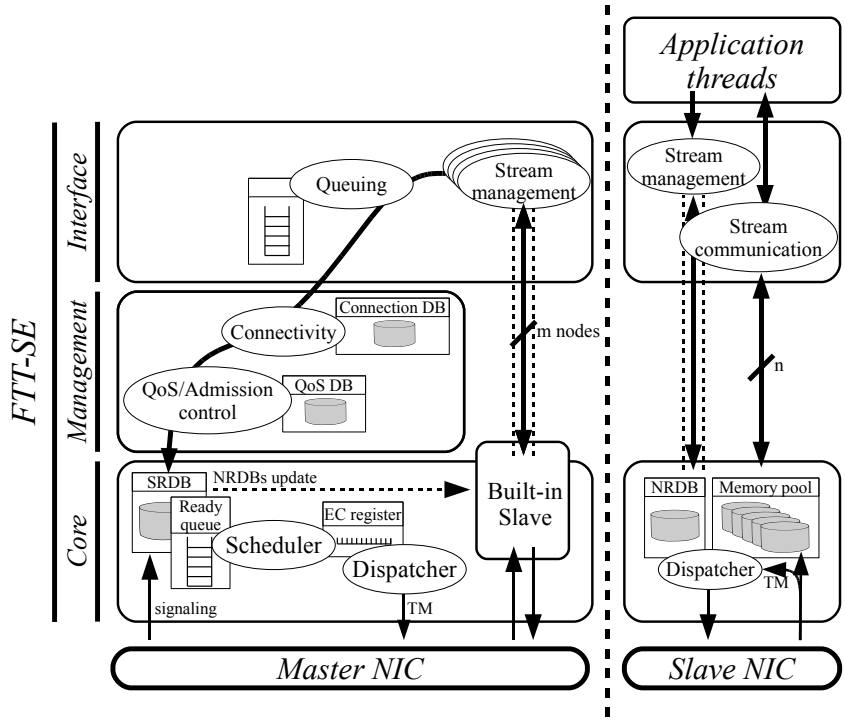


Figure 3.10: FTT-SE internal details.

building blocks that compose the FTT-SE framework. Regarding the middleware abstraction, there are several aspects that deserve a special mention since they contribute to a better application integration. The functional details of each block in Figure 3.10 are described next.

The protocol basic implementation

The basic components of the protocol are pretty much identical to the architecture proposed by Pedreiras [99] for the FTT-Ethernet.

In the master, a system requirements database (*SRDB*) holds the running traffic properties, either synchronous or asynchronous. A *Scheduler* periodically scans the *SRDB*, pushes transactions into the ready queue and builds transaction plan for the trigger message (TM), which is stored in the *EC-register*. This register is then read by the *Dispatcher* and inserted in the TM transmitted at the beginning of the following EC. For each traffic class there is an independent *ready queue* and a corresponding scheduler, which follows any policy with a resolution set by the length of the EC (LEC). Regardless of the scheduling policy, the messages are activated differently whether they follow the synchronous or asynchronous model.

Messages within the synchronous model get activated, i.e., enter the *ready queue*, in a periodic basis. A counter for each variable triggers that activa-

tion every T_i. As for the asynchronous messages, their activation is related to asynchronous triggering of messages in the node, which are intercepted and kept waiting for transmission order. As previously described in Section 3.2.2, at the end of the EC, a signaling message is sent to the master with an indication of the asynchronous messages queued in the node. The master then schedules those messages according to their traffic class. This mechanism allows enforcing minimum inter-transmission time specified for real-time asynchronous messages. Therefore, similarly to the synchronous traffic, a counter tracks the last transmitting time of such message, and enforces a separation of at least $Tmit_i$ between consecutive transmissions. Referring to the asynchronous traffic without real-time requirements that operates in the background in a best-effort basis, it is handled through a FIFO queue, filling the slack left unused by the other traffic, without schedulability or overflow-free guarantees. All the remaining procedures, mainly the polling with the TM, are similar for all messaging models.

In the slaves, the TM is received, decoded and its contents compared against the node requirements database (*NRDB*). For each message that is polled by the TM, if the node is a producer of it, the dispatcher fetches the message from the *memory pool* and orders the its transmission. Conversely, when a slave receives a data message, it checks whether the node is a consumer and, if not, stores the message contents in the *memory pool*. The *memory pool* operates differently whether the message being stored is synchronous or asynchronous. Synchronous messages have state semantics and each message is associated to a buffer holding the latest state update. Asynchronous messages have an event semantics and so are stored in a FIFO queue, both in the sender and receiver. Therefore, the *memory pool* additionally holds the dynamic information related to each stream, such as the queuing status or the message freshness. Still in the *memory pool* component, application signals are associated to each buffering entity in order to support synchronization events to the application upon a transmission or reception of a message.

The *NRDB* component records the model properties of all streams related to that node, i.e., all the static (long-term lasting) information. Such information must be consistent with the information present in the master *SRDB*. There is a clear bound between the streams set in the *NRDB* and the items in the *memory pool*.

As proposed in Section 3.2.2 the slave nodes additionally produce a signaling message that reports the status of the queues of the streams for which the node is a producer. That message, a unicast message is sent to the master node at approximately the same instant as the master sends the TM. For this purpose, the *dispatcher*, which is the same component that decodes the TM and issues the message productions, synchronizes with the every received TM in order to program the instant of transmission of the next signaling message, slightly less than 1 EC later.

Resource allocation consistency

The basic infrastructure described before has no mechanism supporting the requirements consistency between the master *SRDB* and the nodes *NRDB*. If no automatic mechanism is provided, the application necessarily has to ensure that streams are registered consistently across the master and the related slaves. This coordination requires an agreement between these elements in the distributed network specially when addressing the dynamic changes that may occur in the network requirements. Note that the properties of a given stream may vary at run-time, with data being generated, and that not only the FTT-SE stack must be consistent but also the application threads must be aware of the updates taking place and react accordingly.

To address this issue the FTT-SE framework is provided with mechanisms that transparently update the protocol internal databases (*NRDB*) that report to the application threads the updates on the streams to which they are bound. In a way, this is a synchronization mechanism between the distributed threads in a reconfiguration procedure.

The network management interface is confined to the *FTT-SE core* layer in the master node, as illustrated in Figure 3.10. All the configuration commands that modify the network requirements are issued through the master. This facilitates consistency and alleviates the application in the slaves side, suppressing the need for additional synchronization (logic and time) between the distributed nodes.

Every time the master *SRDB* is updated, all related *NRDB*s are notified, using a broadcast channel established within the *FTT-SE core* and connecting the master and the slaves. The channel is created along with the *FTT-SE core* layer, making use of the *Built-in Slave* component in the master that requests the creation of an asynchronous broadcast channel. This channel is unique and serves all slaves. It is configured in all nodes with a reserved default ID that is fixed and hard-coded. The master registers a producing endpoint whereas each slave registers a consuming one.

The *SRDB* interface mechanisms guarantee that the updates are simultaneously committed in the master and all slaves, within the system resolution (one EC). Moreover, depending on the nature of the update, changes may not be considered immediately, having to wait, for instance, for asynchronous messages already queued. In such cases the two instances may co-exist in the databases, to provide a soft transition to the application. For example, consider a thread producing messages considering a set of properties A for the channel. Meanwhile, the stream requirements are modified to exhibit properties B. The producing thread is immediately notified for the new properties and the receiving thread is informed right after the last reception of the message already queued with properties A. Another mechanism provided by the *SRDB* interface is group handling. It provides the ability to perform atomic changes of several modification requests, i.e., the commands

are committed by the system all at once.

The *Built-in Slave* component in the master node is introduced to handle the asynchronous communications of this node, as needed for the network management. In fact, this component provides a full asynchronous communication interface to the master, allowing the registration of synchronous or asynchronous endpoints. The interface to this component is alike in all nodes, except for the *NRDB* updates received in the slaves and the asynchronous signaling messages received in the master that bypass the need for message exchange in the network.

activations that go directly to the master without a transporting mechanism through the network.

With these mechanisms the slave nodes are automatic and accurately synchronized with each other and the master node that defines the network requirements, at the basic protocolar level.

Plug 'n Play

When addressing flexible and, particularly, open systems, the network starts executing without any pre-configuration or knowledge about the system topology. The architecture must be prepared to support the online connection and disconnection of services and nodes in a random order. This demands for mechanisms that detect and act on such events, without disrupting nor jeopardizing the ongoing communications.

When booting-up the system, each of the layers described in Figure 3.9 starts in a bottom-up order, preparing the software components to operate properly. Some of the local components are related and dependent and so they follow a hierarchical boot procedure. Moreover, some components are distributed and also related and dependent, more specifically, between the master node and the slave nodes.

It is thus necessary to provide the mechanisms to firstly detect changes to the network topology and then discover the streams provided by each node. We shall now focus on the former, covering the latter afterwards, along with the *FTT-SE interface* layer.

The boot-up procedure is conducted individually in each distributed node and globally in the distributed system, where the nodes (including the master) may be randomly included. It is however obvious that the network only operates after the master connection and that this cannot be disconnected afterwards. Each node enters the network with zero-configuration, i.e., a plug 'n play procedure has to be conducted. Hence, the slaves must suspend their boot until a Master is detected and resume it afterwards simultaneously, as if they were connected at time zero.

When starting the slave's *FTT-SE core* layer the first procedure, after sensing the master, is to become visible to the Master's *FTT-SE core*. The master aggregates the system configuration and thus requires the knowledge

of all the nodes connected. Only then the slave may prepare all the communication infrastructure upwards. However, the slaves have no communication channels established to inform the master. The only hard-coded configuration is the asynchronous broadcast channel, to which all slaves register as consumers to receive the *SRDB* updates. To notify the master about the presence of slaves, each slave uses the aperiodic signaling mechanism as described before. This channel only conveys minimum sized packets, one per node and per EC, thus not requiring scheduling and being adequate for use on this Plug 'n Plug phase. Hence, on boot, the slaves inform the master through this channel using a special message that contains the node's Ethernet unique address (MAC). The master, then associates an internal ID to that node, updating the network topology, and acknowledges the slave notification through the asynchronous broadcast channel.

After this procedure, the nodes are ready to operate using the basic communication primitives provided by the *FTT-SE core* interface that include binding to an existing endpoint and sending or receiving accordingly. However, the stream must already be registered in the slave and so must have been registered through the master interface. The upper layers assess this registration procedure and agreements between the slaves and the master. There is though an additional service provided within the *FTT-SE core* layer that leverages the connection of the upper layers on this distributed scenario. It basically provides a notification and acknowledgment mechanism from each slave to the master. It extends the hereby described Plug 'n Play procedure for the upper layer (the *FTT-SE interface*). It uses the same channels, i.e, the signaling and the asynchronous broadcast channels, to notify the *FTT-SE interface* in the master whenever a new node is registering and so trigger the respective session establishment procedure, handled within the *FTT-SE interface* layer.

In an open system scenario, in the same way as the nodes are connected asynchronously, they may also be disconnected with no previous report, e.g. on a system failure or just because there is not a soft deallocation procedure, with all the implications that may result to the application. It is thus of utmost importance that the network, particularly the master, is able to maintain its knowledge on the current network topology and act accordingly when changes occur without previous notification, which includes readjusting its network, deallocating the broken streams and reporting to the application about those. Again, we will focus on the former, covering streams deallocation to the *FTT-SE interface* layer.

In the *FTT-SE core* layer, since the slave is not able to notify the master when removed, the master must follow a discovery procedure to detect missing nodes. A simple mechanism to that end comprehends the use of an activity aging counter, easily implemented with the signaling message freshness, which is sent every EC. The master, reading the counter for each node, is able to discover when the node becomes inactive. Upon a pre-specified

sequence of inactivity events, the master removes the slave from the network topology, letting the *FTT-SE interface* layer to be aware of the change and apply the necessary measures that exclude the node from all the running services.

Application interface and management

The purpose of a middleware layer is to provide to the application a communication interface that fully abstracts the protocol internals providing a uniform and transparent access to the communication services throughout the distributed system.

The *FTT-SE interface* layer plays an important role to support such distribution and network abstraction. It provides the interface for the network management and the data communication transactions then established, where it entirely abstracts the network internal namespace for nodes and streams, creating a full separation with the application-defined namespace. The abstraction mapping is achieved with a binding process that associates the namespaces and allows, for instance, application threads to be node independent regarding execution.

Referring to the internal implementation of the communication primitives, the *FTT-SE interface* simply redirects the interchanged data and control signals to/from the *FTT-SE core* layer. It should be noted, though, that these transactions are optimized to minimize the number of copies when going through the protocol stack. Depending on the Ethernet driver, it is possible to accomplish the whole process with a single memory copy transaction.

The management related requests are triggered by the slaves and go through the master, before reaching the *FTT-SE management* layer. The interface layer assures the abstraction of this communication with the master, serializing the requests in a queue and allowing the submission of several requests as a group. This serialization and group handling mechanism is necessary if we consider the need for atomic changes and the fact that nodes may compete asynchronously for the opportunity of modifying the network requirements. This mechanism demands a direct communication link between the slaves and the master. Hence, whenever a node is attached, two asynchronous message streams per node are automatically created. These form a bidirectional channel that is transparent to the application and allows carrying out the requests and acknowledgment with the Master *FTT-SE interface* layer.

Synchronization properties

Regarding to the middleware and services made available to ease the application design, there are three synchronization aspects to highlight related to

the streams communication. The first aspect refers to the synchronization of the threads that handle the streams directly. There are three kinds of threads supporting a stream, the producing thread, one or more receiving threads and one management thread, being the latter the one that negotiates and defines the stream requirements with the FTT-SE network. These threads can execute on any node, all apart in different nodes or even share the same node. The issue here is that all threads are attached to the same logical entity, the stream, and so have to be set together and run as a whole, e.g. there is no point in issuing a transmission if there is no thread receiving and handling the data.

As part of the communication endpoints establishment procedure, each application thread, either producer or consumer, registers itself in the master. However, a single endpoint registration is not enough to set a stream. The stream only becomes valid once associated to one producer and at least one consumer. Therefore, each thread individually, after registering the endpoint, must wait on a binding procedure until the stream is fully registered. In the slaves this binding procedure becomes possible when the stream properties are visible in the *NRDB*.

This stream registration procedure takes place in the Master *FTT-SE management* layer that collects each registered endpoint to build the stream connectivity table, associating the stream application ID S_i with the node IDs. When the stream becomes fully described it is issued to the network, which eventually unblocks all threads synchronously. This mechanism provides a full synchronization of the application right from the stream startup.

The second synchronization feature refers to the notification of communication events on a channel already established and running. In the slaves, whenever a message, synchronous or asynchronous, is transmitted or received, a signal is raised that unblocks any blocked thread waiting for that event. This allows an easier and simpler interaction with the network, specially in event-triggered communication scenarios.

Finally the third synchronization aspect refers to getting all threads updated regarding the stream requirements. When the management thread sets the base requirements, all threads have access to that information, as they share the same design scope. Each thread then produces and receives data according to those requirements, for example a video frame with a given size. However, when the requirements set by the application allow instantaneous variations, as a result of a QoS manager, the application is not able to directly control the properties of a stream on the network. The result of the QoS manager depends on the global requirements set for each stream as well as the current network load. Furthermore, these variations may occur asynchronous, e.g. upon adding or removing another stream that ultimately affects the network current load, forcing the network re-configuration. After a re-configuration all the involved threads must be properly notified of those changes. Failing to do so, may lead to potentially catastrophic consequences,

since the application threads may inconsistently address the produced and consumed data. Another implication of this kind of synchronization is that the application does not stop while updating the running properties of a given stream. The architecture must be able to support those changes online and interact with the application. This latter topic was briefly introduced when describing the online synchronization between the *SRDB* and the *NRDB*s within the slaves.

This online synchronization issue is handled using an tagging mechanism that associates a unique tag to the requirements of a stream. Whenever the characteristics of the stream change, so does the tag. During the transition phase from one state into the other, i.e., one tag to another, the two requirement sets co-exist in the *SRDB* as well as in the corresponding *NRDB*s. This is necessary for a smooth transition, where from one side the pending messages (old-tagged) have to be scheduled and transmitted, and from the other, newly-tagged messages are being produced.

For the application, this tagging mechanism allows a friendly synchronization of each thread to the environment dynamics. Production of old-tagged data is denied from the moment the new tag is visible in the *NRDB*, guaranteeing that no old-tagged messages are buffered. The thread producing that message must then poll the new tag requirements and rearrange the data being produced. In the consuming side, threads get all messages along with the associated tag that informs how the message must be handled and so they are explicitly notified of any changes on the current stream properties.

Figure 3.10 shows some of the components that play a part supporting these synchronization mechanisms. In the *FTT-SE management* layer, the *Connections_ db* stores the connectivity of the streams. After a complete connectivity registration, the stream passes an admission control and the QoS management that determines the final properties submitted to the network. In the slaves, the *memory pool* buffers the transactions data and holds the signaling flags that trigger the communication activity events. These events pass through the *FTT-SE interface* layer where the stream ID is translated and passed to the application layer. Still in the slave nodes, the *NRDB* holds the streams requirements currently registered in the network.

3.4.2 Data addressing modes

The centralized architecture of FTT-SE relies on the master to coordinate the traffic that crosses the switch. In this task, the scheduler handles the switching queues independently, building timelines for each, while allowing parallel forwarding paths.

The Ethernet switch operates in the data link layer, routing Ethernet packets according to the destination MAC address, see Section 2.3. The packets can be forward as unicast, multicast or even broadcast. A unicast

packet is forward to a single output queue, corresponding to the packet destination MAC in the forwarding table. A broadcast packet uses a special destination address, reserved to replicate the packet into all the output port queues, except for the one from where the packet arrives. Finally, the multicast transmission also uses a special MAC address type to forward packets to multiple destination ports. This address is taken out of a range reserved for multicast transmissions, where each address designates a multicast class. The switch forwarding table registers the ports that refer to each multicast group. However, the building procedure of this forwarding table is more complex, as the receivers have to establish a registration to the multicast group. The IGMP protocol, detailed in Section 2.3, handles this registration that is supported by switches with IGMP snooping capability. Once the forwarding addresses are properly set in the table, the multicast packets that match the indexed forwarding addresses are replicated to the respective output queues, similarly to the broadcast address.

In general, broadcast messaging better suits the producer-consumer communication model, which is not the model followed by the FTT-SE protocol. It can also be used as a trade-off replacement for when multicast is not supported and using multiple unicast messages becomes too expensive. Another application scenario for broadcast messaging is on network discovery services, where polls are required to flood the network through all its branches.

The FTT-SE protocol, follows the publisher-subscriber model, which is more efficiently supported by the multicast transmission model. Multicast messaging lies between the unicast and broadcast models that are the two opposite addressing extremes. In fact, it generalizes both, since a multicast can address either a single port (unicast) or all the other ports (broadcast), without additional penalty other than the layer 2 subscription required for the receivers.

The two traffic models within the FTT-SE architecture, synchronous and asynchronous, define for each stream a sending node and one or more receivers, as defined in Equations 3.1 and 3.4. Depending on the specifics of each variable it is possible to enforce a unicast, multicast or even broadcast transmission. Thus, the multicast transmission, if supported by the switch device, is the preferable choice since it is more resource efficient defining the message stream.

Section 3.4.1 described the procedure necessary to register a messaging stream in the protocol. The registration procedure is centralized in the master node and then unfolded to all the nodes that take part in each transaction. During the process, the producer and consumers of each stream get to know the transition details, among which, the destination MAC address. That address can either be a broadcast directive, a specific node MAC address or a multicast defined group. When defining a multicast, the master associates the stream to a multicast group, from the range of MAC addresses reserved by the IGMP protocol, which is then used in the stream destination field.

To complete the multicast session, the receivers have to be associated to the multicast group membership in the switch. Each registered receiver sends an IGMP Host membership report packet to the switch. The switch, snooping the IGMP traffic, associates the ingress port to the multicast group in its forwarding table.

3.5 Simulation and experimental assessment

This section presents both simulation and experimental results that allow assessing the potential of FTT-SE to efficiently use the aggregated switch throughput and to verify the correctness of the implementation as well as the protocol capability to enforce jitter control in synchronous traffic when compared to using a COTS switch without any further transmission control.

3.5.1 Periodic traffic simulation results

The traffic scheduling model used in FTT-SE enforces a strict priority order in the scheduling of messages, even if it leads to the insertion of idle time in the synchronous windows of the ECs. This happens whenever the scheduler moves on to the next EC while there is still capacity left in some links and the ready queue is not empty. Such idle time depends on the specific scheduling policy used and introduces a degradation of the efficiency in the use of the switch aggregated throughput.

To assess such degradation several simulations were carried out with randomly generated message sets using both RM and EDF scheduling. The operational parameters considered an EC duration of 5ms and a maximum synchronous window duration (LSW) of 85% of the EC (4.25ms). The message sets were generated with uniform distributions according to the following parameters: period between 1 and 4 ECs, deadline equal to period, single packet messages with payload between 1200 and 1450 data bytes, Publisher chosen from {A, B, C, D, E, F, G, H} and Subscriber chosen from {A, B, C, D, E, F, G, H, Broadcast}\{Publisher} and considering three different cases: no broadcasts, 50% broadcasts and 100% broadcasts. The two first cases verify the capability for taking advantage of the parallel forwarding paths. Moreover, despite the protocol supporting the specification of offsets among the synchronous streams the simulations considered a synchronous release of all messages since we were interested in detecting worst-case response times.

The message sets were generated in order to obtain a given utilization value of the most loaded link. Thus, new messages were continually appended to the set until one link reached the predefined maximum load. This generation method was used because it prevents queue overflows with an appropriate load threshold.

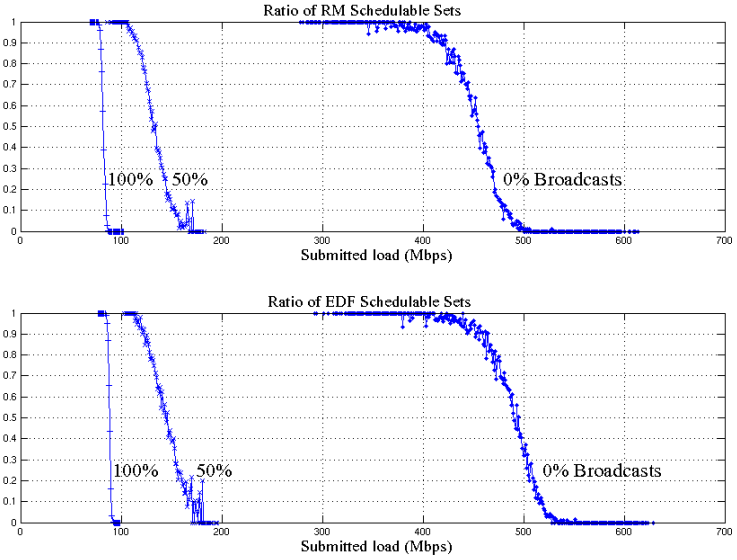


Figure 3.11: Schedulable sets versus the aggregated submitted load with EDF (bottom) and RM (top).

Each of the generated sets was simulated using both EDF and RM scheduling policies. The simulations were carried out until a deadline was missed or the full macro-cycle elapsed in which case the set was considered schedulable. The ratio of schedulable sets for EDF and RM with respect to the total number of generated sets is shown in Figure 3.11. This ratio is shown as a function of the total load submitted to the switch, corresponding to the generated sets. In general, as expected, EDF (bottom) generates more schedulable sets than RM (top) despite the difference being relatively small (less than 10% of the schedulability ratio). Nevertheless, this experiment shows how easy it is to inherit the benefits of EDF traffic scheduling over COTS switches using FTT-SE.

Also as expected, the switch utilization with broadcast traffic is rather low since parallel forwarding paths are not exploited. There can still be a small level of parallelization between the uplinks and downlinks inherent to full duplex but it is rather limited. Conversely, without broadcasts the switch allows for a substantial increment in the utilization of its aggregated bandwidth. In this case, there were 8 publishers connected to the switch through 100 Mbps ports and generating traffic further constrained by the synchronous window with a maximum duration of 85% of the EC. In these circumstances, the maximum aggregated throughput is 680 Mbps. The figures show that, in the unicast-only case, EDF and RM are capable of successfully scheduling all generated sets with aggregated utilization of 55% and 50% of that abso-

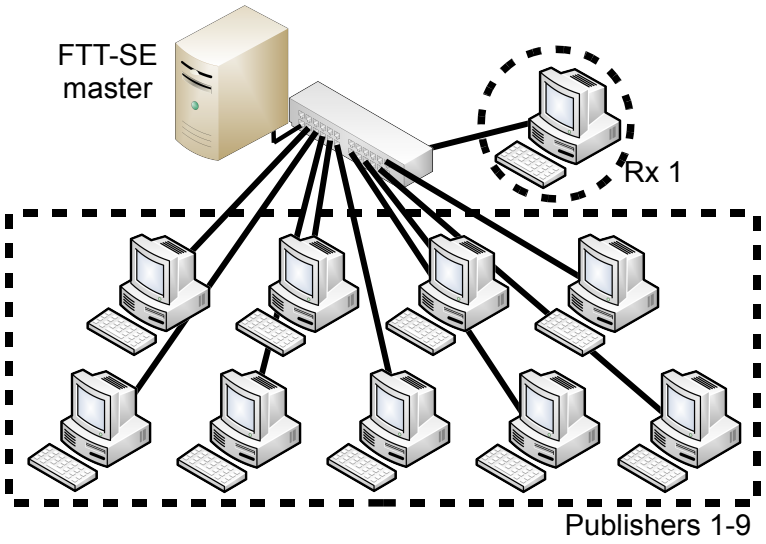


Figure 3.12: The experimental platform.

lute maximum and, in some cases, up to 80% and 73%, respectively. These numbers, however, cannot be generalized since they depend on the evenness of the load distribution across the switch links. Nevertheless, these values show that FTT-SE is capable of efficiently exploiting the switch aggregated capacity, whenever the load is evenly distributed across the links.

The values obtained with 50% broadcasts are intermediate values that illustrate the penalty that these transmissions cause. It is interesting to observe that the schedulability ratio grows approximately 50% when moving from the 100% broadcasts to the 50% broadcasts case but when moving to no broadcasts, such improvement is near 450%. This indicates that broadcasts impose a severe penalty on traffic schedulability even if in low number.

3.5.2 Experimental results

The prototype implementation of the FTT-SE protocol was carried out on the RT-Linux [54][1] real-time operating system over the Ethernet layer provided by the LNet network stack [53][1]. Several practical experiments were carried out to verify the correctness of the implementation as well as the level of jitter control. The experimental platform, shown in Figure 3.12, comprises eleven computers interconnected by an Allied Telesyn model 8024 Ethernet switch, with 24 ports and 2 priority levels. The computers included one Celeron at 2.2GHz, one Pentium III at 550MHz, six Celeron at 735MHz as well as three SBCs with Pentium MMX at 266MHz. The network interface

ID	C($bytes$)	T(=D)(ms)	$\max_{j_w}(\mu s)$	$\max_{j_{wo}}(\mu s)$
2	1000	1	483	996
7	1000	1	174	984
8	1000	1	170	1003
3	3840	3	92	932
1	3840	4	893	559
4	3840	4	316	446
5	3840	4	1000	521
6	3840	4	137	561
9	1480	8	4132	436

Table 3.2: Message set used in the FTT-SE experiments.

cards (NICs) were Intel 8255 and 3Com 3C905B.

One of the computers was dedicated to the FTT Master, nine computers were data publishers, publishing one message each, and the last computer was a subscriber of all those nine message streams. Only one subscriber is used in this experimental assessment to maximize the messages concurrency in a single link, creating a worst-case latency, and probably jitter, situation. The message set is detailed in Table 3.2 and mixes messages with different periods as well as single-packet (messages 2 and 7 to 9) and multi-packet (messages 1 and 3 to 6). Remember that FTT-SE handles transparently the fragmentation and re-assembly of multi-packet messages, allowing preemption between packets. The total load submitted by this set is approximately 68,6 Mbps. Concerning the operational configuration of FTT-SE, the EC duration was set to 1ms and the LSW to 85% of the EC, i.e., 0,85ms. The traffic scheduling was RM.

The same experiments were also carried out with the publishers sending information at the same rate but without the transmission control mechanisms provided by FTT-SE. The inter-arrival instants of all messages at the subscriber node were recorded for both of these configurations referred to as *with* and *without* FTT.

Firstly, an experiment was carried out to assess the *infrastructure jitter*, i.e., the jitter due to the operating system, network device drivers and packet switching. This is the base jitter of the system without any scheduling interference. This preliminary experiment consisted in the transmission of a single periodic message, with 1500 data bytes, transmitted every EC (1ms) by one of the slower computers, during 560s. In this case there is no *scheduling jitter* and thus the jitter observed is only caused by the communication infrastructure. As depicted in Figure 3.13, the infrastructure jitter is lower

[1] RTLinux and LNET were by 2006 supported by FSMLabs Inc. They are currently distributed by Wind River Systems Inc.

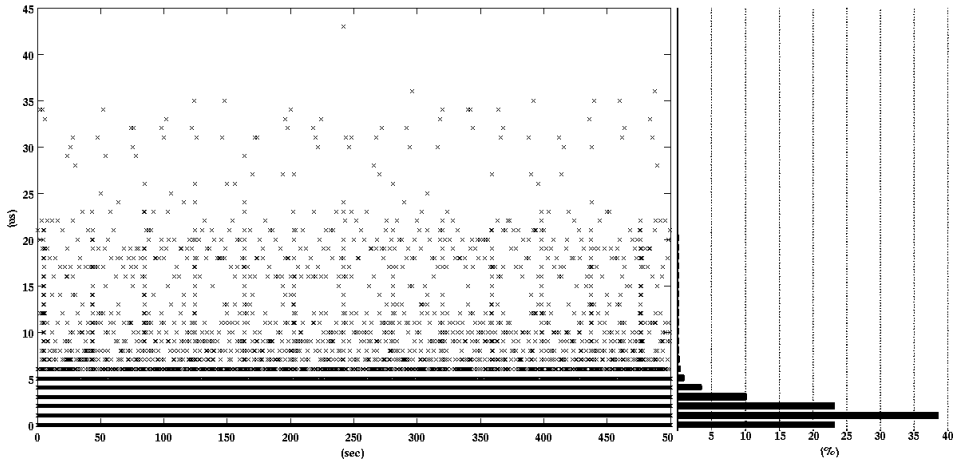


Figure 3.13: The infrastructure jitter.

than $5\mu s$ for 99% of the samples, with a single occasional maximum of $43\mu s$.

The following experiments consisted on the transmission of the full message set defined in Table 3.2 during approximately $300s$, both with and without the FTT-SE protocol. The maximum measured jitter is also shown in the same table in columns $\max_{j_w}$ and $\max_{j_{wo}}$, for the cases of with and without FTT-SE, respectively. The jitter j_w and j_{wo} was measured taking the modulus of the difference of consecutive message arrival timestamps. This is also know as the relative jitter.

Figure 3.14 and Figure 3.15 show the histograms of two selected messages, one high priority and the other with lowest priority, respectively, both with and without FTT-SE. The results obtained are illustrative. In the absence of transmission control, i.e., the case without FTT-SE, the clocks of the various publishers are not synchronized and the respective relative drifts lead to situations that vary from high contention periods, in which all transmissions arrive at the subscriber uplink within a short interval and are queued and thus delayed, to other situations in which the transmissions are de-phased, thus with practically no mutual interference and no delay. This explains the large spread in the inter-arrival time observed in the respective histograms.

Conversely, with FTT-SE the system nodes are globally synchronized by the TM and the submitted load is always within the capacity of the synchronous window of each EC. However, within the EC, the traffic is managed solely by the switch, without control of the FTT-SE. The jitter that results from the system is of two types, *sub-EC*, caused by interference among the streams in the FIFO queues in each EC, and *scheduling induced*, caused by the actual FTT-SE scheduling activity on a time scale with a resolution of 1 EC. The former type is clearly visible in messages 2, 7 and 8, generated by differences in the speed of the respective computers in responding to the

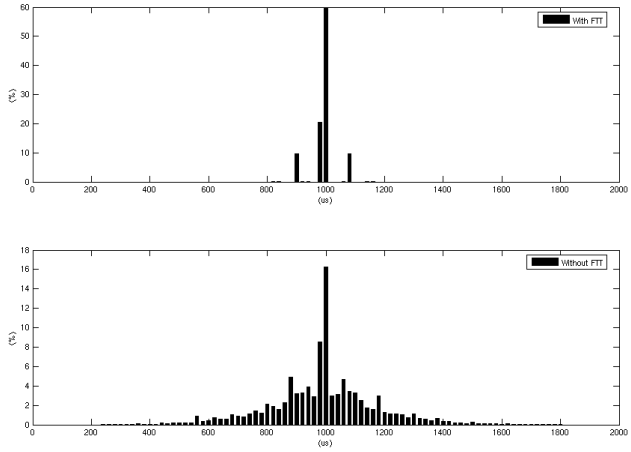


Figure 3.14: Histogram of inter-arrival times for message 7.

TM. In some cases, e.g., multi-packet message 3, the jitter value is especially low because it refers to the last packet, which is sent later in the EC, being queued after all others and thus in a relatively constant position in the FIFO queue. The latter type of jitter, i.e., scheduling jitter, is visible in message 9 which has the lowest priority, thus being sometimes scheduled several ECs later by RM scheduler.

Finally, the transmission control enforced by FTT-SE may also be beneficial under highly bursty loads. Without the transmission control the level of contention at the receivers might be too high for many network device drivers, which simply crash. The transmission control of FTT-SE prevents this abnormal situation by maintaining the submitted load under manageable levels per EC, obviously at the expense of enlarging the processing time for the same load. Nevertheless, this is sufficient to avoid such crashes and keep the system running. This phenomenon has been observed several times during the practical experiments.

3.6 Conclusion

The advent of switched Ethernet has opened new perspectives for real-time communication over Ethernet. However, a few problems subsist related with queue management policies, queue overflows and limited priority support. Meanwhile, several techniques were proposed to overcome such difficulties but they require specific hardware, are inflexible with respect to communication parameters or do not enforce timeliness guarantees. This chapter proposed using the FTT paradigm to achieve flexible communication with high level of control to guarantee timeliness and provide adequate queues

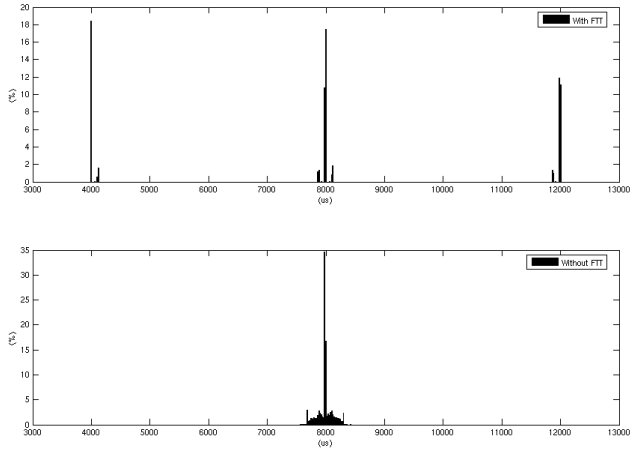


Figure 3.15: Histogram of inter-arrival times for message 1.

management in micro-segmented Ethernet networks. This resulted in the FTT-SE protocol. The chapter presented the mechanisms used to handle the periodic and aperiodic traffic and provided rules that allow building EC-schedules that respect the duration of the predefined transmission windows. Moreover, it described some implementation-oriented mechanisms intrinsic to the protocol that must be considered when instantiating the protocol in a system. Finally, several simulation and experimental results were obtained that exhibit the efficiency of the proposed approach in terms of using the aggregated switch throughput and enforcing different traffic scheduling policies. The results achieved also highlight the higher level of jitter control possible with FTT-SE when compared with COTS switches. However, for applications that are highly sensitive to jitter, further mechanisms might be required to also enforce low sub-EC jitter.

Chapter 4

Traffic schedulability analysis

This chapter presents RM and EDF schedulability tests for the FTT-SE protocol. As described in the previous chapter, the schedulability tests have to be executed online. For this reason, the research has focused on utilization-based tests, which can be implemented with linear time complexity. Additionally, the nature of utilization-based tests facilitates their adoption in dynamic Quality of Service (QoS) management mechanisms supporting a bandwidth-based QoS distribution through the services.

4.1 Introduction

Despite the relatively large amount of work on network-induced delay caused by switches, particularly on Ethernet switches [78][49], low attention has been devoted to utilization-based tests.

Utilization-based analysis are typically more pessimistic, compared to other analysis such as Network Calculus (NC) and Response Time Analysis (RTA), but on the other hand, they are faster, thus more appealing for use in dynamic frameworks such as the FTT-SE. Moreover, utilization-based analysis provide information on the amount of resource, bandwidth in this case, that can be used without compromising the system schedulability, which is of paramount importance to carry out bandwidth distribution among different streams within a dynamic QoS management framework. This topic is extensively described in Chapter 5.

In Ethernet switches, the usual presence of FCFS policy queues complicates using utilization-based tests. The works that consider such tests are built upon modified switches that carry out different scheduling policies such as RM or EDF. Conversely, FTT-SE overrides the effect of those FCFS policy queues and supports any scheduling policy on a COTS switch such as those that are suited for utilization-based tests. Nevertheless, these policies are implemented on a coarse scale (EC) and thus the use of utilization-based schedulability tests in FTT-SE requires some adaptations to the traffic model

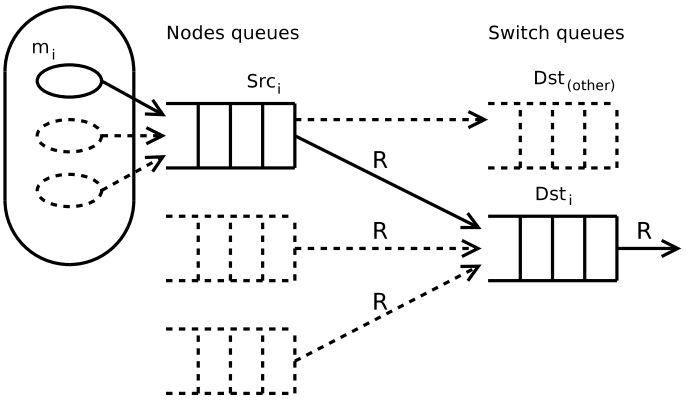


Figure 4.1: Traffic aggregation on switched Ethernet.

of such protocol.

