Universidade do Porto

Faculdade de Engenharia

FEUP

# Intermediate-Level Protocols to Provide Quality of Service in Master/Slave Communication Infrastructures

By

**Filipe de Faria Pacheco**

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor in Electrical and Computer Engineering

November 2008

**Doctoral Committee:**

| | |
|---|---|
| Prof. Fernando Lobo PEREIRA | Chairman |
| Prof. Thilo SAUTER | External Examiners |
| Prof. Luís ALMEIDA | |
| Prof. Manuel Pereira RICARDO | Internal Examiners |
| Prof. Paulo PORTUGAL | |
| Prof. Eduardo TOVAR | Supervisor |
| Prof. Pedro Ferreira do SOUTO | Co-Supervisor |

# Intermediate-Level Protocols to Provide Quality of Service in Master/Slave Communication Infrastructures


*Abstract*

Industrial communication networks have suffered a dramatic change over the last decades. There has been a proliferation of “traditional” fieldbuses and other more application-specific networks, such as the ones relying on power-line communications. Industrial Ethernet solutions have gained a significant market share too. Due to the stringent quality-of-service (QoS) requirements of industrial monitoring and control applications, most of the protocols for this type of applications rely on a master/slave paradigm where one or more master stations control the access to the communication medium, granting medium access to slave stations.

This Thesis was developed in synergy with the RFieldbus and REMPLI European Union projects. Although operating at opposite network scales, they share one main characteristic: a master/slave network was enhanced with communication features previously unavailable.

In the context of the RFieldbus framework, a standard fieldbus network was extended to support multimedia services and wireless/mobility capabilities. These multimedia services run over the TCP/IP stack that in turn runs over the fieldbus low layers network protocols. Since these multimedia services and the “traditional” control traffic converge in the use of the same communication medium, appropriate admission control and scheduling mechanisms were conceived to introduce different traffic classes, in such a way that real-time control traffic is not affected by multimedia traffic which in some cases is typically of best-effort type.

The REMPLI approach is based on a power-line communication protocol that was enhanced with additional capabilities such as the ability of supporting large-scale deployments - both in terms of number of network stations and in terms of geographical area under coverage - and new metering-focused end-to-end services. This required a rethinking of the Data Link, Network and Transport Layer protocols in a cross-layered perspective that had end-to-end QoS requirements in mind.

The initial hypothesis was that providing add-ons to existing protocols to achieve the required level of QoS and additional functionalities would present major advantages over all-new network protocols or using stripped-down versions of existing network protocols. This hypothesis is confirmed through experimental and simulation validations.



***Keywords***: Real-time Systems, Master/Slave Networks, Quality-of-Service, Fieldbus, Power-Line Communications, Network middleware, Cross-Layered Design

# Protocolos de Nível Intermédio para Conferir Qualidade de Serviço a Infra-estruturas de Comunicação Baseadas em Protocolos Mestre/Escravo


*Resumo*

As redes de comunicação industrial passaram por uma mudança extraordinária nas últimas décadas, observando-se uma proliferação de redes "tradicionais" e outras redes mais específicas, tais como as baseadas em comunicação pela rede de energia eléctrica. A chamada *Ethernet* Industrial também conquistou uma representatividade significativa em poucos anos. Os rigorosos requisitos de qualidade de serviço (QoS) das aplicações industriais de controlo e monitorização levaram a que muitas destas soluções se baseassem no paradigma mestre/escravo, segundo o qual, uma ou mais estações mestre controlam o acesso ao meio de comunicação, concedendo então acesso às estações escravo.

Esta Tese foi desenvolvida em sinergia com os projectos europeus RFieldbus e REMPLI. Embora de certa forma situados em extremos opostos do espectro das redes de dados, as redes utilizadas no âmbito desses dois projectos possuem uma característica comum: redes com controlo do acesso ao meio baseado no paradigma mestre/escravo foram utilizadas como base para a introdução de novas funcionalidades de comunicação.

Na abordagem RFieldbus, uma rede de comunicação industrial normalizada foi actualizada com mecanismos inovadores que permitiram suportar serviços multimédia e funcionalidades de rede sem fios/mobilidade. Os serviços multimédia operam sobre a pilha de protocolos TCP/IP que por sua vez opera sobre os protocolos de nível baixo da rede de comunicação industrial. Tendo em conta que estes novos serviços e o tráfico de controlo "tradicional" convergem na utilização de um mesmo meio de comunicação, foi necessário desenvolver mecanismos de controlo de admissão e escalonamento de tráfego apropriados de modo a introduzir diferentes classes de tráfico, permitindo assim que o tráfego de controlo tempo-real não seja afectado pelo tráfico multimédia.

A abordagem REMPLI foi baseada num protocolo de comunicação de dados pela rede de energia eléctrica, o qual foi complementado com funcionalidades adicionais adequadas ao suporte de redes de grande dimensão – quer em termos de número de estações de rede, quer em termos de distribuição em área geográfica – e novos serviços focados na gestão da distribuição de energia. Estes melhoramentos obrigaram a um reequacionar dos diversos níveis da pilha protocolar numa perspectiva holística e tendo em conta os requisitos de QoS entre os pontos de disponibilização dos serviços.

A hipótese validada por esta tese é a de que estendendo os protocolos existentes de forma a atingir os níveis requeridos de qualidade de serviço e de funcionalidades, resulta em vantagens relevantes quando comparado com soluções alternativas baseadas em sistemas de rede desenhados de raiz ou usando versões reduzidas de redes de comunicação existentes. A hipótese foi confirmada através de validação experimental e simulação.



***Palavras-chave***: Sistemas de Tempo-real, Redes mestre/escravo, Qualidade de serviço, Serviços de comunicação de dados, software de nível intermédio para redes de dados.

# Protocoles de Niveau Intermédiaire pour offrir Qualité du Service en Infrastructures Maître/Esclave de Communication


*Résumé*

Les réseaux de communication industriels ont connu une évolution notable durant ces dernières années du fait de la prolifération de réseaux traditionnels et d'autres applications réseaux spécifiques tels que celles basées sur les réseaux électriques. Les solutions Ethernet industrielles ont également remporté un gain significatif du marché au sein de quelques années.

Toutefois, les exigences strictes de qualité de services (QdS) pour les applications de contrôle industriel ont notamment conduit à un grand nombre de solutions reposant sur le paradigme Maître/ Esclave. De ce fait, un ou plusieurs machines Maître stations contrôlent l'accès aux moyens de communication, tout en garantissant un accès aux stations de type Esclave.

Cette thèse a été développée en collaboration avec les projets Européens RFieldbus et REMPLI. Bien que ces réseaux soient situés à l'extrémité opposée de la gamme de réseaux ordinaires de données, ils ont une caractéristique en commune présentée par l'amélioration des réseaux Maître/ Esclave par des configurations de communication précédemment invalides.

Dans le Project RFieldbus, un réseau de communication industriel normalisé a été étendu pour le support des services multimédia et des fonctionnalités réseaux sans fil/ mobilité. Les services multimédia fonctionnent sous TCP/IP qui, à son tour, fonctionne sur le réseau de communication industrielle.

Etant donné que ces nouveaux services et le trafic de contrôle "traditionnel" existant dans le même moyen de communication, il s'avère important de concevoir et de mette en place des mécanismes appropriés de commande et d'ordonnancement d'admission pour définir différentes classes du trafic, de telle manière que le trafic de contrôle temps réel ne soit pas affecté par le trafic de multimédia de meilleur-effort.

Le projet REMPLI, construit sur la base d'un protocole de communication de données pour le réseau électrique, a été complété par des fonctionnalités supplémentaires. A travers ces nouveaux fonctionnalités, de nouveaux services axés sur la gestion de la distribution d'énergie ainsi que le support des déploiements à grande échelle que ce soit en termes de nombre de stations du réseau, que en termes de répartition par une vaste zone géographique ont été mis en place.

Ceci a exigé une révision des couches liaison de données, réseau et transport dans une perspective multi-couche tout en tenant compte des conditions de QdS bout en bout. L'hypothèse initiale était l'extension des protocoles existants pour atteindre le niveau exigé de QdS et les fonctionnalités additionnelles présenteraient des avantages importants par rapport aux nouveaux protocoles ou à ceux existants. Cette hypothèse a été confirmée par des validations expérimentales et des simulations.



***Mots-clés***: Réseaux Maître/Esclave, Qualité de Service, Services de Réseaux de Communications

# Acknowledgements

It is always hard to find the most adequate words to show appreciation to all that made this huge task possible. To all those that are not in this few lines and feel that they should be, please accept my most sincere apologies and gratefulness for their encouragement.

First, I would like to thank my supervisor, Eduardo Tovar, for his leadership and his reviewing work. I would also like to express my gratitude to the support provided by my co-supervisor, Pedro Souto.

My deepest gratitude also goes to Mário Alves and Miguel Pinho for their friendship and reviewing work.

Additional acknowledgments to the RFieldbus team Nuno Pereira, Sandra Machado, Luis Ferreira (ISEP), Heiko, Jörg and Lutz (IFAK), Klaus, Gerhard (Siemens), Peter and Thomas (Softing), Αθανάσιος, Χρήστος and Σταύρος[1] (ISI) and Frabrice (ST2E).

The REMPLI team also deserves my sincere thank you: Luis Marques, António Barros (ISEP), Maksim, Albert, Gerhard and Thilo (ICT), Peter (TCE), Greg, Jüergen and Xavier (iAd), Raul and YeQiong (Loria).

I would also like to extend my most friendly appreciation to Isabel Praça, Miguel Losa, Berta Batista, Veríssimo Lima and Armando Sousa for their unconditional support.

Finally, a special thank you note goes to my family and friends, mainly to João but also extended to my brothers and parents. Without them, it would all have been useless…

Formal Acknowledgments:

This work was partly supported by the RFieldbus (IST-1999-11316) and the REMPLI (NNE5-2001-00825) European projects, by the Portuguese Science and Technology Foundation (PRODEP) and by my school (ISEP/IPP)

Porto, 24/Nov/2008

Filipe de Faria Pacheco

[1] "Athanassios, Christos and Stavros" in Latin Alphabet transliteration

# Table of Contents

# Table of Figures

# Part I
# Research Context

# Chapter 1
# Overview

There is a growing trend in industrial computer networking to incorporate new services and to provide adequate levels of Quality-of-Service (QoS) as an added-value to the users. This is in some way defines some of the technological context of this Thesis..This chapter presents the context, defines the hypothesis, summarises the main contributions and provides a view of the overall organization of this Thesis.

## *1.1 Introduction*

In the last few decades, there has been a proliferation of fieldbus and other application-specific communication networks. Due to the quality-of-service (QoS) requirements usually imposed by industrial monitoring/control applications, namely timeliness, most of these network protocols rely on master/slave paradigms, where one or more master stations control medium access, granting other stations (namely slaves) permission to acknowledge or to respond to master station's requests.

In many situations, it is preferable to extend an existing network technology to support additional services/functionalities, rather than designing new solutions from scratch. This idea forms the baseline for this Thesis.

In this Thesis, two distinct industrial communication frameworks were re-designed to support functionalities that were previously unavailable, in a way that the original applications quality of service requirements would still be respected.

Firstly, a well-known fieldbus protocol – Profibus[2] (PROFIBUS, 2008) – was redesigned and extended to support multimedia TCP/IP (Transport Control Protocol / Internet Protocol) applications, without interfering with the timeless requirements of the control traffic. That research framework is described in Section 1.2 of this chapter.

Secondly, a power-line communication-based energy management system was also re-designed, extended and adapted to be able to provide end-to-end quality of service in large-scale deployments. That research framework is briefly described in Section 1.3.

In this chapter the Hypothesis is stated in Section 1.4, and the contributions of this research work are summarized in Section 1.5.

[2] The official representation is "PROFIBUS", all caps. In this thesis, we will in most cases a more text-friendly version with only the capital "P" to refer to the same standard.

## *1.2 Factory Automation Scenario*

The factory-floor looks rather hermetic to innovative technologies, eluding the widespread usage of the so-called gadgets in everyday such as cellular phones, personal data assistants and digital cameras, even with less technological-aware users.

After the fieldbus revolution on the 80's in the last century, the factory-floor has seen an increased use of more and more powerful programmable logic controllers and user interfaces, but the way they are used remains almost the same. Too many times the "new" graphical user interfaces are simple copies of the previous synoptic boards with light bulbs and buttons replaced by pixels on a screen. We believe (Pacheco and Tovar, 2002) however that new user-computer interaction techniques, including multimedia and augmented reality combined with now affordable technologies such as wearable computers and wireless networks, can change the way the factory personal work together with the machines and the information system on the factory-floor. This new age is already in place with innovative uses of communication networks on the factory-floor either using "standard" networks (Pokam *et al.*, 1995) or through enhanced industrial networks with multimedia (Tovar *et al.*, 2001) and wireless capabilities (Pereira *et al.*, 2001).