4.2 Interference in the switch architecture

Multiple services competing for a common shared resource inevitably cause interference on each other. Contention among services is managed serializing the access to the resources under consideration. The most common approach to handle serialization is by using a ready queue, where all services wanting to use the resource are placed. The order in which the services are placed is defined by the scheduling policy.

On a switched Ethernet network, the nodes are connected by means of parallel links with each link being an active shared resource. The flows that traverse the switch establish interdependencies that make the traffic scheduling more complicated than just the scheduling of multiple independent resources.

Figure 4.1 sketches the interference path for a generic message m_i, when issued to a switched Ethernet network. The interference appears when aggregating traffic from different sources into bandwidth limited channels. Firstly, in the output queue of the sending node, m_i competes for the access in the uplink that conveys all messages from node Src_i to the switch. Once in the switch, m_i is forward to the destination link, Dst_i, where it eventually competes with messages from other source nodes.

Therefore, the transmission delay results from combining the delays on both queues, from the moment the application issues the packet transmission until it arrives at the destination node. Unless stated otherwise, in the following analysis the propagation times and switching delays are neglected, as they are fixed and easily accounted within the packet size. Also the computational delay within the nodes, sender and receiver, is included within the propagation time.

The scheduler within the FTT-SE protocol controls both, the nodes outgoing queues and the switch forwarding queues. However, while the former ones are directly managed by the scheduler, the latter ones, that are typically FIFO queues can only be overridden indirectly by controlling the submitted traffic.

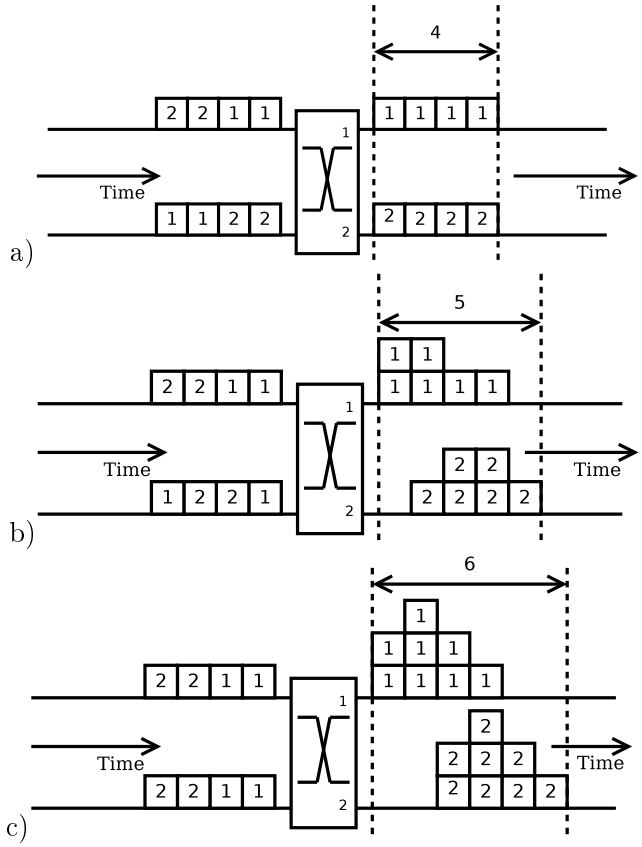


Figure 4.2: Inserted Idle Time on switch output queues (the numbers denote the destinations).

The interference in the nodes outgoing queues is easily assessed given the scheduler and the traffic activation models. However, evaluating the impact of the switch queues depends on the traffic pattern generated by the other nodes. As depicted on Figure 4.1, message m_i suffers interference on Dst_i from messages issued by nodes other than Src_i, but equally forwarded to Dst_i. To further complicate the analysis, these messages also accumulate the interference pattern from the first queuing step in the respective sources. Evaluating the interference becomes a complex task that depends on how the patterns are merged in the switch FIFO queues. Figure 4.2 illustrates the problem under consideration, by showing different interference patterns and the resulting message latencies. The scenario denotes a dual port switch. On

the left side are represented the uplinks and on the right side the downlinks. The boxes represent messages and the numbers inside, the destination port. On this example, packets are supposed to be ready at $t = t_0$ and issued as described on the uplinks. For instance, on Figure 4.2.a, the packets are issued in such a way that at each instance, packets are sent to different destinations. Clearly, in this case, there is no contention and the packets are immediately forward. On Figure 4.2.b, at $t = t_0$ the first transmissions are issued to the same output port (port 1). The switch promptly sends one of them while storing the other. After a complete transmission, the second transmissions are issued, with one of the messages forward to port 1 and queued. On the following transmitting steps, the same transmitted pattern is repeated but for the other output port (port 2). Finally, 4.2.c exacerbates this interference phenomena issuing consecutive sets of messages for the same port, leading to a maximum momentaneous backlog of 2 units. It is also noticeable the inserted idle time (IIT) of 2 units in the output port 2 caused by the non-existence of traffic to that port, which extends the transmission time by 2 units. Comparing these three scenarios it is noticeable the final transmission delay that varies according to the transmitting traffic patterns and how they correlate in time. It is evident the relationship between the additional transmission delay, the IIT and the message transmission pattern.

4.3 Interference within the FTT-SE periodic model

The previous section generically described the interference phenomenon that occurs with several traffic sources in a switched Ethernet network. To address this interference within the FTT-SE, the analysis has to conform with the periodic activation model, address the simultaneous scheduling of multiple inter-dependent links and fit additional constraints of the architecture, namely the additional inserted idle time (IIT) that results of strictly enforcing the synchronous window duration. The periodic scheduling model in FTT-SE considers N_s message streams (m_i) which descriptors are stored in the Synchronous Requirements Database Γ (Equation 3.1).

The streams are mainly characterized by the periodic activation release (T_i), message size (C_i) that can extend over several Ethernet packets and the forwarding directions that include the sending link (S_i) and one or more receivers ($\left[R_i^1..R_i^{ki}\right]$).

4.3.1 Window confinement

The FTT-SE scheduler strictly confines the periodic traffic to the synchronous window, with maximum size of LSW. This means that when a message cannot be fully transmitted within this window, it is kept in the ready queue and the scheduling is suspended and resumed in next EC, as illustrated in

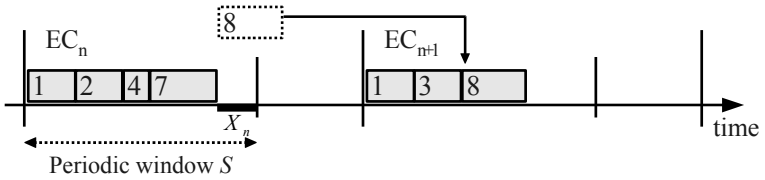


Figure 4.3: Impact of inserted idle time.

Figure 4.3. This property prevents the window overrun but it introduces idle time that must be accounted for in the analysis.

The effect of such inserted idle time has been thoroughly studied in [27] for fixed-priorities scheduling and [28] for EDF. Therein, it was shown that when scheduling within a periodic partition of length S and period E, as long as the message transmission time C_i is inflated with a compensation factor, becoming C'_i, as in Equation 4.1, with X being the maximum inserted idle time upper-bounded by the maximum packet length. Any schedulability analysis for preemptive scheduling with fixed or dynamic priorities applies.

$$C'_i = C_i \frac{E}{S - X} \tag{4.1}$$

When considering utilization-based tests, the compensation factor can be applied to reduce the utilization least upper bound while the transmission times C_i are kept unchanged as shown in Equation 4.2. We will use this approach in the remainder of this chapter.

$$\sum \frac{C_i}{T_i} \leq U^{lub} \times \frac{S - X}{E} \tag{4.2}$$

4.3.2 Deferred release in the downlinks

The interference between messages referred to in Section 4.2 also affects the FTT-SE periodic traffic. Such interference is responsible for the appearance of deferred release with jitter (J_i) in the downlinks. In fact, while in the uplinks, i.e., in the nodes, the periodic activation pattern can be enforced easily by the traffic scheduler, the same is not true in the downlinks. The amount of time that a message can be deferred in a downlink is directly related to the interference that it can suffer in the respective uplink from the messages that are sent to different destinations (Figures 4.3 and 4.2). This interference may vary from instance to instance, leading to a jittered arrival of the message at the downlink.

The deferred release phenomenon also occurs at intra-EC scale, as shown in Figure 3.7 and discussed in Section 3.3.2, when building each EC-schedule. However, here we do not account for any intra-EC aspects and we deal solely with the traffic scheduling at an EC scale and thus the deferred release jitter we are now concern with also occurs at this scale.

4.3.3 Scheduling multiple links

The scheduler operates over the message requirements data-base (SRDB) and builds a single global ready queue, serializing messages according to their priority, either static or dynamic. However, when dispatching the messages in the global ready queue, the protocol attempts to fill up the uplinks and downlinks in each EC in order to exploit as much as possible the parallel forwarding paths available in the switch.

This scheduling methodology approximates the model to a link oriented scheduling. The global ready queue is parsed in several ready queues, one per link, with the messages that traverse that link, and a scheduler that operates quasi-independently on each link with the restriction of scheduling messages together (in the same EC) in the uplinks and the downlinks. The intra-EC scheduling is handled according to the rules in Equation 3.2 that enforce the window protection and the causality effect between the uplink and the downlink.

Therefore, the traffic on a specific downlink is in fact constrained by the traffic explicitly submitted to that port and also the traffic scheduled in the uplinks of each message, causing interference. In a multicast or broadcast transmission scenario, messages are only scheduled if fitting the uplink and all downlinks in a given EC. In this case, it is enough to have one downlink full to prevent that message from being transmitted in that EC even if the other downlinks and uplink are lightly loaded. This represents a very strong interference. In a unicast message scenario this type of interference does not apply, since only one uplink and one downlink are involved. The interference verified in the uplink does not propagate to other downlinks.

In terms of schedulability analysis of a downlink, in the unicast scenario, only the interference from the uplinks is considered. However, in a multicast or broadcast messaging scenario, a multitude of interfering sources affecting the downlink must be considered. The interference in the downlinks is not confined to the traffic in the uplinks but also to the traffic in other downlinks, which by themselves may suffer interference from other uplinks and downlinks in a cascade procedure, pushing the complexity of the analysis to a higher magnitude.

Therefore, in this work we focus in the unicast scenario, only. At the end of this chapter a few considerations are done addressing the multicast/broadcast scenario.

4.4 Schedulability test - unicast

This section presents a schedulability test for the unicast only scenario. For all messages we consider the deadline equal to the activation period ($\forall_i, Di = Ti$) to prevent excessive efficiency degradation of RM and EDF schedulability utilization tests, and the release offsets equal to zero, i.e.,

synchronous release. The simplified traffic model is thus defined as:

$$SRT = \left\{ m_i : m_i = (C_i, T_i, S_i, \{R_i^1\}), i = 1..N_s \right\} \tag{4.3}$$

As described in Section 4.3.1, the packet transmission within this Ethernet-based implementation is non-preemptive. However, the FTT-SE framework allows the definition, at the logical level, of multi-packet messages, which can be preempted between packet transmissions.

The scheduling model is issued at the EC resolution scale. Without loss of generality, we consider the SRT to be sorted by ascending periods, i.e., $\forall m_i \neq m_j : i < j \Rightarrow T_i < T_j$.

We now consider two different cases that create different dependencies and interference. In the first case, each sender sends to a single destination, only, i.e., one uplink cannot issue traffic to distinct downlinks. In the second one, more general, the sender may generate messages to different destinations. Note that for both cases only unicast messaging is considered. A schedulability bound is then derived for each of the cases.

4.4.1 One node sending to one destination, only

In this first scenario all traffic generated within a node is sent to a single destination, only. Although simple, this is a frequent scenario where nodes send one or more data streams to a single destination node. In this case the traffic and interference in the uplinks is exactly transposed to the downlinks, with the plus that one downlink might hold traffic from different uplinks, which increases the interference. In this scenario, downlinks come to be the operational bottleneck. Hence, the schedulability of the system can be checked applying the utilization-based test in Equation 4.2 to each of the downlinks, as follows:

$$\forall_j \sum_{m_i \in l_j^d} \frac{C_i}{T_i} \leq U_{RM,EDF}^{lub} \times \frac{LSW - \max(C_i)}{E} \tag{4.4}$$

The rationale behind this test is that the model considers a strictly periodic generation of messages in the uplinks thus, schedulability under RM or EDF can be checked with an utilization test as in Equation 4.2. From the uplinks, two situations can occur in the downlinks. If the downlink receives from just one uplink, its traffic pattern will be the same, thus testing the downlink is enough and the test in Equation 4.4 applies. Similarly, if the downlink receives from several uplinks, there is higher interference in the downlink but without any deferred release effect since all interfering packets go to the same downlink. Moreover, there is no queuing inside the switch (across ECs) since the master polls the uplinks at a sufficiently low rate to prevent an overload in the downlink and following strictly the scheduling

criterion, be it RM or EDF. The uplinks always have similar or lower load and need not to be tested, thus test (4.4) also applies.

4.4.2 One node sending to multiple destinations

This is a more general and complex situation since the downlinks can now exhibit deferred release with jitter, which prevents the direct use of the existing utilization-based tests, as in the previous case. As mentioned in Section 4.3.3, the downlinks are subject to variations in the message arrival patterns due to scheduling in the uplinks that introduces variable delays, deflecting the strictly periodic pattern that is underlying the easily-coped tests in the previous scenario, in which there was no additional interference in the uplinks.

Despite the existence of several analysis that take release jitter into account, such analysis are mostly response time based, being hard to use online and particularly for dynamic QoS distribution.

Therefore, we developed a new utilization-based analysis that incorporates the release jitter of a task as a new virtual task that is added to the task set and we use such analysis for the downlinks. The analysis for the uplinks is still as presented in Equation 4.2. The schedulability test for this scenario is, then, the one in Equation 4.5.

$$\forall_j \begin{cases} \displaystyle\sum_{m_i \in l_j^u} \frac{C_i}{T_i} \leq U_{RM,EDF}^{lub} \times \frac{LSW - \max(C_i)}{E} & \text{(uplink)} \\ \underbrace{\underbrace{\displaystyle\sum_{m_i \in l_j^d} \frac{C_i}{T_i}}_{real\ load} + \frac{\max\limits_{m_i \in l_j^d} J_i}{T_1}}_{virtual\ load} \leq U_{RM,EDF}^{lub} \times \frac{LSW - \max(C_i)}{E} & \text{(downlink)} \end{cases} \tag{4.5}$$

Looking at the the lower condition in Equation 4.5, which applies to the downlinks, we see that it includes the actual utilization load of all messages forward to the downlink in analysis (*real load*), plus a term that represents the interference load (*indirect load*), computed dividing the maximum jitter ($maxJ_i$) among the various messages issued to the downlink by the period of the fastest message (T_1). Together, both components represent the *virtual load* in the downlink. We call it "virtual" in the sense that it includes the *indirect load*, which is not physically present in the link and corresponds to the inserted idle times caused by the interference in the uplinks. The proof for this test is presented in the following section.

4.4.3 Schedulability utilization bounds with release jitter

In this section we present the new utilization bounds for RM and EDF scheduling for general periodic and preemptive task sets with release jitter. For the sake of generality, the theorems use the expressions such as *task* (symbol τ) referring to *message* and *execution time* referring to *transmission time*.

These bounds directly support the lower schedulability condition in Equation 4.5, after applying the compensation factor due to the inserted idle time inside the synchronous window ($(LSW - \max(C_i))/E$).

Theorem 4.1. *A preemptive task set* $\Gamma \equiv \{\tau_i (C_i, T_i, J_i), i = 1..n\}$ *with* $D_i = T_i$ *is schedulable under RM if:*

$$\forall_{i=1..n} \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{\max_{j=1..i} J_j}{T_i} \leq U_{RM}^{lub}(i) \tag{4.6}$$

Proof:

In this model, the worst-case situation occurs when all tasks suffer the maximum release delay and are activated all at the same time and from then on, all following activations suffer no release delay [31]. Consider that $t = 0$ corresponds to the virtual periodic activation of some generic task τ_i that is released J_i later, in a worst-case interference scenario, illustrated in Figure 4.4 (top). Under these circumstances, the response time of task τ_i is upper bounded by the earliest instant t after $t = J_i$ that satisfies the following equation (Rwc_i):

$$Rwc_i = \min_{t>J_i} : t = \sum_{j=1}^{i-1} \left\lceil \frac{t - J_i + J_j}{T_j} \right\rceil C_j + C_i + J_i$$

Let us now advance the time scale so that $t' = 0$ will coincide with the earliest virtual periodic arrival among all tasks with priority equal to or higher than τ_i, i.e., $t \rightarrow t' = t + \max_{j=1..i} J_j - J_i$. In this situation, the response time Rwc'_i is now given by:

$$Rwc'_i = \min_{t' > \max_{j=1..i} J_j} : t' = \sum_{j=1}^{i-1} \left\lceil \frac{t' - \max_{k=1..i} J_k + J_j}{T_j} \right\rceil C_j + C_i + \max_{j=1..i} J_j$$

Since $t' - \max_{j=1..i} J_j = t - J_i$, we can conclude that $Rwc'_i \geq Rwc_i$.

The sum can be upper bounded by replacing J_j with $\max_{j=1..i} J_j$. Let

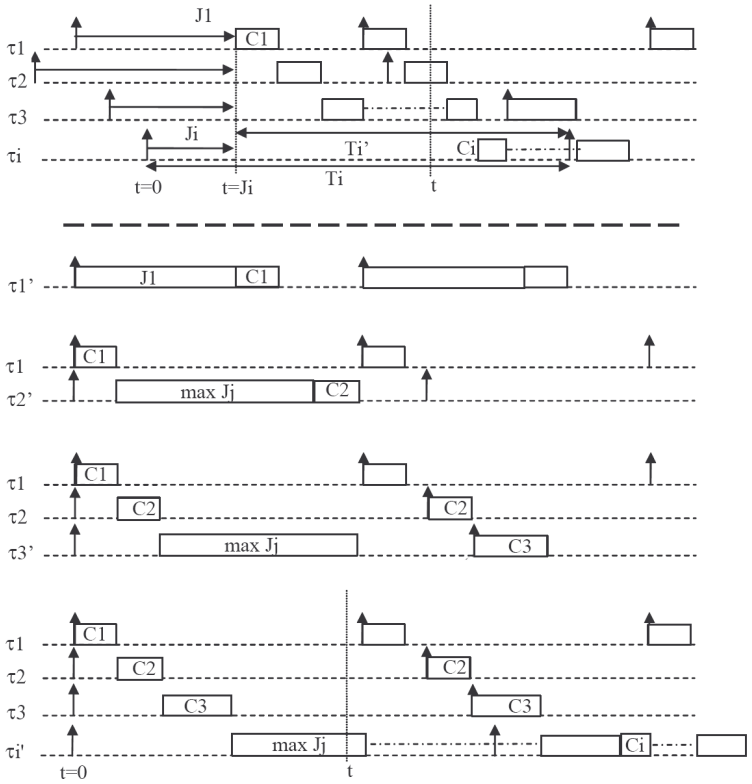


Figure 4.4: Worst-case situation with release jitter and the virtually added task in RM.

Rwc_i'' be the solution to the equation:

$$Rwc_i'' = earliest\ t' > \max_{j=1..i} J_j :$$
$$\sum_{j=1}^{i-1} \left\lceil \frac{t'}{T_j} \right\rceil C_j + C_i + \max_{j=1..i} J_j = t'$$

Necessarily, $Rwc_i'' \geq Rwc_i' \geq Rwc_i$. However, this last equation corresponds to a response time equation of a set of periodic tasks, without release jitter, but with task τ_i inflated to $C_i + \max_{j=1..i} J_j$. This transformation is shown in Figure 4.4. Thus, if this adapted task set is schedulable, so is the original task set with release jitter.

Finally, since the transformed task set is purely periodic, the Liu and Layland's utilization-based schedulability test applies [37], which proves the theorem. □

Corollary 4.1. *A preemptive task set* $\Gamma \equiv \{\tau_i (C_i, T_i, J_i), i = 1..n\}$ *with* $D_i =$

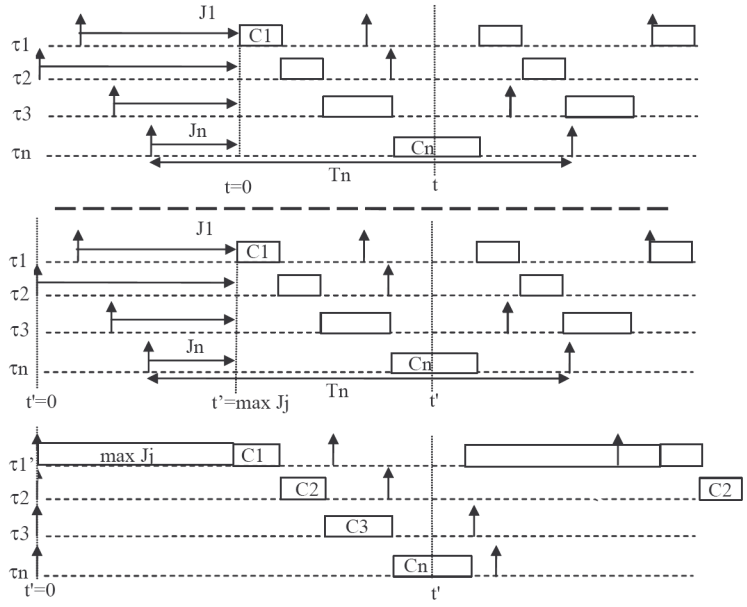


Figure 4.5: Worst-case situation with release jitter and the virtually added task in EDF.

T_i is schedulable under RM if:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{\max_{i=1..n} J_i}{T_1} \leq U_{RM}^{lub}(n) \tag{4.7}$$

Proof: Just note that $\forall_i, \max_{j=1..i}(J_j)/T_i \leq \max_{j=1..n}(J_j)/T_1$,

$$\forall_i, U_{RM}^{lub}(i) \geq U_{RM}^{lub}(n)$$

□

Theorem 4.2. *A preemptive task set $\Gamma \equiv \{\tau_i(C_i, T_i, J_i), i = 1..n\}$ is schedulable under EDF if:*

$$\forall_{i=1..n} \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{\max_{j=1..i} J_j}{T_i} \leq U_{EDF}^{lub} = 1 \tag{4.8}$$

Proof:

With this model, the worst-case situation in terms of highest consecutive processor demand also occurs in a similar manner as in the previous theorem [121]. The situation is also illustrated in Figure 4.5 (top), in which all tasks

are released as late as possible and all at the same instant. Let us define such instant as $t = 0$. Then, following a processor demand approach, this set is schedulable if:

$$\forall_t, h(t) = \sum_{i:(T_i - J_i) \leq t} \left\lfloor \frac{t + J_i}{T_i} \right\rfloor C_i \leq t$$

$$\Leftrightarrow \forall_t, h(t) + \max_{i=1..n} J_i \leq t + \max_{i=1..n} J_i$$

Now, let us carry out a scale shift making $t' = 0$ coincide with the earliest virtual periodic arrival, i.e., $t' \rightarrow t' = t + \max_{i=1..n} J_i$. The previous schedulability condition can be written as:

$$\forall_{t'}, h(t' - \max_{i=1..n} J_i) + \max_{i=1..n} J_i \leq t' \quad \Leftrightarrow$$

$$\sum_{i:(T_i - J_i + \max_{j=1..n} J_j) \leq t'} \left\lfloor \frac{t' - \max_{j=1..n}(J_j) + J_i}{T_i} \right\rfloor C_i + \max_{j=1..n}(J_j) \leq t'$$

Again, the sum is upper bounded by replacing J_i with $\max_{j=1..n} J_j$. Thus, the following condition implies the previous one and still guarantees task set schedulability.

$$\forall_{t'}, \sum_{i:Ti \leq t'} \left\lfloor \frac{t'}{Ti} \right\rfloor C_i + \max_{j=1..n}(J_i) \leq t'$$

In order to obtain an utilization-based expression we divide both sides by t' leading to:

$$\forall_{t'}, \sum_{i:Ti \leq t'} \left\lfloor \frac{t'}{Ti} \right\rfloor \frac{C_i}{t'} + \frac{\max_{j=1..n}(J_i)}{t'} \leq 1$$

Observe that,

$$\left\lfloor \frac{t'}{Ti} \right\rfloor \leq \frac{t'}{Ti} \Rightarrow \left\lfloor \frac{t'}{Ti} \right\rfloor \frac{C_i}{t'} \leq \frac{C'_i}{t'} * \frac{t'}{Ti} = \frac{C'_i}{Ti} (as \; \frac{C'_i}{t'} \geq 0)$$

Therefore, the term inside the sum is upper bounded by C_i/T_i. Thus, the following condition also implies the previous and still guarantees task set schedulability.

$$\forall_{t'}, \sum_{i:Ti \leq t'} \frac{C_i}{T_i} + \frac{\max_{j=1..n}(J_i)}{t'} \leq 1$$

This condition needs to be evaluated at the instants in which t' is an integer multiple of any of the tasks periods. This is particularly relevant for $T_1 \leq t' \leq T_n$. Within this interval, each time t' reaches the first period of any task, say τ_i, the corresponding term C_i/T_i is included in the sum. At the same time, the term $\max_{j=1..n} J_j/T_i$ becomes lower as i grows since the set

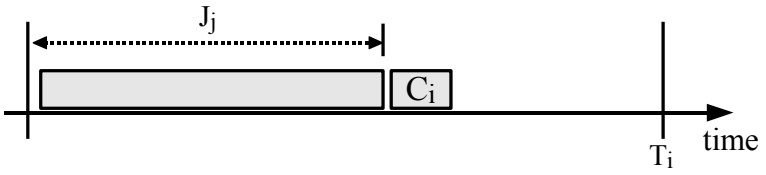


Figure 4.6: Interfering task.

is sorted by ascending periods. For $t' > T_n$, the left term in the condition decreases monotonically and the condition is trivially verified if it was verified in the previous interval. □

Corollary 4.2. *A preemptive task set* $\Gamma \equiv \{\tau_i (C_i, T_i, J_i), i = 1..n\}$ *is schedulable under EDF if:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{\max_{i=1..n} J_i}{T_1} \leq U_{EDF}^{lub} = 1 \tag{4.9}$$

Proof: Again just note that

$$\forall_i, \max_{j=1..i} (J_j)/T_i \leq \max_{j=1..n} (J_j)/T_1$$

□

4.4.4 Upper bounding the indirect load

Inequalities 4.6 and 4.8, for RM and EDF respectively, are computed considering the *real load* directly obtained from each task utilization load and the *indirect load* that represents the interference. The latter term is not assessed using each task utilization load though. For this reason, the following proposed test aims at upper bounding this *indirect load* in a way to become utilization-based.

The *indirect load* in Inequalities 4.6 and 4.8 is equivalent to the impact of an additional task with a load given by U_{ind_i} (Equation 4.10).

$$\begin{aligned} U_{ind_i} &= \frac{\max_{j=1..i} J_j}{T_i} \quad , \forall i = 1..n \\ &= \max_{j=1..i} \frac{J_j}{T_i} \end{aligned} \tag{4.10}$$

For simplicity let $U_{ind_{i,j}} = \frac{J_j}{T_i}$ be the interference caused by the j^{th} task (τ_j) in τ_i. Figure 4.6 illustrates one such task for which the load is $U_{ind_{i,j}}$.

Coming back to the switch case, remember that the release jitter that affects message m_j in its downlink is a consequence of messages in the same

uplink that are transmitted to other downlinks. Call $\mathbb{C}^{RM}$ and $\mathbb{C}^{EDF}$ the set of such messages for RM and EDF scheduling, respectively. For each case, we can also express J_j as the sum of the transmission times of the interfering messages leading to an *indirect load* that can be expressed as in Equation 4.11 and 4.12 for RM and EDF, respectively.

$$U_{ind}{}_{i,j}^{RM} = \frac{\sum_{\mathbb{C}^{RM}} \left\lceil \frac{J_j}{T_k} \right\rceil C_k}{T_i}, \text{ where } \mathbb{C}^{RM} \equiv \forall m_k : \begin{cases} S_k{=}S_j \\ R_k{\neq}R_j \\ k{<}j \end{cases} \tag{4.11}$$

$$U_{ind}{}_{i,j}^{EDF} = \frac{\sum_{\mathbb{C}^{EDF}} \left\lceil \frac{J_j}{T_k} \right\rceil C_k}{T_i}, \text{ where } \mathbb{C}^{EDF} \equiv \forall m_k : \begin{cases} S_k{=}S_j \\ R_k{\neq}R_j \end{cases} \tag{4.12}$$

Knowing that $J_j < T_i - C_i < T_i$ an upper bound to the interfering load is given by Equation 4.13, as follows [1]:

$$\begin{aligned} U_{ind}{}_{i,j} &< \frac{\sum \left\lceil \frac{T_i}{T_k} \right\rceil C_k}{T_i} \\ &< \frac{\sum \left(\frac{T_i}{T_k} + 1 \right) C_k}{T_i} = \sum \frac{C_k}{T_k} + \sum \frac{C_k}{T_i} \qquad \Leftrightarrow \\ U_{ind}{}_{i,j} &< \sum U_k + \sum \frac{C_k}{T_i} \end{aligned} \tag{4.13}$$

Thus, we obtain the schedulability test for RM and EDF given by Inequalities 4.14 and 4.15 that must be conducted to each message m_i of a given downlink.

RM:

$$\forall_{i=1..n} \sum_{j=1}^{i} U_j + \max_{j=1..i} \left(\sum_{\mathbb{C}^{RM}} U_k + \sum_{\mathbb{C}^{RM}} \frac{C_k}{T_i} \right) \leq U_{RM}^{lub}(i) \tag{4.14}$$

EDF:

$$\forall_{i=1..n} \sum_{j=1}^{i} U_j + \max_{j=1..i} \left(\sum_{\mathbb{C}^{EDF}} U_k + \sum_{\mathbb{C}^{EDF}} \frac{C_k}{T_i} \right) \leq U_{EDF}^{lub}(i) \tag{4.15}$$

This test, that requires the verification of n conditions, can be simplified to one condition, only, upper bounding the terms in the sums leading to a

[1] For the sake of clarity we omit the indexes in the sums.

pair of tests, for RM and EDF, that contain one single condition, Equations 4.16 and 4.17.

RM:

$$\sum_{m_i \in l_j^d} U_j + \max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{RM}} U_k \right) + \frac{\max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{RM}} C_k \right)}{T_1} \leq U_{RM}^{lub}(i) \tag{4.16}$$

EDF:

$$\sum_{m_i \in l_j^d} U_j + \max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{EDF}} U_k \right) + \frac{\max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{EDF}} C_k \right)}{T_1} \leq U_{EDF}^{lub}(i) \tag{4.17}$$

These tests only contain additive utilization terms that particularly well suited to our purpose of supporting dynamic QoS management. In order to use these terms in FTT-SE we just need to include the idle time compensation factors in the right side of the inequalities, and replace the lower condition in 4.5.

4.5 Simulation results

The previous analysis provides the conditions to assess the schedulability of the traffic submitted to the switch. However, the conditions contain several approximations that introduce a non-negligible level of pessimism. In order to validate the analysis and develop a notion of the pessimism embedded in these schedulability tests we carried out simulations with randomly generated traffic. For each generated set, it is verified whether the set passes the schedulability tests and if so, the message set is submitted to the scheduling simulator that runs for the LCM of the message periods to determine the set schedulability (note that simulations consider a synchronous release).

The message sets are pseudo-randomly generated with a load around the theoretical schedulability bound, in order to have schedulable and non-schedulable sets. The pseudo-random generation uses the following parameters:

- 4 ports switch with 100 Mbps links
- Packet length uniformly distributed within [100 1500] bytes
- EC of 1ms
- Periods uniformly distributed within [1, 5] ECs
- Unicast messages, only
- 1, 2 or 3 destinations per uplink (source)
- Scheduling with EDF and RM

- Inserted idle time compensation factor of 88% (already including EC length, TM transmission, turnaround, and maximum packet length)
- 200000 points per utilization point in the graphs in the part where schedulability starts to reduce.

The theoretical tests in Equations 4.5 and the extension to the downlink test in Equations 4.16 and 4.17 compare the traffic utilization with the EDF or RM utilization bounds. The simulation procedures differ only in the pseudo-random generator that for each case generates message/task sets around the respective schedulability limits.

For the sake of clarity, note that the designation virtual load is defined in the downlinks to include *real load* plus the interference load given by

$$\max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{RM,EDF}} U_k \right) + \frac{\max_{m_i \in l_j^d} \left(\sum_{\mathbb{C}^{RM,EDF}} C_k \right)}{T_1},$$

in Equations 4.16 and 4.17. For simplicity, let us extend this designation to the uplinks in which case virtual load coincides with the *real load*. The switch links, either uplinks or downlinks, can now be analyzed inter-changeably with the same per link utilization limit U^{lub}. The virtual load is the per link load used to verify the schedulability in the uplinks and downlinks on each of the switch ports (4 ports in this simulation scenario).

Each message set was generated computing random messages that were sequentially added to the set while keeping the virtual load on each link under a given value, lets say x Mbps. New messages were continuously added until there were 1000 consecutive failed attempts,i.e., attempts that cause an excess to the desired load. This guarantees that the virtual load generated in any link is no more that x Mbps and yet very close to that limit on most of the links, mainly in the downlinks that tend to reach the limit before, given the interference load. In fact, the real traffic load on the downlinks is much lower than the corresponding virtual load.

The generated sets were simulated with synchronous release during LCM of the periods and the percentage of schedulable sets was plotted as a function of the maximum virtual load (x Mbps). The following plots were obtained forcing the maximum allowed virtual load per link. They also show the Liu and Layland schedulability bounds for RM and EDF, after the inserted idle time compensation factor, which become 61% and 88%, respectively. Note that the analysis only guarantees positive scheduling results when the vitual load in all links is below these thresholds. The points of virtual load larger than the schedulability bounds with 100% schedulable sets is a measure of the pessimism that is implicit in the analysis.

Figure 4.7 shows the case of one destination per source, i.e., messages from one uplink go all to the same downlink, for which schedulability test in

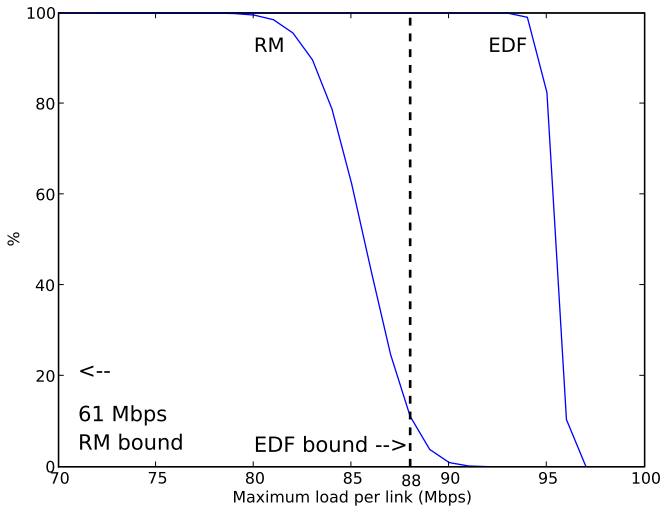


Figure 4.7: Schedulability vs. Utilization(1→1).

Equation 4.4 applies. In this case, the virtual load coincides with the real load which is necessarily less than the links capacity of 100 Mbps. These results are similar to those for RM and EDF scheduling in one CPU, as expected, showing close to optimal performance for EDF (93 Mbps actually schedulable against the test bound of 88 Mbps) and a larger deviation from optimal for RM (79 Mbps against 61 Mbps respectively).

Figure 4.8 shows the cases of 2 and 3 destinations per source with the impact of interference included in the virtual load, which can now grow beyond the link capacity of 100%. As shown in the plots, Figure 4.8, positive scheduling results are obtained with virtual load beyond the 100 Mbps of the link capacity. This happens because the considered worst-case scenario for the interference in the uplinks may never occur in practice. Note that the *real load* is much smaller, not exceeding the physical limits as illustrated also in Figure 4.8 (dotted plot).

In all scenarios plotted in Figure 4.8 we may observe that the first deadline misses (with growing utilization) occur for message sets with a maximum virtual load above the 61 Mbps and 88 Mbps schedulability bounds of RM and EDF scheduling, respectively. No violation of these bounds was found, considering the maximum virtual utilization per link, confirming the schedulability analysis. The first deadline misses occur for virtual loads of 89 Mbps and 97 Mbps, for RM and EDF, respectively.