The RFieldbus project (RFieldbus Project, 2008) aimed at facilitating the use of these solutions by enabling TCP/IP usage over traditional factory-floor fieldbus networks, without detrimental side effects on the original (control-oriented) network usage.

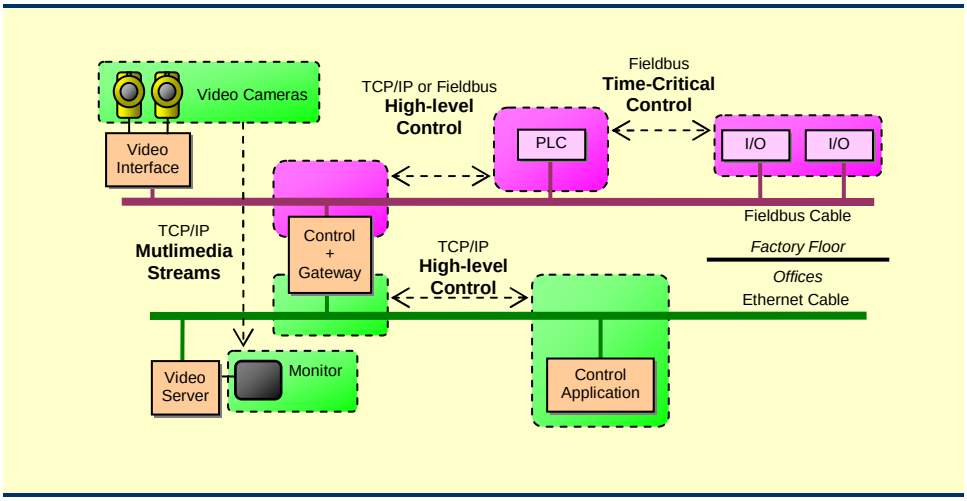



**Figure 1.1: Example of a System TCP/IP services coexisting with fieldbus services**

To illustrate some of the concepts and challenges, consider Figure 1.1. In the system described, two video cameras are connected to the fieldbus network and eventually the video streams generated by them are then used by a remote video monitoring system in a network station connected to the Ethernet network. Therefore,

there will be video streams eventually "circulating" in the fieldbus network. While these may not impose end-to-end time guarantees, they cannot jeopardize the timeliness guarantees required for the time critical control traffic in the fieldbus network. This imposes some careful design options on how to support this multimedia type of traffic with the fieldbus network.

Also typically, the remote video monitoring applications will use standard TCP/IP Application Program Interfaces (APIs) to communicate. This is a natural application productivity requirement. *Per se* it poses and important challenge into the system design: IP applications are typically symmetric in the sense that any IP network node can have communication assess while in typical fieldbus networks some nodes (e.g., slaves) will not have communication initiative.

Also important, typical fieldbus networks are optimized for short messages related to sensor reading or actuation. Conversely, multimedia information such as video or audio involves higher amounts of bytes in simple transactions.

Another functionality that should be supported and is illustrated in Figure 1.1 is a high-level Control Application connected via TCP/IP to a Gateway and the latter connected via a fieldbus to a Programmable Logic Controller (PLC). The time-critical control may be performed within the fieldbus level between the PLC and I/O stations but eventually also through the Gateway between the Control Application and the PLC using TCP/IP over Ethernet and the fieldbus.

Both solutions are possible in the RFieldbus architecture, thus enabling a much greater flexibility at the factory-floor. The I/O stations can be common fieldbus stations that are "unaffected" by the TCP/IP traffic at the fieldbus level.

## *1.3 Energy Distribution Management Scenario*

Like discrete manufacturing companies, utility providers (energy, water, heating, etc.) have also considered using emerging technologies to optimise and improve the set of offered services and at the same time to reduce costs.

One service considered strategic by utility companies is to have cost-effective remote meter reading technologies. Utility companies benefit from these technologies by obtaining detailed information about how energy is consumed by the end-users. They can even take corrective measures since the data can be gathered in real-time (although this can limit the scalability of the system).

In addition, remote metering technologies can also be used to harvest information about the status of the energy distribution grid itself. Based on the availability of fine-grained energy consumption data at the end-users site, the energy flow is easier to control and leakages detected more efficiently. In particular, peak load situations can be better managed with extreme benefits for both the utility providers and the consumers given the fact (Rowlands, 2007) that in peak situations a very small increase in load can have a dramatic effect in the energy cost to the utility company. Additional services such as the remote switching or termination of the supply of energy can also be supported, if required for either management services or services not yet generally available in the market (e.g., pre-paid systems or time-bounded uses).

This type of systems could also be used to provide additional information such as monthly energy cost on the actual metering device and make new interactive services available like user-selectable alternative billing services.

This is the context of the Real-time Energy Management via Power-Lines and Internet (REMPLI) European project (Pacheco *et al.*, 2005a). The project aimed at designing and implementing a communication infrastructure for real-time distributed data acquisition and control operations, exploiting the power-line as the communication medium; therefore exploiting a Power-Line Communication (PLC) system.

According to the overall project goals, the primary usage of this infrastructure is remote meter reading and remote control. Besides that, the communication platform is open to various types of add-on services. Figure 1.2 illustrates the architecture of the communication network.

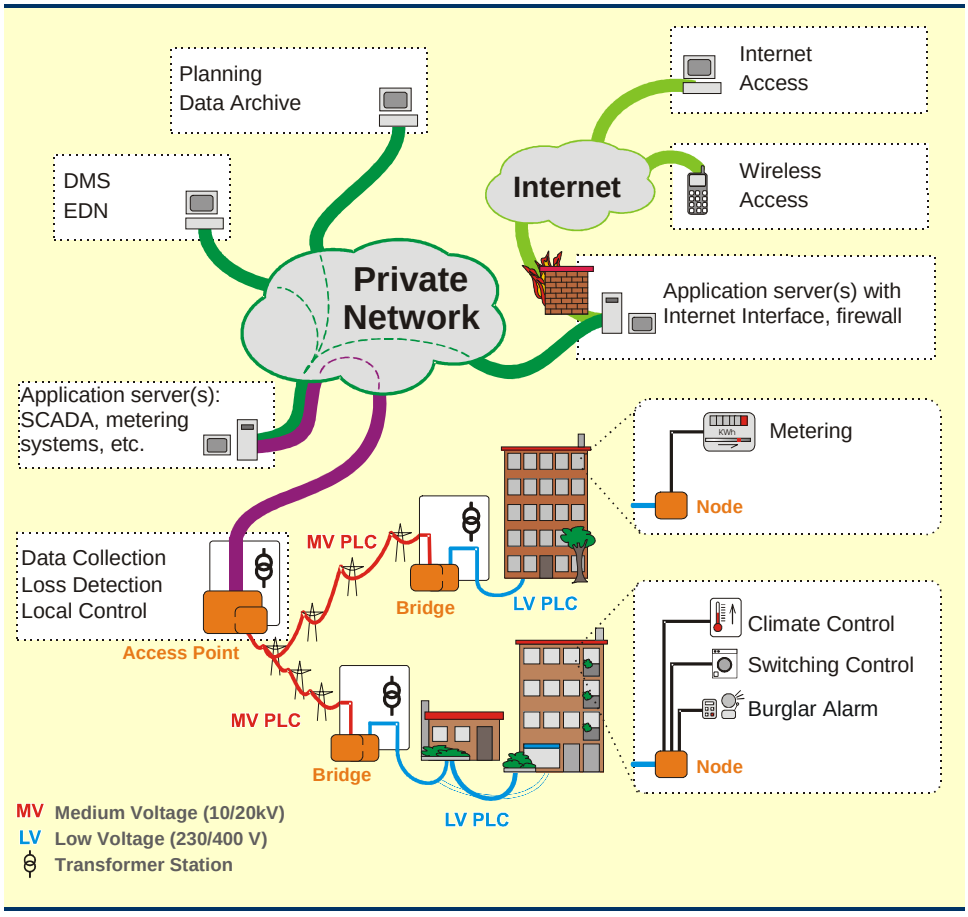



**Figure 1.2: Energy Management System**

There are diverse systems available to access remote information on meters using telephony, radio frequency and satellite communications technologies. However, PLC systems have the clear advantage in the electrical utility market that the medium is available in all client sites.

There is however a number of challenges that needed to be tackled in order to attain such a system. Most of them result from the fact that power-line communications are typically based on robust time-slotted master-slave communication paradigms (Sebeck and Bumiller, 2000) while electrical networks may have rather non-linear and multi-tiered topologies as will be explained in Chapters 3 and 7 of this thesis.

## *1.4 Hypothesis*

The main hypothesis in this thesis is that *providing add-ons to existing protocols to achieve the required level of quality of service and additional functionalities would present major advantages over all-new network protocols or using stripped-down versions of existing network protocols*.

Taking into account the research and technological context briefly presented in the previous sections, the approach was to devise in fact the appropriate solutions to network ensembles built upon existing and proved low-level master-slave communication network mechanisms. As can be inferred from the rest of this Thesis, this hypothesis is confirmed through the design of novel architectures that are validated through actual implementations and simulations.

## *1.5 Research Contributions*

This thesis contains a number of important contributions. As it will become clear in Section 1.6, this thesis is organised into two main parts, one related to the contributions for factory communications and another related to power-line communication systems. Therefore, the contributions are organized according these to main parts.

Concerning factory communications, the main contributions are the following:

1. *The specification of a dual stack architecture that provides traffic independence between TCP/IP applications and native Profibus applications.* This architecture also provides transparent access of both types of applications to the network allowing for rapid deployment of mixed systems.
2. *A system that provides TCP/IP traffic selection and system-wide scheduling capabilities.* Traffic generated by TCP/IP applications can be divided in best effort or in several classes with diverse quality of service parameters. These levels of service are guaranteed even if the TCP/IP application resides on a slave (Profibus) station.
3. *A methodology for determining the parameters for setting up a RFieldbus-like network.* The correct operation of a RFieldbus-like system relies on the proper setting of several configuration parameters that influence the timeliness properties and overall performance of the system.

4. *The definition of a test system architecture* for the contributions listed above providing validation of the results.

Regarding the energy management communication system, the main contributions of this thesis are the following:

5. *The design of a dual-level network architecture that is easily deployed in the last two voltage levels of most electrical power distribution grids*. This architecture provides transparent access to stations regardless of their location in the grid.
6. *The definition of a set of utility-oriented services that are scalable not only concerning the number of stations reachable but also concerning the deployment over large geographical areas*. Features like a flat address space, a run-time editable mapping mechanism between stations and the address space and the close integration of the base network services enable an end-to-end set of services focused on metering applications not only for electrical power but also on other utilities.
7. *The deployment of a distributed traffic-selection mechanism*. The base system forwards remote queue and link quality information to the network entry points. This "global view" of the network is then used to select the best route for requests. The scheduler can be easily upgraded and tested using the simulation/emulator tools developed within the timeframe of the project.
8. *The definition of a development/simulation architecture* where the same transport layer code can be used for simulation tests and end-device execution enables not only an easier deployment but also future developments and research.

## *1.6 Structure of the thesis*

This thesis is structured as follows.

There are essentially 4 parts. The first part is devoted to research context. It includes Chapter 1 (Overview), Chapter 2 (Related Work on Factory Communications) and Chapter 3 (Related Work on Power-Line Communication Systems).

Chapter 2 provides a description and discussion of research issues and technology related to the factory communications research framework developed in this thesis. Besides a general background on fieldbus networks and issues related to the interoperability of these with TCP/IP networks, the chapter provides, with some emphasis, details on Profibus networks. In fact, Profibus is used as the basis for the RFieldbus approach.

Chapter 3 provides also research and technology context for energy management communication systems. Likewise the previous chapter, related work is described and discussed. Emphasis is given to the base power-line network used as the building block for the wider approach. Intended services to provide to higher layers by the transport layer are discussed as well. Finally, since most of the approach is validated also through simulation, the OMNeT++ simulation system, over which the transport layer development was built and validated, is briefly presented.

The second part (Part II) of this thesis is devoted to the contributions related to the factory communications framework. It includes Chapter 4 (Protocol Stack Architecture), Chapter 5 (Other Design and Implementation Issues) and Chapter 6 (Validation).

In Chapter 4, the main concepts of the (dual) protocol stack are presented, including the various sub-layers (IP Mapper, DP Mapper, IP ACS, and the DP/IP Dispatcher) used to achieve the required functionalities at the required qualities of service. The slave initiative mechanism, traffic classes, IP fragmentation, scheduling, routing and other mechanisms that allow the coexistence of Profibus and (tunnelled) IP traffic in the same bus are discussed, proposed and detailed.

Chapter 5 deals with implementation specificities of the concepts and mechanisms described and proposed in Chapter 4.

Finally, to close Part II, Chapter 6 describes a pilot implementation and field trial tests that used to validate the proposed system. The results of the tests and respective conclusions are discussed accordingly.

The contributions related to the energy management communication system framework of this thesis are organized in Part III, which has basically the same chapter organization of Part II.

Therefore, in Chapter 7 the main concepts related to the transport layer is discussed and novel solutions are proposed. The routing and network topology information gathering system is described, followed by a specific solution for the slave initiative issues and the distributed scheduling mechanisms. The alarm service, due to its specificity is also discussed in that chapter.

In Chapter 8 the main implementation-related issues are dealt with, including the base network services and the mixed simulation/development system based on OMNeT++. Fragmentation issues are given additional emphasis stress the difficulties resulting from the limited resources available in the stations and network that occur typically with the extreme range of packet lengths therefore leading to very specific PDU header structures. Implementation options of the algorithms for the main services provided by the transport layer are discussed.

Finally, Chapter 9 deals with the validation of the system done via simulation scenarios. Several tests and related results are presented and discussed.

The thesis concludes with Part IV, that summarizes, in Chapter 10, the various contributions, and, on Chapter 11, potential future work.

# Chapter 2
# Related Work on Factory Communications

An overview of technologies related to factory automation networks is provided in this chapter. The first section presents the historic context of fieldbus networks followed by details on the diverse standards available and the Profibus protocol in particular.

## *2.1 Overview of fieldbus networks*

Since one of the research frameworks of this thesis is factory communication systems based on a fieldbus network, we first start with a historic perspective of the area.

Typically a computer-controlled system can be decomposed into a set of three subsystems: the controlled object; the computer system; and the human operator (Kopetz, 1997). The job of the computer system is to react to stimuli from the controlled object or the operator. The computer system should be able to accept status data of the controlled object, compute new instructions according to the references provided by the user, and transmit those new commands to actuators.

A computer-controlled system can have a centralised architecture, with the field devices (e.g., sensors and actuators) connected to the computer system via point-to-point analogue or digital links. In traditional systems, there was a main control box in a central location and wires were connected to each sensor and actuator using analogue signals. The analogue signals had problems with limited distance, wire-to-wire noise and lack of unified protocols. The usage of digital links made it possible to cover much larger distances and reduce the noise problems dramatically. It also made much easier to use digital processing units on the central location. However, the protocol problem remained since each manufacturer had its own digital protocol making device integration from different manufacturers difficult or even impossible.

In addition, the wiring issue was not solved: even with digital links, there where still end-to-end wires connecting each device to the central box. After the digital end-to-end link, the obvious leap was to use a digital bus network. The main advantages (Thomesse, 2005) include lower installation and maintenance costs, bidirectional communication, more accurate information, easier interface to the data (possible using handheld devices on the field), and easier expansion due to the modular nature of the network. The ability to support distributed control algorithms is another advantage achievable by the use of field level networks.

Most computer-controlled systems are also real-time systems, e.g. systems that must react within a pre-defined maximum delay. In general, the issue of guaranteeing real-time requirements is one of checking, prior to run-time, the feasibility of the system's task set; that is, checking if the worst-case response time of the tasks is smaller than their admissible response time. In distributed computer-controlled systems, where some of the application tasks are also communicating tasks, one has to take into account the transmission delays when considering the real-time characteristics of the system.

The timing constraints needed to guarantee real-time characteristics in fieldbus networks have, however, some drawbacks that should be taken into account when using them. First, there is the planning problem: even the most complex real-time networking protocol will fail if it is overloaded with traffic not planned adequately. Secondly, there are the efficiency issues: in order to guarantee the maximum delays, packets have to be limited in length; and this leads to a significant overhead when using the fieldbus network to send large data payloads. Scheduling and dispatching techniques help on the first point making it easier to guarantee that the planned traffic does not affect the real-time characteristics of the network. The second point is not normally taken into account since these networks where designed precisely to send very small packets very quickly. However, new applications are pushing the networks to the limit and efficient solutions for this issue are possible.

A solution that has very low cable costs is a "bus network": all stations are electrically interconnected to the data wires of the network. The physical layout of a bus network is normally very flexible improving even further the cabling costs including line, star and tree topologies. In a bus network, Protocol Data Units (PDUs) are transmitted from a source station to destination station(s) via the shared communication medium. As in any broadcast network, it is necessary to control – using a Medium Access Control (MAC) mechanism – the situations where two or more stations attempt to send PDUs via the shared medium at about the same time.

Considering this scenario there are several solutions for the MAC that can be used: Time-Division Multiple Access (TDMA), Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) and master/slave systems are some of the most used.

On a TDMA network, the bus is divided into time "slots" and the system guarantees that each slot is used by, at most, one network station. Besides the obvious issue of guaranteeing the time synchronization between stations, the assignment of the slots and their duration is a problem with multiple solutions from the simple fixed allocation scheme to complex systems with dynamic allocation of slots in runtime. One of the dynamic allocations schemes is the Timed Token (Malcom and Zhao, 1994), where a token is rotated between stations with timing limitations to guarantee responsiveness and bandwidth usage to each station.

CSMA/BA networks, such as the CAN (Controller Area Network) fieldbus, use another approach: the physical layer has the capability to accept simultaneous transmission of bits and guarantee that one of the logical levels is "dominant" over the other. If two (or more) stations send the same bit, all the receiving stations (including the ones transmitting) receive the bit correctly. However, if the bits are different then the result is always the "dominant" bit. This enables a bit-wise priority mechanism where lower-priority stations will back-off when they detect that their bit transmissions where changed by a dominant bit sent by another station. With the adequate planning, this

protocol guarantees (Tindell and Burns, 1994) real-time message response in the network.

On the other hand, master/slave networks are simple in their approach: one station controls all the communications on the bus and decides when to contact the other stations and if they have or not the chance to send a PDU back. With appropriate timing constrains it is relatively simple to implement a real-time network with this paradigm. Some fieldbus networks extend the basic master/slave concept with extensions enabling multiple-master capability, dynamic network configuration and slave-initiative PDUs under controlled circumstances

## *2.2 Fieldbus standards*

In the beginning of the 1980s several national fieldbus projects where initiated (Thomesse, 2005). In 1982 with the support of the French government the FIP fieldbus, now known as WorldFIP, was presented. In Denmark a set of institutions were involved in the development of P-NET, while in Germany it was the same concerning the Profibus project (PROFIBUS Nutzerorganisation e.V., 1992) in 1984. In terms of industry players, Bosch developed the specifications (Robert Bosch GmbH, 1991) of the Controller Area Network (CAN) in 1983, which was initially targeted to in-vehicle (e.g., cars) applications.

Then, the international level process started within the TC65 of the International Electrotechnical Committee (IEC) that was supported as well by the Instrument Society of America group SP50. However, only in 1993 the physical layer specification was approved (IEC DIS 1158-2) and it did not include the *de facto* standards Profibus and WorldFIP. In 1996, CENELEC decided that it was better to have three standards than no standards at all and soon the European standard (EN 50170) was approved containing three different profiles: part 1 for P-NET (Danish national standard), part 2 for Profibus-FMS (German national standard) and part 3 for WorldFIP (Franch national standard). In 2000 the EN 50170 had an addendum to include Foundation Fieldbus, ControlNet and Profibus-PA.

In 1996, CEN and CENELEC started preparing the EN 50254 under the title *high efficiency communication subsystems for small data packages*. This was approved in 1998 also as a multi-profile document that included Interbus, Profibus-DP and WorldFIP.

In March 1998 part 3 (Data Link Service Definition), part 4 (Data Link Protocol Specification) and parts 5 and 6 (Application Layer Service and Protocol) of IEC FDIS 61158 were submitted to a vote and not approved. However, 6 of the negative votes were later discarded due to being justified by general, not technical opinions and merely untrue statements (Instrument Society of America, 1999), and so the document was approved in November 2000. On this standard there where 8 Types of non-interoperable link layer networks (defined in parts 3 and 4): Type 1 – Proposed compromise (Foundation Fieldbus based); Type 2 – ControlNet; Type 3 – Profibus (including DP, PA

and FMS); Type 4 - P-NET; Type 5 - Fieldbus Foundation[3]'s High-Speed Ethernet (HSE); Type 6 – SwiftNet; Type 7 – WorldFIP; Type 8 – Interbus. In addition to these eight Types, there are two additional Types for the Application Layer standard (defined in parts 5 and 6): Type 9 – Foundation Fieldbus H1 and Type 10 – PROFInet.

Outside the IEC 61158 are sensor networks that were included in the IEC 62026 standard including DeviceNet, SDS, CANOpen and AS-i. On Europe EN 50325 included also profiles derived from the CAN protocol (DeviceNet, SDS, CANOpen) and EN 50295 defines the actuator and sensor protocol (AS-i),

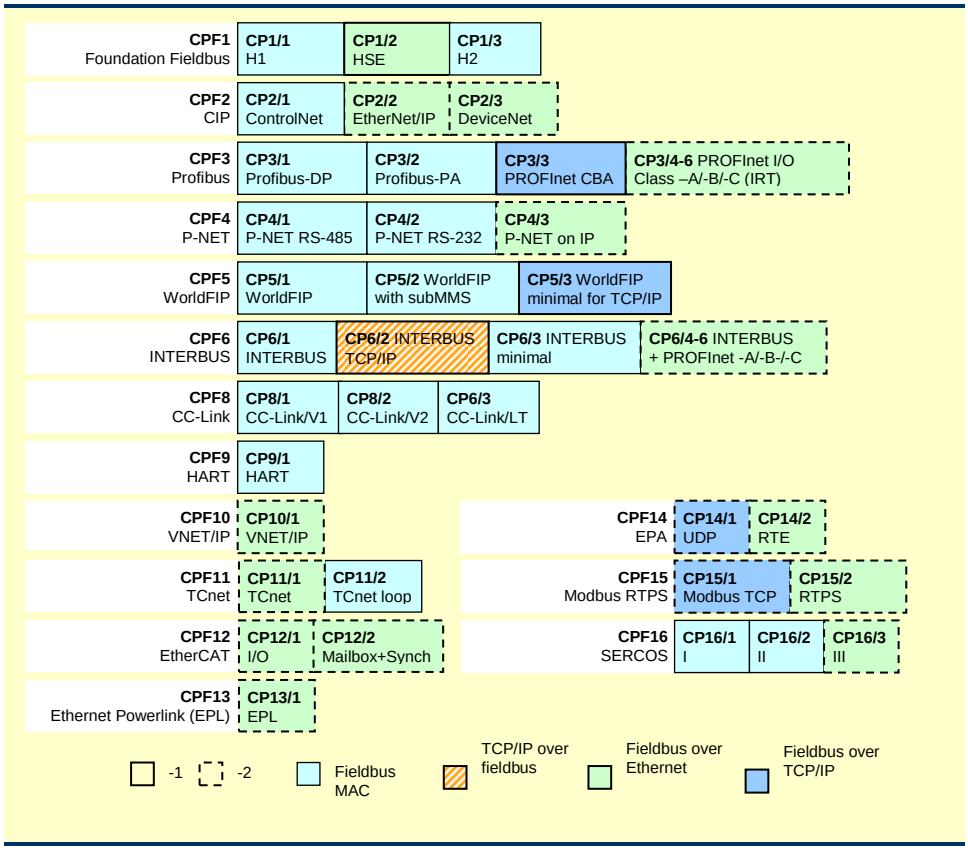



**Figure 2.1: IEC 61784-1 (2nd Ed.) and -2 (1st Ed.) communication profiles**

One issue with the IEC 61158 standard was that each fieldbus could have features in each layer assigned to different types. The solution for this problem was the EN 61784 where Communication Profile Families (CPF) specifies the complete stack of each

[3] "Fieldbus Foundation" is the organization that promotes the "Foundation Fieldbus" communications standard

fieldbus based on the technical data in IEC 61158. The first part EN 61784-1 covers "current" CPFs and was approved in 2003, for example, the Foundation Fieldbus is CPF 1, with H1 specified in CPF 1/1, and HSE specified in CPF 1/2. In mid-2003 the work started on IEC 61784-2 that includes "new" CPFs including PROFInet, EtherCAT, etc.