The number of schedulable sets beyond the proposed scheduling bounds reveals the bounds schedulability penalty. There are three sources for this penalty, the one already included in Liu and Layland's RM test, the intra-

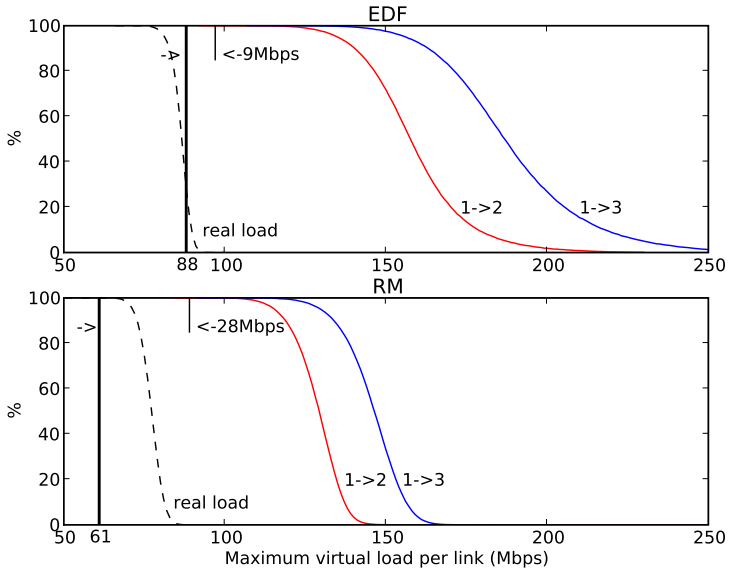


Figure 4.8: Schedulability vs. Utilization using maximum virtual load per set $(1 \rightarrow 2, 3)$.

EC inserted idle time compensation and the interference term $\max_i(J_i)/T_1$ in the downlinks. For EDF scheduling only the two latter factors apply, thus resulting in a lower degree of pessimism. This is confirmed by the simulations observing the difference between the first non-schedulable sets and the respective bounds, which is 28 Mbps for RM and 9 Mbps for EDF.

In Figure 4.8 we may also observe that when increasing the diversity of destinations per source (range of downlinks per uplink) the scheduling penalty also grows. This is revealed by the observation that with the scenario of 3 possible destinations (1->3) there are more schedulable sets beyond the bounds than with the scenario of 2 possible destinations (1->2). In fact, with more downlinks per uplink the number of uplinks interfering in a downlink may grow and thus we are increasing the chance for a higher term $\max_i(J_i)/T_1$ in Equation 4.5. Moreover, the impact of the deferred release on each downlink is more visible at run-time when all interfering uplinks are sending to different downlinks, which, for a higher diversity of downlinks per uplink, is more likely to happen.

4.6 Multicast/Broadcast analysis

Considering a pure unicast messaging model greatly simplifies the system analysis, specially when it comes to evaluate the interference in the down-

links. When considering a multicast or broadcast scenario, the traffic interference observed in the downlinks is due to other traffic patterns triggered to the respective uplinks, as well to traffic issued to downlinks, other than the one in analysis. The reason is that all transmissions in all involved downlinks must occur in the same EC.

Referring to the downlinks analysis test within the unicast model, when it comes to include the interference as release jitter, it is considered the maximum possible jitter generated on each of the uplinks. In the multicast scenario however, evaluating the jitter is not as simple. In this case, the jitter must include the interference within the uplinks, which may additionally suffer interference related to the scheduling on other downlinks and recurrently other uplinks. This recurrent procedure may reach high complexity levels that vary with the system scale and with the message dependencies across the resource components used (uplinks and downlinks).

The strong penalty that this kind of traffic can have was already illustrated in Section 3.5.1 in Figure 3.11. In the limit, if usinf only broadcast messages, the throughput of the switch is reduced to that of a single link. The analysis for this situation is out of the scope of this thesis.

4.7 Conclusion

In order to support efficiently dynamic QoS distribution, the FTT-SE framework needs a fast schedulability analysis tool that provides operational bounds, supporting quantitative information on the amount of free resources. Often associated to the periodic activation model, utilization-based tests are one possible choice to meet such requirements.

However, on switched Ethernet, the switch queues introduce distortion to the periodic model and so the well known utilization-based analysis does not directly apply. This chapter addressed this topic and proposed a new schedulability test based on the well known utilization bounds for EDF and RM preemptive scheduling. These tests have very low time complexity and can be performed online in dynamic environments. Another property of this analysis is the ability to consider the links as different resources. These properties make this analysis an important building block to support dynamic QoS management. However, due to the large complexity of analyzing the multicast/broadcast transmissions, only the unicast transmission model was analyzed and validated with simulations. The former models are left for future work.

This theoretical work, for the schedulability test that includes considering the interference on a periodic message caused by release jitter is general and applicable to the FTT-SE framework as well as to a set of periodic tasks scheduled preemptively with RM or EDF and suffering release jitter. The exact-case characterization of the proposed analysis and comparison with

other known ones, such as response time based tests, is also left for future work.

Chapter 5

Dynamic QoS management

This chapter discusses QoS adaptation techniques suitable for use within a dynamic real-time network. Existing network QoS management techniques either cannot cope with dynamic communication requirements or lack efficiency in bandwidth utilization. In this regard, some generic models are proposed for the QoS distribution among communication streams, as well as the corresponding optimization. Finally, this chapter discusses the integration of these QoS management techniques within the FTT-SE protocol.

5.1 Introduction

Real-time systems have traditionally neeb designed considering worst-case requirements that are fixed throughout the system lifetime. However, real-time systems may have to deal with dynamic environments. In this case, considering fixed worst-case requirements (static) penalizes the resource utilization efficiency since, even when not required, the resources are permanently reserved for the worst-case scenario (over-provisioning). Pushed both by the need to reduce costs or by deployment constraints, there has been a greater interest on providing more flexibility to the resource allocation, supporting dynamic reconfigurations and dynamic adaptability based on Quality of Service (QoS) rules.

Nevertheless, predictability is still an issue in the the design of real-time systems. Hence, the introduction of operational flexibility demands for appropriate models for both the application platform sides, capable of guaranteeing minimum QoS levels. This introduces a different design paradigm, where the application should be able to specify acceptable service ranges as well as utility functions to relate the provided service levels with the application performance. On such scenario, predictability can still be assured using the minimum usage levels, while at run-time, a dynamic QoS manager distributes the remaining capacity among the subscribed services, to maximize the global performance.

The focus of this work is on studying such dynamic QoS management schemes, given the requirements for operational flexibility. In this regard, several distribution models are presented and compared according to their computational complexity. While some of the results are generic, the work herein presented is mainly focused on the FTT-SE framework, addressed in this thesis. Therefore, the QoS models and distribution schemes are presented within this framework.

5.2 The QoS management problem

Consider a system composed of several subsystems that compete for the access to system resources, e.g., network links or nodes CPUs. For a broad range of applications of this kind there is a direct relationship between the amount of resources allocated to the application services and their individual performance. This is the case of many control systems in which, within certain limits, higher sampling frequencies and thus higher CPU utilization and network bandwidth in distributed architectures, provide a better control quality (lower error, more stable control signals,...)[29].

A natural and simple solution for the resource capacity distribution would be to give maximum service level to each of the service subsystems involved. However, if done without control, it would eventually end up with an overload or a worst-case situation. If the system becomes overloaded, ultimately, the services get degraded in an uncontrolled manner, which is not acceptable in most systems.

Another simple approach would be to give only the minimum possible service level to each subsystem. In this case, it is possible to have a situation where there are available resources that are not allocated to subsystems that could take advantage of them, thus the global application performance is sub-optimal.

A third option can be followed by enforcing service degradation in a controlled manner. Modeling the services with QoS ranges allows managing the QoS degradation of each specific service with respect to its maximum requirements in order to maximize the overall resource usage and assure the system correctness. Along with this model, it is essential the deployment of proper management policies with specific strategies regarding the resource capacity distribution.

This section describes such resource capacity distribution strategy based on Quality of Service management rules associated to the service model. Yet, one question raises:

What exactly do we mean by Quality of Service?

Quality of Service (QoS), in a broader sense, is a measure of the performance level provided by a generic server. The term is sometimes confused

with *service quality*, which more closely refers to the quality experienced by the user [64]. QoS frequently denotes a more precise and objective measure, usually associated to a negotiation contract between a service vendor and the respective client [50].

There are different metrics to measure the QoS level, varying with the application class and purpose. For example, in generic processing it is common having, for general purpose jobs, metrics that minimize the time by which a job is delayed, or in control-oriented applications metrics that minimize the sampling execution jitter.

In the communication networks domain, a direct relationship is commonly established between the service bandwidth and the QoS delivered. For example, in video streaming, within certain limits, higher bandwidth results in higher user perceived quality or Quality of Experience [55].

The role of the QoS manager comprehends the coordination and management of all resource parameters that play actively in the QoS level. In the networking domain, it includes the management of the communications load, that must be kept within the network capacity, and the traffic scheduling.

In real-time systems, the QoS guarantees come along with the ability to meet the timeliness constraints set within the service description model. Those guarantees are the result of a schedulability analysis performed at the admission phase. The role of a QoS manager on such a real-time network is then focused on the bandwidth distribution and enforcing QoS differentiation between the services, while maximizing the global system performance [39].

For the networking and multimedia communities the QoS management is often considered a service broker [93] that analyzes the resource status and accepts or rejects a service request with a given QoS level. Several negotiation protocols are example of this, DiffServ [94], IntServ [36] or RSVP [136]. These approaches however, do fixed provisioning of network resources at service startup that remains constant for the service duration, thus resulting in a fixed QoS. If a QoS variability range is considered in the reservation contract, defining when the service is practicable, it becomes possible adjusting, at run-time, the QoS of currently active services in order to better accommodate more service requests and maximize the throughput of the resource [65]. This is hereby designated as dynamic QoS management.

Notwithstanding, the following question raises:

> When free network bandwidth is available, how to distribute it among the current services?

And conversely:

> When the network bandwidth is overloaded, how to manage the QoS degradation of current services, needed to eliminate the overload?

These are two perspectives of the same problem: the QoS distribution among a set of services that individually seek for the best QoS, within the resource capacity.

This work focuses on online QoS management schemes for dynamic networked environments. This imposes a constraint since the QoS algorithms must be sufficiently fast to enable fast adaptation.

5.2.1 The resource capacity

Resource capacity is one key property of a shared resource. It defines the resource sharing limits for providing services. Communication networks are resources which the capacity limit is given by the maximum data they can transmit per time unit, i.e., the network bandwidth. However, the use of specific scheduling models poses additional constraints that may reduce such capacity.

A scheduler is an important piece in the resource management, being responsible for temporally organizing the execution of services in the resource, given the services run-time constraints. It is thus an integral and determinant component for the system analyzability.

The schedulability test conducted within the system analysis phase ultimately determines whether a service is admitted or not, which thus means that this test is based on the definition of the resource capacity.Schedulability tests can be more or less accurate and complex.Usually, there is an interdependency between the test accuracy and the associated computational complexity. More accurate tests often imply higher computational overhead, e.g. tests that mimic the scheduler execution or perform a response time analysis. Pessimism is sometimes the price to pay for a faster analysis, meaning tests that are just sufficient, thus potentially rejecting services QoS combinations that would be schedulable.

Dynamic real-time systems often require a bounded response time to the system modification requests. Hence, the amount of time spent in this process must be as low as possible. In a QoS management procedure, this time must allow quantifying the amount of available resource and finding a resource distribution that leads to a global utilization as large as possible and is schedulable.

Therefore the scheduling analysis must rely on a per service metric that represents simultaneously the quality-level and the respective share of the resource capacity. This direct relationship between the quality-level and the share of the resource capacity with an appropriate schedulability test, assures the system correct behavior.

Resources such as networks or processors commonly make use of schedulers with feasibility tests that define bandwidth bounds as guarantees for the system schedulability. Hence, as long as the resource utilization load stays within those limits, the system schedulability is assured. In this case

the resource capacity may be defined as that schedulability utilization bound. Then, the bandwidth distribution mechanism becomes independent from the schedulability test that is implicitly enforced by using a resource capacity equal to that bound.

5.2.2 The application model

Dynamically changing the quality level of a service requires a tight cooperation with the application. For example, if in a video surveillance application the bandwidth allocated to a given stream is suddenly reduced, the application has to adapt accordingly, e.g. by reducing the frame-rate or increasing the compression factor.

Therefore, a reflexive model is needed according to which the resource informs the application of its current QoS level allowing the application to adjust to it.

Such model is compatible with a flexible use of the resource and it allows the application to operate with different requirements and adapt online. In this regard, the application services are categorized in two classes, the inelastic services that only operate under fixed requirements and the elastic ones that operate with different configurations within a predefined requirements range [39].

For the elastic class of services, several service/task models have been proposed toward the extension of the timing requirements beyond the usual hard and soft real-time, based on the assumption that services can have their performance degraded without jeopardizing the application. Generically, the model specifies the services with minimum requirements, i.e., least required QoS, which are granted once the service is admitted. The quality level may then be improved depending on the resource availability given the other admitted services that compete for the shared resource.

Chapter 2 described several of these models in detail. Some of them are based on the enforcement of specific job skipping patterns to alleviate the system load when in overload. For example a control system can skip cycles occasionally or a video transmission that can skip frames. However, these models cause a sudden drop in the instantaneous QoS provided to the application whenever a skip is done that might produce undesired effects. Moreover, there is not a univocal and direct relationship between the chosen skipping patterns and the QoS that is obtained [42]. Additionally, the schedulability analysis techniques become computationally heavier as they require a careful analysis of the patterns and the impact of the interference.

Therefore, other techniques have been proposed to distribute the resource capacity based on the utilization load, mostly by changing the tasks periods. These schemes aim at providing a fair and correct bandwidth distribution of the resource. However, they are either specific to a QoS metric, or require a complex and computationally heavy structure, a problem aggravated by the

fact that in some cases it is necessary to manage multiple resources, thus a multi-dimensional problem. Similarly to the elastic task model (ETM) proposed by Butazzo *et. al.* [38], we propose several QoS distribution models that use the bandwidth as the differentiation metric. However, our proposal goes beyond the specific ETM model that uses a single distribution model to manage the tasks periods since we allow an integrated management of both periods and execution times.

Another important aspect to add relates to the definition of the operating range. The proposed schemes define a continuous bandwidth range, or period range in the ETM, wherein the application operates. However, some applications pose additional constraints to the operating range, for example being only able to operate with discrete bandwidth values. Example of this are video cameras that sometimes operate under certain specific frame sizes and frame rates. This topic is further detailed in Section 5.4, where the application service model is extended to include a discrete model that defines multiple operating points. For now, consider the existence of a valid continuous bandwidth range describing the minimum and the maximum (or preferred) requirements.

Let St denote the set of services concurring for a resource R and U_R denote the resource capacity.

$$St \equiv \{St_i \left(U_{min_i}, U_{lax_i}, qos_i\right), \quad i = 1..n\}$$

Each service St_i defines a minimum bandwidth U_{min_i}, maximum bandwidth $(U_{min_i} + U_{lax_i})$ and importance, qos_i. The system integrity is assured as long as the minimums fit the resource capacity, defined within the schedulability analysis, i.e., $\sum_{i=1}^{n} U_{min_i} \leq U_R$. After distributing those minimums, the remaining bandwidth U_{spare} is available to be distributed by the QoS manager, being given by:

$$U_{spare} = U_R - \sum_{i=1}^{n} U_{min_i} \tag{5.1}$$

The resource is overloaded when the services maximum load exceeds the resource capacity, i.e., $\sum_{i=1}^{n} U_{lax_i} > U_{spare}$, leading to a lack of bandwidth given by:

$$\overline{U_{spare}} = \sum_{i=1}^{n} U_{lax_i} - U_{spare} \tag{5.2}$$

Figure 5.1 depicts the distribution model for two concurrent services St_1 and St_2. U_{lax_1} and U_{lax_2} represent the bandwidth flexible part of each service and U_{spare} the resource bandwidth that remains free for distribution.

The distribution problem is now reduced to distribute U_{spare} by the services given their U_{lax_i} operating ranges. In many applications, different

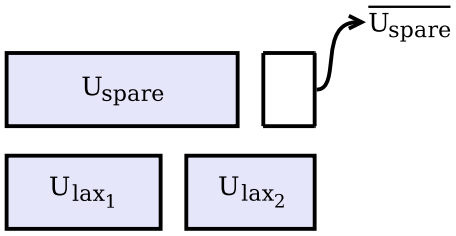


Figure 5.1: Remaining bandwidth.

services have different purposes and impact on the global application performance and thus should be differentiated on that regard. The qos_i parameter, used by several techniques, serves this purpose, allowing to differentiate the importance of different services.

Based on this model, there are several techniques that can be used to distribute the U_{spare} bandwidth among the services and obtain the set of current utilizations $\{U_i, i = 1..n\}$.

5.3 Bandwidth distribution

Online QoS distribution within dynamic real-time networks has to be fast. For this reason, the service bandwidth utilization (load) is used as the QoS metric for the distribution procedure. The bandwidth utilization can be directly computed from the periodic service model, reflecting the impact of the respective task or service in the overall capacity of the resource. The use of this metric also takes advantage of the existence of fast and reliable schedulability analysis techniques based on resource utilization.

The bandwidth that is available for distribution, U_{spare}, is defined by the load of the submitted services and by the resource capacity. That distribution can be achieved using different policies, all based on the parameters provided for each service. Different policies may result in distinct QoS provisioning for the services. Therefore, when specifying the service parameters, the distribution model must be considered, too.

This section addresses the bandwidth distribution among the services. The materialization of such bandwidth into the operational parameters, valid for the services, is addressed in Section 5.4.

Two algorithmic procedures carrying out the bandwidth distribution are presented, one that assigns bandwidth in a greedy basis (Section 5.3.1 - Fixed importance distribution) and another one that considers each service importance in terms of a weighted share (Section 5.3.2 - Weighted distribution). The fixed approach presents a lower computational complexity but is not fair. For systems in which fairness is important the weighted distribution policy is more appropriate. However, this technique presents a higher com-

putational complexity and may require an iterative procedure, discussed in Section 5.3.3, to converge to an optimal solution.

5.3.1 Fixed importance distribution

Perhaps the simplest and most direct policy to discriminate the services when assigning free bandwidth is the one that, starting from the most important (highest qos_i), sequentially concedes all the bandwidth that each one may use until exhausting the resource capacity. However, if the most important services drain all the spare bandwidth, the low rated services get no extra bandwidth share.

This model appears in resource management frameworks such as FRESCOR, see Section 6.1, where an importance parameter is included along with the contract negotiation of each service. The FRESCOR framework further provides a quality weight that differentiates services within the same importance class, i.e., hierarchical composition of two distribution mechanisms.

5.3.2 Weighted distribution

The weighted distribution technique contrasts with the greedy technique by providing a fair distribution based on weights. The weights are the application specifications and influence the bandwidth share obtained.

Each service weight w_i is obtained through service application parameters, namely the qos_i parameter and the bandwidth laxity parameter, U_{lax_i}. This section only covers the impact of using such weights. Section 5.3.4 addresses the weights formulation based on the application parameters. Notice though that we assume the weights to be normalized, i.e., given a set of services $\sum_{i=1..n} w_i = 1$.

After the distribution, a service i gets the bandwidth given by:

$$Uv_i = U_{min_i} + V_i \tag{5.3}$$

where V_i represents the share allocated to St_i from U_{spare}.

Now, two bandwidth distribution mechanisms are presented:

- *Direct Share*, in which the services are given a direct weighted share from U_{spare} (equation 5.4);

$$\forall St_i : V_i' = w_i'.U_{spare} \tag{5.4}$$

- *Indirect Share*, where a weighted share of $\overline{U_{spare}}$ is taken from each of the U_{lax_i} (equation 5.5).

$$\begin{aligned} \forall St_i \in St : V_i'' &= U_{lax_i} - w_i''.\overline{U_{spare}} && (5.5) \\ &= U_{lax_i} - w_i''.\left(\sum_{j=1}^{n} U_{lax_j} - U_{spare}\right) && (5.6) \end{aligned}$$

These are two bandwidth distribution schemes, with distinct mathematical formulation, that fully distribute the resource capacity (U_{spare}), given the proper weights. However, each scheme perceives differently the weights and so produces distinct results. The w_i weights have in fact opposite meanings in the two schemes. While in Equation 5.4, w'_i is proportional to the amount of resource that is granted to service i, in Equation 5.5, w''_i controls the amount of resource that is taken from the maximum resource requirements. Besides the semantic difference on the w_i parameter, there is another, more factual, difference between the direct and the indirect approaches. While in the former one, the U_{lax_i} parameter has no influence on the resource distribution, besides constraining the maximum amount of the resource one can use, in the latter approach, U_{lax_i} is a direct part of the equation, since it is the baseline from which the bandwidth is taken, upon overload.

However, despite of the differences between both models it is still possible to, from the application perspective, map the models one into the other. By putting both equations (5.4 and 5.5) together and assuming equal distribution results ($V'_i = V''_i$) leads to Equation (5.7) that maps w''_i into w'_i.

$$w'_i = \frac{U_{lax_i}}{U_{spare}} - w''_i . \frac{\overline{U_{spare}}}{U_{spare}} \tag{5.7}$$

Notwithstanding, from a design perspective, there are differences between the models. The same variation on the weights results on different distribution, as can be easily shown by taking the respective derivatives (Equation 5.8):

$$\left| \frac{\delta V'_i}{\delta w'_i} \right| \neq \left| \frac{\delta V''_i}{\delta w''_i} \right| \tag{5.8}$$

Therefore, despite the similarities, the two techniques are different and are worth a separate analysis. Moreover, there are QoS distribution schemes in the literature, as mentioned further, that are based on both of these schemes.

Both the direct and indirect approaches are formulated in terms of linear equations, which, at a first glance, makes these QoS distribution techniques exhibit a low computational complexity ($O(n)$). However, the bandwidth specification is upper and lower bounded, and the distribution results of equations 5.4 and 5.5 may fall off those limits. Respectively, they may lead to situations where services receive an amount of bandwidth that exceeds or falls bellow the minimum specified requirements.

Optimality

As mentioned before, each service defines a range of allowed bandwidth values. Assuming that the distribution procedure is optimal if the available bandwidth is fully assigned, the direct application of equations 5.4 and 5.5

does not provide an optimal solution, *per se*. Take for instance the following example, where three services compete for a resource with $U_{spare} = 5$.

i	U_{lax_i}	w_i	V_i'	V_i''
1	2	0.5	2.5	$2 - 2.5 = -0.5$
2	5	0.25	1.25	$5 - 1.25 = 3.75$
3	3	0.25	1.25	$3 - 1.25 = 1.75$

Each service is provided a weight w_i equally provided to both mechanisms, the *Direct Share* and the *Indirect Share*. As referred before, the weights have distinct meanings for each scheme, however, for the purpose of this example this aspect is not relevant. Columns V_i' and V_i'' list the distribution results from equations 5.4 and 5.5 respectively. Focusing, e.g. on service $i = 1$, it is noticeable that for the first approach V_i' is greater than the admissible bandwidth (U_{lax_i}), i.e., this service cannot handle such an amount of bandwidth. Conversely, the result for the second approach (V_i'') shows a negative bandwidth, which indicates that instead of providing additional bandwidth, the service is not receiving the minimum specified bandwidth. Therefore, in both mechanisms there is a violation of the requirements set by the application. It is necessary to create an additional constraint that verifies if the output of equations 5.4 and 5.5 does not violate the specifications (Equation 5.9).

$$0 \leq V_i \leq U_{lax_i} \tag{5.9}$$

When v_i needs to be clipped there is either bandwidth being wasted, leading to non-optimality, or a requirements fault. In either case the bandwidth distribution must be recalculated to confine V_i within the limits. This requires an iteration process to re-adjust the distribution whenever a maximum or a minimum is hit, until the spare bandwidth is assigned. Consequently, the process is no longer linear with respect to the number of services.

5.3.3 Need for iteration

Let i be a service for which V_i at a given moment exceeds its limits. To clip V_i in one of the boundaries, w_i^{clip} is formulated as follows:

$$w_i^{clip} = \begin{cases} 0 & \text{if} \quad V_i < 0 \\ \frac{U_{lax_i}}{U_{spare}} & \text{if} \quad V_i > U_{lax_i} \end{cases} \tag{5.10}$$

This clipping compensation reflects on the weights the distortion introduced by the clipping. Basically, when a service bandwidth reaches its limit, its bandwidth is kept fixed and the available bandwidth recomputed. Necessarily, the weights of the remaining services must be recomputed too, to

assure they continue being normalized for the distribution of the remaining bandwidth. This requires an iterative procedure, repeated until the bandwidth is fully distributed.

The iterative procedure herein presented is similar to the one proposed for the Elastic Task Model (ETM) [38]. However, this model is generalized, comprising the ETM as well as other specific techniques. Furthermore, some improvements on the iterative procedure are presented further on, aiming at reducing its computational overhead.

The iterative procedure is described as follows. Consider a service set Γ that, at every instant, is divided in two subsets: Γ_{clip}, comprising the jobs that have been clipped, and Γ_{unclip} holding the ones still manageable (unclipped). The iterative procedure starts with all services unclipped, as sketched in Algorithm 5.3.1. For now, assume that those starting services are randomly sorted and computed sequentially. Then, at each step k the distribution is computed for the unclipped services. The bandwidth resulting for each service in Γ_{unclip} is analyzed and whenever a service gets clipped, i.e., V_i differs from (alg. 5.3.1-(2)), a new iteration step is issued and that service moved from Γ_{unclip} to Γ_{clip}. The procedure completes after a round with no services being clipped.

Algorithm 5.3.1: (Procedure for a full bandwidth distribution)

---

comment: Γ is an unsorted list of St

comment: $BW function$ computes either (5.11) or (5.14)

$\Gamma_{unclip} \leftarrow \Gamma$
$\Gamma_{clip} \leftarrow \{\}$
repeat
 for each $St_i \in \Gamma_{unclip}$

$$\textbf{do} \begin{cases} V_i = BW function() & (1) \\ \textbf{if not } 0 \leq V_i \leq U_{lax_i} & (2) \\ \quad \textbf{then} \begin{cases} \Gamma_{unclip} \leftarrow \Gamma_{unclip} \setminus \{St_i\} \\ \Gamma_{clip} \leftarrow \Gamma_{clip} \cup \{St_i\} \\ \textbf{break} \end{cases} \end{cases}$$

until no **break** occurs

A simple rationale can be followed to understand the iteration process. The occurrence of clipping means that part of the initial bandwidth block U_{spare} or part of $\overline{U_{spare}}$ was not correctly distributed on one or more services and had to be fixed overriding the initial linear distribution. Therefore, for each new iteration, the remaining bandwidth from U_{spare} or $\overline{U_{spare}}$ has to be re-computed to account for the services that have moved to the Γ_{clip} set,

and so the individual weights of each service, yet in Γ_{unclip}, to satisfy the normalization constraint, as follows:

$$\sum_{St_i \in \Gamma_{unclip}} w_i^{k+1} = 1$$

For the *Direct Share*, $BWfunction()$ model is given by:

$$\begin{aligned} &\forall St_i \in \Gamma_{unclip} : \\ &V_i'^{k+1} = \begin{cases} w_i'^{k+1} \times {U_{spare}}^{k+1} & \text{, if } \left(w_i'^{k+1} \times {U_{spare}}^{k+1}\right) < U_{lax_i} \\ U_{lax_i} & \text{, else.} \end{cases} \end{aligned} \tag{5.11}$$

where,

$$\begin{aligned} {U_{spare}}^{k+1} &= U_{spare} - \sum_{St_i \in \Gamma_{clip}} V_i'^k \\ &= U_{spare} - \sum_{St_i \in \Gamma_{clip}} U_{lax_i} \end{aligned} \tag{5.12}$$

$$w_i'^{k+1} = \frac{w_i'^k}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j'^k} = \frac{w_i'^0}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j'^0} \qquad , \forall St_i \in \Gamma_{unclip} \tag{5.13}$$

And for the *Indirect Share*, $BWfunction()$ model given by:

$$\begin{aligned} &\forall St_i \in \Gamma_{unclip} : \\ &V_i''^{k+1} = \begin{cases} \left(U_{lax_i} - w_i''^{k+1}.\overline{U_{spare}}^{k+1}\right) & \text{if } \left(U_{lax_i} - w_i''^{k+1}.\overline{U_{spare}}^{k+1}\right) > 0, \\ 0 & \text{else.} \end{cases} \end{aligned} \tag{5.14}$$

where,

$$\begin{aligned} {U_{spare}}^{k+1} &= U_{spare} - \sum_{St_i \in \Gamma_{clip}} V_i''^k \\ &= U_{spare} - \sum_{St_i \in \Gamma_{clip}} 0 \end{aligned} \tag{5.15}$$

$$\begin{aligned} \overline{U_{spare}}^{k+1} &= \sum_{St_i \in \Gamma_{unclip}} U_{lax_i} - {U_{spare}}^{k+1} \\ &= \sum_{St_i \in \Gamma_{unclip}} U_{lax_i} - U_{spare} \end{aligned} \tag{5.16}$$

$$w_i''^{k+1} = \frac{w_i''^{k}}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j''^{k}} = \frac{w_i''^{0}}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j''^{0}} \qquad ,\forall St_i \in \Gamma_{unclip} \tag{5.17}$$

Equations 5.11 and 5.14 describe the bandwidth distribution for each service within an iteration step. Only unclipped services are computed. The clipped services, remain constant through the rest of the procedure.

The unclipped weights are permanently modified in each iteration, however, as denoted by Equation 5.13 and 5.17, the weights can be assessed given the initial application weights and still reflect the application specification. Thus, this process respects the relative importance of the services throughout the iterative procedure.

Each ${w_i}^{k+1}$ uniquely depends on the distribution status between Γ_{clip} and Γ_{unclip}. Given $k = m$ to be the last iteration, the weights in m are given by:

$$w_i^m = \frac{w_i}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j} \qquad ,\forall St_i \in \Gamma_{unclip}$$

These weights refer to the distribution of the bandwidth portion left for the last iteration. Translating those final values into the scope of the initial U_{spare} bandwidth, gives:

For the *Direct Share* model:

$$w_{final_i} = \begin{cases} \frac{U_{lax_i}}{U_{spare}} & ,\forall St_i \in \Gamma_{clip} \\ \frac{w_i}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j} \left(1 - \sum\limits_{St_j \in \Gamma_{clip}} w_j\right) & ,\forall St_i \in \Gamma_{unclip} \end{cases} \tag{5.18}$$

And for the *Indirect Share* model:

$$w_{final_i} = \begin{cases} 0 & ,\forall St_i \in \Gamma_{clip} \\ \frac{w_i}{\sum\limits_{St_j \in \Gamma_{unclip}} w_j} & ,\forall St_i \in \Gamma_{unclip} \end{cases} \tag{5.19}$$

Given this final distribution, the respective weights are still normalized:

$$\sum_{St_i \in \Gamma_{unclip}} w_{final_i} + \sum_{St_i \in \Gamma_{clip}} w_{final_i} = 1$$

Observing equations (5.18) and (5.19), the final weights regarding U_{spare}, do not depend on the procedure steps, but only on the result of the distribution between Γ_{clip} and Γ_{unclip}. This is important to understand that the

order by which the services are accounted for during the bandwidth distribution is irrelevant to the final result. However, it is still required to prove that the separation result between Γ_{clip} and Γ_{unclip} is unique given the initial parameters.

The following subsection shows such proof. Then, the bandwidth distribution procedure is evaluated and ways to improve its performance are discussed.

Existence of a unique solution

Theorem 5.1. *There exists a unique solution to the bandwidth distribution problem.*

Proof. The solution generated by the bandwidth distribution procedure is of the format $\{W_{clip}, W_{unclip}\}$, where W_{clip} is the set of weights corresponding to services ($St_i \in \Gamma_{clip}$) that have a final bandwidth share of ($U_{spare}.w_i^{clip}$), and W_{unclip} is the set of weights corresponding to services ($St_j \in \Gamma_{unclip}$) that have a final bandwidth share of ($U_{spare}.w_j^{unclip}$).

The sets are given by

$$W_{clip} = \{w_i^{clip} : St_i \in \Gamma_{clip}\} \tag{5.20}$$

The remaining weights ($1 - \sum_{St_i \in \Gamma_{clip}} w_i^{clip}$) are shared proportionally by the applications in Γ_{unclip}. The weights allocated to Γ_{unclip} are therefore given by:

$$W_{unclip} = \left\{ w_j . \left(\frac{1 - \sum\limits_{St_i \in \Gamma_{clip}} w_i^{clip}}{\sum\limits_{St_p \in \Gamma_{unclip}} w_p} \right) : St_j \in \Gamma_{unclip} \right\} \tag{5.21}$$

As can be seen from the above expression, the weights in Γ_{unclip} are equally scaled by a factor of k, where k is given by,

$$k = \frac{1 - \sum\limits_{St_i \in \Gamma_{clip}} w_i^{clip}}{\sum\limits_{St_p \in \Gamma_{unclip}} w_p} \tag{5.22}$$

This value k is a fingerprint of the solution to the bandwidth distribution problem. There is a one-to-one mapping between k and the sets $\{W_{clip}, W_{unclip}\}$. So far, it has been shown that given $\{W_{clip}, W_{unclip}\}$ (Equations 5.20 and 5.21), a fingerprint k can be found (Equation 5.22). Now, it is shown the other way around, i.e., there cannot be two distribution results with the same k.

From the definitions of the sets $\{W_{clip}\}$ and $\{W_{unclip}\}$, observe that

$$\forall St_i \in \Gamma_{clip} : w_i.k \geq w_i^{clip} \tag{5.23}$$

$$\forall St_j \in \Gamma_{unclip} : w_j.k < w_j^{clip} \tag{5.24}$$

Also,

$$\sum_{St_i \in \Gamma_{clip}} w_i^{clip} + \sum_{St_j \in \Gamma_{unclip}} w_j.k = 1 \tag{5.25}$$

We can observe a one-to-one mapping between the fingerprint k and any solution of the QoS maximization problem $\{W_{clip}, W_{unclip}\}$.

In order to prove the uniqueness of the solution, it is sufficient to prove the uniqueness of the fingerprint.

Case 1: Assume that there exists a solution $(k + \delta)$, where $\delta > 0$. Let the corresponding solution to the bandwidth distribution problem be $\{W'_{clip}, W'_{unclip}\}$.

In this case, the following relationships should also hold:

$$\forall St_i \in \Gamma'_{clip} : w_i.(k + \delta) \geq w_i^{clip} \tag{5.26}$$

$$\forall St_j \in \Gamma'_{unclip} : w_j.(k + \delta) < w_j^{clip} \tag{5.27}$$

$$\sum_{St_i \in \Gamma'_{clip}} w_i^{clip} + \sum_{St_j \in \Gamma'_{unclip}} w_j.(k + \delta) = 1 \tag{5.28}$$

Observe that all services that satisfy Inequality 5.23 also satisfy the Inequality 5.26, hence

$$\Gamma_{clip} \subset \Gamma'_{clip} \tag{5.29}$$

$$\Gamma_{clip} = \Gamma'_{clip} \setminus S_\delta \tag{5.30}$$

since Γ_{clip} is the complement of Γ_{unclip},

$$\Gamma_{unclip} = \Gamma'_{unclip} \cup S_\delta \tag{5.31}$$

$$\Gamma'_{unclip} \subset \Gamma_{unclip} \tag{5.32}$$

$$S_\delta \subset \Gamma_{unclip} \tag{5.33}$$

Rewriting (5.25), gives

$$\sum_{St_i \in (\Gamma'_{clip} - S_\delta)} w_i^{clip} + \sum_{St_j \in (\Gamma'_{unclip} + S_\delta)} w_j.k = 1 \tag{5.34}$$

$$\sum_{St_i \in \Gamma'_{clip}} w_i^{clip} - \sum_{St_p \in S_\delta} w_p^{clip} + \sum_{St_j \in \Gamma'_{unclip}} w_j.k + \sum_{St_q \in S_\delta} w_q.k = 1 \tag{5.35}$$

Subtracting (5.28), gives

$$- \sum_{St_p \in S_\delta} w_p^{clip} - \sum_{St_j \in \Gamma'_{unclip}} w_j.(\delta) + \sum_{St_q \in S_\delta} w_q.k = 0 \tag{5.36}$$

$$\sum_{St_j \in \Gamma'_{unclip}} w_j.\delta = \sum_{St_q \in S_\delta} w_q.k - \sum_{St_p \in S_\delta} w_p^{clip} \tag{5.37}$$

Using the fact that $S_\delta \subset \Gamma_{unclip}$ and Inequality 5.24,

$$\sum_{St_j \in \Gamma'_{unclip}} w_j.\delta < 0 \tag{5.38}$$

Which is not possible, since the weights w_j are positive.

Case 2: Assume that there exists a solution $(k-\delta)$, where $\delta > 0$. Let the corresponding solution to the QoS maximization problem be $\{W''_{clip}, W''_{unclip}\}$.
In this case, the following relationships should hold good:

$$\forall St_i \in \Gamma''_{clip} : w_i.(k-\delta) \geq w_i^{clip} \tag{5.39}$$

$$\forall St_j \in \Gamma''_{unclip} : w_j.(k-\delta) < w_j^{clip} \tag{5.40}$$

$$\sum_{St_i \in \Gamma''_{clip}} w_i^{max} + \sum_{St_j \in \Gamma''_{unclip}} w_j.(k-\delta) = 1 \tag{5.41}$$

Observe that all the applications that satisfy Inequality 5.39 also satisfy the Inequality 5.23, hence

$$\Gamma''_{clip} \subset \Gamma_{clip} \tag{5.42}$$

$$\Gamma''_{clip} = \Gamma_{clip} - R_\delta \tag{5.43}$$

$$R_\delta \subset \Gamma_{clip} \tag{5.44}$$

since Γ_{clip} is the complement of Γ_{unclip},

$$\Gamma''_{unclip} = \Gamma_{unclip} + R_\delta \tag{5.45}$$

$$\Gamma_{unclip} \subset \Gamma''_{unclip} \tag{5.46}$$

Rewriting (5.25), gives

$$\sum_{St_i \in (\Gamma''_{clip} + R_\delta)} w_i^{clip} + \sum_{St_j \in (\Gamma''_{unclip} - R_\delta)} w_j.k = 1 \tag{5.47}$$

$$\sum_{St_i \in \Gamma''_{clip}} w_i^{clip} + \sum_{St_p \in R_\delta} w_p^{clip} + \sum_{St_j \in \Gamma''_{unclip}} w_j.k - \sum_{St_q \in R_\delta} w_q.k = 1 \tag{5.48}$$

Subtracting (5.41), gives

$$\sum_{St_p \in R_\delta} w_p^{clip} + \sum_{St_j \in \Gamma''_{unclip}} w_j.\delta - \sum_{St_q \in R_\delta} w_q.k = 0 \tag{5.49}$$

$$\sum_{St_j \in \Gamma''_{unclip}} w_j.\delta = \sum_{St_p \in R_\delta} w_p^{clip} - \sum_{St_q \in R_\delta} w_q.k \tag{5.50}$$

Using the fact that $R_\delta \subset \Gamma_{clip}$ and Inequality 5.23,

$$\sum_{St_j \in \Gamma''_{unclip}} w_j.\delta < 0 \tag{5.51}$$

Which is not possible, since the weights w_j are positive.