IEC 61158 was revised in March 2003 with adjustments on existing Types, new Type 6, 9 and 10. The 3rd edition of IEC 61158-2 follows this changes (April-May 2003)

IEC 61158-1 had a 2th edition (November 2007) with removal of Type 6 (SwiftNet), inclusion of Types 11 to 20; generalization of Type 1 radio and sub-division of parts 3, 4, 5 and 6 (e.g. IEC 61158-6-2, etc); IEC 61158-2 4th edition follows this changes (December 2007)

IEC 61784-1 2nd edition (December 2007) synchronizes this standard with 61158:2007 including the addition of new Types: CPF 8 (CC-Link, IEC 61158 Type 18), 9 (HART, IEC 61158 Type 20) and 16 (SERCOS, IEC 61158 Type 16). Also in December 2007, functional safety is included in part 3, and installation issues in part 5.

With the "pulverization" of the fieldbus standards by IEC, it is no surprise that IEC itself provides a CD-ROM with the title "Industrial Communication Networks – Fieldbus – The complete collection" (January 2008), a pack that contains 79 standards covering 15 Communication Profile Families and 20 Types. These include IEC 61158-1, IEC 61158-2, IEC 61158-3-* (DLL Service Specification for Types 1, 2, 3, 4, 7, 8, 11, 12, 13, 14, 16, 17, 18, 19), IEC 61158-4-* (DLL Protocol Specification for Types 1, 2, 3, 4, 7, 8, 11, 12, 13, 14, 16, 17, 18, 19), IEC 61158-5-* (Application Layer Service Specifications for Types 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20), IEC 61158-6-* (Application Layer Protocol Specification for Types 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20), IEC 61784-1 (CPF introduction), IEC 61784-2 (Additional CPF), IEC 61784-3 (General rules for functional safety, and Additional specifications for CPF 1, 2, 3, 6), IEC 61784-5 (Installation profiles for CPF 2, 3, 6, 10, 11) and IEC 61918 (Installation in industrial premises).

The 15 Communication Profile Families in the latest version of IEC fieldbus standards are (see Figure 2.1): 1. Foundation Fieldbus; 2. CIP; 3. PROFInet; 4. PNET; 5. WorldFIP; 6. Interbus; 8. CC-Link; 9. HART; 10. Vnet/IP; 11. TCnet; 12. EtherCAT; 13. Ethernet Powerlink; 14. EPA; 15. MODBUS-RTPS and 16. SERCOS.

Some of these Communication Profile Families share characteristics like same physical layer interface, however most Communication Profiles vary greatly in several other characteristics including:

- Specification Availability – Some fieldbuses have their specifications available for free or for very small fees, this contrasts with the usual cost of an IEC standard (the collection for each CPF costs typically from to 550 EUR to 1800 EUR, the full fieldbus collection costs 7500 EUR).
- Industrial Property Protection – some CPs are covered by patents and other industrial property protection mechanisms and cannot be commercialized without prior licensing.
- Detail – some CPs are defined in almost every detail from physical layer up to functional details, others still have space for big interoperability issues.
- Market share/target – in raw numbers some CPs have huge market shares when compared to others, however this comparisons do not reveal the true "success" of

each fieldbus. They do not take into account different market targets and the fact that some CPs are targeted for specific levels of the industrial process (from factory-wide control to more local/simple station interconnections). Some CPs are targeted for specific areas of the industry (e.g. P-Net in shipbuilding, SERCOS in drives…) or even regionally bounded (EPA documentation was only available in Chinese until recently). Other issue here is that there is a huge difference in market phases of the several CPs: some are well established with multiple products from multiple vendors, others do not have a single implementation in the market.

- National or Regional Standards – from the first 3 European-based national standards there is now an extended collection of standards including national standards from USA (CIP), China (EPA), Japan (TCnet, Vnet/IP).
- Physical Layer – Some CPs are built over TCP/IP stacks, others use dual-stack architectures over Ethernet, some use specific changes in Ethernet MAC, and some use their own physical layer.
- Real-Time characteristics vs. standard MAC chipset – some CPs are designed for very fast and time-guaranteed responses using specific chipsets (cycle time less than 100 µs for EtherCAT, jitter less than 1 µs for some flavors of SERCOS and PROFInet), others have less stringent response times (around 10ms) but are built over garden-variety Ethernet chipsets.
- Application level – features available for applications also vary greatly… solutions range from the application-controlled station poling mechanism to a system-managed subscriber-consumer model and even high-level station profiles.

The Profibus organization announced (PROFIBUS & PROFINET International, 2008a) that more than 1.1 million PROFInet stations where installed by the end of 2007, this value does not include infrastructure devices like switches. It also informed that 4.7 million Profibus stations where sold in 2007 bringing the total number of Profibus stations in the field up to 23 million. In 2004 there where "only" 10 million Profibus stations in the field (2008). 4 million Profibus stations in 2002 (Calandrini, 2003).

On the other hand, Fieldbus Foundation stated in February 2008 (Process Engineering, 2008) that it has 68% of market share in sales values in the "Process Industries" against 27% of Profibus. It also stated that about 1 million stations were installed. The market share in 2006 was the same (Fieldbus Foundation, 2007b) in value.

According to ARC Report in 2006, the market of fieldbus in Process Industries was 831 million USD and forecasted a 2280 million USD for 2011 in a total for Automation Systems of 30 billion USD in 2006 and forecasted 57 billion in 2011 (ABB, 2008)

In 2005 there where 625000 Foundation Fieldbus stations and 10000 systems worldwide (Fieldbus Foundation, 2006), in 2004 there where 500000 stations and 8000 systems (Fieldbus Foundation, 2005).

### *2.3 Profibus overview*

Profibus (PROFIBUS Nutzerorganisation e.V., 1992) was the selected base fieldbus for the factory communication system framework. Like other fieldbuses, it uses a master/slave paradigm for medium access control. There are two distinct stations in the

network: masters that control the access to the medium, and slaves that respond to master requests.

Profibus uses a timed token passing mechanism that supports also multi-master networks. The Profibus-DP standard does not however require inter-master user data communication support. In practice a Profibus-DP network behaves like separated logical networks each with its own master and, if properly configured, without interferences between them.

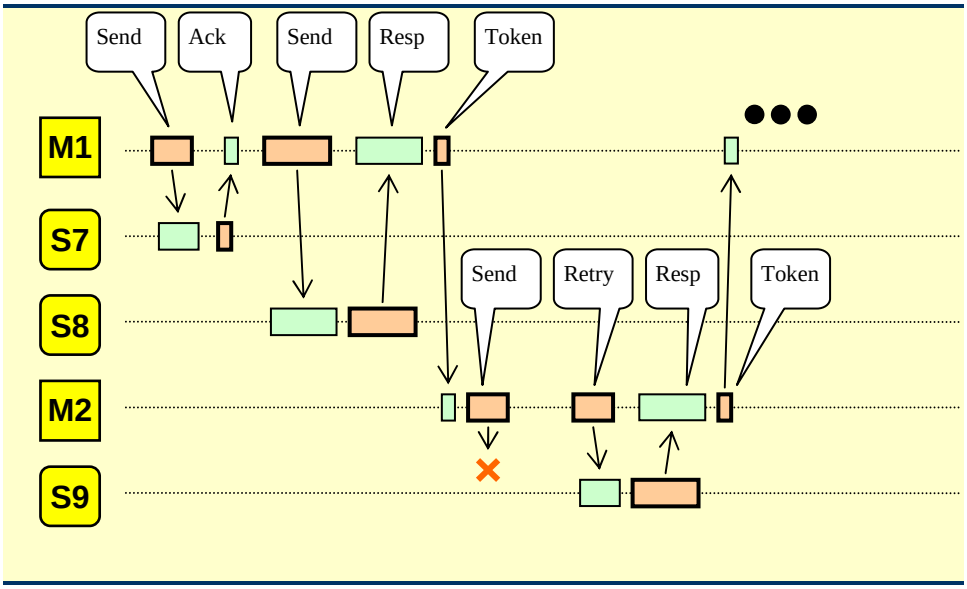



**Figure 2.2: Example Profibus data exchange**

The token, that represents the right to access the bus, circulates in a logical ring composed by the masters (see Figure 2.2).

Profibus allows distinguishing between high priority and low priority PDUs. The latter can further be divided into three subtypes: cyclic low priority PDU cycles (Poll Cycle), that represent the execution of the requests contained in the poll-list; acyclic low-priority PDU cycles, which comprise application and remote management services; and gap maintenance cycles, that are actions taken to determine the status of the others station in order to support dynamic changes in the network.

The medium access control protocol,, the data transfer services as well as the management services are defined according to the standards DIN 19 241-2, IEC 995, ISO 8802-2, ISO/IEC JTC 1/SC 4960 and the all-encompassing IEC 61784 and IEC 61158 standard families, in particular CPF 3/1 (IEC 61784) and Type 3 (IEC 61158) define Profibus-DP stack details. In order to provide transmission synchronism and some redundancy, some characters are encoded in the UART character format: 11 bits with start-stop synchronisation, one data octet and a parity bit.

One important concept on the timing characteristics of Profibus is the *Message Cycle* that includes the *Action Frame* sent by the initiator (always a master) and the corresponding *Acknowledge* or *Response Frame* sent by the responder. After the transmission of the action frame, the initiator waits for a response during the *Slot Time* ($T_{SL}$). If no response is received within that time span then the initiator tries again a number of times up to *max_retry_limit* times.

Profibus has a mechanism to query and update a list of sensors and actuators automatically using a *Poll List*. The processing of the requests in this list is a *Poll Cycle*. The Poll Cycle requests are processed after the *High Priority* PDUs and before the acyclic *Low Priority* PDUs. A Poll Cycle may span several token visits, however only one Pool Cycle is allowed per token visit.

As for the token management, the token is simply passed to the next master in rising address order. The highest address master sends the token to the lowest address one.

In a mono-master network, the token is just passed back to the master, enabling the usage of the same PDU processing mechanisms of multi-master networks. Profibus has also procedures to detect token transmission errors and changes in the number of master stations connected to the network.

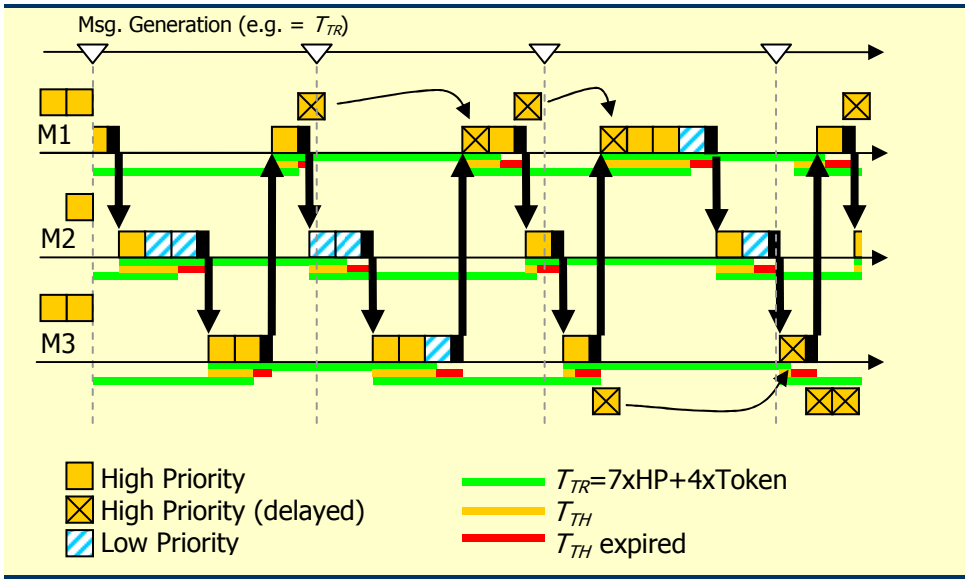



**Figure 2.3: Profibus Low Priority traffic affects High Priority traffic**

When the token arrives, the master computes the *Token Holding Time* ($T_{TH}$), the time available to perform message cycles. This time is the difference between the *Token Target Rotation Time* ($T_{TR}$), the time that the token is expected to take to visit all masters, and the *Token Real Rotation Time* ($T_{RR}$), the effective time measured from the last visit. The master sends one high-priority PDU even if the $T_{TH}$ is negative. The remaining high-priority PDUs are processed until $T_{TH}$ expires.

Message Cycles in Profibus are never interrupted, if $T_{TH}$ expires after the cycle start all the retry processing carries on as usual. After the high-priority PDU queue is empty, then the Poll Cycle requests are processed and, finally, the low-priority acyclic PDUs.