Hence, once proved that there exists a unique fingerprint to the bandwidth distribution problem, the solution to the problem is also unique since there is a one-to-one mapping between the fingerprints and solutions to the QoS maximization problem. □

Computational complexity of the iterative procedure

The QoS distribution mechanisms addressed in this work aim at online use, thus their computational complexity is a key factor.

The computational complexity is related to the number of times V_i is computed (alg. 5.3.1-(1)). In a scenario where the result appears with no need for iterations, the number of times V_i is computed equals the number of services, which is in fact the best-case scenario.

Every time a new iteration is triggered the remaining services in Γ_{unclip} are re-computed, requiring new V_i results. Therefore, given a generic set St and the number of iterations, the worst scenario occurs when V_i is computed

in each iteration as many times as the number of services in the unclipped set.

From the algorithm (5.3.1), the number of iterations is given by the number of times condition (alg. 5.3.1-2) is not verified, i.e., the number of jobs that are clipped.

Let $\Gamma_1, \Gamma_2 \subseteq St$ be two arbitrary subsets of St. As previously shown, the number of iterations (or breaks) that result when running the iterative algorithm will be the same for both lists, even though the number of times V_i is computed varies. For each new iteration, V_i is re-calculated for the services not yet clipped. Therefore, depending on the list arrangement, the computational time might vary. If the list is sorted in such a way that the services that might be clipped head the list, the number of re-calculations is minimized. Conversely, if the list is reversely sorted the number of re-calculations will be the highest, leading to the worst-case situation.

For a given list Γ, let T_Γ be the number of times V_i is computed and n the number of services in the list. If Γ is sorted as in the best-case scenario, the services that do not clip are computed in the last iteration. Therefore there are no services for which V_i is computed more than once. The number of times V_i is computed is then given by:

$$St_{Best\text{-}case}: \qquad T_{\Gamma(sorted)} = n \tag{5.52}$$

To evaluate the impact of Γ being reverse-sorted, leading to a worst-case scenario, we have to estimate the total number of iterations that occur. We have seen that, no matter the sorting order, the number of iterations is unique for a given set St. Let It_{St} represent that number. In the worst-case scenario, we have a situation where in the first iteration all services pass the condition but the last one breaks, causing V_i to be computed n times. Then, in the second iteration, with one less element, it might be that the, now last element (previously last but one), breaks the condition. The number of times V_i is computed is now $n-1$. Assuming this worst scenario for all the following iterations, we have in fact a decreasing arithmetic progression that starts with the value n and ends with $n - It_{St}$. The overall number of V_i computations is then given by:

$$T_{\Gamma(rev-sorted)} = (2.n - It_{St}).\frac{It_{St}+1}{2}$$

Yet, in a real scenario, for a generic St there is no way of predicting It_{St} without computing through all services. Thus, a priori, we have to assume the maximum number of iterations, $It_{St} = n-1$:

$$St_{Worst\text{-}case}: T_{\Gamma(rev-sorted)} = \frac{n+1}{2}.n \tag{5.53}$$

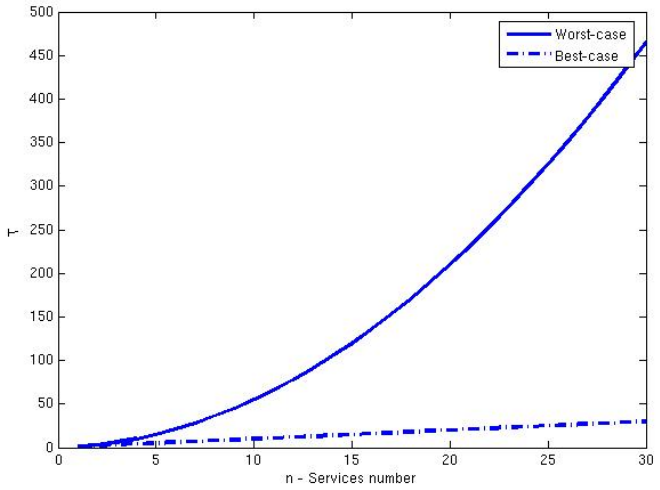


Figure 5.2: Computational complexity, excluding services sort.

Figure 5.2 plots the complexity growth with the number of services, both for the best-case (Equation 5.52) and the worst-case (Equation 5.53). It is clear the advantage of having the list sorted before applying the algorithm. While the best-case is computed with an $O(n)$ complexity, the worst-case requires an $O(n^2)$.

The difference between the two cases is noticeable. However, there are two aspects that so far have not been considered, the sorting computational overhead and the conditional parameters that define the sorting. For now assume that there is a static parameter to guide the sorting. In other words, a parameter constrained to the scope of a single service specification and not influenced by the number nor the parameters of other services.

Regarding the sorting computational time, if appropriate measures are taken in the implementation, such as keeping the list always sorted from the beginning, the sorting procedure is reduced to a binary search with complexity $O(log\ n)$. Hence, the full procedure can be classified as having an $O(n\ log\ n)$ complexity, which is still better than the unsorted approach, with complexity $O(n^2)$.

Sorting the services - Overview

The sorting algorithm is still relying on the existence of a conditional expression that enables an efficient and fast sorting procedure, otherwise, it cannot be neglected when addressing the iterative procedure complexity.

To find that sorting parameter, we must understand what triggers the conditions failure, and the subsequent new iteration.

Direct Share: From (5.4) and (5.9), the necessary condition for having an iteration is given by:

$$\begin{aligned} V_i > U_{lax_i} \quad &\Leftrightarrow \\ U_{spare} > \frac{U_{lax_i}}{w_i} \quad & \end{aligned} \tag{5.54}$$

Indirect Share: From (5.5) and (5.9), the necessary condition for having an iteration is given by:

$$\begin{aligned} V_i < 0 \quad &\Leftrightarrow \\ \overline{U_{spare}} > \frac{U_{lax_i}}{w_i}, w_i > 0 \quad &\Leftrightarrow \\ \sum_{j=1}^{n} U_{lax_j} - U_{spare} > \frac{U_{lax_i}}{w_i} \quad & \end{aligned} \tag{5.55}$$

The QoS weights are provided by the application and are normalized. Each weight is obtained comparing certain parameters across all the services. For the sake of simplicity, assume for now, that w_i, the QoS weight of St_i, is uniquely dependent from St_i parameters, i.e, its calculation does not depend on St_j parameters, for instance.

Based on that assumption, in both inequalities, (5.54) and (5.55), the left member reflects the global impact of all the services set, whereas the right member depends only on St_i service parameters. Generalizing the inequalities members, follows that:

$$\forall St_i \in \Gamma \qquad Const\left(\Gamma\right) > Var\left(St_i\right) \tag{5.56}$$

where $Const()$ function is determined by all the current services, and thus fixed regarding St_i, the service being compared, while the $Var()$ function depends on St_i only.

To define the sorting condition, consider two services $St_j, St_k \in \Gamma$, so that $Var(St_j) > Var(St_k)$. Comparing Inequality 5.56) when applied to both, j and k, we may say that if St_j meets Inequality 5.56, so does St_k, since $Const(\Gamma)$ is the same for both. But the reverse is not valid. Therefore, St_k is more likely to cause an iteration than St_j. Generalizing to the Γ list, if the services are sorted in ascending order regarding the $Var(St_i)$ value, the services that most probably cause an iteration will be heading the list, hence being analyzed first in the iterative algorithm 5.3.1.

Therefore, by assuming w_i independent from other services, it is possible to have a parameter, $Var(St_i)$, that determines if a service is more likely to break the condition, i.e., the sorting condition. However, the previous assumption may not be valid for the QoS mapping models, a subject that is investigated in the next section.

5.3.4 Application mapping models

Section 5.3.3 addressed two distribution models that use parameter w_i (w'_i and w''_i) to distribute the spare load among the services. This section addresses how that parameter is obtained from the application model parameters, i.e., how the application indications are interpreted when to distribute the bandwidth.

The weights are assessed with a function δ of application parameters, as follows:

$$w_i = \delta([qos_j, St_j \in St], [U_{lax_j}, St_j \in St])$$

Function δ has the only constraint of providing weights such that $\sum w_i = 1$. Given this constraint, three mapping models are proposed to be applied along with the distribution models, either the *Direct Share* or the *Indirect Share*.

The combination of the application mapping models and the bandwidth distribution techniques, results on some QoS management techniques similar to other techniques already formulated in the literature. In this case, they are placed side by side and the axiomatic analogy is discussed. For each model it is also addressed the iterative optimization issue, discussed in Section 5.3.3.

Mapping model A

Weighted fair queuing (WFQ) [122] is a data packet scheduler that enforces reduced latency in a per packet basis. It prioritizes services in order to give bandwidth guarantees based on an application given parameter. The approach hereby described, although in a different context, is based on the same principle. When a resource is shared by several services and we want to distinguish between the performance quality of these, the easiest approach is to make a direct mapping between the application requirements and the resource shares.

In this model the QoS level weights, $\{w_i : St_i \in \Gamma\}$, are obtained normalizing the service application parameter, qos_i as follows:

$$\forall St_i \in \Gamma : w_i = \frac{qos_i}{\sum\limits_{St_j \in \Gamma} qos_j}$$

The model is very simple and easy to understand from the application perspective: the higher is the qos_i, the higher w_i becomes. However, as we have seen that w'_i and w''_i, the weights within the *Direct Share* and the *Indirect Share* models respectively, have opposite meanings regarding the bandwidth distribution. Therefore, to equalize those meanings, we obtain:

For *Direct Share*:

$$\forall St_i \in \Gamma : w'_i = \frac{qos_i}{\sum\limits_{St_j \in \Gamma} qos_j} \tag{5.57}$$

For *Indirect Share*:

$$\forall St_i \in \Gamma : w_i'' = \frac{(qos_i^{-1})}{\sum\limits_{St_j \in \Gamma} (qos_j^{-1})} \tag{5.58}$$

Hence, we have for both models the parameter qos_i directly controlling the bandwidth obtained.

Regarding the sorting optimization issue, Section 5.3.3 discussed the existence of a sorting condition given by the Inequality 5.56. Such inequality is characterized by having a member that is constant to all services being compared and another one that is unique to each service. However, it was assumed that w_i depends solely on St_i service parameters, which is not valid for this mapping model, as illustrated in Equations 5.57 and 5.58, since w_i depends on parameters from all the services. Yet, it is possible to obtain an inequality similar to Equation 5.56 with a member depending on Γ and a separate one depending on St_i parameters. The proof is applied for both distribution techniques using this model.

$$Const\,(\Gamma) > \frac{U_{lax_i}}{w_i} \quad ,\forall i \in \Gamma$$

Direct Share:

$$\frac{Const\,(\Gamma)}{\sum\limits_{St_j \in \Gamma} qos_j} > \frac{U_{lax_i}}{qos_i}$$

$$Const\,(\Gamma)' > Var\,(i)' \tag{5.59}$$

Indirect Share:

$$\frac{Const\,(\Gamma)}{\sum\limits_{St_j \in \Gamma} (qos_j^{-1})} > \frac{U_{lax_i}}{(qos_i^{-1})}$$

$$Const\,(\Gamma)'' > Var\,(i)'' \tag{5.60}$$

Both inequalities are similar to (5.56) where the leftmost members are constant and the rightmost depend only on application given parameters, as sought. This result validates the assumption that led the original condition (5.56) and identifies the sorting parameter, $Var(i)' = \frac{U_{lax_i}}{qos_i}$ for the *Direct Share* and $Var(i)'' = U_{lax_i}.qos_i$ for the *Indirect Share*.

The integration of this mapping model with the *Direct Share* or the *Indirect Share* distribution mechanisms and the sorting optimization scheme described in Section 5.3.3, results in a QoS distribution procedure with complexity $O(n\ log\ n)$.

The QoS distribution scheme that results from the *Direct Share* method is perhaps the simplest and most intuitive to understand from the application

point of view, thus being used in resource management frameworks such as FRESCOR [52].

The integration with the *Indirect Share* method is not entirely novel, either. Buttazzo *et al* proposed a mechanism that addresses the problem of overload in task sets that tolerate service degradation, the elastic task model (ETM) [38]. The model is analogous to the physical model of springs being compressed. A generic task (spring) i with a nominal size U_{i_0} is compressed with a QoS level (elastic coefficient) e_i. The task i is compressed along with all other tasks until a final distribution balance is found, with all tasks fitting U_d, the desired total task utilization. Similarly to the iterative procedure in Section 5.3.3, a task is given $U_{i_{min}}$ when the maximum compression is reached. Consider a task set Γ divided in two subsets: a set Γ_f with the fixed tasks, the ones that reached the minimum requirements, and a set Γ_v with the tasks that are still adaptable. U_i denotes the bandwidth given to task i in each iteration, as follows:

$$\forall St_i \in \Gamma_v \qquad U_i = U_{i_0} - (U_0 - U_d + U_f)\frac{e_i}{E_v} \tag{5.61}$$

where U_0 is the nominal utilization load and

$$U_f = \sum_{St_i \in \Gamma_f} U_{i_{min}}$$

$$E_v = \sum_{St_i \in \Gamma_v} e_i$$

The ETM is thus analogous to the one that results integrating this mapping model A along with the *Indirect Share* distribution technique, without loss of generalization. We can map one model into the other, comparing Equations (5.61) and (5.5) results:

$$\begin{aligned} U_i &= V_i'' + U_{i_{min}} \\ \frac{e_i}{E_v} &= w_i'' \\ U_{i_0} &= U_{lax_i} + U_{i_{min}} \\ (U_0 - U_d + U_f) &= \overline{U_{spare}} \end{aligned}$$

To complete this comparison, keep in mind that our model distributes the spare bandwidth after having reserved the minimum requirements, while the ETM operates with the full bandwidth. This small difference introduces an offset in the iterative procedure, which does not affect the result nor the algorithm efficiency.

There is however an efficiency difference regarding the iterative procedure complexity that is reduced from $O(n^2)$ in the ETM to $O(n\ log\ n)$ in our approach.

Mapping model B

For diverse application domains, e.g. multimedia, there is a direct relationship between the dynamic QoS range that a given service can accept and its importance to the global system performance. Consider, for instance, a video surveillance application, comprising diverse services with identical quality expectations, except for one stream with higher quality level requirements. This particular service is able to operate with higher bandwidth requirements, higher QoS range (laxity) and should be getting more bandwidth than the others. It is thus reasonable to establish the relation between the QoS range with which services are specified and the differentiation level of each one.

This is a simple approach to the service differentiation problem that in this case further simplifies the bandwidth distribution procedure. The model uses simple calculus to obtain the weighted parameters based on the bandwidth laxity of each service. While the previous model uses a parameter directly issued to handle QoS management (qos_i), this mechanism instead, uses the slack value, U_{lax_i}, to obtain the distribution weights, as follows:

$$\forall St_i \in \Gamma : w_i = \frac{U_{lax_i}}{\sum\limits_{St_j \in \Gamma} U_{lax_j}} \tag{5.62}$$

This model can be seen as a particular case of the previous model, provided that $qos_i = U_{lax_i}$. However, there are a few characteristics that make this mechanism special.

The first particularity of this mechanism is that there is no difference between integrating this model with the *Direct Share* or the *Indirect Share* mechanism. They become mathematically equal given Equation 5.62. Merging equations (5.4) and (5.5) with the same w_i, follows:

$$\begin{aligned} V'_i &= V''_i \qquad\qquad , w_i = w'_i = w''_i \\ w_i.U_{spare} &= U_{lax_i} - w_i.\left(\sum_{j=1}^{n} U_{lax_j} - U_{spare}\right) \\ w_i &= \frac{U_{lax_i}}{\sum\limits_{St_j \in \Gamma} U_{lax_j}} \end{aligned}$$

Comparing *Direct Share* and *Indirect Share* models for when the result is the same, the only possible weights are the ones given by Equation 5.62, which makes both schemes equal.

Another characteristic that emerges from this model is that the bandwidth distribution procedure executes with no iterations, i.e., the inequalities in (5.9) are always met. This is because each w_i denotes the ratio of

the service bandwidth laxity among all the services that must be shrunk to fit U_{spare}. The proof for this statement is easily followed showing that conditions in (5.9) are always met.

For the *Direct Share*, or *Indirect Share*, we have to show that given the overload condition, the following conditions are always true:

$$\forall St_i \in \Gamma : 0 \leq V_i \leq U_{lax_i} \tag{5.63}$$

where $(U_{min_i} + V_i)$ is the bandwidth allocated to the service St_i, and $(U_{min_i} + U_{lax_i})$ is the maximum possible bandwidth allocated to it.

For the first condition, from (5.5) and (5.62) we have,

$$\begin{aligned} U_{lax_i} - w_i . \left(\sum_{St_j \in \Gamma} U_{lax_j} - U_{spare} \right) &>= 0 \\ U_{lax_i} - \left(\frac{U_{lax_i}}{\sum\limits_{St_j \in \Gamma} U_{lax_j}} \right) . \left(\sum_{St_j \in \Gamma} U_{lax_j} - U_{spare} \right) &>= 0 \\ \frac{U_{lax_i} . U_{spare}}{\sum\limits_{St_j \in \Gamma} U_{lax_j}} &>= 0 \end{aligned}$$

which is always met since both, U_{lax_i} and U_{spare} are always positives. This verifies the first condition.

Similarly, for the second condition, from (5.4) and (5.62) we have,

$$\begin{aligned} w_i . U_{spare} &\leq U_{lax_i} \\ \left(\frac{U_{lax_i}}{\sum\limits_{St_j \in \Gamma} U_{lax_j}} \right) . U_{spare} &\leq U_{lax_i} \\ U_{spare} &\leq \sum_{St_j \in \Gamma} U_{lax_j} \qquad , \forall St_i \in \Gamma : U_{lax_i} > 0 \end{aligned}$$

which is the necessary overload condition. This makes the second condition in (Equation 5.63) to be always valid, and proves that using the weights proposed for this model, the distribution procedure goes with no iterations.

Thus, the use of this mapping model within the bandwidth distribution scheme introduces a substantial reduction on the computational complexity. Avoiding the iterations, it eliminates the need to sort the services sequentially, and so the complexity is reduced to $O(n)$.

Mapping model C

The QoS distribution approach followed for the previous model assumes a tight relationship between the importance given to one service and the requirements range in which that service operates.

However, some applications are strictly confined to operate under specific requirements, which constrains the manageability of the requirements specification towards a given differentiation level. As an example of this constraint, consider a video surveillance system similar to the one presented for the previous model, but where a given service is constrained to operate on a specific range of frame rates, hence not able to increase the bandwidth laxity and so increase the resource share with respect to other services.

To address this class of applications, a model is proposed that uses a virtual operating bandwidth laxity for the QoS management, while the service operates within the real laxity. Compared to the previous approach, this model provides extra flexibility to, at design time, adjust the importance of services with operational constraints. The virtual range is obtained applying a scaling ratio to the effective operational range. The ratio, specified by qos_i, increases or decreases the base operational range (U_{lax_i}), i.e., $qos_i > 1$ or $qos_i < 1$ respectively. The weighted parameters for this model are given by:

$$\forall St_i \in \Gamma : w_i = \frac{(U_{lax_i}.qos_i)}{\sum\limits_{St_j \in \Gamma} \left(U_{lax_j}.qos_j\right)}$$

From a designing perspective, the presented model is easily mapped into the first model (A), giving $qos'_i = (U_{lax_i}.qos_i)$. The only difference for this model is the way the designer distributes the qos_i importance. Addressing separately the *Direct Share* and the *Indirect Share* models, follows:

For *Direct Share*:

$$\forall St_i \in \Gamma : w'_i = \frac{(U_{lax_i}.qos_i)}{\sum\limits_{St_j \in \Gamma} \left(U_{lax_j}.qos_j\right)} \tag{5.64}$$

For *Indirect Share*:

$$\forall St_i \in \Gamma : w''_i = \frac{(U_{lax_i}.qos_i)^{-1}}{\sum\limits_{St_j \in \Gamma} \left(U_{lax_j}.qos_j\right)^{-1}} \tag{5.65}$$

Similarly to model A, the optimal bandwidth distribution is not guaranteed in a single iteration step. It is necessary to follow the iteration procedure as described in Section 5.3.3. For the mapping model A, the performance impact is minimized by having the service list sorted. The same is true with this model.

The performance impact minimization results from the validation for this mapping model of the assumption that led to (5.56). So, taking (5.56) and (5.64) for the *Direct Share*, or (5.65) for the *Indirect Share*:

$$Const\,(\Gamma) > \frac{U_{lax_i}}{w_i} \quad , \forall St_i \in \Gamma$$

Direct Share:

$$\frac{Const\,(\Gamma)}{\sum\limits_{St_j \in \Gamma} \left(qos_j.U_{lax_j}\right)} > \frac{1}{qos_i}$$

$$Const\,(\Gamma)' > Var\,(St_i)' \tag{5.66}$$

Indirect Share:

$$\frac{Const\,(\Gamma)}{\sum\limits_{St_j \in \Gamma} \left(qos_j.U_{lax_j}\right)^{-1}} > U^2_{lax_i}.qos_i$$

$$Const\,(\Gamma)'' > Var\,(i)'' \tag{5.67}$$

Here also, $Var(St_i)'$ and $Var(St_i)''$ are isolated in the rightmost member and depend uniquely on St_i parameters. The iterative impact is minimized if the service list is sequentially issued and sorted by $\frac{1}{qos_i}$ and $U^2_{lax_i}.qos_i$, respectively for the *Direct Share* and the *Indirect Share*. The resulting computational complexity is reduced to $O(n\ log\ n)$.

5.4 Operational parameters mapping

For many applications there is a direct correspondence between the bandwidth that is served and the quality experienced by the application. However, despite being the bandwidth that determines the QoS distribution, the application model demands for an explicit timing specification, beyond the bandwidth, that includes a maximum per job utilization transmission time, C_i that is repeated periodically with period T_i. Additionally, evolving toward dynamic adaptability demands for a model extension to comprise the requirements variability, thus includes ranges or sets of discrete points for both period and transmission time.

Section 5.2.2 introduced a simplified model addressing the requirements variability and the QoS distribution. For each service, U_{min_i} denotes the minimum bandwidth required and $U_{min_i}+U_{lax_i}$ denotes the maximum bandwidth. Upon distribution, each service receives a bandwidth Uv_i (Equation 5.3) within the specified operating range. Then, it is necessary to map that given bandwidth in terms of the service operational parameters, C and T, knowing that C/T denotes the bandwidth being used. For a service St_i, let Uv_i denote that given utilization bandwidth and let Cv_i and Tv_i be the materialization of such bandwidth in operational parameters. Let f be the function that does such mapping.

$$f : (Uv_i) \mapsto (Cv_i, Tv_i) \qquad : \forall_i, \frac{Cv_i}{Tv_i} \leq Uv_i$$

However, f is not bi-univocal since for a given Uv_i there is a virtually infinite range of pairs (Cv_i, Tv_i).

However, this mapping function may be subject to additional constraints derived from the application operational limitations. The application may constrain the function co-domain to specific value sets. An example is a video camera streaming that defines specific discrete values for the frame size (Cv) as well as for the frame-rate (Tv). It is thus necessary for each service St_i, not only, to include such limitations within the application model, specifying a space region of allowed (Cv_i, Tv_i) pairs that correspond to a bandwidth space BW_i, but also, to enforce those limitations within the f function.

When the allowed operational space is discrete, further constraints are imposed as the bandwidth distribution scheme that decrease the efficiency in assigning bandwidth, typically leading to waste. This situation is addressed next.

5.4.1 Complete application model

Section 5.2 presented a QoS-oriented service model $St_i\,(U_{min_i}, U_{lax_i}, qos_i)$, according to which each service gets a bandwidth Uv_i within the range $U_{min_i} \leq Uv_i \leq U_{min_i} + U_{lax_i}$

Let us now focus on the case in which the service operational space is restricted to a set of discrete points.

Let $\mathbb{C}_i$ denote a discrete list of Cs available for St_i and $\mathbb{T}_i$ denote a discrete list of Ts. $\mathbb{Z}_i$ denotes the list of pairs available for St_i, such that:

$$\mathbb{Z}_i \equiv \{(C_j, T_m), \forall C_j \in \mathbb{C}_i, \forall T_m \in \mathbb{T}_i\}$$

The service model can thus be written as:

$$St_i\,(\mathbb{Z}_i, qos_i)$$

For each service, it is specified a list of admissible pairs and the QoS importance parameter, qos_i.

Opposed to the linear bandwidth range referred in the simplified model, in this model, the available bandwidth is denoted with a discrete domain as the basis for the bandwidth distribution. For this reason, the bandwidth distribution urges for a fine-grained procedure that assigns the bandwidth based on such domain.

In order to use the QoS distribution models described in Section 5.3, which are based on a continuous bandwidth adaptation range, we propose defining a continuous range that contains all the discrete points.

Thus, considering the set $\mathbb{BW}'_i$ with all discrete bandwidth points allowed for St_i, such that:

$$bw : z_i^k \in \mathbb{Z}_i \mapsto \mathbb{BW}'_i \equiv bw(z_i^k) = \frac{C_k}{T_k}$$

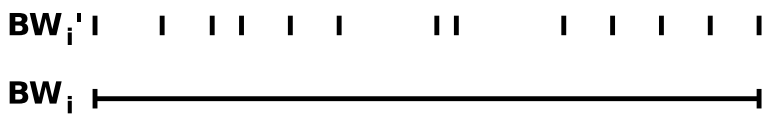


Figure 5.3: Discrete bandwidth to linear range mapping.

we define the bandwidth range $\mathbb{BW}_i = [U_{min_i}, U_{max_i}]$, where:

$$U_{min_i} = \min_{z_i^k \in \mathbb{Z}_i} bw(z_i^k)$$

$$U_{max_i} = \max_{z_i^k \in \mathbb{Z}_i} bw(z_i^k)$$

Figure 5.3 illustrates the $\mathbb{BW}'_i$ space and the corresponding continuous range $\mathbb{BW}_i$ for a service St_i.

After applying the bandwidth distribution to $\mathbb{BW}_i$ we obtain, for each service St_i, the bandwidth Uv_i, which then needs to be mapped onto a point $Uv'_i < Uv_i$. This condition is necessary to maintain the schedulability of the bandwidth distribution.

This procedure is non-optimal in terms of bandwidth distribution and resource usage because the difference between the granted bandwidth and the one actually used, $Uv_i - Uv'_i$, is wasted.

For systems where wasting bandwidth means a significant degradation penalty on the resource management, it is possible to reclaim that bandwidth and reassign it in one or more iterations. This topic is further addressed in Section 5.4.2.

However, we still need to address the assignment of a concrete (C, T) pair whenever there are several possible pairs for the same given Uv'_i bandwidth value. In this case, given a bandwidth Uv'_i, there must be a policy that chooses the pair that best meets the application goals.

The option for any mapping rule is independent from the bandwidth distribution scheme in place. In fact, it is even possible to allow that different services be managed by different rules. For this purpose and since the choice for these rules is mainly application oriented, an additional parameter is attached to the service description model, P_i, to denote this policy. The final service model used for the bandwidth distribution is thus:

$$St_i\{\mathbb{Z}_i, qos_i, P_i\}, \forall k : Z_i^k \in \mathbb{Z}_i$$

5.4.2 Bandwidth reclaiming

At this point we address the reclaiming of bandwidth left free because of using discrete operational points as refered in the previous section.

For a service St_i, with granted bandwidth Uv_i and allowed discrete bandwidth Uv'_i, we define the wasted bandwidth W_i as follows:

$$W_i = Uv_i - Uv'_i$$

The basis of the bandwidth reclamation is to group all the individual wastes of all services and redistribute it again.

$$W_{sum} = \sum_{i=1..n} W_i$$

As mentioned in Section 5.3, there are two possible distribution schemes, the greedy scheme and the weighted one. For the greedy scheme, the spare bandwidth is allocated to each service in sequence, staring from the highest priority to the lowest priority one. Each service either gets all the bandwidth it can use or the remainder. Going through all services the wasted bandwidth W_{sum} will eventually be completely allocated to the services or a certain amount will remain that cannot be used by any service.

However, the situation is different with the weighted scheme. I this case, repeating the bandwidth distribution over W_{sum} with the same differentiation weights (w_i), will result in a distribution for each service equal to W_i, thus unusable. Therefore, a distortion to the weights must be done for the sake of efficiency. We proposed two methods. One is to distribute W_{sum} to the services sequentially based on their weight (w_i), from the largest to the smallest one, and grant them the bandwidth that they can , until all W_{sum} is assigned. This procedure is identical to the greedy bandwidth distribution scheme in which the weights (w_i) are used as priorities. Therefore, the procedure is performed in a single pass with reduced complexity ($O(n)$).

Despite simple, the above procedure violates the weighted bandwidth distribution principal. The other method tries to improve the approximation of the reclaimed bandwidth distribution to the initial weights by sequentially removing some services from the process and distribute the bandwidth by the remaining ones. Each time a service j is removed from the process, its bandwidth share (W_j) is made available to others that might be promoted to the discretized output bandwidth, Uv'_i, reducing W_i.

When iterating this process, in order to improve the chances of assigning more bandwidth to the services with larger weight, in each iteration we remove from the service set the one with the lowest weight. Therefore, after the main bandwidth distribution and the related discretization mapping, where all services are included, the distribution is re-issued with one less service, the one with lower weight coefficient. The process is repeated iteratively, removing the services one by one, starting from the lower weighted service, until all the wasted bandwidth is fully distributed or all services have been removed. For each new iteration the remaining services adjust the distribution parameters in order to account with the already assigned bandwidth

and so the next distribution focus on the wasted bandwidth. Algorithm 5.4.1 depicts this procedure.

Algorithm 5.4.1: (Weighted redistribution)

---

comment: $\Gamma \equiv \{St_i\left(\mathbb{Z}_i, qos_i, P_i\right), i = 1..n\}$

comment: U_R - the resource capacity.

$$\Gamma' \leftarrow \Gamma : \Gamma' = \left\{ St'_i \left(\begin{array}{l} U_{min_i} = \min_j \left(bw(Z_i^j) \right) \\ U_{lax_i} = \max_j \left(bw(Z_i^j) \right) \\ qos_i = qos_i \end{array} \right), i = 1..n \right\}$$

$U'_R \leftarrow U_R$

repeat

$\{Uv_1...Uv_n\} = BW\_distribution(\ U'_R,\ \Gamma')$

$\{Uv'_1...Uv'_n\} = Discretization(\ \{Uv_1...Uv_n\},\ \Gamma)$

$$\left\{ \begin{array}{rcl} U'_R & \leftarrow & U'_R - Uv'_k \\ \Gamma' & \leftarrow & \Gamma' \setminus \{St_k\} \end{array} \right. , k : w_k = \min_{St'_j \in \Gamma'} (w_j)$$

until $\Gamma' = \{\}$ **or** $(\forall i = 1..n, Uv'_i = Uv_i)$

This iterative procedure to re-assign the wasted bandwidth increases the overall complexity of the QoS management. Despite the fact that it reduces the services cardinality on each step, this procedure still increases the computational complexity of the inner distribution algorithm by a power of 2, thus $O(n^2)$.

This bandwidth redistribution problem is in fact a bin-packing problem, where the bandwidth to be distributed (W_{sum}) is the bin and the discretized bandwidth levels of each service the packages. The problem is known to be NP-hard.

Given the limitations to bound the time complexity of this procedure, it is not possible to get an optimal solution regarding the resource distribution. The purpose of the proposed redistribution methods is thus to, improve the distribution efficiency in a best-effort fashion, while following a logical distribution policy. The penalty incurred by the second method may not be worth since the redistribution may refer to a small share of the overall bandwidth distribution.

5.5 QoS management on FTT-SE

The dynamic QoS management model presented in this chapter focused a constrained resources. At this level, the QoS management provides to each service the best possible quality along with a more efficient resource usage.

However, there are limitations to the deployment of this model regarding the type of resource since it must be able to support online reconfigurability, while enforcing the requirements setup for each given service. Additionally, when addressing a distributed resource, it demands for a consistent distribution agreement in order to globally enforce the resource management. Finally, and perhaps the most demanding feature is, related to the scheduler and the schedulability test that is part of the admission control. The admission test must be fast and execute in bounded time in order to be suited for dynamic environments. Moreover, in order to accomplish the QoS distribution and maximize the resource usage, the schedulability test must be based on the individual load of each service and their sum, which must be compared with the resource capacity, i.e., there must exist a utilization bound defining the schedulability region.

The FTT-SE protocol, described in Chapter 3, supports all these features. It is a hard real-time communication protocol deployed on top of a switched Ethernet network, suited for dynamic environments and enforcing the service requirements by means of a centralized traffic control architecture that consistently issues the services transmissions. Additionally, the protocol supports any traffic scheduling policies, namely EDF and RM, for which there are utilization-based schedulability tests.

However, given the multiple links of a switch the analysis has the particularity of addressing the network as a multi-dimensional scheduling problem, where each uplink and downlink represents a resource with dependent relationships. Nevertheless, the problem is reduced to several uni-dimensional utilization-based analysis, in spite of being assessed globally for the complete switch.

Similarly, the integration of the QoS distribution has to be conducted individually on each link. In the following we discuss this integration within the FTT-SE protocol.

The QoS distribution is intimately related to the capacity provided to each link by the schedulability bound. Although the links have similar initial capacity, the utilization load is irregular with the different forwarding paths generated and the dependencies that thereby result.

Recalling the distribution base model described in Section 5.2.2, for a single resource, after assigning the default bandwidth of each service, the U_{spare} bandwidth is distributed among the services according to a distribution policy. The same principle applies here to each individual link. The difference though, is that different links have distinct loads and the same service may be seen in different links, resulting on multiple bandwidth reservations for the same service. There may be a different reservation value for each resource where a service takes part.

The distribution model is generically defined as follows. Let M be the list of m individual resources, uplinks and downlinks, and St the list of s services following the QoS distribution model. Each M_j resource holds a list

of services from St, denoted by S_{M_j}, which may appear in different resources, co-existing with different other services. Therefore, assessing each resource individually results on different reservation values for the same service.

However, despite the variety of reservation values per service, there must exist a global agreement regarding the resources assigned to the service that must be unique since it is not possible to execute a service with different characteristics across resources. Therefore, the bandwidth reserved for a service is determined by the resource that provides the lowest reservation level, as follows:

$$Uv_i = \min_{\forall M_j \in M} (Uv_{(i,j)})$$

where $Uv_{(i,j)}$ denotes the bandwidth reserved within the resource j for service i.

The consequence of such limitation is the bandwidth being left unused, leading to a more inefficient distribution. For each resource, the wasted bandwidth is given by:

$$\forall M_j \in M, \qquad W_{sum\,j} = \sum_{St_i \in S_{M_j}} (Uv_{(i,j)} - Uv_i)$$

This bandwidth waste is inevitable when addressing the distribution of the resources independently. Though, when considering the distribution method that assigns the bandwidth on a fixed importance basis (Section 5.3.1), it is possible to avoid this waste, comprising all the resources together in the same distribution flow. Starting from the most important service, the bandwidth is reserved with the exact amount, relating all the resources. This avoids inconsistency on reserving across several resources and leverages a more efficient distribution.

However, the same reasoning cannot be followed when addressing the weighted distribution method. The resource distribution is resource independent and cannot be merged in a single procedure flow. Yet, the bandwidth can be reclaimed and redistributed among the services, following a similar procedure as when mapping the bandwidth onto operational parameters (Section 5.4). In this case, after distributing and truncating the bandwidth by the minimums, the remaining bandwidth can be redistributed as depicted in the following algorithm.

Algorithm 5.5.1: (Multi-resources weighted redistribution)

Given:
$m \equiv$ number of resources
$r_i \equiv \{a_j, j = 1...m\} : a_j = \begin{cases} 1 \Leftarrow St_i \in S_{M_j} \\ 0 \Leftarrow \text{otherwise} \end{cases}$
$St \equiv \{St_i \left(U_{min_i}, U_{lax_i}, qos_i, r_i\right), i = 1...s\}$
and $U_R = \{U_{Rj}, j = 1...m\}$, denoting the capacity per resource.

$\Gamma \leftarrow St$
$U'_R \leftarrow U_R$

repeat

$$Uv_{(S,M)} = \begin{bmatrix} Uv_{(1,1)} & \cdots & Uv_{(s,1)} \\ \vdots & \ddots & \vdots \\ Uv_{(1,m)} & \cdots & Uv_{(s,m)} \end{bmatrix} = BW\_distribution(U'_R,\ \Gamma)$$

$$\{Uv''_1 \cdots Uv''_n\} \leftarrow Uv_{(S,M)} : Uv''_i = \min_{j=1...m} Uv_{(i,j)}$$

$$\begin{cases} U'_R & \leftarrow & U'_R - (Uv''_k \times r_k) \\ \Gamma & \leftarrow & \Gamma \setminus \{St_k\} \end{cases} , k : w_k = \min_{l=1...\#\Gamma}(w_l)$$

until $\Gamma = \{\}$ **or** $(\forall_{St_i \in \Gamma, M_j \in M}, Uv''_i = Uv_{(i,j)})$

The function referred as *BW_distribution* distributes the capacity of each resource (U_{Rj}), by the related services, listed in $S_{M_j} = \{St_i : r_i(j) = 1\}$. The result is a set of bandwidth reservations per service and per resource.

The bandwidth assigned to each service is then derived in Uv''_i, taking the minimum reserved values across all the related resources. The bandwidth reclaiming procedure is finally conducted, issuing new distributing iterations while removing services to the Γ set of services. In each new iteration, the service to be removed is the one with minimum distribution weight, for which the bandwidth obtained in the previous iteration is kept. Simultaneously, the share corresponding to the outgoing service is removed from the distributing capacity. The iterative procedure ends when either no bandwidth is being wasted or Γ becomes empty.

In terms of complexity, the procedures hereby presented are similar to the ones presented in Section 5.4, addressing the bandwidth reclaiming for the discrete model mapping. Similarly, the procedure that reassigns bandwidth in a fixed importance basis adds an $O(n)$ complexity order, while the weighted distribution procedure increases the complexity of the individual distribution by a power of 2.

5.6 Conclusion

The embedded systems in general and the network communications in particular are following a trend that includes making a better use of the resources, by efficiently adjusting online the requirements according to the application dynamics, while guaranteeing the system correct behavior regarding timeliness issues.

On this regard, applications have to be supported with models that define the environment dynamics and how each individual application service or job adapts facing those dynamic changes. However, the search for these models is confined to the ability of delivering fast and reliable adjustments as demanded by the online properties of the system.

This chapter aims to categorize several of those models, oriented in a design perspective of providing quality-levels that distinguish the preferred services when the resource, or network, is overloaded. The models provide the means so the application may define the quality of service requirements for each service contract as well as defining how that quality evolves before other competitor services.

This whole QoS management problem is decomposed in several conceptual phases, 1) the application interpretation and mapping of the model into actual quality-levels, 2) the bandwidth distribution of the resource capacity that based on those quality-levels assigns bandwidth to the services, 3) the mapping of the obtained bandwidth into the operational parameters used by the application, and finally, 4) the management of a resource in a multi-dimensional perspective as in the FTT-SE protocol.

The models are categorized according to their distribution fairness on regard to the application specifications and according to the computational complexity overhead. Through the QoS management decomposition layers several approaches are conducted to minimize the impact on that complexity and optimize the bandwidth distribution, where one of the issues comprehends the minimization of bad resources allocation, i.e., bandwidth waste.

Concerning the deployment of these models within the FTT-SE framework, the proposed models fit under the constraint that distribution has to be carried on multi-dimensionally. To validate such integration Chapter 6 carries an experiment with surveillance cameras where this QoS distribution is actually implemented as the mean to deliver the best instantaneous throughput of the network and considering the application defined expectations concerning QoS.

Chapter 6

FTT-SE case studies

This chapter compiles a collection of case-studies developed on top of the FTT-SE architecture. Firstly, it addresses the integration of the protocol within a wider-scoped resource management framework, which includes reserving for multiple resources with distinct class types (Section 6.1). In Section 6.2 the FTT-SE framework is used in a real application scenario where its reconfigurability and QoS management properties play an active role. Then, Section 6.3 shows how the protocol is able to support complex scheduling structures such as aperiodic servers and hierarchical composition, in order to ease the application design. Finally, Section 6.4 describes a preliminary validation of a work-in-progress and future work related implementation that includes the master node within the switch.

6.1 Integration in the FRESCOR framework

Networked Embedded Systems (NES) were originally associated with industrial supervision and control applications, which employed simple sensors, actuators and controllers. However, a steep evolution in this application domain is being experienced, pushed by the growing number of sensors and overall complexity present at the plant level. As an example, the use of imaging sensors, both for supervision and control purposes, is spreading widely in classes of applications such as mobile robotics, traffic control and assembly lines inspection. Consequently, the sensors become inherently more complex, so as the flows of information exchanged at the cell and plant levels, integrating periodic and aperiodic flows of short and large data, some of multimedia nature, with considerable variability during run-time.

The new demands and increased complexity posed by these applications pushed the development on new techniques and design methodologies. Two key aspects in this regard are the complexity management and the resource management. Complexity management is being addressed by the adoption of adequate middleware layers in NES (e.g. CORBA, Java RMI, DCOM,

etc)[134], which abstract away distribution, aiming at transparent interaction mechanisms between objects, components or applications. Regarding the resource management (e.g. CPU, memory, network, energy, etc) several approaches have been proposed recently, aiming at fulfilling the needs of those emerging applications in aspects like dynamic configuration and QoS management, support for new and more efficient scheduling techniques, etc.

The FIRST Scheduling Framework (FSF) [25] provides a high-level abstraction for real time resource schedulers while maintaining predictability and performance efficiency. It provides a homogeneous interface so it can be used in different platform architectures. This framework was initially designed to cope with the application needs for processor and network management, although with some limitations in the latter. The Framework for Real-time Embedded Systems based on COntRats (FRESCOR) [66] aims at extending the FSF framework for multi-resource reservation, comprising various classes of resources commonly found in NES applications.

Regarding the communication subsystem, an early implementation has been proposed to address negotiation over an Ethernet resource. This is the Real-Time Ethernet Protocol (RT-EP) [90]. The proposed Flexible Time Triggered communication over Switched Ethernet (FTT-SE) (Chapter 3) provides flexible and deterministic real-time communication services combined with dynamic Quality of Service (QoS) management. This protocol has been developed specifically to address the requirements presented by the emerging applications referred above, combining real-time requirements with a high degree of adaptability. It looks, thus, a natural network candidate for inclusion in a contracting framework, to efficiently exploit and enrich the high level of flexibility that it already offers.

This section the integration of the FTT-SE protocol in the scope of the contract model framework and describes how this integration can be performed fulfilling issues like distribution consistency, reservation guarantees and run-time flexibility. The work has been carried out in cooperation with Michael González Harbour, Daniel Sangorrín e a Julio L. Medina from the University of Cantabria in Spain [86].

6.1.1 FRESCOR background

The FRESCOR framework is based on the notion of contracts between the application and the system resource manager. These contracts are created, managed and enforced by a *Contract Layer*, which assures that sufficient resources capacity is available. The framework is divided in modules that allow abstracting away the specificities of the resources typically found in NES. Of particular interest to this work are the *Core* module, which contains the basic contract information that must exist in all contracted resources, the *Spare capacity* module, which defines how the application may take advantage of currently unused resource capacity, and the *Distribution* module that deals

with issues of distributed applications.

The main contract parameters associated to these modules are referred in Table 6.1. The *Label* is a unique identifier inside one resource (here a network resource) that distinguishes the contracts globally, the *Resource type* and the *Resource id* inform about the kind of, and which resource the contract refers to. The *Spare capacity* is meant to provide some operational flexibility for Quality of Service (QoS) management by maximizing the number of running services in the system while complying with their minimum requirements, and simultaneously getting the maximum usage from the resources. In this scope, the *Minimum budget*/*Maximum budget* and *Maximum period*/*Minimum period* define the minimum and maximum application resource requirements, respectively, limiting the contract negotiation range. It is also possible, inside such range, to define discrete utilization tuples using the *Utilization set* parameter. As more and more services are bound to a resource, the allocation eventually reaches an utilization limit and the system becomes overloaded. At this point the services utilizations are readjusted within the specified ranges, according to a given QoS management policy, possibly degrading performance of the already admitted services. The QoS policy can make use of several parameters to discriminate the application services, namely the parameters *Importance* and *Weight*, which allow prioritizing the contracts associated to one resource when distributing spare bandwidth, and the parameter *Stability time* that defines the settling time that must be respected before re-adjusting each contract bandwidth. The *Distribution* module allows defining network dependent information, which, in this work, will be used to support FTT-SE parameters. It also specifies the *Queuing info* parameter that describes the size of the message queue used for sending messages through the contract, in terms of the maximum number of messages that it can hold, and the rejection policy used when a message arrives at a full queue: oldest, or newcomer. A value-based communication in which the most recent value of a given variable is transmitted can be easily implemented by specifying a maximum queue size of one message, and a rejection policy of oldest.

FRESCOR network adaptation layer

The FRESCOR architecture is designed to be modular in terms of supporting a wide range of resource types while maintaining the same application interface. The contract itself is divided in several modules, each associated to a different resource type, and which can be plugged into the proper resource class interface. The *network* is one such class that joins all possible network types, possibly using different topologies, different media, different medium access and even different communication semantics. The network class interface is called the FNA (FRESCOR Network Adaptation layer) to which different networks are plugged to.

Core	Label Resource type Resource id Minimum budget Maximum period Deadline
Spare capacity	Granularity Maximum budget Minimum period Utilization set Importance & Weight Stability time
Distribution	Protocol dependent information Queuing info

Table 6.1: Contract Parameters.

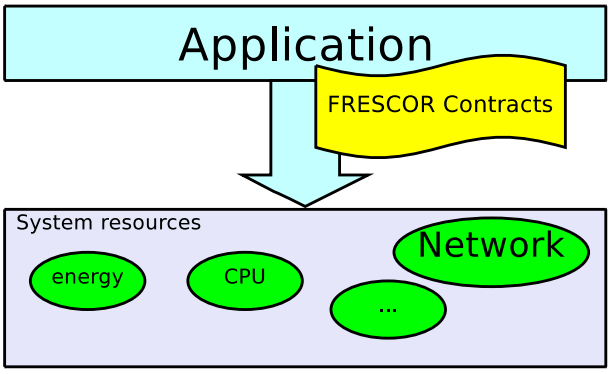


Figure 6.1: FRESCOR resources.

The FNA specifies services for negotiation and renegotiation of contracts, for creating endpoints for sending and receiving messages, for binding the endpoints to contracts, and for sending and receiving messages through the endpoints.

FRESCOR application model

FRESCOR uses a simple application model based on threads that need to acquire the necessary resources, such as CPU, memory or energy, by means of negotiated contracts. Those threads access the system resources by means of *Virtual resources (Vres)* residing in the *Contract Layer*. Each *Vres* holds one associated contract.

In distributed applications, this layer is also distributed comprising the network resource management. A notion of *Stream* is introduced to model

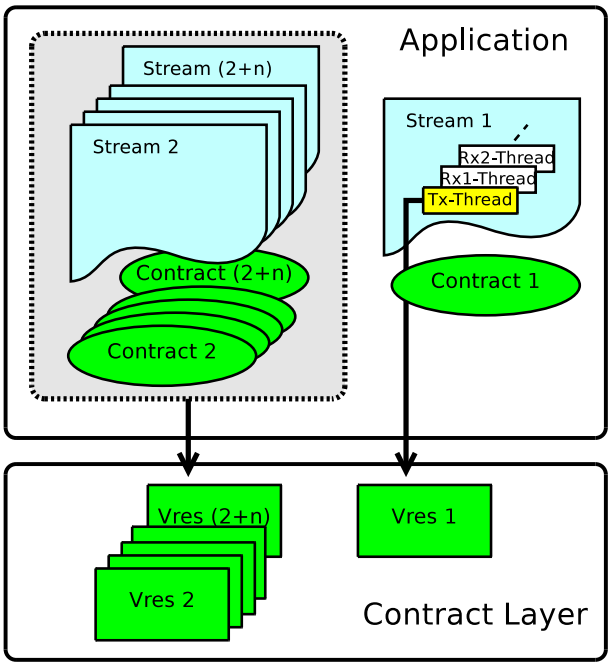


Figure 6.2: Network application model.

the network transactions. In the application model a *Stream* is associated to several *Threads* (Figure 6.2), i.e., one sender and one or more receivers, which must also be negotiated in the subsystem environment of their respective nodes. A *Stream* negotiation is conducted as a multi-resource group contract that includes the network and the CPU contracts at the end nodes. Such multi-resource contracts are handled within FRESCOR in an horizontal layer that spans across all the distributed system.

For the sake of clarity in the remainder of this section the network resource is addressed alone, excluding the CPU resources at the end nodes. It is thus considered the *Stream* negotiation as a simple contract that, if succeeded, is mapped onto a single *Vres* in the contract layer. Each stream has a unique identifier mapped on the *Label* and used as a stream descriptor by the application when accessing the *Contract Layer*.

For all network resources, the *Vres* objects are stored in the associated module plugged to the FNA interface, but they can also be instantiated in a remote node, as seen later on. Inside the same network resource (*Resource type* + *Resource id*) several contracts may also be atomically negotiated as a group, ensuring that they are either all accepted, or all rejected as a whole.

Finally, the network contract negotiation for a given stream can be initiated by any network node and even by more than one node but in this case the application must ensure the respective consistency. Figure 6.2 shows the FRESCOR distributed application model with a network resource highlight-

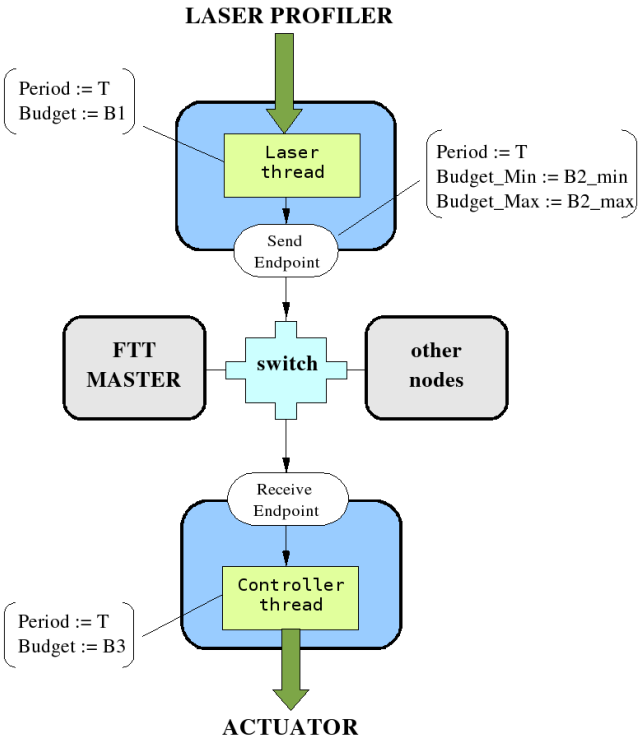


Figure 6.3: FRESCOR example.

ing the network contracts in two possible situations: when a contract *Virtual resource* is created in a transmitter node (Stream 1), or in a contract group situation (Streams 2 .. (2+n)). Details on the FRESCOR interface services for networked applications are better described in Section 6.1.3

6.1.2 FRESCOR application example

The following example illustrates the use of FRESCOR through a very simple distributed system that integrates a welding system. The system captures an array of points from a laser profiler representing the depth of the welding area and then send it to another node where a controller activates an actuator, for moving the welding torch. The distributed system is supported by an FTT-SE network, which provides communication channels upon negotiation.

As shown in Figure 6.3, three contracts must be negotiated:

- **Network contract:** this is the contract needed to send the array of points to the controller. It specifies a range of budgets so that if the network has enough capacity it can send all the points, but if the network becomes overloaded, e.g. in consequence of new contracts being negotiated, a lower capacity would be granted by the system,

although always over the minimum required. This is the functionality provided by the *Spare Capacity* module.

- **Laser thread CPU contract:** this is the thread in charge of capturing, processing and sending the points obtained from the laser profiler. In each period it reads the current budget assigned to the network contract to see how many points can be sent. Depending on this information it will process and send more or less points.

- **Controller thread CPU contract:** this is the thread in charge of receiving the data from the network and actuating on the motors controlling the movement of the welding torch. Its budget must be prepared for processing the maximum size of the array.

On the other hand, CPU budgets are fixed and must cover the requirements for processing the whole set of points of the laser. When only a subset of the points is needed to be used in the transaction, threads will use less capacity and the FRESCOR *dynamic reclamation* module will give that capacity to other threads.

The pseudocode of the threads involved is depicted in Figure 6.4. In this simple example, where FRESCOR is used at a low level of abstraction, all the contracts involved in the transaction have a fixed period. If renegotiations in the transaction were necessary to switch to different periods, they should be made through the FRESCOR high-level transaction manager which will provide the necessary distributed coordination for a global negotiation of the diverse resources involved in the transaction (including CPU, networks, memory or even bus accesses). A deeper discussion of the FRESCOR renegotiation mechanisms description is outside the scope of this work.

6.1.3 FTT-SE under FRESCOR

One important goal of the integration is to keep the performance level of the FTT-SE real-time communication services throughout the abstraction process associated with the creation of a middleware. The FRESCOR framework was selected as middleware because it facilitates achieving this goal given its simple and generic application interface and real-time concerns. Moreover, its modular flexibility extends the resource management to an holistic application perspective which reduces the project design complexity.

This section describes how the FTT-SE protocol can be integrated as a FRESCOR pluggable resource. This integration allows abstracting away the network access from the application perspective. It defines two sets of services, the negotiation procedure and the communication access primitives. The former handles the contract (re-)negotiations requested by the application to change the Stream properties provided by the network resource.

Main threads

```
vres := negotiate CPU contract
create thread & bind to vres
```

Laser thread

```
n_vres := negotiate network contract
create send_endpoint & bind to n_vres

loop
   c := get budget (n_vres)
   read laser profiler
   process points (c)
   send points (c)
   frsh_timed_wait
end loop
```

Controller thread

```
create receive_endpoint

loop
   read points
   process points
   send command to actuator
end loop
```

Figure 6.4: FRESCOR example pseudocode.

Once a contract is accepted the application may start using the respective communication Stream through the services provided by the latter.

As referred before, the FRESCOR modules used when integrating FTT-SE are the *Core*, *Spare capacity* and *Distribution* modules. Each of these takes its role in the contract negotiation with the parameters described in Table 6.1. The *Distribution* module needs special attention since it contains network protocol dependent information. For the RT-EP distributed resource no special parameters were required, thus the *Core* parameters *Minimum period* and *Maximum Budget* were enough to carry out the network management. However, FTT-SE includes several features that require appropriate configuration and management, which must thus be included in this module:

- Two communication triggering models are provided by the network,

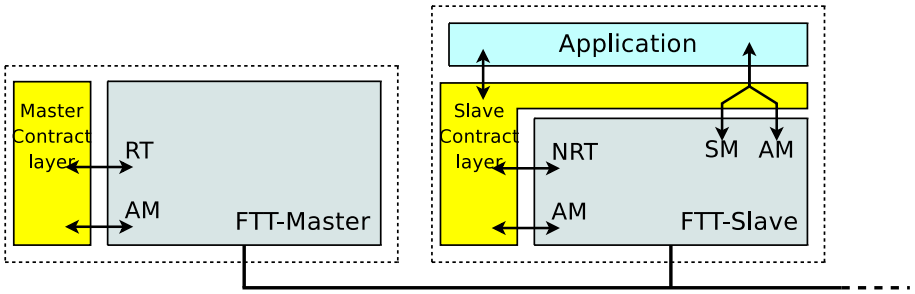


Figure 6.5: Architecture overview.

namely time-driven and event-driven, which must be defined in the contract specification;

- To take advantage of the multiple forwarding paths in the network switch and still provide real-time guarantees, the contract must include the specific switching path used by each channel, i.e., the identification of the producer and consumers involved and the switch ports they are connected to;

- To exploit the explicit synchronization between time-driven channels supported by the network, the contracts, or contract groups, must include two additional parameters, one describing the *Label* of the channel to synchronize with, and another one specifying the desired synchronization offset. If no synchronization is specified the channel is considered as float and the network will arbitrarily allocate relative offsets to the contract;

The integrated FRESCOR / FTT-SE architecture is sketched in Figure 6.5. The network contract negotiation procedure is centralized in the FTT master node and it is handled by the *Master Contract Layer*. This is a natural choice since the FTT master centralizes all the real-time requirements of current communication channels and controls the network access. The contracts are then reflected on the involved slave nodes. The *Slave Contract Layer* handles network contract requests, holds local contract copies and makes them available to the application threads. This centralized approach is substantially different from the one taken in the RT-EP implementation, where the negotiation procedure is fully distributed requiring every node to keep a consistent replica of all running contracts.

FTT-SE interface for FRESCOR

The communication between the *Master Contract Layer* and the *Slaves Contract Layer*, both for conveying negotiation requests and publishing the contract copies, uses permanent bi-directional channels between the master and

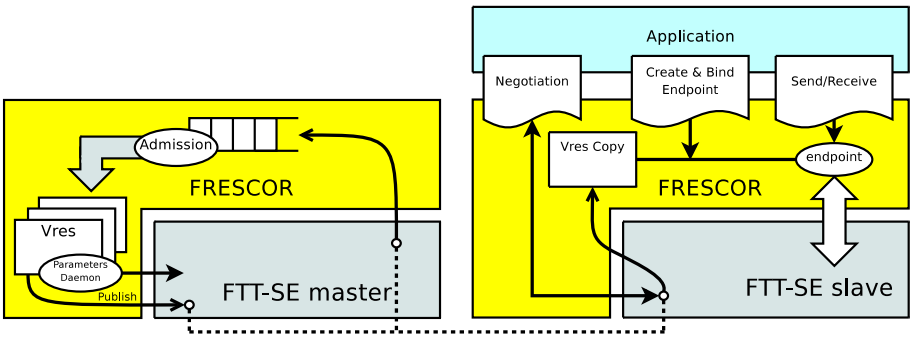


Figure 6.6: FRESCOR interface for network contracts.

each slave node in the network, implemented with FTT-SE asynchronous messages (AM).

The network contracts in the *Master Contract Layer* are reflected in the FTT Master, in the SRDB. This layer also provides interfaces to the application, to negotiate contracts and to access the communication services, both synchronous (SM) and asynchronous (AM).

Supporting the application interface

Figure 6.6 shows the FRESCOR common resource interface and the objects involved in network contracts. The *Negotiation* service allows the system to establish the required resource reservations and, in this case, involves communication with the *Master Contract Layer*. Upon a negotiation success, the respective contract *Virtual resource(s)* is(are) created/updated in the *Master Contract Layer* and the *Virtual resource copies* are created/updated in the *Slaves Contract Layer*. The access to the contracted resource, a Stream in this case, is made through an *endpoint*, which is created and bound to the *Vres* by the *Create & Bind Endpoint* services. Finally, the *Send/Receive* services allow the communication through the respective endpoint.

The network *Virtual resources* in the contract layer must be consistent with the communication parameters within FTT-SE so that the protocol actually enforces the contracted communication parameters with its control mechanisms. Therefore, a *parameters daemon* is used to keep such consistency. Whenever a server is created or updated dynamically, the daemon communicates such changes to the FTT-SE layer.

6.1.4 Internals of the contracting procedure

The setup of network contracts with FTT-SE, as referred before, requires an interaction between the *Slave Contract Layer* of the involved nodes and the *Master Contract Layer*. The process is triggered by the thread that manages

the contract or the contract group and its sequence diagram is depicted in Figure 6.7.

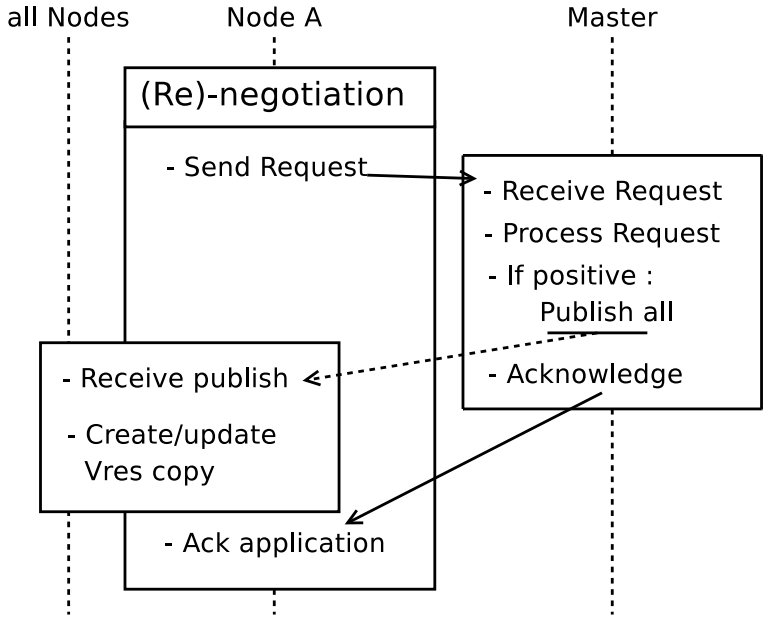


Figure 6.7: Negotiation steps.

The request is enqueued in the *Master Contract Layer* until it can be processed (Figure 6.8). At that point, it is removed from the requests queue and submitted for admission process, which involves the admission control in the FTT-SE master. If the contract is accepted, which may result in changes to other contracts, the master updates the FTT-SE internal structures and publishes all *Vres* that were updated. The respective slaves receive this information and update/create the respective *Vres copies*. Then, the master acknowledges the negotiation result.

Linking with the FRESCOR example described in the Subsection 6.1.2, after the Laser thread is created and bound to a CPU Vres, it triggers the negotiation of the network contract through a call to the FRESCOR Core module. This call will be mapped through the FNA layer to FTT-SE, where a negotiation request message will be sent to the Master via a preallocated AM negotiation channel and stored in a queue. While the Laser thread is blocked waiting for the finalization of the negotiation process (asynchronous notifications are also supported for renegotiations), the master will read the request from the queue and perform an admission test using the minimal requirements of all the contracts in the network (if the *stability time* of a Vres has not expired the currently assigned parameters will be used), in this case a minimum budget *B2_min*. After that, the *Spare capacity* will be distributed among the network contracts according to their importance, weight

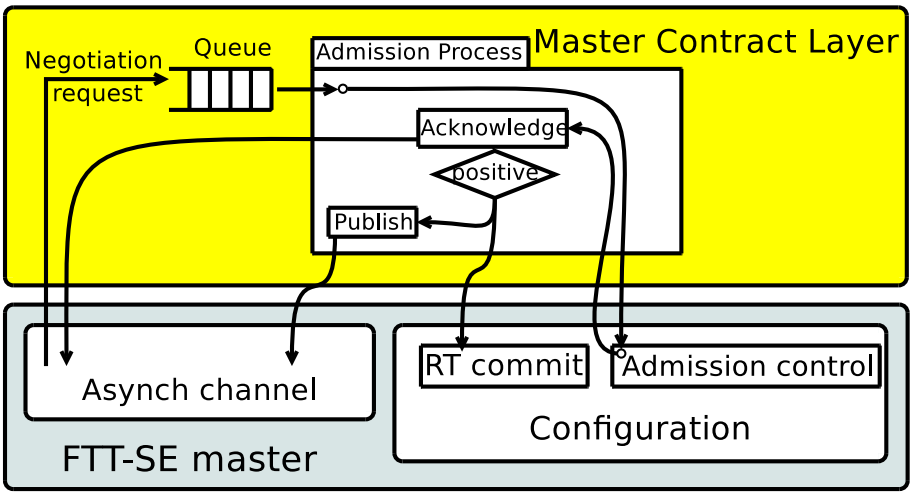


Figure 6.8: Master contract negotiation procedure.

and maximum requirements. A budget value between $B2\_min$ and $B2\_max$, called $Current\_B2$, will be assigned to the resulting network virtual resource and other, previously existing, Vres might also get new values.

The results of the negotiation will be published to the corresponding FTT-SE nodes (in FTT-SE this is efficiently undertaken internally) and the associated Vres will be updated by the FTT-SE FNA layer. After the publication of the results an Acknowledgement message will be sent to the Laser node and the Laser thread will be waken up. This Acknowledgement is necessary to make sure that all the nodes have received and processed the new Vres values.

After binding a Send_Endpoint to the Vres, the Laser thread will start sending the points periodically. For doing that, in each period it will get $Current\_B2$ and send more or less points per period depending on its value. As a copy of the current parameters is stored in the node, getting data associated to the Vres is not resource consuming. Meanwhile the Master node will have updated its scheduling table and will send the corresponding trigger messages using it.

6.1.5 (Re-)negotiation procedure time

A network (re-)negotiation procedure is typically triggered by the application in either of two situations: during system startup, where the negotiation time is not so critical, and in a dynamic situation where the negotiation is a consequence of an environmental or structural change. In this latter case, the negotiation request may be subject to time constraints determined by the environment dynamics. Therefore, it is important that the system designer can estimate and bound the time required by the negotiation procedure,

Neg_Rt. This subsection identifies all the variables that play a role in that time bound and show that the bound is determined in polynomial time.

According to the negotiation steps depicted in Figure 6.7 and Figure 6.8, Equation 6.1 summarizes the time spent in the generic i^{th} negotiation procedure (Neg_t_i) considered in isolation, i.e., no other request is currently being handled.

$$Neg\_t_i = AM\_t_{S_i} + P\_t_i + max(Set\_t_i, AM\_t_{M_i}) \tag{6.1}$$

$AM\_t_{S_i}$ is the time taken by the transmission of the asynchronous message from the slave to the master, P_t_i stands for the processing time spent by the admission control algorithm, Set_t_i is the time taken to update the FTT-SE internal mode changes and finally the acknowledgement time using an asynchronous transmission from the master to the slave that is accounted for in $AM\_t_{M_i}$. Set_t_i refers to the settling time enforced by the FTT-SE manager to start the new traffic pattern without jeopardizing the timeliness of the currently running traffic. This is a network specific and predictable delay depending on the responsiveness of the asynchronous transactions undertaken between the network nodes when updating the resource communication requirements status.

However, in the general case it may happen that when a negotiation request arrives at the master, another previously issued request(s) is(are) already pending to be served, thus causing further interference (Figure 6.8). Thus, the generic request i will not only take the amount of time imposed by the negotiation process itself as given by Equation 6.1 but it will also suffer a queuing delay that corresponds to the time taken by the admission control of all negotiation requests that arrived before, considering a FIFO queue. Equation-6.2 allows deducing such bound, Neg_Rt.

$$\begin{aligned} Neg\_Rt_i &= AM\_WCRt_S \\ &+ \sum_{j=1}^{max_c} P\_Wt_j \\ &+ max(Set\_Wt_i, AM\_WCRt_M) \end{aligned} \tag{6.2}$$

The asynchronous message response time bounds, AM_WCRt_S and AM_WCRt_M, are obtained in [101], so as the settling time (Set_Wt).

To bound the delay caused by the enqueued requests the queuing policy must be considered, which in this case is a FIFO, and thus it must be assumed that the queue contains all possible other requests for application contracts. Let max_c be such maximum number of requests admissible by a certain application. Given the worst-case execution time for the admission control (P_Wt), the interference delay caused by the queue is upper bounded by $(max_c - 1) * P\_Wt$.

Therefore, an upper bound for the negotiation time Neg_Rt^+ can be obtained from Equation 6.3.

$$\begin{aligned} Neg\_Rt_i^+ &= AM\_WCRt_S \\ &+ (max_c - 1) * P\_Wt \\ &+ max(Set\_Wt_i, AM\_WCRt_M) \end{aligned} \tag{6.3}$$

6.1.6 Summary

The FRESCOR framework has been proposed recently to cope with the growing application complexity and interoperability requirements in embedded systems. The approach followed by FRESCOR allows abstracting the management of the application resources, which are accessed through a common interface based on contracts.

This section discussed the integration of the FTT-SE real-time communication protocol within the FRESCOR contracting framework. This framework efficiently exploits the FTT-SE natural ability for dynamically adapting the network resource usage while maintaining predictability. Another positive aspect in this symbiosis is that the interface given by the framework to the application can be commonly applied together with other shared resources of the system.

Previously, only the RT-EP network protocol had been integrated within the FSF/FRESCOR framework. Such protocol works over a shared medium with a priority based event-triggered messaging paradigm. The integration of FTT-SE brings in the features of a different network paradigm and topology, namely time-triggered and event-triggered communication over switched Ethernet. The dynamism of FTT-SE positively impacts the efficiency of the network management in the FRESCOR framework. On the other hand, the abstraction provided by FRESCOR benefits the FTT-SE protocol in terms of its usability and applications development.

Finally, this section presented a worst-case analysis for the time taken by the negotiation process, provided a few network parameters, easily determined, as well as a bound on the number of possible simultaneous negotiation requests. Such bound guarantees the timeliness for the negotiation procedure, which is of utmost importance for adaptive critical systems operating within dynamic scenarios.

6.2 Industrial multimedia application

The use of Media Control Applications [57] (MCA) such as machine vision, automated inspection, object tracking and vehicle guidance [109, 75, 43, 69] in today's industry is increasing strongly. Such applications can be classified in two broad classes [57], *Supervised Multimedia Control Subsystems* and *Multimedia Embedded Systems* (MES). While the former class emphasizes the quality of the media processing, relieving the real-time constraints to a soft real-time level, the latter applications class (MES), in addition to the media processing concerns, establishes hard real-time requirements on their application goals. These applications are typically complex and heterogeneous, comprising several real-time activities in addition to the media processing ones. Thus, the interference of the multimedia processing components must be limited and predictable.

Many MES applications rely on networked embedded systems (NES) that depend on real-time communication protocols to support the necessary real-time services. However, multimedia traffic in general, and video streaming in particular, have specific characteristics that conflict with the operational model of conventional real-time protocols. In particular, video compressors generate highly variable bit rate (VBR) transmission patterns that mismatch the constant bit rate (CBR) channels typically provided by real-time protocols, severely reducing the efficiency of network utilization. Matching a VBR source to a CBR channel is not trivial and may lead either to a waste of bandwidth or rejection of frames. This difficulty becomes particularly challenging with the emergence of the MES applications described above, which typically impose reliability and timeliness requirements that cannot be fulfilled with standard network protocols [125], due to lack of temporal isolation and consequent unbounded mutual interference between streams.

This section is based on a work conducted in cooperation Javier Silvestre from the Polytechnic University of Valencia in Spain [63, 113], which proposes taking advantage of the dynamic QoS management features of the FTT-SE protocol to carry out the referred adaptation with MJPEG video streams. In particular, it proposes managing in an integrated way both the compression parameters and the frame acquisition properties, which drive the encoding of frames that must fit strictly inside the bandwidth allocated to each stream. This approach aims at maximizing the QoS level provided by each communication channel and by that means minimizing the compression used at each instant.

The QoS adaptation is presented in two stages, regarding the variation degrees of the acquisition properties. In a first stage only the frame acquisition period is manageable to best reflect the bandwidth provided. The frame size, although actually variable with the compression result, is considered to fit a fixed size container, property of the FTT-SE allocated channel. In a second stage both parameters, period and frame container are manageable

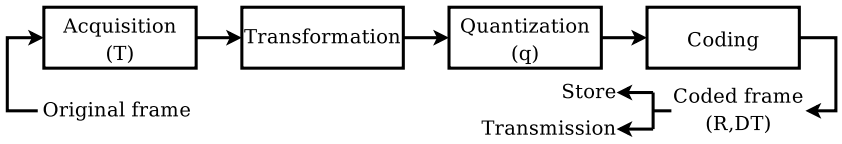


Figure 6.9: Generic encoding process.

to deliver the best QoS level when adapting the QoS stream to the provided bandwidth.

The work is presented by firstly addressing some background issues regarding multimedia transmission. Then, it describes the system architecture and details on the two approaches for QoS management. Finally some simulation and experimental results are presented.

6.2.1 Related work

Multimedia compression standards

When addressing multimedia transmission or storage, compression plays an important role to reduce the footprint of the media stream. It provides data reduction, yielding either faster transmissions or lower storage requirements. On this regard, two phases are identified, a compression phase, where the compression algorithm identifies redundant data that can be removed, and the decompression phase, where the reverse algorithm obtains the original or similar stream. The compression process is generically described in Figure 6.9. In this process, the most important factors are the coding bit rate R and the quality obtained DT (Distortion), which depends on the quantification factor q, acquisition parameters like period T or resolution r, among others.

Many multimedia compression standards exist today and the choice for one typically results of a trade-off between compression ratio, data loss (distortion), encoding/decoding time and computational requirements. Depending on the operation principle, the compression algorithms can be divided in two main classes: still image compressors and video compressors. Still image compressors use intra-frame compression, i.e., the data being compressed is exclusively within the frame, while video compressors exploit the temporal redundancy that exists in sequentially acquired images. The most common still image compressor standards are the JPEG [8] and, more recently JPEG2000 [89]. With respect to video compressors, the most common standards are MPEG-2 [9], H.263 [10] and MPEG-4 part 2 [14] and part 10 [1], also known as H.264 or MPEG-4 AVC.

Selecting the most adequate compression technique is not a trivial matter. The use of video compressors, can potentially reduce the storage and

bandwidth with respect to, for instance, motion JPEG (MJPEG) by ratios as high as 10:1. However, taking the example of surveillance/recording systems, images are frequently captured at low rates and sometimes multiplexed, which reduces or even eliminates the temporal redundancy and thus seriously compromises the efficiency of these compression algorithms. In addition, video algorithms apply techniques that compensate variations in the network load or processing throughput, adjusting the video image quality, which may be adequate for monitoring applications, but not for MES or surveillance/recording applications [18], often found in industrial environments. Moreover, still image frames tend to be better accepted within systems that interpret frames individually as a sensor data quantum, e.g. a control system that periodically acquires a frame. The use of motion JPEG (MJPEG)is generally found only in digital cameras and monitoring systems (e.g., Lumera, Cast, Axis [18], Mobotix [21]). Concerning the use of JPEG2000, it was expected by 2001 that this compressor would replace JPEG in 2 or 3 years. Though, some drawbacks have made this transition more difficult, among others, the lack of backward compatibility and the additional complexity in terms of memory footprint requirements and computing time that nearly triples for the compression and decompression algorithms. Also, the gains in terms of perceived quality are not relevant, being only noticeable with extreme compression levels and the licensing for the JPEG2000 codec limits the adoption such technology.