The processing of PDUs in Profibus is simple but leads to some unforeseen results when used in practice even in mono-master networks. Since a message cycle is never interrupted, if $T_{TH}$ expires then the token will be late in the next visit… this in turn means that only one high-priority PDU can be sent. If we have a burst of high-priority PDU the network has (Monforte *et al.*, 2000) a awkward pattern like {token, 1 high-priority PDU, token, *n* high-priority PDUs} until all the high-priority PDUs are exhausted.

In addition, it must be noted that if a high-priority PDU is queued right after the token arrival and there are many low-priority PDUs, then the PDU can be delayed more than $T_{TH}$. For multi-master networks if one of the masters expires $T_{TH}$ then all (Tovar and Vasques, 1999a) the remaining masters see a late token until the token is received again by the master that expired $T_{TH}$ (see Figure 2.3).

In order to prevent the priority inversion, a constrain on the low-priority traffic is proposed by (Tovar and Vasques, 1999b) that avoids late tokens without changing the Profibus MAC. The idea is that if one limits the maximum low-priority traffic at each master station and the $T_{TR}$ is set accordingly, then the token is never late due to low-priority PDUs, and so the high-priority traffic is not affected by the low-priority traffic in the network.

Regarding Profibus services, the Fieldbus Data Link (FDL) provides the functions for sending and receiving data over the network (Data Link Layer functionality). Protocol Data Units (PDU) are packaged, delivered and checked. Acknowledgements, responses, retries and timeouts are used to guard against Line Protocol Errors (e.g., frame, overrun and parity) and Transmission Protocol Errors (e.g., start and end delimiters, frame check, frame length and response times).

A Profibus PDU data payload is restricted to 246 bytes. For most industrial applications, the PDU data size should not exceed 32 bytes to reduce transmission delays. In addition to the data, a PDU of variable length contains an 8-byte header; a PDU of fixed length (8 bytes) has a 6-byte header. Various acknowledgement and response frames are also defined (see Figure 2.4).

The Profibus FDL offers three acyclic and one cyclic data transfer service: Send Data with Acknowledge (SDA); Send Data with No Acknowledge (SDN); Send and Request Data with Reply (SRD) and Cyclic Send and Request Data with Reply (CSRD).

The SDA service allows the initiators to send a PDU and immediately receive the confirmation. The responder can either acknowledge the received data or respond sending data itself. The SDN is an unacknowledged service. Therefore, it is mainly used for multicast or broadcast transmissions.

Finally, the SRD is based on a reciprocal connection between an initiator and a responder, and requires either an acknowledgement or a response. Using this service, the initiator sends data in the request and it receives data from the addressed station in the response.

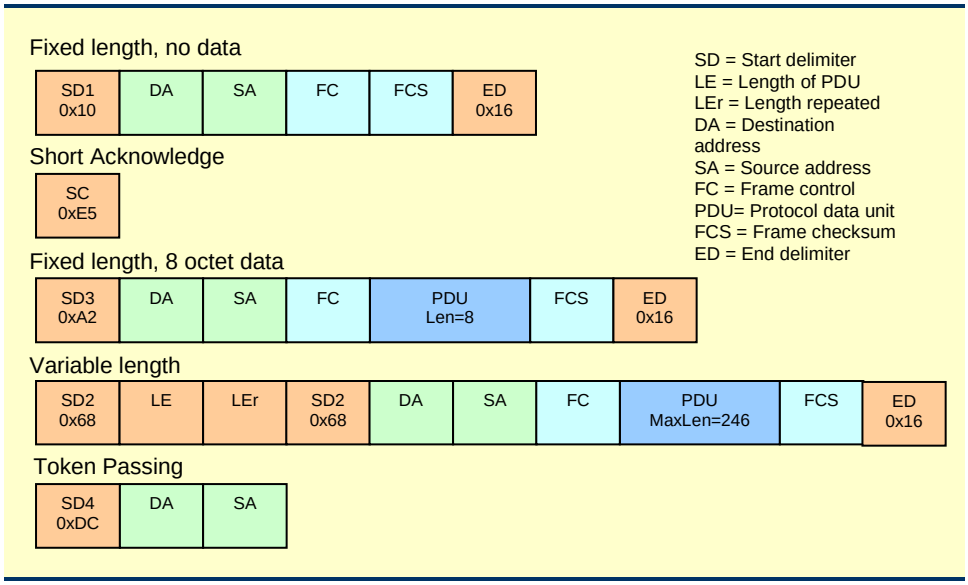



**Figure 2.4: Profibus most used frame formats**

The Profibus FDL layer also offers a cyclic service (based on the acyclic SRD). This service is Cyclic Send and Request Data with reply (CSRD), and is used to poll simple field stations, such as sensors and I/O racks. The list of the stations to be polled is called the *Poll List*.

Stations may have addresses from 0 up to 125; additionally address 127 is used to broadcast PDUs. The eighth bit of the *Address Field* can be used for an extended addressing mechanism used in networks with multiple segments. The FDL supports optional *Service Access Points* identification, which provides similar functionality of the TCP/IP port numbers and station addressing.

## *2.4 Wired/Wireless Profibus Networks*

Systems like RFieldbus extend Profibus not only with new services but also with a new transmission medium: wireless radio. In RFieldbus, the wireless medium has different data rates as compared to wired Profibus. Rfieldbus also introduces additional headers and trailers for the wireless messages. The RFieldbus includes also a mobility support functionality that introduces some time overhead. These aspects are discussed in detail in (Alves, 2003) and are briefly presented in the rest of this sub-section.

In terms of interconnection between heterogeneous medium (wired/wireless), the RFieldbus solution is a physical-level one, using repeater-like interlinking devices. There are alternative approaches such as the ones proposed in (Ferreira, 2005), which use higher-level solutions but then requiring some modifications to the standard operation of the stations.

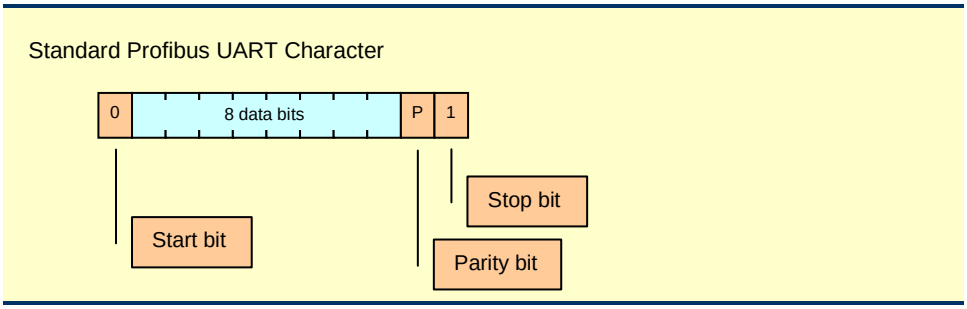



**Figure 2.5: Profibus UART frame**

Profibus specifies that master stations must leave the medium unused between message cycles (see Figure 2.6) for a minimum *Idle Time* ($T_{ID1}$) that is given by:

$$T_{ID1} = \max(T_{SYN} + T_{SM}, \min T_{SDR}, T_{SDI}) \quad \textbf{(2.1)}$$

where:

- $T_{SYN}$ is the synchronisation time, the minimum time interval during which each station must receive idle state from the physical medium (33 bits);
- $T_{SM}$ is a safety margin;
- min $T_{SDR}$ is the minimum station delay of responders
- $T_{SDI}$ is the station delay of the initiator.

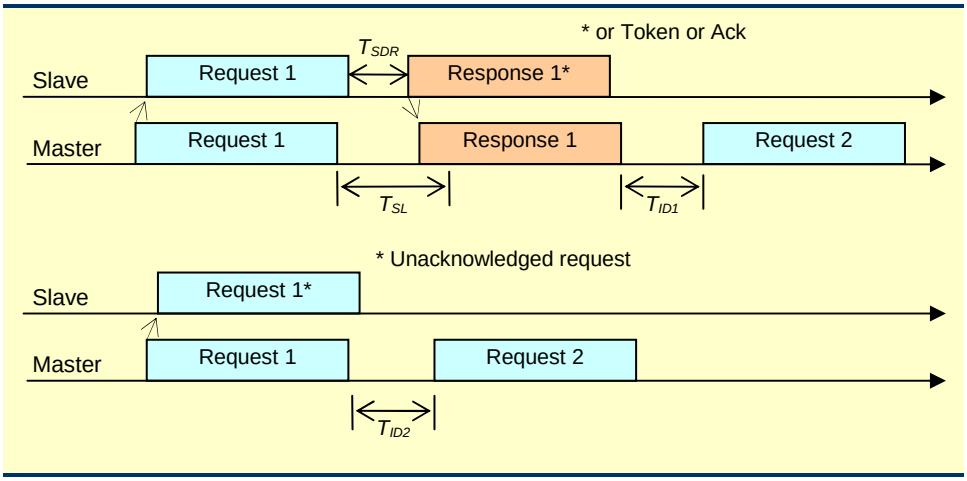



**Figure 2.6: Profibus Idle Times and Slot Time**

In addition, for unacknowledged PDUs we have:

$$T_{ID2} = \max(T_{SYN} + T_{SM}, \min T_{SDR}) \quad \textbf{(2.2)}$$

The values of $T_{ID1}$ and $T_{ID2}$ are set in a per-station (master) basis.

Another important parameter is the *Slot Time* ($T_{SL}$): after the master sends the last bit of a confirmed request to a slave it waits for the response until $T_{SL}$ expires, if it expires then the retry mechanism is started.

In more formal terms, there are two components for this value, $T_{SL1}$ is used for confirmed requests and is defined as follows:

$$T_{SL1} = 2 \cdot T_{TD} + \max T_{SDR} + T_{UART} + T_{SM} \quad \textbf{(2.3)}$$

where $T_{TD}$ is the line transmission delay and $T_{UART}$ is the time needed to detect a character (11 bits, see Figure 2.5)

$T_{SL2}$ is used for Token transmission, and its value is defined as follows:

$$T_{SL2} = 2 \cdot T_{TD} + \max T_{ID1} + T_{UART} + T_{SM} \quad \textbf{(2.4)}$$

The final value of $T_{SL}$ is:

$$T_{SL} = \max(T_{SL1}, T_{SL2}) \quad \textbf{(2.5)}$$

This value is configured in all master stations of the network since it is a parameter of the token passing mechanism.

It is clear that these parameters have to be adjusted when using a hybrid (wired/wireless) network since the reaction times are different when the PDU is forwarded between the different physical domains.

It is also necessary to avoid queuing delays (e.g. when one PDU is not forwarded immediately because another one is still being transmitted) or else we cannot guarantee the real time characteristics of the network. Since both request and responses are also forwarded between multiple physical domains and not only the segments including the master and the slave, it is then necessary (see Figure 2.7) to have an Inserted Idle Time.

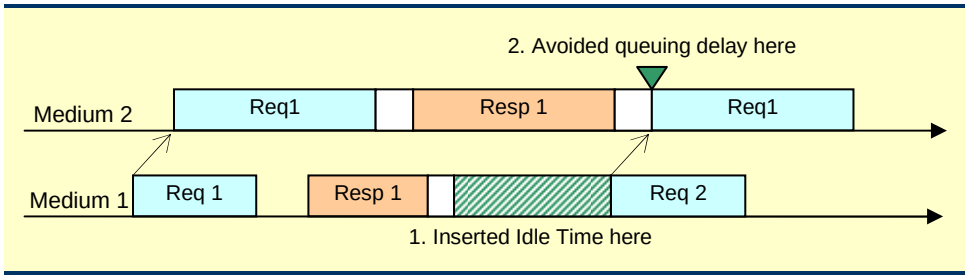



**Figure 2.7: Profibus Inserted Idle Time**

The main concept is that a new request can only begin when it can be guaranteed that it is not subject to queuing delays. The same reasoning applies to token

transmissions and for unacknowledged PDUs. This means that in hybrid networks there will be higher values of $T_{ID1}$ and $T_{ID2}$ on the master stations and these values can be different on each station depending on the PDU sizes and characteristics of the stations it interacts with. Further details are available in (Alves, 2003).

## *2.5 Connecting Fieldbuses to TCP/IP and Ethernet networks*

One of the objectives is to allow transparent interconnection between the TCP/IP and fieldbus realms. There are diverse solutions and technologies used for this purpose.

TCP/IP and fieldbus interconnection solutions include IDA (Interface for Distributed Automation), Ethernet/IP (DeviceNet based), Modbus/TCP (Modbus based) and HSE - High Speed Ethernet (Foundation Fieldbus based).

These solutions enable remote control even over the Internet using standard TCP/IP hardware and software including Virtual Private Network (VPN) tunnelling if needed (Hoon *et al.*, 2002).