Furthermore, still image transmission is more robust than video transmission. This conclusion can be drawn from the fact that in still image compression the frames are independent of each other, and thus losing one image (or parts of one image) has no consequence for the following images. In turn, video transmission uses different frame types, namely I-frames (independent frames), P-frames (inter-frames coded depending on previous frames) and sometimes B-frames (on pass and future frames). Only I-frames are self-contained, thus the loss of a frame (or part of a frame) may have impact on several of the following frames, until the arrival of another I-frame. This effect is further aggravated by a common practice that consists in enlarging the distance between I-frames to reduce the bandwidth utilization. Another aspect that penalizes the use of video compression is related to the fact that in industrial applications the images are frequently captured at low rates and sometimes are multiplexed. Any of these situations can reduce severely the temporal redundancy and thus the level of compression that can be attained.

Whenever timeliness requirements come into play, the lower latency of JPEG with regard to other video/still image compression techniques presents a significant advantage. The relevance of this aspect is growing in consequence of the use of increasingly higher resolution image sensors, which have evolved from the traditional CIF (Common Intermediate Format) and QCIF (Quarter CIF, 352x288 and 176x144 pixels), with a maximum size of 4CIF 704x576 pixels, to VGA (Video Graphics Array, 640x480 pixels), XGA

(Extended Graphics Array) with a 1024x768 pixels, SXGA with 1280x1024 pixels, WUXGA with 1920x1200 pixels, arriving to the WHUXGA format, with 7680x4800 pixels. Thus, the amount of data to process in each frame is growing exponentially, emphasizing the importance of the compressor latency.

Multimedia transmission

Multimedia transmission over the Internet has been the subject of intense research in the recent years [35, 130]. Typical solutions are based on the TCP/IP protocol stack complemented by other protocols, e.g., RTP/RTCP (Real Time Protocol/Real Time Control Protocol [108]), RTSP (Real Time Streaming Protocol [107]) or SIP (Session Initiation Protocol [58]), which measure key network parameters, such as bandwidth usage, packet loss rate and round-trip delays to control the load submitted to the network.

The main drawback of these technologies concerning their use for industrial communications is the latency introduced. For example, there are video algorithms [110] that use memory buffers between the producer and the consumer to smooth out the bit rate variations. The estimation of the required bandwidth and of the amount of buffers can be done offline, for stored video, or based on a number of images buffered before their transmission, for live (non interactive) video streams. The quality given by this smoothing mechanisms is however highly dependent on the application requirements. This latency problem is addressed by the low-delay Rate Control algorithms [106, 129], such as those used in TMN8 [11], which achieve a high level of performance for the applications for which they have been designed, mainly videophone and video-conference. However, these algorithms are based on the use of only one I-frame and a following sequence of P-frames, an approach that can be used in soft real-time applications like videophone and video-conference, but not in MES applications. For example, within an industrial control process based on vision, the appearance of a new object in the scene will either generate a traffic peak, if the compression is kept unchanged, or a strong quality reduction if the compression is changed to keep the bandwidth utilization stable. Any of these situations is susceptible to introduce errors into the system, thus making the use of this class of protocols unsuited for MES applications.

From the above discussion it can be concluded that the solutions developed for generic video transmission, despite efficient and flexible, do not fill in the requirements of MES, towards an efficient fit of a VBR source within a CBR channel. Taking a conservative approach, it is possible to reserve a channel with a capacity equal to the maximum bandwidth of the multimedia source. While taking this approach guarantees that no frames are lost, the fact is that the bandwidth generated by multimedia sources typically exhibit

high variance, leading to a potentially significant bandwidth waste. An alternative approach to overcome this inefficiency problem would be reserving a channel with a capacity equal to the average bandwidth necessary. Despite more efficient, from the bandwidth point of view, this approach can lead to additional delays or even frame losses, depending on the existence and size of buffers at the source nodes, whenever the instantaneous bandwidth exceeds the average value. At this point it should be stressed that many applications comprise the transmission of several multimedia streams, thus multiplying the impact of these sources of inefficiency.

The difficulty in fitting VBR sources in CBR channels motivated the work presented in this section that also illustrates the capabilities of the FTT-SE framework on supporting dynamic QoS management. The dynamic QoS management features present in FTT-SE are used to dynamically adapt the bandwidth of the real-time communication channels. Such integration allows adjusting the QoS of each stream given as inputs the relative importance of a media stream, the currently allocated bandwidth and the compression level of each multimedia source, as well as the global network utilization. The goal is to provide, at every instant, the best possible QoS to each stream. The admission control and scheduling faculties of the FTT-SE protocol allow carrying out seamless online transitions without losing the real-time guarantees, thus being suitable for MES applications.

6.2.2 System Architecture

This case-study addresses a generic monitoring industrial application, where p multimedia sources, also called *producers*, send a set $\mathbb{M} \equiv \{M_i, i = 1..p\}$ of video streams to c multimedia sinks, called *consumers*, via a local area network executing the FTT-SE protocol (Figure 6.10). Besides the video streams, the network may also support other traffic sources, potentially with stringent real-time requirements, e.g. related with real-time control, as well as non-real-time, e.g. configuration or even general-purpose Internet access. This communication scenario is effectively supported by the FTT-SE protocol, which provides dynamic QoS management while guaranteeing the real-time performance. Moreover, the protocol enforces mutual isolation between traffic classes and thus Therefore, these additional traffic classes are not considered throughout this work.

Several producers may reside in one single node but, for the sake of simplicity, in the scope of this work a single producer per transmitting node is considered. Figure 6.11 illustrates how the application uses the protocol. The QoS manager within the FTT-SE master node, concentrates the requests from the system nodes, holding the network requirements for the contracted streams, including operational and QoS aspects. It receives channel change requests from different system nodes and assigns the bandwidth to each of the streams. Then, each node holds a QoS sublayer that addresses the QoS

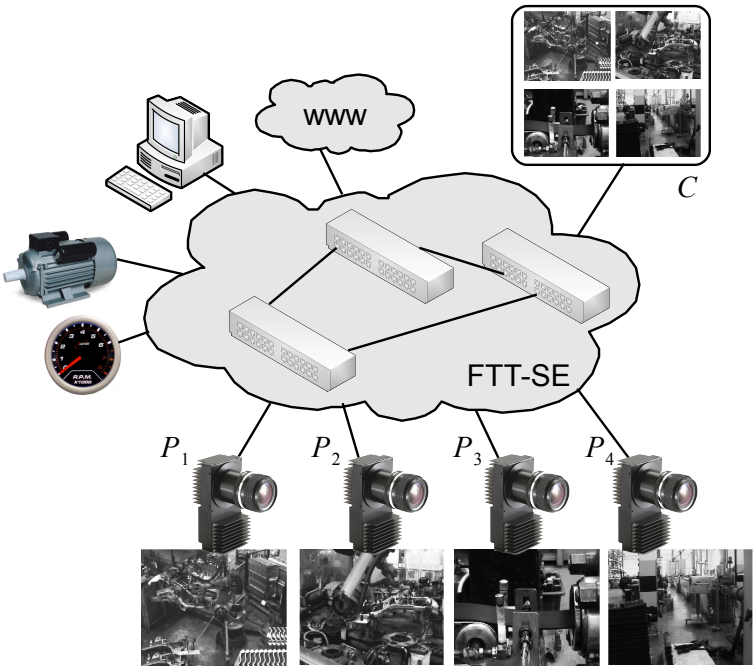


Figure 6.10: System architecture.

management requirements in terms of a video-specific application.

The QoS sublayer is included to handle the QoS contracts of each stream, while conserving their run-time properties. As depicted in Figure 6.11, this layer acts in between the application and the FTT-SE protocol, adapting the instantaneous requirements of each video stream (VBR source) to the FTT-SE communication channels (CBR). This mapping is straightforward with $Pr_i^{FTT} = Pr_i^{APP}$, $\mathbb{T}_i^{FTT} = \mathbb{T}_i^{APP}$ and $C_i^l = B_i^l$. The upper bound of the transmission buffer size range C_i^u is determined in each QoS negotiation, expressing the desired buffer size from the QoS sublayer perspective at that moment.

At the application level, the QoS model considers each stream being characterized by a normalized relative priority Pr_i^{APP} such that $\sum_{\forall_i} Pr_i = 1$), a range of allowed quantification factors that imply different compression levels $\mathbb{Q}_i^{APP} \equiv \left[q_i^l, q_i^u\right]$, a range of possible frame sizes after compression $\mathbb{B}_i^{APP} \equiv \left[B_i^l, B_i^u\right]$ and a set of possible inter-frame intervals corresponding to the n_i allowed frame rates $\mathbb{T}_i^{APP} \equiv \{T_i^j, j = 1..n_i\}$, as follows:

$$M_i^{APP} = \{Pr_i, \mathbb{Q}_i, \mathbb{B}_i, \mathbb{T}_i\}^{APP} \tag{6.4}$$

As described back in Section 5.3, the FTT streams are characterized by a priority that reflects the relative channel importance Pr_i^{FTT}, a range of

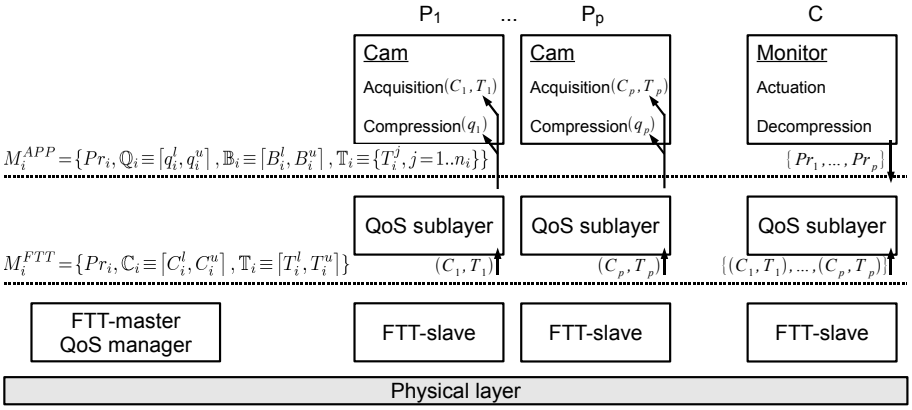


Figure 6.11: QoS management model.

admissible transmission buffer sizes $\mathbb{C}_i^{FTT} \equiv \left[C_i^l, C_i^u\right]$ and a range of possible transmission periods $\mathbb{T}_i^{FTT} \equiv \left[T_i^l, T_i^u\right]$, as follows:

$$M_i^{FTT} = \{Pr_i, \mathbb{C}_i, \mathbb{T}_i\}^{FTT} \tag{6.5}$$

The output of the QoS manager is the actual bandwidth u_i assigned to each channel at each instant and materialized as a (C_i, T_i) duplet that is communicated back to the QoS sublayer and application. Note that $u_i = \frac{C_i}{T_i}$. Then each application node is responsible for satisfying this constant resource allocation, keeping the bandwidth R_i below u_i, i.e., $R_i \leq C_i/T_i$. The QoS sublayer dynamically adjusts the video properties in order to fit the source stream in the granted channel, eventually adapting the compression level or even discarding frames. Also, whenever appropriate, the QoS sublayer re-negotiates the bandwidth with the FTT-SE QoS manager, updating long-lasting variations on the streaming requirements. At the FTT-SE level, the QoS manager provides constant bandwidth channels with a budget C_i and a period T_i. Although, the channels properties are constant, they are not permanent and may change when the network adjusts to a new workload scenario. As mentioned (Equation 6.5), that variation is delimited and parameterized by the application. The QoS manager is thus a major piece on this architecture that ultimately adapts the VBR source to the CBR channels and realizes the necessary adjustments to the contracted requirements, avoiding over-estimated resource provisioning.

Each multimedia stream i is composed by a succession of streams, associated to a producer. The j^{th} frame is compressed with the quantification level q_i^j, set by the QoS adaptation layer, resulting on a payload with size f_i^j. Both the acquisition and compression parameters are set by the QoS sublayer, reflecting the QoS management in a controlled and predictable way. These parameters (q_i^j, B_i and T_i) are provided to the application in the constraints set when registering the stream. The QoS adaptation must be such

that the compressed streams fit within the allocated resources in terms of bandwidth and bucket size, as follows:

$$f_i^j \leq B_i \quad \wedge \quad \frac{f_i}{T_i} = R_i \leq u_i \qquad ,\forall_i, \forall_j \tag{6.6}$$

6.2.3 QoS Management

To understand the QoS management and how it actually manages the streams properties towards a better fit to the CBR channel and better serviced quality, this subsection describes the video stream variation model, its content scaling and finally, the metrics to assess the video quality throughput.

R(q) model

Characterizing the video stream is important to accurately address the fitness of a VBR source through the CBR channel. On this regard, the $R(q)$ model provides a parameterized model of the stream evolution in terms of load. With such model it is possible to obtain the compression q_i^j value that at a given moment provides the best quality on the currently available bitrate. Knowing T_i^j it becomes possible to adapt q to the w assigned in each moment to satisfy the bandwidth restriction in Equation 6.6. One of the models with better results [48] defines $R(q)$ as:

$$R(q) = \alpha + \frac{\beta}{\overline{q}^{\lambda}} \tag{6.7}$$

where α and β are parameters of a curve in which λ regulates the curvature. The model was developed for MPEG, where q, the quantification factor, varies from 1 (lowest compression) to 31 (highest compression). To apply the model to a JPEG stream, $\overline{q} = 100 - q$ is the compression level, varying symmetrically with respect to the quantification factor. For each video frame, a different curve is obtained relating the compression factor and the bandwidth necessary to transmit it. Hence, the j^{th} frame holds the parameters $(\alpha_i^j, \beta_i^j, \lambda_i^j)$. In order to obtain the optimal q_i value for a specific R_i output, the curve has to be estimated for each frame. However, obtaining this curve may involve iterating the compression algorithm for several points, increasing the computation latency and eventually turning impractical its applicability for online calculations.

On a monitoring application with fixed cameras, as hereby addressed, it can be assumed that frames are acquired sequentially on a scenario without much abruptly, i.e, consecutive frames have a strong similarity. Therefore, it can be assumed for a stream i that $\forall_j \alpha_i^j = \alpha_i^j$ and $\forall_j \lambda_i^j = \lambda_i^j$, reducing the model to $(\alpha_i, \beta_i^j, \lambda_i)$. Equation 6.7 now establishes the relationship between β_i^j and q_i^j for a given bitrate $R_i \leq u_i$. It is this relationship that yields the adaptation of the quantification factor q.

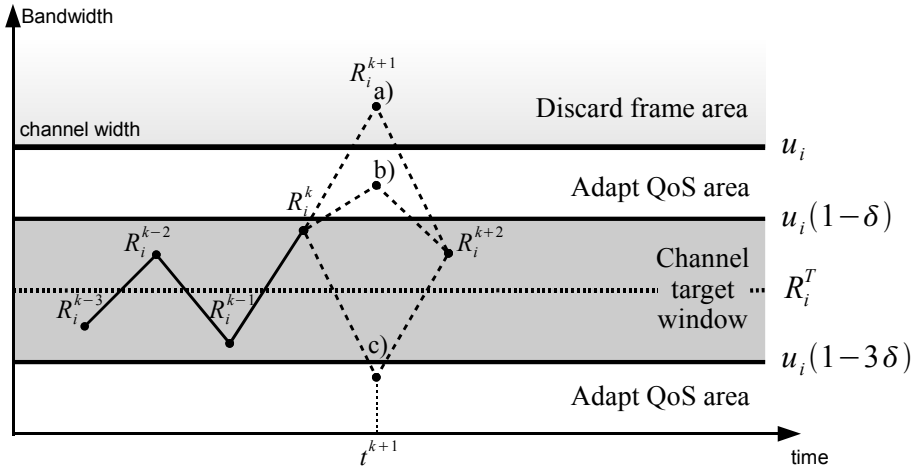


Figure 6.12: q_i adaptation.

Based on this model, the QoS manager is able to tune each frame compression level in order to better handle the channel given capacity u_i. The model allows deriving at instance k an estimate of the quantification level for the next frame q_i^{k+1} that will generate a bandwidth R_i^{k+1} within a *channel target window*. As long as the frame bandwidth falls inside such window, q is kept and its adaptation is not invoked, thus reducing the frequency of adaptations and saving overhead. This window is controlled by a parameter, δ, resulting in $[u_i(1-3\delta), u_i(1-\delta)] \equiv [R_i^T \pm u_i\delta]$. The nominal coding bitrate is thus defined as $R_i^T = u_i(1-2\delta)$, as depicted in Figure 6.12. The value of q_i^{k+1} is then defined as a function of the current frame bandwidth R_i^k, the current channel bandwidth u_i and δ. The δ factor is a pre-defined relative fraction of the channels bandwidth, equal for all channels, that sets a compromise between higher efficiency in channel bandwidth utilization (lower δ values) and lower frequency of compression level adaptation invocations (higher δ values).

However, when estimating the best q_i^{k+1}, it is not guaranteed that the next frame will fall inside to the channel target window.

When R_i^k falls out that interval, either the bandwidth used is too high, with a potential to originate frame-drops, or too low leading to an under utilization of the channel bandwidth. Figure 6.12 illustrates the three possible scenarios at time t^{k+1} in which the resulting frame bandwidth falls outside the channel target window. In scenario a) the generated frame bandwidth R_i^{k+1} exceeds the current channel width u_i causing a frame-drop, in b) it is within the channel width but over the target window, while in c) it is below the target window leading to an under utilization of the channel bandwidth. In all three scenarios the adaptation is invoked to compute an estimate of the quantification level q_i^{k+2} that will generate an R_i^{k+2} that falls within the

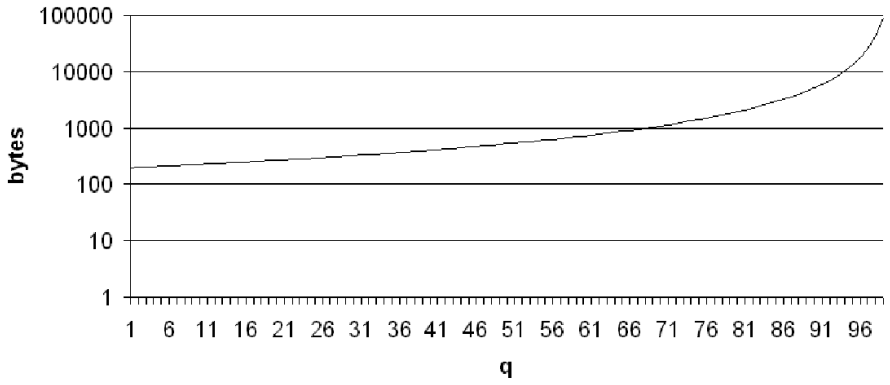


Figure 6.13: Error in frame size caused by $\Delta q_e = 1$ for different values of q.

channel target window.

To bring coding the bitrate back inside the operating window we tune the quantification level for the following frame. In fact, this is the most effective way to control the output bitrate, hence varying the stream QoS.

When computing the following frame $(k+1)$, if the current frame (k) is inside the target window, the same quantification factor is used, $q_i^{k+1} = q_i^k$. Otherwise, q_i^{k+1} is adjusted based on the desired variation of $R(q)$, i.e., based on $\Delta R_i^{k,k+1}$ needed to bring the current frame bitrate to the nominal value. The calculation of the new q_i^{k+1} factor is carried out using the following sequence of operations. Firstly, we compute β^{k+1} as in Equation 6.8.

$$\beta^{k+1} = \Delta R^{k,k+1} q^{\lambda} + \beta^k \tag{6.8}$$

With this value we can compute q_i^{k+1} to be use in next frame, using the $R(q)$ model as in Equation 6.9.

$$q_i^{k+1} = \left(\frac{R_i^T - \alpha}{\beta^{k+1}} \right)^{1/-\lambda} \tag{6.9}$$

However, this estimation becomes more inaccurate as the quantization value becomes higher. Let the difference between the estimated q_i^{k+1} and the actual q_i that generates a perfect match with the frame nominal bitrate R_i^T be Δq_e. Figure 6.13 plots the deviation between the predicted and the effective frame sizes for a fixed $\Delta q_e = 1$, while increasing the compression factor. The higher the compression, the higher the deviation observed. For example, for $q < 65$ and $\Delta q_e = 1$ the error in frame size is below 1000 bytes, which represents about 200 kbps when using a frame period of T = 40ms. When $q > 90$ the error may reach 1 Mbps, in the same conditions.

Therefore, it is advisable to keep the quantification factor bounded, as denoted in the application model with an upper and lower limits.

Content scaling

To attenuate the problem of adjusting the QoS near the compression boundaries, we propose the extension of the $R(q)$ model to include the management of the stream properties such as the acquisition period (T) and the transmission budget C. However, such properties require a more complex and coordinated management beyond the simple compression adjustment. Those properties are intimately related to the channel provided by the network. If for any reason the $R(q)$ model is not able to obtain a q value within the allowable range, then the application must renegotiate its requirements with the network, ultimately increasing or decreasing the demands for bandwidth. On the other hand, renegotiating the channel properties involves a time and resource usage penalty and thus cannot be issued on a frame-by-frame basis. Thus, the frequency by which QoS changes are requested should be limited.

Additionally, in many applications QoS changes have a negative impact on the perceived quality, and thus its occurrence should be as sparse as possible.

Therefore, to reduce the number of QoS changes there is a hierarchy of procedures to be followed. Algorithm 6.2.1 summarizes the hierarchy of procedures involved in the QoS adaptation process.

Algorithm 6.2.1: $q'_i = qosAdaptation(R_i, q_i)$

---

comment: computes q_i for next frame

$q'_i \leftarrow q_i$
if $R_i > u_i(1-\delta)$
  then $\begin{cases} q'_i \leftarrow R^{-1}(R_i^T) \\ over\_cnt \leftarrow over\_cnt + 1 \end{cases}$
  else $over\_cnt \leftarrow 0$
if $R_i < u_i(3-\delta)$
  then $\begin{cases} q'_i \leftarrow R^{-1}(R_i^T) \\ under\_cnt \leftarrow under\_cnt + 1 \end{cases}$
  else $under\_cnt \leftarrow 0$
if q'_i **not in**$[q_i^l, q_i^u]$
  then $\begin{cases} q'_i \leftarrow saturation(q_i^l, q_i^u) \\ \textbf{if } over\_cnt > QCT \textbf{ or } under\_cnt > QCT \\ \quad \textbf{then } qosRenegotiation() \end{cases}$
return (q'_i)

Firstly, the QoS adaptation layer autonomously adjusts the compression factor, in a frame-by-frame basis, trying to keep the stream bandwidth

within the target window. Whenever changing the compression factor is not enough to keep the stream bandwidth within the target window, i.e., q reaches the upper or lower limit while the stream bandwidth is above/below the target window, a renegotiation has to be issued. To prevent excessive requests two event counters are implemented, namely the $over\_frames$ and $under\_frames$ counters, which count the number of consecutive frames that fall above and below the target window. A channel renegotiation is triggered as soon as any of these counters exceeds the *Quality Change Threshold* QCT. This threshold is a system parameter that controls the frequency of autonomous channel renegotiations. The higher this parameter is, the longer it will take for the system to trigger a channel renegotiation. Such procedure relegates short term conditions to be handled locally by the QoS adaptation layer, while long-lasting situations, resulting for example from infrastructural changes in the image or explicit QoS changes by the application, eventually leading to a global QoS renegotiation.

Negotiating

On the FTT-SE interface, the requirements for bandwidth are expressed by the QoS requirements defined for each CBR channel, including the flexibility profile of each. Those requirements are set as described in Equation 6.5, with the transmission periodicity values that are acceptable to be used by the application (acquisition frame-rate) and the variation range for the transmission budget. Modifying those parameters ultimately leads to changes on the bandwidth delivered to each of the contracted channels (u_i).

When re-negotiating channels, the QoS sublayer starts by estimating the new desired transmission buffer sizes C_i^u, to be reported to the FTT-SE QoS manager. These are determined in a way to fulfill the maximum bandwidth requirement of each stream at each instant, i.e., using the $R(q)$ model with the minimum quantification levels, considering the current inter-frame interval T_i. Such values are then capped to the application specified upper bound on the frame size B_i^u. The following expression shows how these values are computed.

$$C_i^u = \min\left(B_i^u, \frac{R(q_i^l)}{(1-2\delta)} \times T_i\right) \tag{6.10}$$

Once the desired channel bandwidths are determined the QoS sublayer hands them over to the FTT QoS manager through the FTT QoS interface. The first operation of the QoS manager is to compute, for each channel, the desired bandwidth ($u_i^d = C_i^u/T_i$) as well as the minimum bandwidth ($u_i^{min} = C_i^l/max_j T_i^j$). The minimum bandwidth is always checked upon addition of a new stream as part of an admission control that is embedded in the QoS management. In fact, a stream can only be accepted if all the minimum channel bandwidths, including its own, can be granted.

Then, the FTT-SE QoS manager executes a bandwidth distribution procedure, as described in Chapter 5. Different policies can be seamlessly used within the QoS manager inside the FTT Master, without requiring any further changes in the rest of the system. Algorithm 6.2.2 shows a fixed priorities-based policy, implemented on this industrial multimedia application.

Algorithm 6.2.2: $\mathbb{W} = uDistribution(\mathbb{M}, U^S)$

---

comment: distributes the system bandwidth capacity

$U_{spare} \leftarrow U^S - \sum_{\forall i} u_i^{min}$
for each $M_i \in \mathbb{M}$, *sorted by* Pr_i

$$\textbf{do} \begin{cases} \textbf{if } (u_i^d - u_i^{min}) < U_{spare} \\ \quad \textbf{then } u_i' \leftarrow (u_i^d - u_i^{min}) \\ \quad \textbf{else } u_i' \leftarrow U_{spare} \\ U_{spare} \leftarrow U_{spare} - u_i' \\ u_i \leftarrow u_i^{min} + u_i' \end{cases}$$

return $(\mathbb{W} = \{u_1, \ldots, u_n\})$

It starts from the minimum bandwidth requirements (u_i^{min}) and distributes the remaining bandwidth among the channels following a strict priority order according to the Pr_i parameter and until there is no more system bandwidth to assign. Channels registered with the same priority level are treated equally, receiving an equal amount of bandwidth and being updated simultaneously in the course of any negotiation. The system bandwidth U^S is the admissible load per link, for example, as dictated by a schedulability criterion that assures that the deadlines of all streams are met.

In most cases there will not be enough system bandwidth to satisfy all channel requests. In such circumstance, some channels will get the requested bandwidth, others will just get their minimum requirement bandwidth while others will get an intermediate value of bandwidth between the previous two cases.

After completing the bandwidth allocation procedure, the QoS manager re-maps the allocated bandwidth into actual (C_i, T_i) parameters. As mentioned before, the stream period (T_i) is specified as a restricted set of discrete values that matches the video source frame-rate, while the frame size parameter (C_i) can be any natural number comprised within the specified range. The lower bound of the frame size C_i^l is kept immutable during the service lifetime, while the upper bound C_i^u is sporadically renewed (renegotiated) by the video encoder to fit the estimated image frame size. The bandwidth mapping is not biunivocal and thus several approaches can be followed. Algorithm 6.2.3 illustrates the technique employed in this work, which attempts

to maximize the value of C_i.

Algorithm 6.2.3: $\{(C_i, T_i)\forall_i\} = uCTmapping(\mathbb{W}, \mathbb{M})$

for each $M_i \in \mathbb{M}$

do
$$\begin{cases} T_i = \min\left\{T_i^j, \forall_{j=1..n_i} : T_i^j \geq C_i^u/u_i\right\} \\ \textbf{if } T_i = \emptyset \\ \quad \textbf{then} \begin{cases} T_i = \max\left\{T_i^j, \forall_{j=1..n_i}\right\} \\ C_i = u_i \times T_i \end{cases} \end{cases}$$

This is done by taking the highest value for C_i within the defined range $[C_i^l, C_i^u]$ and then using the period T_i that allows the closest under approximation to u_i. If this cannot be achieved with any of the available discrete periods then the longest of such periods is used (T_i^{max}) and C_i is recomputed as $C_i = u_i \times T_i^{max}$.

Measuring the Quality of Service

The mean-square-error (MSE) and peak-signal-to-noise ratio (PSNR) are the quality metrics most frequently used to evaluate the performance of codecs and video transmission systems. However, their capacity to match up the perceived degradation as well as the human vision factors is poor. There are numerous attempts to include characteristics of human visual systems (HVS) in objective quality assessment metrics. These attempts try to get a numerical method with a good correlation to subjective methods. In the subjective methods, test sequences are presented to instructed non expert observers, in a controlled environment, which perform an evaluation according to predefined quality scales (ITU-R BT.500), obtaining the MOS (Mean Opinion Score) value.

The method proposed by Z. Wang [135] uses the structural distortion measurement instead of the error, since the HVS is highly specialized in extracting structural information, and not in extracting the errors. Being f the original image, and g the distorted one, the image quality index (QI) can be calculated as:

$$QI = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \frac{2\hat{f}\hat{g}}{\hat{f}^2 + \hat{g}^2} \frac{2\sigma_f \sigma_g}{\sigma_f^2 \sigma_g^2} \tag{6.11}$$

where $\hat{f}$ and $\hat{g}$ are the intensity mean, σ_f and σ_g its variance and σ_{fg} the covariance. The QI index assumes values in the range [-1,1], being $QI = 1$ when the both images are identical.

However, this QI metric does not entirely reflect the influence of the priority on the QoS obtained nor accounts for the correct use of the available

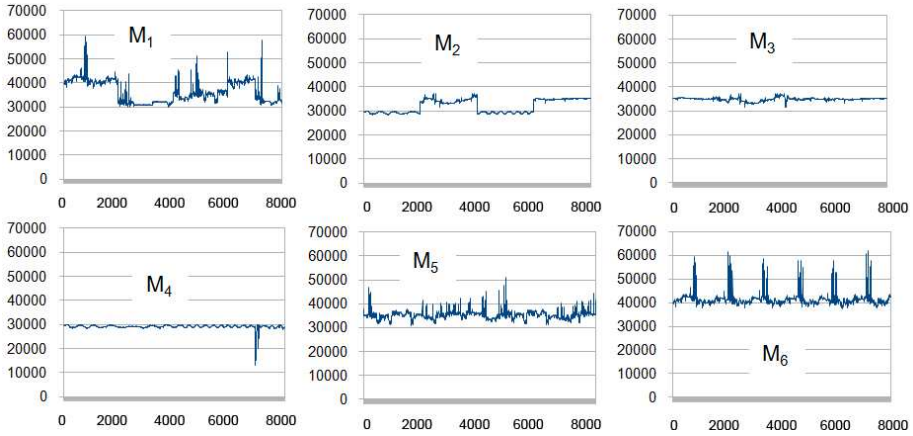


Figure 6.14: Frame size evolution in time ($q = 55$).

bandwidth, which is a factor of paramount importance in industrial networks. For these reasons, we propose a new metric that also accounts for the efficient use of the channel bandwidth by favoring the streams that present lower wasted bandwidth. The formula is the following, where nf is the number of frames and Wb_i the wasted bandwidth in stream M_i.

$$QoS'_i = \frac{Pr_i}{1 + Wb_i} \sum_{k=1}^{nf} QI_i^k \tag{6.12}$$

The global QoS' is also computed as the average of the QoS'_i parameters. In the following experiments, we will use QI and PSNR to characterize the quality of each individual stream and the QoS' metric for assessing the aggregated QoS of each experiment.

6.2.4 Experimental results

In order to assess the performance of the proposed multidimensional content scaling technique, a set of experiments was conducted, also illustrating the dynamic QoS management properties of the FTT-SE framework.

The streams used in the experiments address industrial scenarios. Figure 6.14 illustrates, for those streams, the variability of the size required by each frame, when the streams are compressed with a constant $q = 55$, illustrating their dynamics, complexity and requirements. Stream M_3 and M_4 are representative of an industrial surveillance application, showing nearly constant bandwidth requirements. Streams M_5 and M_6 present smooth variations alternated with strong peaks, representing harsh scenarios with sudden changes in the environment (e.g. sparks from an industrial welding machine). Finally stream M_1 and stream M_2 result of composing alternated

	M_1	M_2	M_3	M_4	M_5	M_6
d1-d4						
q_i^l	20	40	40	20	30	15
q_i^u	70	70	70	70	50	55
$T_i^l(ms)$	40	40	40	40	40	40
$T_i^u(ms)$	160	120	160	120	120	120
$B_i^l(B)$	30k	30k	30k	30k	20k	25k
$B_i^u(B)$	50k	50k	50k	50k	60k	55k
Pr_i	0.166	0.166	0.166	0.166	0.166	0.166
d5						
q_i^l	30	30	30	20	30	20
q_i^u	70	50	50	40	70	70
$T_i^l(ms)$	40	80	80	80	40	40
$T_i^u(ms)$	80	160	160	200	80	120
$B_i^l(B)$	30k	20k	20k	30k	20k	25k
$B_i^u(B)$	60k	60k	60k	60k	70k	65k
Pr_i	0.25	0.10	0.10	0.10	0.25	0.20

Table 6.2: Stream properties for dynamic experiments.

frames (multiplexing), from streams M_5 and M_6 for the first and M_3 and M_4 for the second.

A total of five different dynamic experiments were carried out. The first group of experiments, denoted by d1 to d4, were designed to assess the influence of the δ and QCT parameters. Experiment d5 illustrates the impact of the QoS priority. Table 6.2 depicts the QoS parameters used for each stream on these experiments. For experiments d1 to d4 the tuple (δ,QCT) takes the values (0.1,1), (0.05,1), (0.1,2) and (0.05,2), respectively, while the stream priority is kept equal for all streams. Conversely, in experiment d5 the tuple (δ,QCT) is equal to experiment d2, while the streams receive different priorities.

To establish a baseline for the performance gains of having dynamic adjustments of the streams, a set of static experiments with fixed q, C and T was also carried out (s1-s4). Experiments s1 to s3 use C = 44KB and T = 80ms, resulting in a total bandwidth of 26.4 Mbps, with compression factors set to 50, 55 and 60, respectively. In experiment s4, C was set at 22KB and T to 40ms, yielding a similar bandwidth, while the compression factor q was set to 15.

Tables 6.3 and 6.4 report the experimental results obtained. For each experiment and video sequence the tables show the number of dropped frames (DrF), the wasted bandwidth (Wb in Mbps) and the quality according with the $PSNR$ and QI criteria.

d1 (0.1,1)	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	9	0	0	0	4	16	4.83
Wb	0.89	0.90	1.0	0.86	0.78	0.98	0.90
$PSNR$	32.5	34.7	34.3	35.2	32.0	31.9	33.4
QI	0.88	0.91	0.92	0.90	0.87	0.89	0.89
d2 (0.05,1)	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	15	2	0	8	30	17	9.5
Wb	0.4	0.41	0.47	0.39	0.34	0.42	0.40
$PSNR$	32.6	34.7	34.6	35.2	32.9	32.9	33.81
QI	0.88	0.91	0.92	0.90	0.87	0.89	0.89
d3 (0.1,2)	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	6	0	0	0	5	15	4.3
Wb	0.88	0.90	1.0	0.87	0.78	0.98	0.90
$PSNR$	32.5	34.7	34.3	35.2	32.0	31.8	33.41
QI	0.88	0.91	0.92	0.90	0.87	0.89	0.91
d4 (0.05,2)	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	24	2	0	4	36	17	13.83
Wb	0.41	0.41	0.48	0.40	0.34	0.43	0.41
$PSNR$	32.5	34.7	34.6	35.2	32.0	31.7	33.45
QI	0.88	0.91	0.92	0.90	0.87	0.89	0.89
d5 (0.05,1)	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	24	0	2	11	27	24	14.6
Wb	0.47	0.30	0.35	0.25	0.45	0.55	0.39
$PSNR$	33.0	33.6	33.4	33.7	32.7	32.6	33.16
QI	0.89	0.90	0.89	0.87	0.87	0.90	0.88

Table 6.3: Results with dynamic scenarios.

Table 6.3 shows that the parameter δ has a noticeable effect on the system behavior. Reducing δ causes a consistent reduction on the wasted bandwidth, as expected. However, this reduction is achieved at the expense of an increasing number of dropped frames. This effect is particularly visible in streams that exhibit higher dynamics (e.g. M_5), while for streams with more stable requirements the impact is minor or even null (e.g. M_3). The impact on the number of dropped frames is not, however, always reflected in the per frame quality metrics ($PSNR$ and QI). The justification for this fact is that making δ narrower increases the number of dropped frames but, at the same time, also raises the target bandwidth window, leading to a higher efficiency in using the channel width, allowing the QOS adaptation layer to use lower quantification values. In most of the streams the increase in quality compensates the higher number of dropped frames, with the final difference not being statistically significant. However, for streams with lower dynam-

s1	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	7	0	0	0	2	10	3.16
Wb	1.0	1.38	1.1	1.66	1.09	0.55	1.16
$PSNR$	29.8	32.7	31.16	33.48	29.12	29.25	30.91
QI	0.82	0.88	0.89	0.88	0.8	0.85	0.83
s2	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	47	0	0	0	16	87	50
Wb	0.78	1.2	0.91	1.48	0.88	0.3	0.93
$PSNR$	32.66	34.37	33.87	34.78	32.38	32.0	33.34
QI	0.88	0.90	0.91	0.90	0.87	0.89	0.89
s3	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	1871	2	2	2	60	60	332.8
Wb	1.54	0.93	0.64	1.23	0.62	0.62	0.93
$PSNR$	25.24	34.6	34.12	34.98	32.36	32.05	32.22
QI	0.68	0.90	0.91	0.90	0.87	0.89	0.85
s4	M_1	M_2	M_3	M_4	M_5	M_6	*mean*
DrF	11	0	0	0	1	41	8.83
Wb	1.04	1.37	1.06	1.68	1.18	0.64	1.12
$PSNR$	30.55	31.23	30.66	31.88	30.82	30.00	30.85
QI	0.82	0.83	0.85	0.84	0.85	0.83	0.85

Table 6.4: Results with static scenarios.

ics, such as M_3, reducing δ actually improves the $PSNR$ metrics, since the sequence is not affected by dropped frames.

The QCT parameter controls the frequency of QoS renegotiation requests. Its impact on the QoS metrics depends strongly on the characteristics of each stream. When the streams have narrow and strong bandwidth peaks, higher QCT values increase the QoS renegotiation latency, potentially leading to a quality deterioration. This effect can be observed in Table 6.3, where for experiments d2 and d4 an increase in the QCT from 1 to 2, leads streams M_1 and M_5 to experience a significant increase in the number of dropped frames and, together with M_6, a deterioration in the $PSNR$ figure. In the other streams, that either have bandwidth requirements that stay constant or nearly constant during relatively long periods of time, higher QCT values do not lead to a significant number of dropped frames and, furthermore, help filtering spurious changes that could, otherwise, lead to unnecessary QoS renegotiations, as in the case of M_4 in experiments d2 and d4.

One aspect that should be highlighted is the low sensitivity of the system to particular values of δ and QCT. In fact, the $PSNR$ and QI metrics do not change significantly with any of these parameters, thus facilitating system

	s1	**s2**	**s3**	**s4**	
QoS'	0.41	0.48	0.46	0.40	
	d1	**d2**	**d3**	**d4**	**d5**
QoS'	0.47	0.64	0.47	0.63	0.67

Table 6.5: QoS' results.

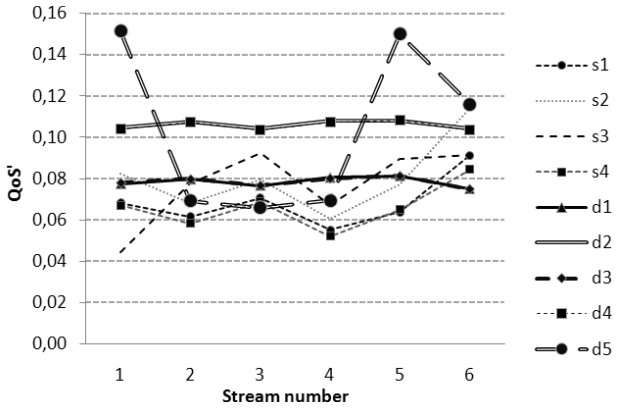


Figure 6.15: Contribution of each stream to QoS'.

set-up.

Comparing Tables 6.3 and 6.4 clearly shows that the dynamic approach leads to significant improvements in all key aspects. The number of dropped frames is strongly reduced, mainly in the streams with higher dynamics (e.g. M_1 and M_5). The quality metrics (PSNR and QI) are also consistently similar or better. It should be remarked that these results are achieved with better bandwidth utilization. The exception is for s2, which, with a constant $q = 55$, attains similar quality levels, with a low number of dropped frames. In fact it is possible in some cases to find the best static q for each stream. However, this procedure has to be done offline, and thus it is not suitable for MES applications.

Table 6.5 presents the QoS' values for each experiment. The first conclusion that can be withdrawn is that, for properly selected δ parameters, the QoS attained with the dynamic approach can be significantly higher than with the static approach, e.g., d2 and d4 versus s2 and s4. Considering the meaning of this metric, one can conclude that higher quality levels can be attained both by allocating more bandwidth to the streams that can make better use of it as well as by reducing the wasted bandwidth. The impact of the wasted bandwidth in this metric can be also observed in the significant difference, around 35%, between experiments d1 and d2, and d3 and d4.

Experiment d5 aims at illustrating the system behavior when the assigned

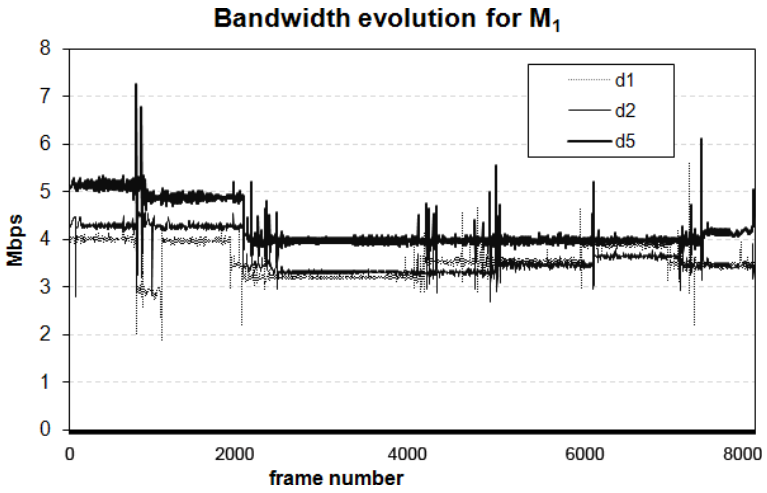


Figure 6.16: Bandwidth evolution of stream M_1.

priorities are not uniform. The Pr values used in d5 imply a bandwidth distribution where streams M_1, M_5 and M_6 obtain more resources in detriment of streams M_2, M_3 and M_4, as can be see in Figure 6.15. This matches the requirements of many applications in which some streams have a higher impact on the global system performance and thus should be favored. Figure 6.16 shows the bandwidth used by stream M_1 in experiments d1, d2 and d5. It can be observed that in experiment d5 the scheduler assigns more bandwidth to stream M_1 than in experiments d1 and d2. This observation is particularly clear when comparing experiments d2 and d5, which have an equivalent parameterization except for the priority. Observing Table 6.3, it is possible to conclude that the higher priority streams have a gain between 0.5 and 1 dB, at expenses of decrease between 1 and 1.2 dB in the lower priority ones. Thus, the priority mechanism proves its effectiveness in differentiating the streams, providing more resources to the ones that have higher impact in the global system performance.

6.2.5 Summary

Using multimedia streams in real-time applications requires appropriate support from the underlying network. A common used technique to address this issue is allocating CBR channels to different streams, which favors temporal isolation. However, most multimedia streams are naturally coded with variable bit rate (VBR). In order to fit the VBR source within the CBR channel, either it is allowed degradation on the streaming quality, if the channel is designed for the average requirements, or a significant bandwidth waste, if the channel is designed to fit the worst-case requirements. This section presented a multidimensional dynamic QoS adaptation mechanism

that allows dynamically changing the channel bandwidth according to the effective stream needs and overall available bandwidth. The capacity of the CBR channels is dynamically adjusted, from time-to-time, reflecting the instantaneous requirements of each stream.

This adaptation mechanism is extensively assessed, with its performance being compared against a corresponding situation with static CBR channels, using a set of stored video sequences from industrial environments. The performance level is assessed with a QoS metric that includes both the image quality, the stream QoS priorities and the capacity of the system to reduce the wasted bandwidth. The results obtained show a consistent superiority of the dynamic approach over the static one, specially when streams of different priorities are in place. Moreover, the adaptation is carried out with reserved channels, thus maintaining the temporal isolation feature among the streams and other real-time traffic, thus being suitable for integration in complex MES systems, integrating real-time sources of diverse natures, e.g. closed-loop control.

This industrial application scenario illustrates the benefits from using a dynamically adaptable system, based on sing the framework proposed in this thesis (FTT-SE) along with its dynamic QoS management capabilities. It provides an example of the integration of these features with the application and its dynamics.

6.3 Server-SE

In Chapter 2 we have seen the major benefits of an Ethernet switch-based architecture to support real-time communications. Several limitations remain, though, and thus, several protocols have been proposed to provide real-time services over such architectures, including FTT-SE, proposed in this thesis. However, in spite of the particular mechanisms employed, they share a common difficulty in the efficient handling of real-time messages with different arrival patterns, such as periodic and aperiodic, by treating them in different ways. The approach proposed in this section is different in the sense that it aims at providing a uniform model, through which messages are scheduled in an integrated way, with no distinction between periodic and aperiodic.

The core of this approach is the integration of the FTT-SE (Section 3) and the Server-CAN [95] protocols. The FTT-SE architecture facilitates enforcing full control over streams of messages, due to the master/slave operation, no matter of their corresponding arrival patterns. Furthermore, the centralization of the scheduling decisions on the master node also facilitates the implementation of arbitrary server policies as well as their hierarchical composition. This property allows complex applications to be decomposed into sub-applications, each one requiring a share of the bandwidth. Although common in CPU scheduling, this level of flexibility, permitting the implementation of arbitrary server mechanisms as well as their hierarchical composition, is new in the context of RTE networks. Summarizing, this approach for SE systems (1) is free of queues overflows, (2) supports advanced traffic scheduling policies and (3) can enforce real-time guarantees even in the presence of aperiodic communication and/or time-domain faults, e.g., babbling idiots.

This section is based on work conducted in cooperation with Thomas Nolte from Mälardalen University in Sweden [87] and Nuno Figueiredo [88], while he was pursuing his Master thesis. The following section provides an overview of server-based CPU scheduling as well as server-based traffic scheduling techniques. Then, Section 6.3.2 proposes using the FTT-SE protocol to allow a local management of all servers, facilitating their online creation, deletion, adaptation and composition. It advocates that such a centralized management of the servers provides the required support for open distributed real-time systems as well as for dynamic QoS management. Moreover, the proposed approach also allows any CPU-oriented server-based scheduling policy to be implemented for network scheduling, possibly with hierarchical composition, increasing the flexibility of the system.

Afterwards, we present a prototype implementation that validates the framework by means of a case study based on a control-based testbed.

6.3.1 Server-based scheduling

In the real-time scheduling literature many types of server-based schedulers have been presented for Fixed Priority Systems (FPS) and Dynamic Priority Systems (DPS). These schedulers are characterized partly by the mechanism for assigning deadlines, and partly by a set of parameters used to configure the servers, e.g., bandwidth, period and capacity. The Polling Server (PS) [116] is one of the simplest FPS servers. A PS allocates a share of a resource to the users of the server. This share is defined by the server period and capacity. The Deferrable Server (DS) [123] improves the responsiveness of the PS by permitting deferring its execution whenever there are no user requests. In general the DS gives better response times than the PS at expenses of a lower schedulability bound. By changing the way capacity is replenished for a server, the Sporadic Server (SS) [116] is a server-based scheduler for FPS systems that allows high schedulability without compromising too much the responsiveness.

Examples of Earliest Deadline First (EDF) based DPS servers include, e.g., the Dynamic Sporadic Server (DSS) [118]. A very simple (implementation wise) server-based scheduler that provides faster response time compared with SS is the Total Bandwidth Server (TBS) [118]. TBS makes sure that the server never uses more bandwidth than allocated to it, yet providing a fast response time to its users (under the assumption that the users do not consume more capacity than what they have specified). When the users desired usage is unknown, the Constant Bandwidth Server (CBS) [24] can be used, guaranteeing that the server users will never use more than the server capacity. This is particularly efficient to enforce temporal isolation between the server users and the remainder of the system.

In the network domain, probably for historical reasons, the names given to servers are different. For example, a common server used in networking is the *leaky bucket*. This is a specific kind of a general server category called traffic shapers, which purpose is to limit the amount of traffic that a node can submit to the network within a given time window, bounding the node burstiness. These servers use a technique similar to those described in the previous section, based on capacity that is eventually replenished. Many different replenishment policies are also possible, being the periodic replenishment as with PS or DS, the most common. However, it is hard to categorize these network servers similarly to the CPU servers referred in the previous section because networks seldom use clear fixed or dynamic priority traffic management schemes. For example, there is a large variability of Medium Access Control (MAC) protocols, some of them mixing different schemes such as round-robin scheduling with fixed priorities, first-come-first-served, first-come-first-served with multiple priority queues, etc.

This work advocates that, using an adequate protocol, such as one based on the FTT paradigm [28, 84, 101], it is possible to control the traffic in a way

that allows implementing any of the CPU-oriented server-based scheduling techniques.

6.3.2 The Server-SE protocol

The integration of the FTT-SE architecture along with the server-based scheduling capabilities creates a new protocol framework, the Server-SE, aiming to handle general message streams in switched Ethernet with arbitrary arrival patterns while still providing timing guarantees. Hierarchical server composition should also be provided in order to support efficiently the decomposition of complex applications into sub-applications, each with its own share of the bandwidth.

The FTT-SE architecture enables a nearly seamless deployment of the server-based management since the master has complete control of the server executions (message transmissions) and knowledge of the status of all queues in all the nodes (via a signaling mechanism). Alike the message scheduling in the FTT-SE protocol, the servers management is centralized in the master node, while the serving queues are kept in the transiting nodes. The communication between the Slaves and the Master node, required to update the queuing status, is supported by the asynchronous signaling mechanism, presented in Section 3.2.2. Then, upon a transmission signaling request, the Master schedules the messages and controls its dispatching via the EC-schedule conveyed in the TM. The message transmission within the Server scope is basically handled as the asynchronous messages in the FTT-SE protocol. However, it is still possible to keep a synchronous window to differentiate such kind of traffic if desired.

Server allocation and management

At run-time, all nodes must negotiate with the Master the creation of adequate servers to handle specific types of traffic. This negotiation is carried out using asynchronous control channels dynamically created and removed when nodes join and leave the system. The Master answers using the following TM in which it piggybacks the appropriate information, e.g., whether the server was actually created or not. Typically, the communication requirements would be expressed in terms of admissible ranges according to different levels of admissible QoS. These ranges can be used by the master to manage dynamically the QoS of the servers already running, e.g. to accommodate new requests. A suitable negotiation guarantees that all requirements of the real-time messages will be satisfied and the physical resources will be enough.

This mechanism is not transparent for the nodes. For legacy applications it is possible to add a wrapper to carry out the QoS negotiation before initiating the application itself so that the servers are created when the application is started. Alternatively, it is possible to tweak the network driver

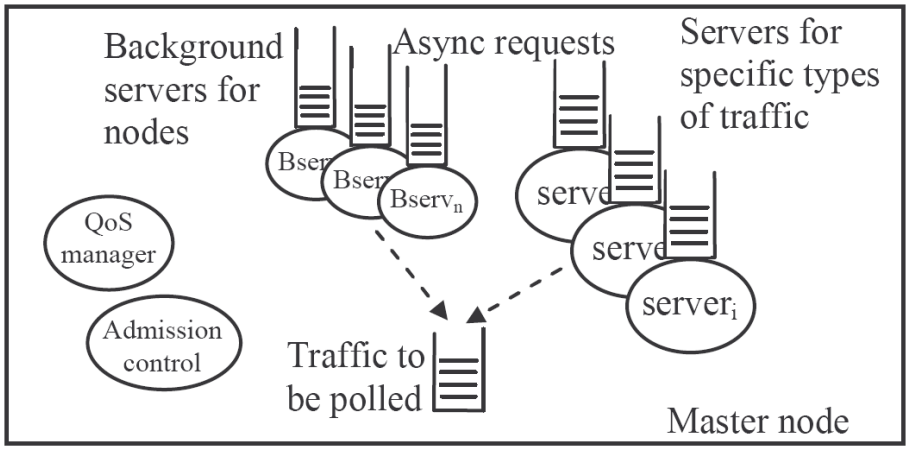


Figure 6.17: Internals of the Server-SE Master.

and to automatically create a background server per node. This server behaves as a "black box" , serving all the application components executed on the node without the need to know any details about their respective traffic characteristics. By default this channel is associated with the NRT window. The bandwidth is divided among all nodes and the polling is carried out in a per-node basis. This background server does not provide minimum QoS guarantees, since it uses the bandwidth left available by other higher priority servers, but grants any node some immediate communication capabilities without the need for specific server negotiations. If needed, specific servers can be created later on for specific traffic.

Server integration in FTT-SE: architectural overview

The integration of the server mechanism in the FTT-SE master is carried out associating one server instance to each asynchronous stream. Each server instance holds the server properties (e.g. budget, period) and manages the server status during run-time. The node's requests, received via the aforementioned signaling mechanism , are decoded and added to a *RequestList*. This list is ordered and keeps track of all the pending requests from all the asynchronous messages. The server status dictates whether the requests present in the *RequestList* are eligible for scheduling or not, thus forming a *ReadyQueue* from the *RequestList*. This step makes the server insertion transparent to the Master scheduler, that only has to scan the *ReadyQueue* and schedule the messages according to the implemented scheduling policy, as if it was the case for normal asynchronous messages.

The integration of the servers in the FTT framework results in the server hierarchy depicted in Figure 6.18. At the top level the FTT EC structure divides the traffic into synchronous and asynchronous classes, associated with

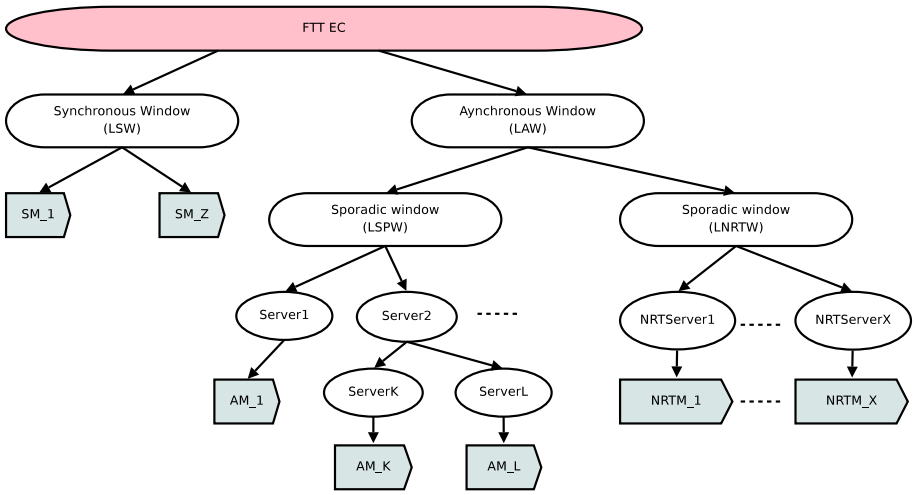


Figure 6.18: Server-SE Server Hierarchy.

disjoint windows that fill in the whole EC length. The synchronous window is associated with a polling server period $T_{sw} = E$ and a (maximum) capacity $C_{sw} = LSW$, resulting in a (maximum) bandwidth utilization of $U_{max_s w} = \frac{C_{sw}}{T} = \frac{LSW}{E}$. On its hand the asynchronous window receives the remaining bandwidth, derived implicitly from the EC size, synchronous window length and protocol overheads (bandwidth reclamation). Thus, it has a (minimum guaranteed) capacity of $C_{aw} = LAW$, resulting in a (minimum) bandwidth of $U_{max_a w} = \frac{C_{aw}}{T} = \frac{LAW}{E}$. Note that E and LSW are FTT configuration parameters that can be tuned to suit the global application needs. The EC period E establishes the granularity of the remaining servers; their periods are constrained to be an integer multiple of E. On the other hand LSW defines, indirectly, the length of the asynchronous window and so how much bandwidth is managed by the servers, or equivalently, the global server budget. Note that LSW can take any value from 0 to near E (in fact it has to be lower than E due to protocol overheads), thus the system is highly flexible in this regard.

The second level of the hierarchy manages the sporadic and the NRT traffic, the former having real-time requirements and thus being always scheduled before the latter, which is handled by a background server. Thus, at this level, the sporadic window inherits the bandwidth and (maximum) capacity of its ancestor server ($C_{spw} = C_{aw}; U_{spw} = U_{aw}$).

The third level of the hierarchy is where the additional servers can be plugged-in. Arbitrary scheduling policies can be implemented at this level and the sole constraints are that the base time granularity is E and the budget of the ancestor server.

For illustrative purposes the implementation of a Deferrable Server is described. This server uses a internal variable, designated *activationcounter*,

that controls the replenishment period. This variable is set to zero upon creation, meaning that the server is eligible for transmitting. Whenever a message is sent, the variable is set to the replenishment period (*mit*) and is decremented every EC until its value reaches zero. This event means that the server is replenished and thus any pending messages present in the *RequestList* are moved to the *ReadyQueue*, becoming eligible for transmission. The process repeats itself. As can be seen, the integration of the server-based mechanism produces minor implications in the FTT-SE architecture. The changes were restricted to the message activation procedure, maintenance of the *activationcounter* and *ReadyQueue* management. More complex servers can require additional structures, specially when comprising several messages in a single server scope, similarly to the case of server mechanisms for processor scheduling.

Limitations and alternative approaches

The Server-SE approach hereby proposed relies on the FTT-SE protocol and, consequently, also inherits most of the limitations and drawbacks of this protocol. From a conceptual point-of-view the main issues that can be identified in the current approach are the requirement for a cooperative architecture and the explicit signaling mechanism. FTT-SE is a master/multi-slave protocol and thus, as any other master/slave protocol, depends on the cooperation of all the nodes in the system. Malfunctioning or non-compliant nodes that use the transmission medium without respecting the mediation of the Master can jeopardize all the temporal guarantees. This aspect constrains one of the goals of Server-SE that is to efficiently support open systems, in which nodes can leave and join at will, while still providing timeliness guarantees. On the other hand, the explicit signaling mechanism induces an undesirable latency. Nodes can only report their status in discrete time instants, namely the beginning of the ECs. Thus the signaling latency is, at minimum, 1 to 2 ECs, and can be bigger for systems with a relatively high number of nodes (> 30, see Section 3.3.4).

Both of these problems can be solved by an alternative FTT-SE implementation currently under development and addressed in Chapter 7 as future work. A preliminary implementation for proof of concept is described in Section 6.4.The main idea is to integrate the FTT master within a custom Ethernet switch, confining, by itself, the non-FTT traffic to configurable predefined windows, thus preventing negative effects from non-compliant nodes. Furthermore, the modified switch carries out traffic policing, i.e., messages that violate the agreed time domain parameters are trashed and, preserving the correct behavior of real-time traffic. Finally, the explicit signaling mechanism is no longer needed and the nodes are allowed to transmit asynchronous messages autonomously. Incoming messages are queued, classified and verified by the switch. Thus, servers can be notified internally on the

arrival of new messages, update their status and turn messages eligible for transmission when appropriate. This is on-going work that is currently being explored.

6.3.3 Experimental results

This section presents experimental results obtained from a prototype implementation of Server-SE. The first experiment is a basic test meant essentially to verify the correctness of the implementation of the server mechanisms. The second experiment involves a mixed environment composed by a distributed control system, with timeliness requirements, and other sources of traffic, eventually causing overloads. This experiment aims at assessing the suitability of server-based mechanisms to handle time sensitive distributed systems comprising different traffic sources with distinct activation and timeliness requirements.

Traffic confinement

For this experiment we created FTT-SE system comprising one Master node and two slave nodes. The EC duration is set to 1ms and the network operates at 100 Mbps. The application submits three aperiodic streams to the network stack with the following properties: AM_1 and AM_2 sent by Slave 1 with 2kB and Mean Time Between Activations ($MTBA$) of 6 and 10 ECs, respectively, and AM_3 sent by Slave 2 with 1kB and $MTBA$ 5 ECs. Three sporadic servers [37] were set, one for each message. Each server has a budget equal to one instance and a replenishment period equal to $MTBA$. The servers were scheduled according to Rate Monotonic, considering the replenishment periods. LAW was constrained to be $240\mu seconds$, which was sufficient to schedule the three servers. In this configuration the servers can schedule no more than 2 packets of the referred messages per EC.

The generation of the message transmission requests was carried out randomly by an application program, which uniformly distributes the activations along the EC time frame, respecting the message $MTBA$. Figure 6.19 shows the results for message AM_3, with the time expressed in ECs. The top histogram shows the inter-arrival times of the requests (signaling messages arrived at the Master). The lower histogram shows the inter-arrival times of message AM_3 as scheduled by the Master (and transmitted by the respective Slave). The measures were actually carried out within the Master (difference between consecutive requests arrivals and difference between consecutive scheduled transmissions). It is clear that, despite the frequent bursts of consecutive requests, the transmissions respect the $MTBA$ corresponding to the respective sporadic server replenishment period. The histograms of the other two messages show similar behavior and thus are not presented.

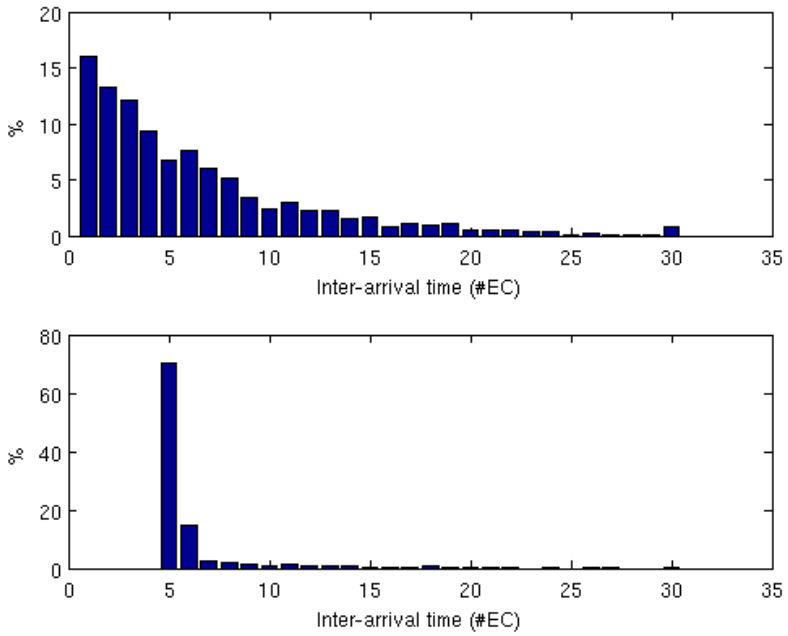


Figure 6.19: Top histogram: inter-arrival times of the AM_3 requests; Lower histogram: inter-arrival times of AM_3 messages.

Case study: distributed control system

This case study setup is based on a "ball-on-plane" mechatronic setup. This setup aims at keeping one ball as near as possible to a predetermined point in an horizontal plane. The plane is controlled by two servo-mechanisms that actuate on two orthogonal axes, X and Y. The ball position and corresponding error is obtained using a video camera connected to a computer.

The platform components form a distributed system based on the Server-SE protocol (Figure 6.20). The network bit rate is set to 100 Mbps. The platform comprises i) a **Sensor Device** that captures images and sends them to the controller; ii) a **Controller Device** that processes the received images, computes the ball coordinates in the plane, executes the control algorithm and sends the setpoints to the actuators (servos) iii) **Actuator Device** that receives the commands sent by the controller and position the servos accordingly.

Beyond, the sensor, controller and actuator nodes, the system comprises the Server-SE Master node as well as two other computers that simulate a scenario where two surveillance cameras, producing variable load traffic, share the same network. The surveillance cameras also direct the image streams to the controller node in order to overload its network link and cause interference in the control traffic.

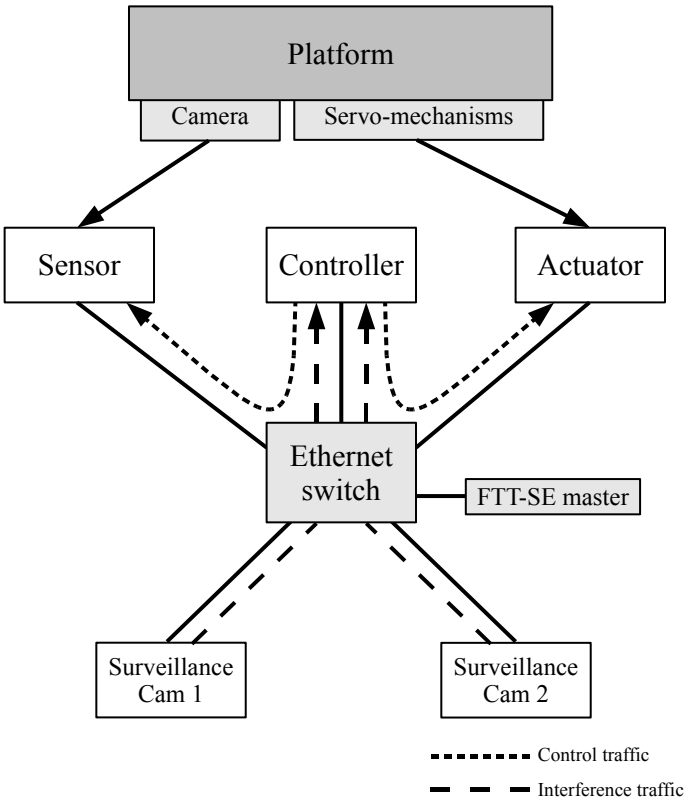


Figure 6.20: Platform connections diagram.

Typically, these surveillance cameras have no external triggering mechanism and thus it is not possible to synchronize their operation with the network. Hence, it was decided to use servers to support them. A total of 3 servers, one for each camera, have been created. The controller node was synchronized with the Server-SE network, and thus the control message is configured as a periodic one and sent within the Server-SE synchronous window. In this scope the utilization of servers is important because it guarantees bandwidth isolation (no device can use more bandwidth than the allowed one) and prevents the occurrence of unbounded traffic bursts, guaranteeing a regular and predictable access to the network (upper bounded by the server period). The critical link is, in this case, the one between the switch and the controller. For this reason the measurements presented in this case study were performed on this link.

In this experiment the control-related traffic was kept constant and the surveillance cameras have been used to simulate a varying load. Eight different experiments, corresponding to increasing link utilization rates, have been carried out. Table 6.6 summarizes the cameras configuration and global link utilization for each one of the experiments. SC stands for Sensor's Cam-

Test	SC (fps)	VC1 (fps)	VC2 (fps)	Utilization Rate
1	30 fps	0 fps	0 fps	19.776%
2	30 fps	15 fps	15 fps	38.816%
3	30 fps	29 fps	24 fps	53.414%
4	30 fps	38 fps	28 fps	61.665%
5	30 fps	37 fps	38 fps	67.377%
6	30 fps	42 fps	42 fps	73.089%
7	30 fps	49 fps	53 fps	84.513%
8	30 fps	57 fps	57 fps	90.859%

Table 6.6: Load characterization - Controller downlink.

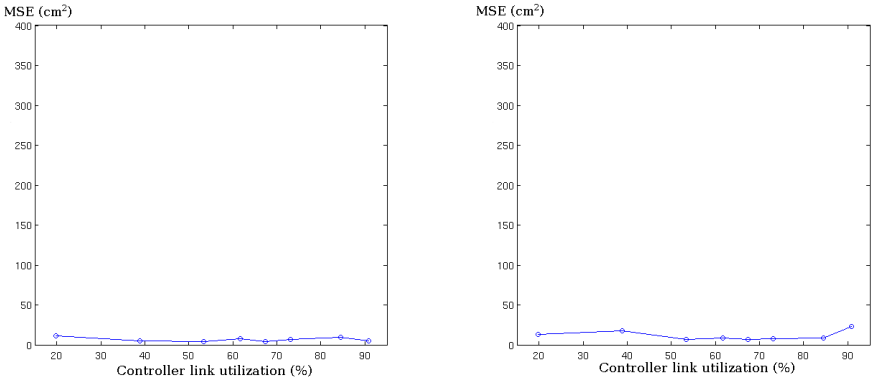
era while VC stands for Video surveillance cameras. The utilization value, indicated in the rightmost column of Table 6.6, includes all the protocol overheads.

To assess the impact on the control performance, the mean square error of the ball in both the X and Y axes was computed, for each one of the different load conditions. As shown in Figure 6.21, the control performance remains essentially constant throughout the different load conditions. This is a direct result of the presence of the servers, which guarantee the QoS independently of the particular load conditions, preserving the property of composability of the real-time property among streams of messages scheduled through separated servers. Also, the control message, transmitted in the synchronous window, does not suffer from any kind of negative impact due to the surveillance cameras induced overload.

The same scenario was implemented with RAW Ethernet, i.e., without network control enforcement. Figure 6.22 shows the results of the control performance for the same network utilization loads as with the setup using the FTT-SE framework. Comparing both implementations, it is noticeable the control degradation in the RAW implementation for high utilization loads in the network (above 70%), leading to the control system instability.

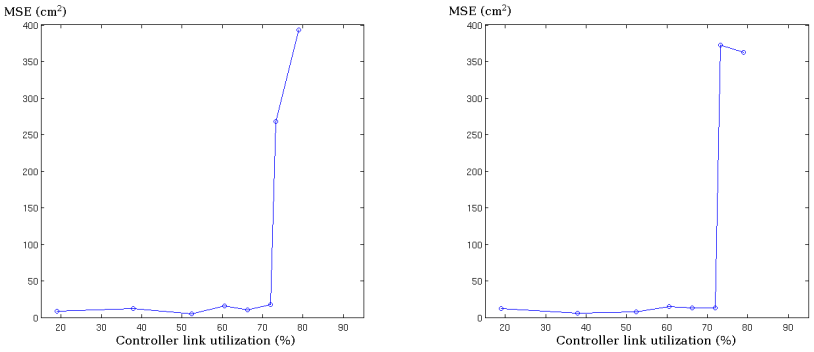
6.3.4 Summary

Real-Time Ethernet (RTE) protocols have difficulties in the efficient handling of aperiodic message streams with arbitrary arrival patterns, while at the same time supporting the derivation of timeliness guarantees. This section presents a server-based mechanism for switched Ethernet real-time networks, integrating concepts from the Server-CAN protocol on the FTT-SE protocol. This approach enables an efficient implementation of arbitrary server schedulers as well as their hierarchical composition. Moreover, this approach is very suitable for open systems as servers can easily be added,



(a) Position X (b) Position Y

Figure 6.21: Mean Square Error of the ball with FTT-SE.



(a) Position X (b) Position Y

Figure 6.22: Mean Square Error of the ball with Raw Ethernet.

changed and removed during run-time. We include a case study based on a distributed control application. The obtained results illustrate the correct operation of the server-based protocol, showing the capability of the framework in providing strict timeliness guarantees to the real-time traffic in spite of the interference with arbitrary arrival patterns and load variations.

6.4 FTT-SE enabled switch

The FTT-SE protocol exhibits a traffic regulation that is based in a common time-line that globally coordinates the traffic submitted to the switch, allowing for fast and atomic online updates to the set of streams. This traffic enforcement is centralized in a single entity, called master, connected to a standard switch.

While using non-standard hardware conflicts with some of the key arguments supporting the use of Ethernet in real-time applications (e.g. cost, availability, compatibility with general purpose LANs), custom switch implementations with enhanced traffic control and scheduling capabilities allows important performance and service breakthroughs, and so a number of approaches of this class have also been proposed in the recent years (e.g. [62], [132], [2]).

This section provides a proof-of-concept study for integrating the traffic management and transmission control mechanisms of the FTT-SE in an Ethernet switch. The resulting framework allows obtaining important performance gains in the following key aspects:

- A noticeable reduction in the switching latency jitter found in common Ethernet switches;
- An important performance boost of the asynchronous traffic, which in this case is autonomously triggered by the nodes instead of being pooled by the master node;
- An increase in the system integrity since unauthorized transmissions can be readily blocked at the switch input ports, thus not interfering with the rest of the system;
- Seamless integration of standard non FTT compliant nodes without jeopardizing the real-time services.

6.4.1 Switch Architecture

Figure 6.23 depicts the functional architecture of the FTT-enabled Ethernet switch integrating the FTT master traffic management services (shaded area). The System Requirements Database (SRDB) is the central repository for all the information related to the traffic management, namely the message attributes for both synchronous and asynchronous traffic (e.g. period/minimum inter-arrival time, length, priority, deadline), information about the resources allocated to each traffic class (e.g. phase durations, maximum amount of buffer memory) and global configuration information (e.g. elementary cycle duration, data rate). Change requests to the message set are submitted to an admission control (plus optional QoS manager), ensuring continued real-time traffic timeliness. The SRDB is periodically scanned

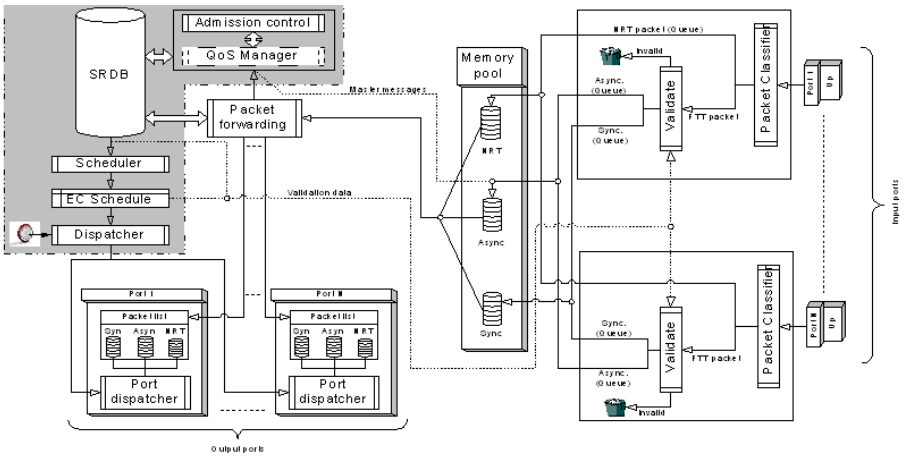


Figure 6.23: FTT-enabled switch functional architecture.

by a scheduler, which builds a list of synchronous messages (EC-Schedule) that should be produced in the following EC. A dispatcher task periodically broadcasts the EC-schedule within the Trigger Message.