For instance, Modbus/TCP (Modbus IDA, 2007) follows a Client/Server model and exchanges data using TCP connections in port 502. Each Modbus/TCP PDU has a header that is different from serial Modbus. Modbus/TCP header (see Figure 2.8) starts with a 2-byte *Transaction Identifier* to support multiple open client requests to a particular server; a 2-byte *Protocol Identifier* that enables multiple protocol support (for Modbus/TCP this is always 0). The header has also a 2-byte *Length* field; this is required since in TCP a single request in the source can be split into several blocks on destination. It has also a 1-byte *Unit Identifier* that is used for intra-system routing like when a Modbus/TCP server is used to connect several serial Modbus stations to the TCP network. Since TCP guarantees the integrity of the data, no check information is needed on the Modbus/TCP PDUs and so the Serial Modbus *CRC* field is not used.

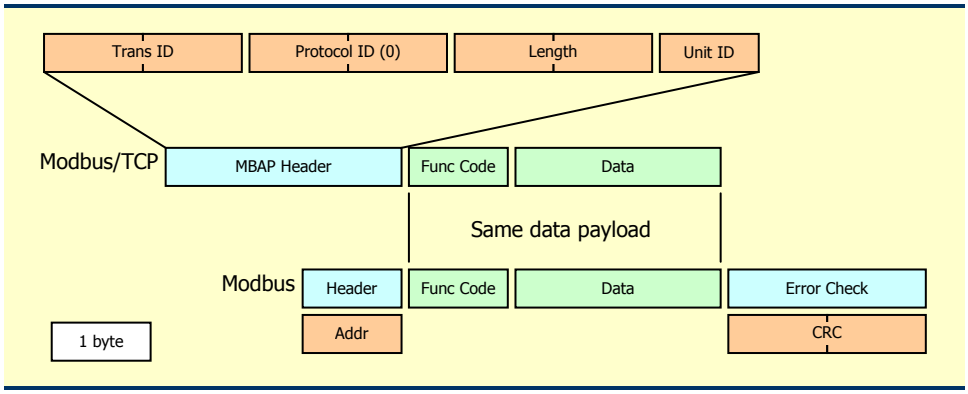



**Figure 2.8: Modbus/TCP and Modbus Serial PDU**

The "over-IP" solutions have advantages like compatibility and hardware availability, but lack both hard-real time performance and very quick response times

since there are obvious overheads not only on the network with the physical layer overhead and IP plus higher layer headers but also in the software stacks used.

PROFInet (Profibus based) and EtherCAT have Ethernet-specific capabilities using specialized hardware (or embedded software) that aim to overcome these issues and even beat traditional fieldbuses in terms of reaction time (Prytz, 2008). The economic cost penalty of these solutions is small since the Ethernet MAC (see Figure 2.9) and respective supporting hardware is standard. Both solutions support other Ethernet traffic (e.g. TCP/IP) in the same network and can even have stations with fieldbus-protocol capabilities and TCP/IP stacks. Both protocols also support standard IP communication.

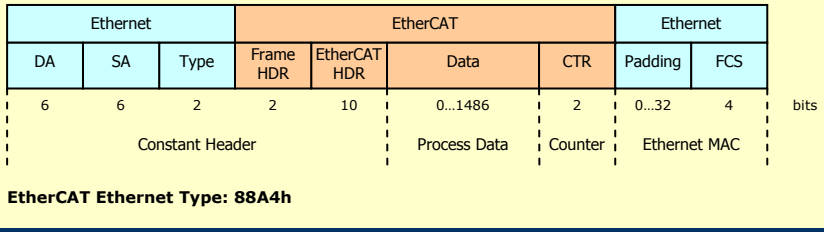



**Figure 2.9: EtherCAT Ethernet PDU**

The EtherCAT shows another possible solution: slave stations have two Ethernet connectors, when data is forwarded from connector A to connector B an EtherCAT-specific Field-Programmable Gate Array (FPGA) reads and changes specific bits on the EtherCAT Ethernet PDU, non-EtherCAT PDUs are forwarded unmodified. This forwarding is very fast and this justifies the quick cycle times of 11µs for 256 bits of data up to 300µs for 12000 bits of data (that fit a single Ethernet frame). The last station on a branch puts data back on the connector A. PDUs on this "back-channel" using Ethernet full duplex capability are forwarded until the master station. The master station has a standard Ethernet card and the duplex capability can be used to realize a double ring topology using two Ethernet cards on the master station. In case of a break in one cable, the system forwards PDUs over the two open branches of the ring.

## *2.6 Multimedia content over fieldbus and automotive on-board networks*

Outside the RFieldbus project, Profibus has been shown to be capable of sending image data with limited capabilities (image data of 17 Mbps reduced by compression to 800 kbps) without impairing the control traffic (Sempere and Silvestre, 2003), careful configuration of the Profibus network is also important as shown in (Silvestre *et al.*, 2002). These solutions have in common the fact that each implements its own method of transferring the multimedia content over the network and interoperability issues where not addressed.

A technology similar to RFieldbus in the multimedia capabilities is Interbus TCP, a system that enables the transmission of TCP/IP data over an Interbus-S fieldbus. In a 500 kbps Interbus network this service provides a TCP/IP bandwidth equivalent to a

14.4 kbps modem (Volz, 2001). Interbus TCP uses (Burmann *et al*., 2004) the Point-to-Point Protocol (Simpson, 1994) to serialize the data on the Interbus link, and has mechanisms to guarantee interoperability between TCP and non-TCP capable stations in operations like software transfer using gateway stations. This solution relies on the serial point-to-point capabilities of Interbus system and adds 2- to 10-byte overhead to each IP packet.

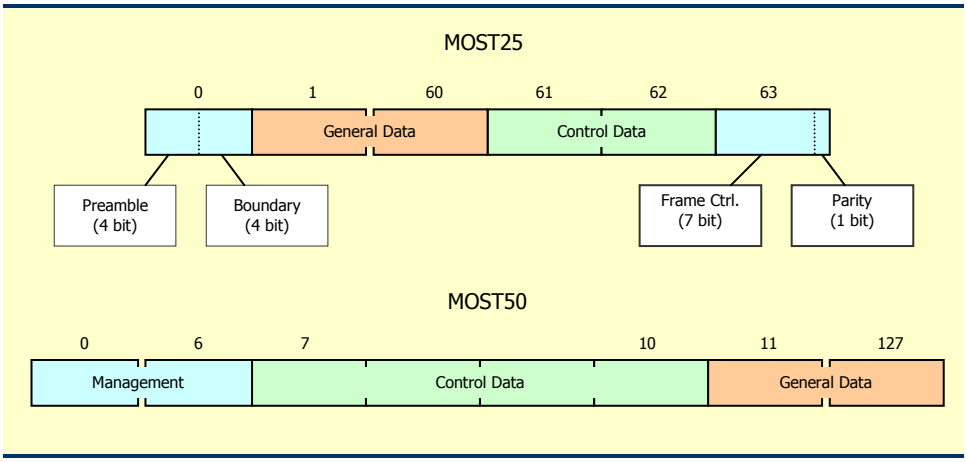



**Figure 2.10: MOST Frame Structure**

The Media Oriented Systems Transport (MOST) bus is a multimedia capable system for automobile distributed applications normally used over plastic optical fibber with ring topology, but supports other topologies and cabling. It features data rates of 25 Mbps (MOST25) and 50 Mbps (MOST50) shared by asynchronous and synchronous (sampling rates from 30 kHz to 50 kHz) data. Up to 64 stations can be interconnected. In practice MOST25 supports 15 simultaneous stereo CD quality uncompressed audio streams at a typical sampling rate of 44100 samples per second, or one 5.1 surround 24 bit uncompressed audio stream, but cannot handle uncompressed video streams, however multiple MPEG compressed video streams are possible (SMSC, 2006). MOST50 supports up to 29 stereo channels of CD quality uncompressed audio at a typical sample rate of 48000 samples per second.

The MOST technology specifies (MOST Cooperation, 2006) not only the physical layer but also all the layers up to the application layer in order to provide interoperability between different manufacturer stations.

MOST PDUs have pre-defined sizes and a variable boundary that divides stream (synchronous) data and packet data (asynchronous) inside the “general data” payload of a particular PDU (see Figure 2.10). In MOST25 networks, this boundary is fixed once the network is setup and at most 60% of the bandwidth can be used for asynchronous data. For MOST50 networks this boundary can be changed on the fly and so bandwidth can be divided by asynchronous and synchronous data at will (in, fact to be precise, at least one byte of synchronous data must be sent in a frame).

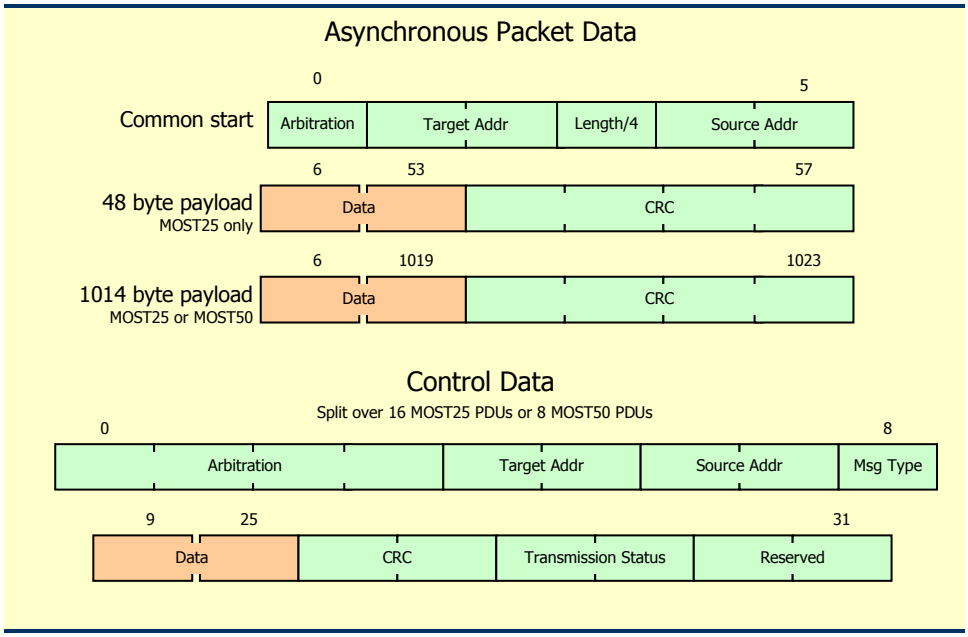



**Figure 2.11: MOST data structures**

Besides the general multimedia data, MOST reserves a small section of the PDU for control data (e.g. turn on/off devices, volume adjustment, etc…). MOST25 data PDUs are 64 bytes long with 60 of general data plus 2 bytes of control data, while MOST50 data PDUs are 128 bytes long with 117 bytes of general data plus 4 bytes of control data (see Figure 2.11).

The synchronous capabilities of the MOST networks are guaranteed by a Timing master station, and accurate synchronization is vital for the network in order to avoid the need of buffering in MOST stations than handle synchronous data flows. The management of the synchronous/asynchronous bandwidth is also done by the Timing Master.

In MOST25, data can be grouped in blocks of 16 PDUs of 64 bytes each, totalling 1024 bytes. In MOST50 data can be grouped in blocks of variable size.

MOST also supports packaging of Ethernet frames into the asynchronous payload (MOST Cooperation, 2003), the packaging is straightforward with the Ethernet MAC 14-byte header information converted to 4 bytes, 2 bytes are used in the MOST MAC destination address, and the other 2 bytes are sent as asynchronous data of MOST. Additional 4 bytes are used for packaging management resulting in a usable MTU of 1008 bytes. Using the packaging presented, real data rates of 800 kbps have been recorded sending IP packets with 8000 bytes and using 28 bytes of the 60 bytes available for general data in a MOST25 frame.

# Chapter 3
# Related Work on Power-Line Communication Systems

The energy management communication architecture is built integrating power-line communications (PLC) with higher level protocols. This chapter first provides a brief introduction of the main PLC concepts relevant to the architecture and then the overall description of the supported services and internal architecture is provided.

## *3.1 The DLC1000 Power-Line Communication System*

For the energy management system framework the communication services are based on the DLC1000 Power-Line Communication System. The DLC1000 system provides time-slotted master-slave communication (Sebeck and Bumiller, 2000) in single-voltage networks. In DLC1000 systems, multiple networks may be supported in the same medium using frequency division and/or time division.