The main advantage of integrating the FTT-SE master inside the switch is the tight control of the packet flow and resource utilization that becomes possible. At the beginning of each EC the global dispatcher directly accesses the port dispatcher, which sends the trigger message and keeps temporal information about each of the phases within the EC. Each output port has 3 queues, one for each traffic class (Sync RT, Async RT and NRT). During the EC the port dispatcher transmits messages submitted to each of these queues, according to the EC phase. This mechanism confines the different traffic classes to the respective phases. If, for example, a malfunctioning node sends a synchronous message outside of the synchronous phase, the message is discarded and does not interfere with the asynchronous or non real-time phases. On the other hand aperiodic messages (either Async RT or NRT) do not need to be polled, contrarily to what happened for FTT-SE. The port dispatcher only transmits messages from the asynchronous or NRT queues if the time left within the respective window is enough.

Both FTT and non-FTT-compliant nodes can be seamlessly attached to the FTT enabled switch. Thus, on the ingress side the first operation carried out is the packet classification, which consists only in inspecting the Ethernet type field. When the message is identified as an FTT message it is subject to a verification process and, if valid, is appended to the synchronous or asynchronous message queues, according to its nature. Conversely, if the message is non-FTT it is simply appended to the NRT queue. The segmentation of the global memory pool, keeping the messages of each class in independent subdivisions, allows avoiding memory exhaustion for the real-

time messages, a situation that standard switches do not guarantee [102]. Note, however, that the amount of memory needed for the real-time messages is typically small given their temporal properties that are enforced at the switch ingress. It is also known in advance since the RT traffic is subject to an explicit registration. During the registration process the producers must state the flow properties, in particular the length and period for periodic messages or minimum inter-arrival time for sporadic ones. With this data it is possible to compute and pre-allocate the amount of memory that RT traffic requires and thus guarantee that the resources are enough for all admitted messages.

The validation process of the RT classes gathers data both from the EC-schedule and from the SRDB. Regarding synchronous messages, the analysis of the EC-schedule allows detecting failures in the time domain, namely the transmission of unscheduled messages or the late transmission of scheduled messages resulting from malfunctioning nodes. An equivalent set of tests (e.g. minimum inter-arrival time, burstiness) is also performed for asynchronous messages with those that fail the validation process being trashed. The policing and enforcement of the traffic attributes in the time domain guarantees the timeliness of the real-time traffic even in the presence of malfunctioning nodes.

Whenever a message is placed in the global memory pool, a packet forwarding process is executed. Control messages, targeted to the master are submitted to the Admission control/QoS manager module and possibly result in changes on the SRDB. Data messages should be forwarded to the target nodes. The forwarding mechanism of FTT messages is based on a producer-consumer model, and does not depend on MAC addresses. Whenever an FTT message arrives the Packet Forwarding module inquires the SRDB to determine the set of ports having consumers attached, and updates the output queue (synchronous or asynchronous, depending on the message nature) of each one of these ports. Non-FTT messages are forwarded according to the normal procedures of standard Ethernet switches, based on the MAC address. From the point of view of NRT traffic, the FTT-enabled switch behaves as a common switch but with restricted bandwidth arising from the confinement of this traffic to the NRT phase in the EC.

The integration of the FTT-SE master in the switch is transparent to the nodes in what concerns the synchronous traffic. There are, however, several simplifications for the aperiodic traffic. Nodes that do not produce real-time messages can use any standard Ethernet driver. The transmission control of this class in FTT-SE is no longer required here, being replaced by adequate confinement by the switch. Nodes that produce real-time messages require the FTT-SE network driver to be updated to include different queues for the three traffic classes and avoid blocking of the synchronous traffic in the uplinks.

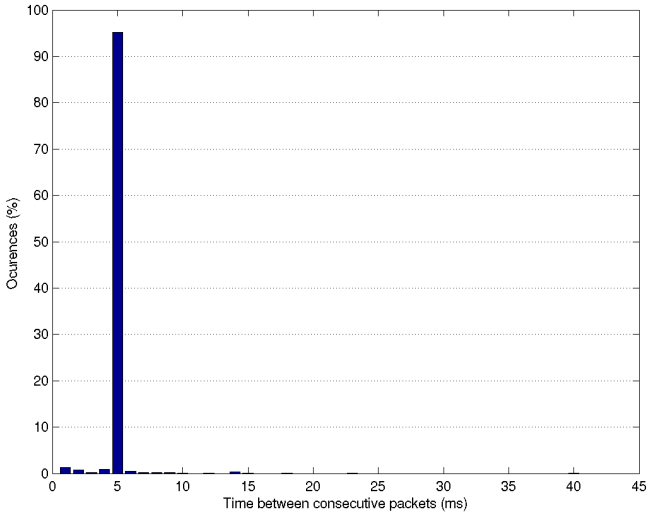


Figure 6.24: Histogram of the differences between consecutive NRT messages (uplink).

6.4.2 Experimental results

A prototype implementation, based on the RT-Linux real-time operating system with the Ethernet layer provided by the LNet network stack, was carried out to validate the extended services provided by the FTT-enabled switch. This prototype switch is based on a Pentium III PC at 550MHz with four 3Com 3C905B PCI network interface cards.

The first experiment consists in the implementation of a policing service for the synchronous traffic. The ID of incoming synchronous messages is matched against the ECschedule and discarded if a positive match is not found. This way only scheduled messages are disseminated, guaranteeing that the synchronous window is not overrun. To verify the correct behavior of the policing service we configured a setup with one synchronous message with period Ti=3ECs while the respective producer slave was tampered to send that message every EC. We observed that the consumer node only received the scheduled messages, one every 3ECs, and the extra messages were discarded. The second experiment consists in the verification of the enforcement of the traffic temporal isolation. The experimental setup is configured with an EC of 40ms, with the last 3ms of the EC dedicated to the NRT traffic. The NRT test load consists in UDP packets carrying 1400 data bytes, periodically sent every 5ms. The load is generated with Pack-Eth (http://packeth.sourceforge.net/) running on a plain Linux distribution (RedHat 9.0). Figure 6.24 depicts the histogram of the time differences between consecutive NRT messages in the uplink.

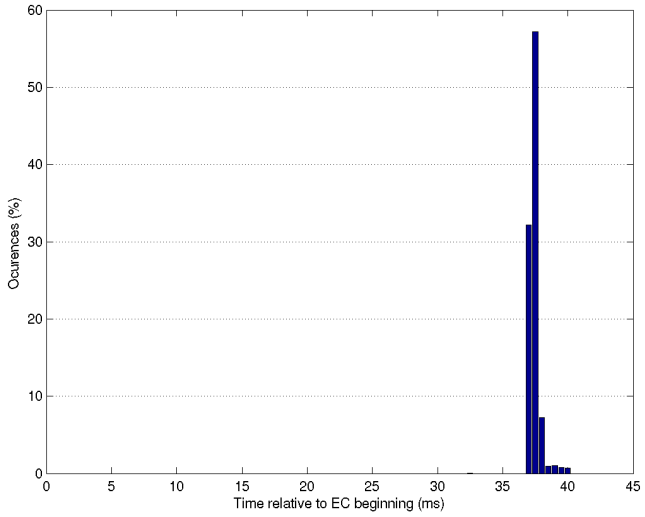


Figure 6.25: Histogram of the differences between the beginning of the EC and NRT messages (downlink).

While NRT messages can be submitted at any time instant, the FTT-enabled switch only forwards them to the output port(s) in the NRT window, which in this setup is configured to use the last 3ms of the EC. This confinement mechanism significantly changes the message transmission pattern between the uplink and the downlink. Figure 6.25 shows the time difference between the beginning of the EC and the reception of the NRT messages being clear the confinement of these to the NRT window (37 to 40ms after the EC start). Therefore the NRT traffic does not interfere with the synchronous or asynchronous real-time traffic, despite being generated at arbitrary time instants by a standard node not implementing the FTT protocol.

6.4.3 Summary

The advent of switched Ethernet has opened new perspectives for real-time communication over Ethernet. However, a few problems subsist related with queue management policies, queue overflows and limited priority support. While several techniques were proposed to overcome such difficulties, the use of standard Ethernet switches constraints the level of performance that may be achieved. Therefore, we proposed an enhanced Ethernet switch, implementing FTT-class services. The resulting architecture inherits the FTT features, namely flexible communication with high level of control to guarantee timeliness, while providing a noticeable reduction in the switching latency jitter found in common Ethernet switches, an important performance increase of the asynchronous traffic, seamless integration of standard Ether-

net nodes and a substantial increase in the system integrity as unauthorized transmissions from the nodes can be readily blocked at the switch input ports. On-going work (Chapter 7) addresses the FPGA implementation of this switch.

6.5 Conclusion

The core contribution of this thesis is a new real-time communication protocol for switched Ethernet, namely FTT-SE,that is capable of handling the dynamic communication requirements as needed for dynamic QoS management or simply for open and dynamically reconfigurable systems. This chapter compiled four case-studies that highlight such capability and provide some insight over the performance and usability of such protocol.

The first case-study addresses the integration of the FTT-SE protocol in an open-scoped resource handling framework (FRESCOR). FRESCOR provides a unified and common application interface for handling together the management of several resources, needed by a service. It handles a number of resource types in classes. The integration with the FTT-SE protocol brings in a new network communication class type with different communication paradigms, namely time and event-triggered, and switched Ethernet topology. Moreover, FTT-SE supports in a direct and efficient way the integrated contracts management of FRESCOR. This results on a smooth integration of FTT-SE in the global application design scope.

The second case-study shows the use of FTT-SE QoS management in improving the performance of a multimedia industrial system. The scenario includes the transmission of video streams over an FTT-SE network with specific QoS requirements that may vary within a given quality range, depending on the network load. We propose a feedback mechanism based on tuning the parameters of MJPEG streams that directly affect the service quality and the network load, such as, the compression factor, the periodic transmission rate and the frame transmission buffer. Given that FTT-SE already assures a QoS-aware resource management based on bandwidth distribution, this case-study illustrates how a multimedia application can take advantage of the QoS-aware resource management to carry out dynamic QoS adaptation. The QoS management includes the adaption of variable bitrate (VBR) services, such as multimedia streams, in deterministic constant bitrate (CBR) channels, such as the ones provided within the FTT-SE framework. The results of this case-study endorse the use of dynamic QoS management policies in resources that support flexible reconfiguration.

The third case-study shows how FTT-SE can be used to support both synchronous and asynchronous traffic in an integrated way using servers. The centralized scheduling architecture of FTT-SE enables a seamless deployment of a variety of servers, allowing the designer to choose the best one for an application domain and even to compose several servers hierarchically, promoting a good component-oriented integration. In order to illustrate and validate the server-based traffic scheduling approach, we used a distributed control application. The results demonstrate the capability of the servers in providing strict timeliness guarantees to the real-time traffic in spite of interfering traffic with arbitrary arrival patterns and load variations.

Finally, the fourth case-study presented a proof-of-concept for pushing further the FTT-SE framework to a different level, including some of the Master functionalities inside the switch. As presented in this thesis, the FTT-SE protocol relies on an external Master node to enforce traffic timeliness as well as the cooperative behavior of every single node.

Hence, only FTT-SE-enabled nodes are allowed to connect to the switch. Otherwise, the communication timeliness would be compromised. Thus, we propose enhancing the switch with special capabilities to directly enforce, at the packets ingress, the results from the scheduler and enforce policing rules that prevent non-conforming nodes from communicating outside the defined windows. This preliminary implementation used a PC with several network interfaces, acting as a modified switch. Further exploration of this line of work is stated as future work, as described in the following chapter.

Chapter 7

Conclusions

7.1 Thesis, contributions and validations

The central proposition of this thesis, supported by this dissertation, claims that it is possible to develop a communication protocol over a switched Ethernet network able to support hard real-time communications combined with operational flexibility, including the ability to reconfigure the network properties online and adapt to variable requirements supporting an efficient QoS management in dynamic environments. On the quest for this achievement the following network characteristics have been identified as required:

- Support for time- and event-triggered traffic with temporal isolation;
- Real-time support in dynamic systems, with online message scheduling;
- Support different scheduling policies based on dynamic or static requirements, including aperiodic servers;
- Support for on-line reconfigurability and admission control;
- Support open systems with a contract-based resource management;
- Yield a seamless deployment for dynamic QoS management policies;
- Support for multiple parallel forwarding paths available at the Ethernet switches.

Current state of the art protocols for switched Ethernet cannot fulfill all these characteristics, a situation that motivated the proposal for a new protocol, the Flexible Time-Triggered over Switched Ethernet (FTT-SE), which is one of the core contributions of this work. This protocol is based on the master/slave FTT paradigm that defines a conceptual framework to combine flexibility and timeliness in communication networks. The following contributions have been made on this regard:

- Support for parallel multi-path traffic forwarding in the switch (unicast and broadcast models), while maintaining the protocol requirements for predictability and support any traffic scheduling policy (Chapter 3);
- Handling the asynchronous traffic with a novel message signaling mechanism that improves its scheduling flexibility, particularly supporting open server-based scheduling approaches, and allows a seamless integration between asynchronous and synchronous traffic (Chapter 3);
- System analysis including the schedulability analysis of the periodic traffic, based on a utilization bound that accounts for the impact of release jitter (Chapter 4).

In addition, it has been proposed a systematic methodology for the deployment of dynamic QoS management mechanisms over communication resources such as the FTT-SE. Contributions on this research line include the proposal of several bandwidth distribution approaches to handle specific QoS level requirements. The methodology is general and includes other existing QoS management techniques as particular cases (Chapter 5).

The thesis and its related contributions were validated with:

- Simulation and practical experiments using synthetic workloads. The FTT-SE framework has been implemented in COTS hardware and some of its properties validated, including hard real-time enforcement and support for multiple parallel forwarding paths (Chapter 3). Simulations with different workload scenarios have also been conducted to validate the hard real-time message scheduling as well as the proposed schedulability analysis, which is the basis for the online admission control (Chapter 4);
- The integration in a global contract-based resource manager/middleware. In Section 6.1 the FTT-SE is proposed to integrate FRESCOR, a resource management framework, validating the architecture in terms of supporting contract-based negotiation.
- An industrial video surveillance application with dynamic QoS management (Section 6.2), validates the protocol support for on-line reconfigurability, adaptability and the seamless deployment of dynamic QoS management policies;
- A distributed control platform with high interference workloads (Section 6.3), validates the protocol efficiency on supporting temporal isolation and its correct behavior when implementing server-based scheduling. Asynchronous traffic is used in the servers as well as during all the contract negotiation procedure, which validates the proposed asynchronous message signaling mechanism.

7.2 On-going and Future research

The work conducted for this thesis unveiled some interesting research ideas that worth future consideration.

Servers and hierarchical composition

The FTT-SE framework relies on a centralized architecture to schedule the network and enforce temporal determinism, providing a scheduling environment similar to the one found for processor scheduling. This yields the possibility of deploying, on a distributed network, any scheduling policy as for processor scheduling and enables a flexible handling of any kind of messages, a property specially suited for systems with dynamic behavior concerning the communication requirements as well as open systems that accept new components online, possibly legacy components developed with specific communication models.

Moreover, while addressing open systems, such flexibility supports the deployment of servers and services that might join, leave, or be modified at run-time, leveraging a composable design perspective.

From the design perspective, the use of servers to handle the communication requirements allows the integration of different and independent applications. The servers define operational boundaries from the resource underneath and provides to the application an abstracted interface. This component-oriented integration ultimately leads to a full abstracted and composable design, where servers are composed hierarchically, allowing a flexible resource utilization.

The use of servers in the FTT-SE framework has been verified in Chapter 6 with an implementation of a server handling the sporadic traffic of a control system. This case-study is a preliminary proof-of-concept for a server-based scheduling approach. It unveils promising research results on the integration of a communication framework in general, and the FTT-SE in particular, within a compositional design.

FTT-SE enabled switch

The design of the FTT-SE framework had in consideration its deployment over regular Ethernet compliant hardware, in order to guarantee a more general hardware integration and maintain the hardware costs reduced. The protocol then relies on a protocol stack to intermediate every node transmission and so enforce temporal determinism. Thus, it is not possible to include non-FTT-SE compliant nodes. Such functional and deployment limitations endorse the pursuit for a more robust and generally applicable solution for the FTT-SE protocol. A possible approach is developing a customized switch that enforces some of these properties. The main benefits that outcome from an FTT-SE enabled switch are the possibility to enforce traffic policing rules at the incoming ports and directly actuate at the output queues, classifying

the packets according to their temporal requirements and allow connecting any legacy node, with no FTT awareness, mixing in the same network FTT traffic with strict temporal guarantees and regular Ethernet traffic in a best-effort fashion. Another feature emerging is the possibility for a complete rethought on the mechanism that handles the asynchronous traffic. The possibility of scheduling and directly managing the switch queues allows the nodes to autonomously issue the asynchronous traffic.

This evolutionary trend that includes the Master node within the Ethernet switch promises to be of great interest for a wider integration scale, greater communication responsiveness and reduced complexity at the end-nodes. Moreover, the system becomes more robust and more dependable concerning bad behaved nodes.

Schedulability analysis

Chapter 4 proposes a utilization-based schedulability analysis for FTT-SE. However, although correct, the results are more pessimistic comparing to other, more exact, methodologies such as response time analysis or network calculus. On the other hand, the proposed analysis provides schedulability results in linear time bounds and facilitates the QoS distribution among the running services, which makes it ideal for certain application domains. In this context the evaluation of the pessimism incurred with this utilization-based test seems an interesting line of research as well as the comparison with other existing schedulability results.

Another interesting line of research is the extension of the utilization-based analysis to include the multicast/broadcast model. Chapter 4 only covers the unicast transmission model, leaving unstudied the multicast/broadcast model. As mentioned in Section 4.6 a schedulability test for such a scenario is more complicated. While in a unicast model scenario the interference at the downlinks is located in the uplinks, only, in a multicast scenario such interference may also result from other downlinks that may be affected by other uplinks. This interference model cannot be directly handled with the proposed technique for unicast.

It is of great relevance extending the proposed utilization-based analysis to cope with the extra interference sources in order to make the the multicast/broadcast transmission model available in the FTT-SE framework.

Bibliography

[1] *Coding of audio-visual objects - Part 10: Advanced Video Coding (AVC) ISO/IEC 14496-10, ITU-T recommendation H.264 AVC for generic audio visual services.*

[2] Real-time PROFINET IRT. `http://www.profibus.com`.

[3] DIX Ethernet V2.0 specification. IEEE, 1982.

[4] IEEE 802.3ae-2002 - 10Gbps. IEEE, 1982.

[5] IEEE 802.3c 100BASE-T standard. IEEE, 1982.

[6] IEEE 802.3i 10BASE-T standard. IEEE, 1982.

[7] IEEE 802.3z 1000BASE-T standard. IEEE, 1982.

[8] *ISO/IEC 10918-1:1994 - Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*, February 1994.

[9] *ISO/IEC 13818-2 Information technology – Generic coding of moving pictures and associated audio information*, May 1994.

[10] *Video Coding for Low Bit Rate Communications ITU-T recommendation H.263*, April 1995.

[11] Video codec test model: TMN8, June 1997.

[12] IEEE 802.1p. IEEE, 1998.

[13] IEEE 802.3ac-1998. IEEE, 1998.

[14] *Coding of audio-visual objects - Part 2: Visual, ISO/IEC 14496-2 (MPEG-4 Visual Version 1)*, April 1999.

[15] *ARINC 664, Aircraft Data Network, Part 1: System Concepts and Overview*, 2002.

[16] *ARINC 664, Aircraft Data Network, Part 2: Ethernet Physical and Data Link Link Layer Specification*, 2002.

[17] *ARINC 664, Aircraft Data Network, Part 7: Deterministic Networks*, 2003.

[18] *Axis Communications WHITE PAPER: Digital video compression, Reviewing the methodologies and standards to use for video transmission and storage.* `http://www.vns.net/axis/documents/compression_standards.pdf`, June 2004.

[19] IEEE 802.1D-2004. IEEE, 2004.

[20] IEEE 802.1Q-2005. IEEE, 2005.

[21] Mobotix. What IP did next. Security Installer. `www.mobotix.com/ger_DE/file/33573/Security+Installer+bench+test+M22M.pdf`, July 2006.

[22] Ethernet Powerlink protocol. `http://www.ethernet-powerlink.org`, 2008.

[23] Foundation Fieldbus HSE. www.fieldbus.org, 2008.

[24] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. of the 19th IEEE Int. Real-Time Systems Symposium (RTSS'98)*, pages 4–13, Madrid, Spain, December 1998. IEEE Computer Society.

[25] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J.M. Drake, G. Fohler, P. Gai, M.G. Harbour, G. Guidi, J.J. Gutierrez, T. Lennvall, G. Lipari, J.M. Martinez, J.L. Medina, J.C. Palencia, and M. Trimarchi. FSF: A Real-Time Scheduling Architecture Framework. In *Proceedings of the 12th IEEE Real Time on Embedded Technology and Applications Symposium (RTAS'06)*, pages 113–124, April 2006.

[26] Luís Almeida and J. A. Fonseca. The planning schedule: compromising between operational flexibility and run-time overhead. In *Proceedings of the IFAC Int. Symposium on Infromation Control and Manufacturing (INCOM'98)*, June 1998.

[27] Luís Almeida and José Alberto Fonseca. Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling. In *Proc. of the 13th EUROMICRO Conference on Real-Time Systems (ECRTS'01)*, June 2001.

[28] Luís Almeida, Paulo Pedreiras, and José A. Fonseca. The FTT-CAN Protocol: Why and How. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, December 2002.

[29] A. Antunes, P. Pedreiras, L. Almeida, and A. Mota. Dynamic Rate and Control Adaptation in Networked Control Systems. In *IEEE Int. Conf. on Industrial Informatics, 2007*, volume 2, pages 841–846, June 2007.

[30] Aeronautical Radio Inc. ARINC 429. *ARINC Specification 429*, 2001.

[31] 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*, 8(5):284–292, September 1993.

[32] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: the deadline-monotonic approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, May 1991.

[33] F. Bogenbergerand, B. Muller, and T. Fuher. Protocol overview. In *Proceedings of the 1st FlexRay Int. Workshop*, April 2002.

[34] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag New York, Inc., New York, NY, USA, July 2001.

[35] Ch. Bouras and A. Gkamas. Multimedia transmission with adaptive QoS based on real-time protocols. *International Journal ofCommunications Systems, Wiley InterScience*, 16:225–248, 2003.

[36] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*. RFC 1633, July 1994.

[37] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[38] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, 1998.

[39] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.

[40] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. `http://www.ietf.org/rfc/rfc3376`, October 2002.

[41] A. Carpenzano, R. Caponetto, L. Lo Bello, and O. Mirabella. Fuzzy traffic smoothing: an approach for real-time communication over Ethernet networks. In *Proceedings of the 4th IEEE Int. Workshop on Factory Communication Systems (WFCS'02)*, pages 241–248, August 2002.

[42] Thidapat Chantem, Xiaobo Sharon Hu, and M. D. Lemmon. Generalized Elastic Scheduling. In *Proc. of the 27th IEEE Int. Real-Time Systems Symposium (RTSS'06)*, pages 236–245, Washington, DC, USA, 2006. IEEE Computer Society.

[43] C.-S. Cho, B.-M. Chung, and M.-J. Park. Development of Real-Time Vision-Based Fabric Inspection System. *Industrial Electronics, IEEE Transactions on*, 52(4):1073–1079, Aug. 2005.

[44] M. Christensen, K. Kimball, and F. Solensky. Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches. `http://www.ietf.org/rfc/rfc4541`, May 2006.

[45] Rene L. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[46] Rene L. Cruz. A calculus for network delay, Part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[47] M. Di Natale. Scheduling the CAN bus with earliest deadline techniques. In *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, pages 259–268, 2000.

[48] Wei Ding and Bede Liu. Rate control of MPEG video coding and recording by rate-quantization modeling. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(1):12–20, Feb 1996.

[49] Xing Fan, Magnus Jonsson, and Jan Jonsson. Guaranteed Real-Time Communication in Packet-Switched Networks with FCFS queuing. Technical report, School of Information Science, Computer and Electrical Engineering, Halmstad University, Sweden, April 2007.

[50] Paul Ferguson and Geoff Huston. *Quality of service: delivering QoS on the Internet and in corporate networks.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[51] Sebastian Fischmeister and Klemens Winkler. Non-blocking Deterministic Replacement of Functionality, Timing, and Data-Flow for Hard

Real-Time Systems at Runtime. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 106–114, Washington, DC, USA, 2005. IEEE Computer Society.

[52] G. Fohler. FRSCOR - Initial application requirements collection. FRESCOR deliverable D-RA1, November 2006.

[53] Inc. FSMLabs. *LNet Programming with RTCore*, June 2006.

[54] FSMLabs Inc. *Real-time Programming in RTCore (v2.2.3)*, 2006.

[55] E. Gallo, M. Siller, and J. Woods. An Ontology for the Quality of Experience framework. In *Int. Conf. on Systems, Man and Cybernetics (SMC'07)*, pages 1540–1544, October 2007.

[56] L. George, N. Riviere, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA, Le Chesnay Cedex, France, 1996.

[57] F. Gomez-Molinero. Real-Time Requirement of Media Control Applications. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS '07)*, pages 4–4, July 2007.

[58] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol, RFC 2543, March 1999.

[59] M. González Harbour and J. C. Palencia. Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, page 200, Washington, DC, USA, 2003. IEEE Computer Society.

[60] H. Hassan, J. Simo, and A. Crespo. Enhancing the flexibility and the quality of service of autonomous mobile robotic applications. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 213–220, 2002.

[61] H. Hoang and Magnus Jonsson. Switched real-time Ethernet in industrial applications – asymmetric deadline partitioning scheme. In *Proceedings of the 2st Int. Workshop on Real-time LANs in the Internet age (RTLIA'03)*, July 2003.

[62] Hoai Hoang, Magnus Jonsson, Ulrik Hagström, and Anders Kallerdahl. Switched real-time Ethernet and earliest deadline first scheduling - protocols and traffic handling. In *Proceedings of the 16th Int. Conf. on Parallel and Distributed Processing Symposium (IPDPS '02)*. IEEE Computer Society, 2002.

[63] Javier Silvestre, Luís Almeida, Ricardo Marau, and Paulo Pedreiras. MJPEG Real-Time transmission in industrial environment using a CBR channel. In *Proc. of 16th Int. Conf. on Computer and Information Society Engineering (CISE'06)*, Venice, Italy, 24 November 2006. (also published on Enformatika Trans. on Engineering, Computing and Technology, Vol 16, Nov. 2006 ISSN 1305-5313).

[64] R. Johnston and G. Clark. *Service operations management*. Pearson Education, 2001.

[65] Michael B. Jones. Adaptive Real-Time Resource Management Supporting Composition of Independently Authored Time-Critical Services. In *In Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 135–139, 1993.

[66] Frescor project home page. `http://www.frescor.org`.

[67] H. Kopetz. Specification of the ttp/c protocol , version 0.5. Technical report, TTTech Computertechnik, July 1999.

[68] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[69] A. Kumar. Computer-Vision-Based Fabric Defect Detection: A Survey. *Industrial Electronics, IEEE Transactions on*, 55(1):348–363, Jan. 2008.

[70] Seok-Kyu Kweon and K.G. Shin. Achieving real-time communication over Ethernet with adaptive traffic smoothing. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 90–100, June 2000.

[71] Chang-Gun Lee, Chi-Sheng Shih, and Lui Sha. Online QoS optimization using service classes in surveillance radar systems. *Real-Time Systems*, 28(1):5–37, 2004.

[72] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the 10th IEE Real-Time Systems Symposium (RTSS'89)*, pages 166–171, Dec 1989.

[73] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, pages 201–209, Dec 1990.

[74] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pages 110–123, Dec 1992.

[75] Xiaoli Li, S.K. Tso, Xin-Ping Guan, and Qian Huang. Improving Automatic Detection of Defects in Castings by Applying Wavelet Technique. *Industrial Electronics, IEEE Transactions on*, 53(6):1927–1934, Dec. 2006.

[76] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[77] H. Lonn and J. Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, pages 142–149, 1999.

[78] Jork Löser and Hermann Härtig. Low-Latency Hard Real-Time Communication over Switched Ethernet. In *Proc. of the 16th EUROMICRO Conference on Real-Time Systems (ECRTS'04)*, pages 13–22. IEEE Computer Society, July 2004.

[79] Jork Löser and Hermann Härtig. Using Switched Ethernet for Hard Real-Time Communication. In *Proc of Int. Conference on Parallel Computing in Electrical Engineering (PARELEC'04)*, pages 349–353, September 2004.

[80] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback control real-time scheduling: framework, modeling, and algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23:85–126, 2002.

[81] Nicholas Malcolm, Wei Zhao, and A. Stankovic. Hard Real-Time Communication in Multiple-Access Networks. *Real-Time Systems*, 8:35–77, 1995.

[82] R. Marau, P. Pedreiras, and L. Almeida. Enhanced Ethernet Switching for Flexible Hard Real-Time Communication. In *Proc. on the 5th Int. Workshop on Real Time Networks (RTN'06)*, Dresden, Germany, July 2006.

[83] R. Marau, P. Pedreiras, and L. Almeida. Signaling asynchronous traffic over a Master-Slave Switched Ethernet protocol. In *Proc. on the 6th Int. Workshop on Real Time Networks (RTN'07)*, Pisa, Italy, 2 July 2007.

[84] Ricardo Marau, Luís Almeida, and Paulo Pedreiras. Enhancing real-time communication over COTS Ethernet switches. In *Proc. of 6th Int. Workshop on Factory Communication Systems (WFCS'06)*, pages 295–302, Torino, Italy, 27 June 2006. IEEE.

[85] Ricardo Marau, Luís Almeida, Paulo Pedreiras, M. González Harbour, Daniel Sangorrín, and Julio M. Medina. Integration of a flexible network in a resource contracting framework. In *Proc. of the WiP session of the 13th Real-Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, 3 April 2007.

[86] Ricardo Marau, Luís Almeida, Paulo Pedreiras, M. González Harbour, Daniel Sangorrín, and Julio M. Medina. Integration of a flexible time triggered network in the FRESCOR resource contracting framework. In *Proc of the 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'07)*, Patras, Greece, 25 September 2007. IEEE.

[87] Ricardo Marau, Luís Almeida, Paulo Pedreiras, and Thomas Nolte. Towards Server-based Switched Ethernet for Real-Time Communications. In *Proc. of the WiP session of the 20th Euromicro Conference on Real-Time Systems(ECRTS'08)*, 2 July 2008.

[88] Ricardo Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and Thomas Nolte. Server-based Real-Time Communications on Switched Ethernet. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS-RTSS'08)*, 2008.

[89] Michael W. Marcellin, Ali Bilgin, Michael J. Gormish, and Martin P. Boliek. An Overview of JPEG-2000. In *DCC '00: Proceedings of the Conference on Data Compression*, page 523, Washington, DC, USA, 2000. IEEE Computer Society.

[90] J. Martínez, M. González Harbour, and J. Gutiérrez. RT-EP: real-time Ethernet protocol for analyzable distributed applications on a minimum real-time POSIX kernel. In *Proceedings of the 2st Int. Workshop on Real-time LANs in the Internet age (RTLIA'03)*, July 2003.

[91] R. Moghal and M. S. Mian. Adaptive QoS-based resource allocation in distributed multimedia systems. In *Proceedings of Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2003.

[92] Jami Montgomery. A Model for Updating Real-Time Applications. *Real-Time Systems*, 27(2):169–189, 2004.

[93] Klara Nahrstedt and Jonathan M. Smith. The QoS Broker. *IEEE Multimedia*, 2:53–67, 1995.

[94] K. Nichols, S. Blake, F. Baker, and D. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474, December 1998.

[95] T. Nolte. *Share-Driven Scheduling of Embedded Networks.* PhD thesis, Department of Computer and Science and Electronics, Mälardalen University, Sweden, May 2006.

[96] J. C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 26–37, 1998.

[97] J. C. Palencia and M. González Harbour. Response time analysis of EDF distributed real-time systems. *Journal of Embedded Computing (JEC)*, 1(2):225–237, November 2005.

[98] J.C. Palencia and M.G. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 3–12, July 2003.

[99] Paulo Pedreiras. *Supporting Flexible Real-Time Communication on Distributed Systems.* PhD thesis, University of Aveiro, Aveiro, Portugal, 2003.

[100] Paulo Pedreiras and Luís Almeida. *Approaches to Enforce Real-Time Behavior in Ethernet.* CRC Press, 2005.

[101] Paulo Pedreiras, Paolo Gai, Luís Almeida, and Giorgio C. Buttazzo. FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *IEEE Transactions on Industrial Informatics*, 1(3):162–172, August 2005. ISSN: 1551-3203.

[102] Paulo Pedreiras, Ricardo Leite, and Luis Almeida. Characterizing the Real-Time Behavior of Prioritized Switched-Ethernet. In *Proc. of the 2nd Workshop on Real-Time LANs in the Internet Age (RTLIA'03 satellite of ECRTS'03)*, July 2003.

[103] D. Prasad, A. Burns, and M. Atkins. The valid use of utility in adaptive real-time systems. *Real-Time Systems*, 25(2-3):277–296, 2003.

[104] Ala' Qadi, S. Goddard, Jiangyang Huang, and S. Farritor. A performance and schedulability analysis of an autonomous mobile robot. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 239–248, July 2005.

[105] Prasad Raja and Guevara Noubir. Static and dynamic polling mechanisms for fieldbus networks. *ACM Operating Systems Review*, 27(3):34–45, 1993.

[106] J. Ribas-Corbera and Shawmin Lei. Rate control in DCT video coding for low-delay communications. *Circuits and Systems for Video Technology, IEEE Transactions on*, 9(1):172–185, Feb 1999.

[107] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP), RFC 2326, April 1998.

[108] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications, RFC 1889, 1996.

[109] V.M. Sempere and J. Silvestre. Multimedia applications in industrial networks: integration of image processing in Profibus. *IEEE Transactions on Industrial Electronics*, 50(3):440–448, June 2003.

[110] S. Sen, J. L. Rexford, J. K. Dey, J. F. Kurose, and D. F. Towsley. Online smoothing of variable-bit-rate streaming video. *IEEE Transactions on Multimedia*, 2(1):37–48, 2000.

[111] Lui Sha and Shirish S. Sathaye. A systematic approach to designing distributed real-time systems. *IEEE Computer*, 26:68–78, 1993.

[112] C.P. Shelton and P. Koopman. Improving system dependability with functional alternatives. In *Procedings of Int. Conf. on Dependable Systems and Networks (DSN'04)*, pages 295–304, June 2004.

[113] Javier Silvestre, Luís Almeida, Ricardo Marau, and Paulo Pedreiras. Dynamic QoS management for multimedia real-time transmission in industrial environments. In *Proc. of the 12th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA'07)*, September 2007.

[114] T. Skeie, S. Johannessen, and O. Holmeide. The road to an end-to-end deterministic Ethernet. In *Proceedings of the 4th IEEE Int. Workshop on Factory Communication Systems (WFCS'02)*, pages 3–9, 2002.

[115] Yeqiong Song, Anis Koubaa, and Loria Inria Lorraine. Switched Ethernet for real-time industrial communication: Modelling and message Buffering delay evaluation. In *Proc. of the 4th Int. Workshop on Factory Communication Systems (WFCS'02)*, pages 27–30. Springer-Verlag, August 2002.

[116] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[117] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *Real-Time Systems*, 1:27–60, 1989.

[118] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proc. of the 15th IEEE Int. Real-Time Systems Symposium (RTSS'94)*, pages 2–11, San Juan, Puerto Rico, December 1994. IEEE Computer Society.

[119] J.A. Stankovic, Chenyang Lu, S.H. Son, and Gang Tao. The case for feedback control real-time scheduling. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, pages 11–20, 1999.

[120] John A. Stankovic and K. Ramamritham. *Tutorial: hard real-time systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1989.

[121] John A. Stankovic, Krithi Ramamritham, and Marco Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[122] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking (TON)*, 6(5):611–624, Oct 1998.

[123] 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*, 44(1):73–91, January 1995.

[124] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE Trans. Comput.*, 44(1):73–91, 1995.

[125] J.-P. Thomesse. Fieldbus Technology in Industrial Automation. *Proceedings of the IEEE*, 93(6):1073–1101, June 2005.

[126] J.P. Thomesse. Time and industrial local area networks. In *Proceedings of Computers in Design, Manufacturing, and Production (COMPEURO'93)*, pages 365–374, May 1993.

[127] K. Tindell, A. Burns, and A. Wellings. Analysis of hard real-time communications. *Real-Time Systems*, 9:147–171, 1995.

[128] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing & Microprogramming*, 40:117–134, 1994.

[129] Jyi-Chang Tsai. Rate control for low-delay video using a dynamic rate table. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(1):133–137, Jan. 2005.

[130] Bobby Vandalore, Wu chi Feng, Raj Jain, and Sonia Fahmy. A survey of application layer techniques for adaptive streaming of multimedia. *Real-Time Imaging*, 7(3):221–235, 2001.

[131] S. Varadarajan. Experiences with EtheReal: a fault-tolerant real-time Ethernet switch. In *Proceedings of the 8th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA'01)*, pages 183–194 vol.1, October 2001.

[132] S. Varadarajan and T Chiueh. EtheReal: A host-transparent real-time fast Ethernet switch. In *Proc of the 6th Int. conference on network protocols*, pages 12–21, October 1998.

[133] P. Veríssimo and Luís Rodrigues. *Distributed systems for system architects*. Kluwer Academic Publishers, 2001.

[134] N. Wang, D. Schmidt, K. Parameswaran, and M. Kircher. Towards a reflective middleware framework for QoS-enabled CORBA component model applications. IEEE Distributed Systems Online special issue on Reflective Middleware (Vol. 2, No. 5), May 2001.

[135] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[136] L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP)*. RFC 2205, September 1997.

[137] H. Zimmermann. OSI reference model: The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.