In a frequency division solution, each network is assigned a particular frequency range and there is no support to any type of inter-network communication. It is possible for a station to "move" from one frequency band to another, but this operation takes some time and results in loss of connectivity with the previous network (and thus loss of the previous communication streams).

In a time-division solution (see Figure 3.1), different networks share the same frequency band and other physical layer parameters. Masters in the network must be synchronized and they manage the medium access via fixed pre-programmed time slots cycles. A station may be in several time-division networks in the same frequency band, being each time-division network accessed by a particular Network Unit on the station.

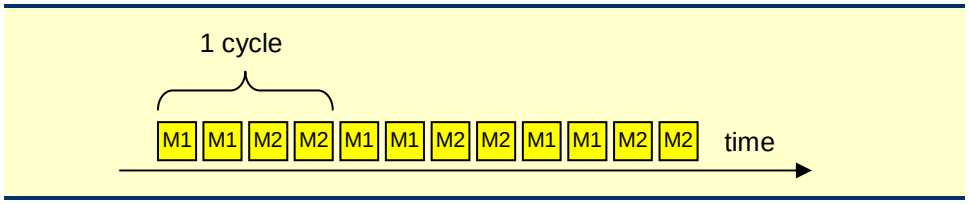



**Figure 3.1: DLC1000 Network Layer time division**

It is possible to mix these two solutions into one same system, with frequency division used first to divide domains, and time-division used independently in each domain.

In a particular DLC1000 network, a station is always either a master or a slave; however a station with two network interfaces may be master in a network and a slave in a different network.

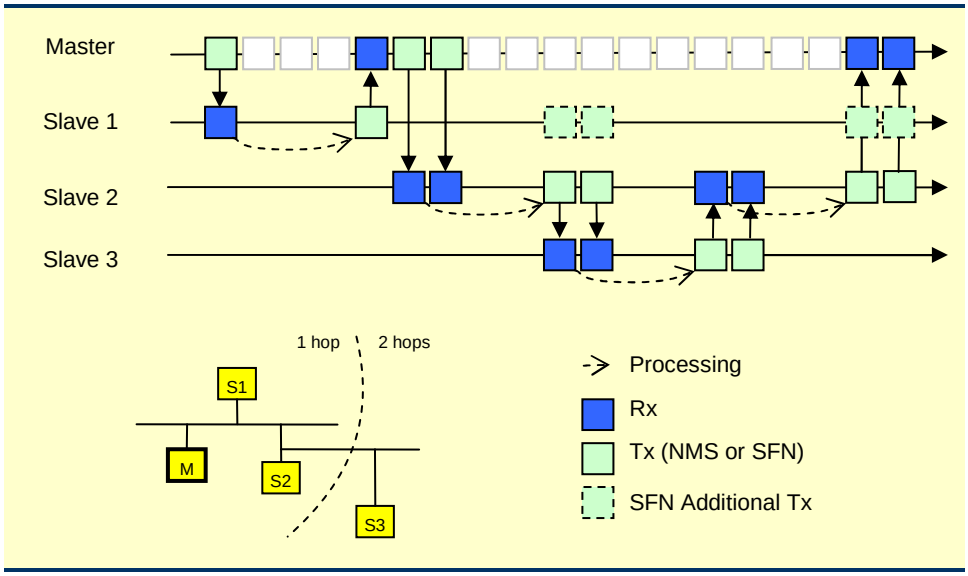



**Figure 3.2: DLC1000 Network Layer timing**

A particular feature of this medium, and a consequence of hardware processing delays, is that slave responses are time-interleaved in *logical channels*. With a time-interleave of four slots – depicted in Figure 3.2 – the master sends four requests to slaves in consecutive slots, and the slave responses (or forwarding PDUs) are expected four slots after the request. This mechanism provides much better network bandwidth utilization in a simple master/slave network. A side effect of this method is that from the application point of view the system behaves as if it had parallel communication channels (4 in the example above). The interleave factor is configurable with a minimum value of three due to hardware processing delays.

In order to extend the network range it is possible to use slave stations as forwarding agents to remote slaves. The master reserves several slots for a particular request, and, in the additional slots, intermediate slaves forward the request to the final destination slave.

DLC1000 supports two different forwarding mechanisms: in the pre-existent Network Management System (NMS) mode (Bumiller, 2001) the master builds a map of the slave routes, and when a request needs routing it is sent to a particular slave with the routing information embedded in the request header. This solution is simple to implement on the slave stations, but uses additional bandwidth for routing data. On the master side, the resources needed for route calculation grow exponentially with the

number of slave stations. For these reasons, the number of hops in NMS is limited to two.

An alternative forwarding mode, identified as "Single Frequency Network" or SFN (Le Phu Do *et al.*, 2005), uses specific hardware to build a forwarding "wave" of requests: the master sends a request with a particular destination signalling the number of hops needed; all the slaves that receive the request forward simultaneously back to the network decrementing the hop-count. The process goes on until hop-count reaches zero. This latter process is much simpler in terms of software management and uses less network resources than the former process, but timing synchronization of the stations may limit the forwarding capability in general networks.

SFN can also be used in very long network lines (that have only one station on each end) with several requests flowing at the same time in different points of the line. Two different frequency bands are used in this solution, one for master-to-slave and the other in the opposite direction. Due to its advantages, SFN was selected and developed within the context of the REMPLI project.

The services that the system supports are thus highly dependent of the underlying communication medium. The data services for a master station include Unicast Unconfirmed Request; Multicast (or broadcast) Unconfirmed Request; Unicast Confirmed Request and Data Arrival.

Automatic retry of Confirmed Requests is configurable if needed. Multicast and Broadcast requests are supported using an 8-bit *Group Address*: each slave is configured with a list of the groups it belongs to, with all slaves belonging to group 255 (broadcast). Unicast requests use 12-bit *Network Layer Addresses* that are assigned at the station's login to the network.

On a slave station, the main data services are Confirmed Request Data Arrival; Unconfirmed Request Data Arrival; Multicast Request Data Arrival and Send Data Request.

The Send Data simply puts the data in the slave output queue. It is not possible to guarantee that the data is sent in response to a particular Confirmed Request. Due to the network timing (and specially the time needed for the encoding and decoding of frames), it is challenging to generate the Send Data Request in response of the Confirmed Request Data Arrival event in time for the data to be usable. Using a larger interleave factor would help on this matter, but then response times would be worsen for other services. Since it is expected that the delay needed to query sensors connected to the slave station are much larger than the request/response capabilities this is not a critical issue to the system design. However, if packets have multiple fragments, then they should be delivered as efficiently as possible.

The main reason for this efficiency-urge is that with the medium access implementation used in REMPLI, automatic additional Confirmed Requests are generated to empty the queues of each slave without the intervention of higher protocol layers. Out-of-band information is sent by the lower level layer on the slave to inform on the current queue state.

Data services requests are queued in priority queues; there are three priority queues for masters and two for slaves. In the simplest implementation, higher-priority queues are emptied before lower-priority queues are served. However, the system is built such

that other more complex scheduling algorithms can be later incorporated in particular scenarios.

There are also special services for managing the status of the network. These support services include login/logout notifications, link quality information and the Status Pool service. The Status Pool service automatically queries each slave on the network for its 8-byte *Status Information*; it provides a simple way to keep at the master an image of slowly changing information of each slave.

When a slave station is activated, it starts by scanning the configured frequency bands. When it finds an active frequency it tries to synchronize with it: the master sends periodic special Physical Layer Configuration Packets for this feature. Afterwards, the slave tries to logon to a master; once again the master sends special packets to enable new slaves to start the logon process. This process involves exchanging *Unique Serial Numbers* between master and slave, and assigning a 12-bit *Network Layer Address* (*NLAddr*) for each newly connected slave. A slave may be connected to several masters using one or more times slots in each cycle. The *NLAddr* of a slave is specific to each master connection. The master keeps track of the link quality of each logged on slave automatically pooling the station if needed.

### *3.2 The REMPLI System Services*

The REMPLI communication infrastructure connects several Application servers (on a Private Network) and devices in the power distribution grid providing end-to-end connectivity (Figure 3.3). It consists of: *low-voltage segments*, which cover groups of energy consumers (for example, a segment can span across one staircase of apartments within an apartment block, or cover a single production branch); *medium-voltage segments* between the primary and secondary transformer stations; *TCP/IP or IEC 60870 based segments* between the primary transformer stations and the Application server(s); and *TCP/IP communication* between the Application Servers and their clients. The interfaces provided by the Application Servers can be available only within the Private Network or also by Internet clients (e.g., SCADA server/client communication).

The bottom-level of the communication infrastructure is comprised of *REMPLI Nodes*, each coupled with a PLC interface (usually a low-voltage PLC modem, in certain cases it can have a medium-voltage PLC modem). A Node is usually installed at the consumer site, e.g., inside an apartment or apartment block, and has a number of metering inputs (such as S0, for electrical energy meters). Nodes are also equipped with digital outputs that allow switching off and on electrical/heat/gas/water supply for a particular consumer, upon commands from the utility company.

At the top-level of the infrastructure is the TCP/IP-based *REMPLI Private Network*, where Application Servers of utility companies are connected. Application Servers perform dedicated functions, such as metering, billing or SCADA. Special Application Servers can also offer access to data, collected and processed by the REMPLI system, to end-user clients, located in the open Internet, or even to wireless terminals.

All Application Servers access Nodes in the PLC network via a *REMPLI Access Point* (AP) – a station that interconnects TCP/IP and PLC-based segments and, optionally, implements a number of additional services.

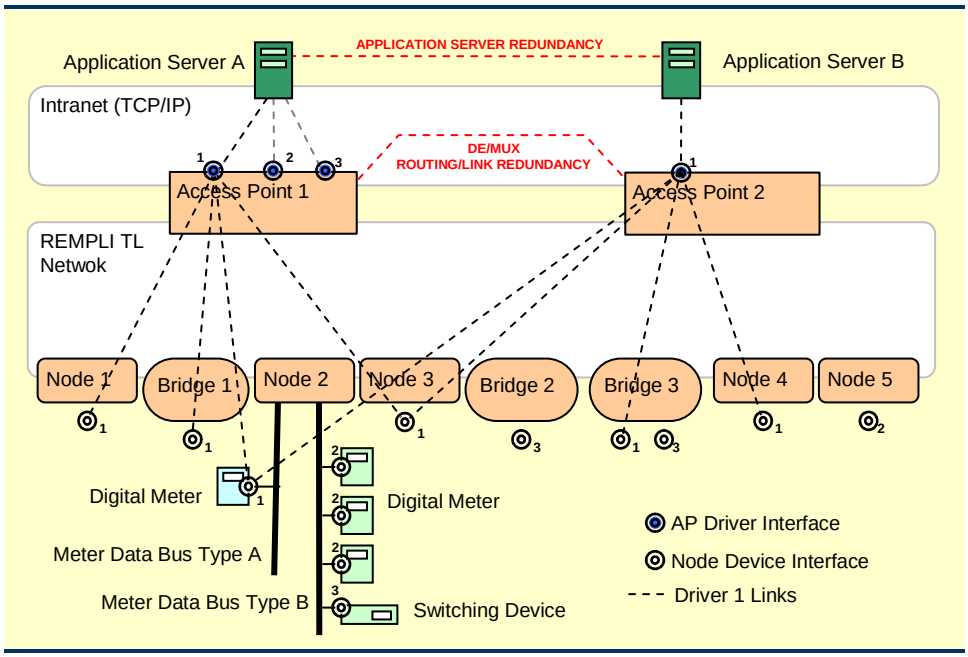



**Figure 3.3: REMPLI Upper Layer Functionality ("outside" view)**

The software architecture of a Node allows running different types of applications, each provided with an interface to the PLC environment. Any application running at the Node is "visible" on the other side of the communication system. Hence, Application Server(s) can access data collected by a Node application (e.g., retrieve metering values) or provide inputs into the application (configure the application itself, or control peripheral devices – such as relays – via the application).

A power-line network can contain other PLC stations (i.e. non-REMPLI) as well, not represented in Figure 3.3. These stations are equipped with the same type of PLC interface as REMPLI Nodes; however, they run different software and perform different functions, whilst sharing the available PLC bandwidth with the REMPLI communication infrastructure. The REMPLI system co-exists with them, although not providing any facilities for communicating to foreign stations.

All Nodes within a typical REMPLI installation are connected to a cascaded power-line network. The power-line network consists of one Low-Voltage and one Mid-Voltage segment, with the word segment being used in the logical sense: several independent wired segments may exist in each Low-Voltage and Mid-Voltage "segments". Communication at both levels is master/slave-based. Low- and Mid-Voltage segments are coupled by one or more *REMPLI Bridges*, which are usually installed at

the secondary transformer stations, between the two parts of the cascade. Physically, the Bridge is comprised of a station that has both a high-voltage PLC modem and a low-voltage PLC modem. The link, established by a Bridge, is transparent for the data payload seen by applications: requests are forwarded from the upper part of the cascade into the lower one, responses are passed back. Hence, the whole PLC network of two segments becomes a single request/response communication environment.

In some installations, where a utility company needs to collect information from the secondary transformer station itself or to control it, the Bridge can be combined with a Node, i.e. all the services available in a Node are also available on the Bridge. It is also possible to equip a secondary transformer station only with the Node, and not with the Bridge. In the latter case, the transformer station becomes a communication end-point, and no data transmission occurs into the Low-Voltage segment. Other Nodes can be connected directly to the Mid-Voltage network, e.g. for utility internal metering control purposes or for clients with direct Mid-Voltage electric power network supply (e.g. industrial clients).

### *3.3 REMPLI System Internal Architecture*

From the point of view of Applications that use REMPLI resources, the network presents a flat address space (*REMPLI Node Address*) with direct connection from the *Access Point Driver Interface* at each AP to the *Node Driver Interface* at the Nodes or Bridges (Pacheco *et al*., 2005a).

APs provide interfaces on the Intranet network to multiple Application Servers. Depending on the purpose of the system, a Server can be connected to one or more APs (this redundancy in access can be managed by the Server). However, the REMPLI system provides its own redundancy services between APs. This means that, if needed, an Application Server can be connected to a single AP and still have access to all devices in the network.

In order to implement the interfaces to the external word, e.g. some specific Application Server and a Digital Meter, special modules (Drivers) are provided. Drivers implement protocol-specific functionalities to the REMPLI network on top of the Transport Layer providing services that connect each driver on the AP to a specific driver on the Nodes. Typically, a different driver pair is used for each type of metering or control devices.

Management of shared resources (e.g., if drivers share the same physical bus), has to be implemented at the drivers' level. In the example of Figure 3.4, Application Server A and B can connect to AP Driver 1 Interface (e.g. a TCP Server Port) at either AP1 or AP2, linking up to Nodes 1, 2, 3 and 4 and Bridges 1 and 3. At Node 2, Drivers 2 and 3 share the same data bus and thus some resource management mechanism must be in place to avoid conflicts between the two modules.

In the REMPLI project, Drivers for IEC 1107/EN 62056, IEC 60870-5-104 and EN 1443.3/M-Bus where implemented (REMPLI Project, 2008). The IEC 60870-5-104 implementation is a simple translate and tunnel interface. The IEC 1107 Driver, however, had to implement local handshakes in order to cope with the standard's timings. In the M-Bus driver, the dialled number command used to connect applications

to devices using phone-line modems was "translated" to a connection to a specific Node (the number is the *Node Address*); in this case a proxy application is used in the application server to forward communication port PDUs to the AP driver. A "transparent" point-to-point driver was also implemented and this enabled the SSH and Telnet standard Internet application connections to Nodes (used for instance in the field trials for remote debugging of software).

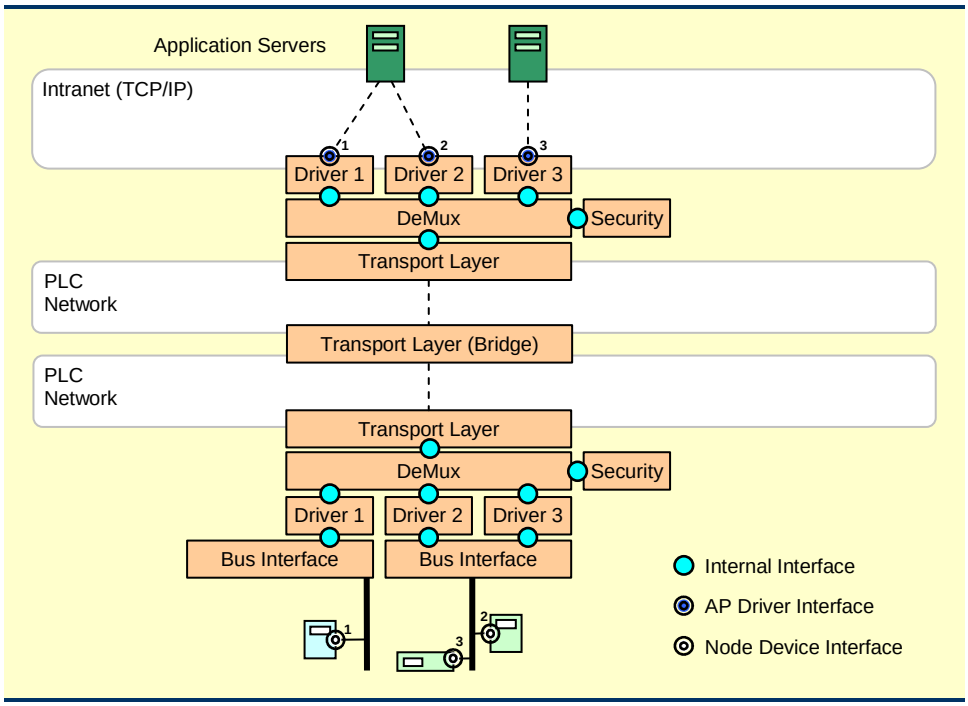



**Figure 3.4: REMPLI Upper Layer Functionality ("inside" view)**

The DeMux layer interconnects all the Drivers in one station to the lower software levels, merging the multiple driver data to a single channel. It also provides security services integration including encryption and authentication of data travelling the PLC network. The security services are supported by smartcard technology (Treytl *et al.*, 2004). The DeMux can also route requests made by a Driver in one AP to another AP using the Intranet. Depending on the current network conditions the response is routed back to the original AP automatically. In terms of implementation, the interfaces of the DeMux are internal TCP connections as a server to Drivers, and as a client to the Transport Layer.

The Transport Layer (the central layer of the architecture proposed in this thesis) provides REMPLI with bi-directional end-to-end communication services between APs and Nodes (Pacheco *et al.*, 2005b). It deals with routing via Bridges when needed, with providing QoS capabilities and with the support mechanisms to very large data payloads

(16 MiB in the current configuration, up to 4 GiB with code rebuild). The main services provided by the Transport Layer include sending a packet from an AP to a Node; sending a request from an AP to Node with response in the opposite direction; and broadcasting a packet from an AP to all available Nodes. A Node can also send a packet to, at least, one of the available APs, selected at run-time by the Transport Layer. Finally, the Transport Layer manages system-wide link quality information and link connection/disconnection information.

The PLC Network is the base master/slave network with point-to-point communication of small packets and link-quality information services. The interface between the Transport Layer and the Network Layer is a Linux character device driver. From the point of view of the Transport Layer, the Network provides services in a master station like send data, send confirmed data, data reception and status information reception. On slave stations, the services are data transmission, data reception and status information setting. Since the Network Layer supports TDMA for multiple-master capability, a station can have multiple Master Network Units, each managing a group of TDMA slots, and multiple Slave Network Units (each connected to a single master). Bridges have both master and slave interfaces active. Connection between masters and slaves is dynamic and fully automatic: when a slave is started up, it searches for information on the current network characteristics (the REMPLI network can use multiple frequency bands and multiple TDMA configurations) and tries to connect to the available masters. All stations have a *REMPLI Unique Serial Number* (RUSN) that is used to keep track of the slave logins at the master but can also be used to build simple “access lists” that forbid certain slaves to login in certain masters. This feature is more a management feature than a security question.

# Part II
# Factory Communications Framework

# Chapter 4
# Protocol Stack Architecture

This section presents a novel architecture for supporting multimedia TCP/IP services over a standard fieldbus protocol – Profibus. This communication architecture enables the transmission of multimedia traffic such as sound and video, in conjunction with the "traditional" real-time control traffic, through appropriate admission control, scheduling and traffic differentiation mechanisms.

## *4.1 Introduction*

One of the main objectives of the RFieldbus system architecture (RFieldbus Project, 2000) is to allow that multimedia TCP/IP applications and native Profibus-DP (PROFIBUS & PROFINET International, 2008b) applications coexist transparently supported by the same physical network infrastructure. Traffic differentiation must be guaranteed, i.e. different traffic classes must be defined in a way that real-time traffic is not affected by "multimedia" traffic, typically best-effort traffic. The solution that is proposed and discussed is achieved through a DP/IP Dispatcher that merges TCP/IP traffic and "native" Profibus-DP traffic (see Figure 4.1).

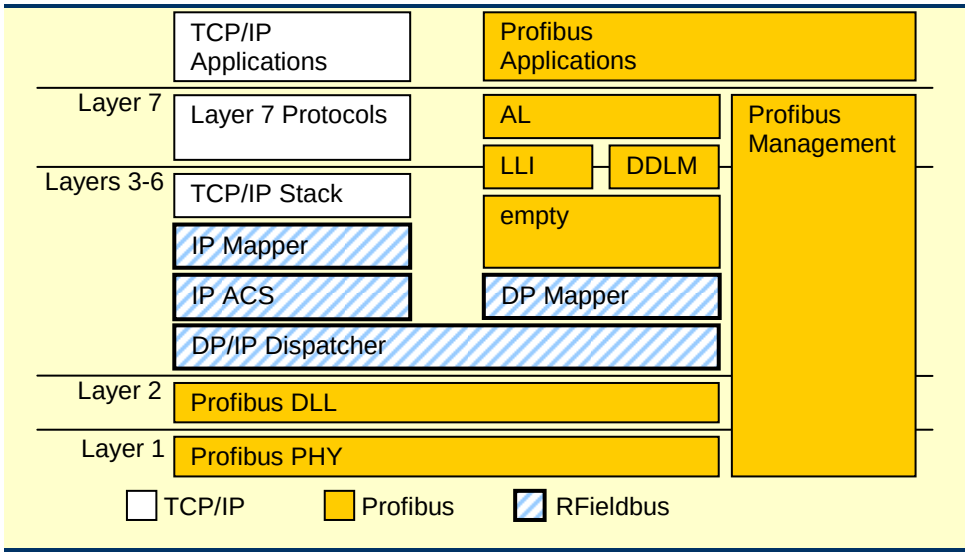



**Figure 4.1: RFieldbus Protocol Stack Architecture**

Traffic from each protocol stack is divided into five classes: DP High-Priority, DP Low-Priority, DP Best Effort, IP with QoS requirements and IP Best Effort. The DP/IP Dispatcher (or just "Dispatcher", for short) can reserve a minimum bandwidth for each of the traffic classes. It can also guarantee that local Profibus traffic does not impact real-time traffic generated by other network stations, which is something that even the standard original Profibus protocol could not guarantee without some care in configuring the diverse network parameters (Tovar and Vasques, 1999a).

Since an objective is to rely on an unchanged Profibus DLL (Data Link Layer), the Dispatcher does not use the token arrival information. Alternatively, it executes periodically controlling the medium utilization time by means of transmissions time estimations of each frame sent. The Dispatcher receives these time values from the IP Mapper and DP Mapper sub-layers.

A structural limitation must also be overcome: the Profibus protocol follows a master/slave paradigm (stations play different roles), while the TCP/IP protocol does not. The basic concept in master/slave networks is that some stations – the masters - control the access to the medium and other stations – the slaves – only respond to requests from the masters. On the other hand, in TCP/IP networks all stations have equal initiative rights. Therefore, in order to support TCP/IP applications it is essential that slave stations can behave like IP traffic sources without previous explicit consent from a master station.

To grant slaves initiative, without changing the base network protocol, it must be guaranteed that all slaves (or at least the ones requiring initiative) receive a request/response PDUs periodically.

In our proposal, this is achieved through two main mechanisms. First, the IP ACS Scheduler at the master side reserves some data slots for Request with Response PDUs for a particular slave. These are sent even if the applications at the master do not issue any request to the slave. The reserved slots are also used for application data, if that data is available. In this way, it is possible to guarantee a minimum bandwidth for the slave-to-master communication, but additionally an equivalent bandwidth is also available to master-to-slave traffic. Secondly, when a master receives a packet, it checks if it is for itself or to another station. In the latter case, the message is forwarded to the appropriate destination; note that the destination can be an IP station outside the fieldbus network if the master embeds a gateway, or a slave station in the fieldbus network.

The implementation of these mechanisms imply just some small additions to the IP ACS Scheduler. The IP Mapper routing mechanism becomes also quite straightforward. The main difficulty is using the correct packet identifiers for the fragments that are sent back to the network.

This was solved using an ID Generation mechanism that is used when new packets are sent to the network and when packets are forwarded. Figure 4.2 depicts an illustrative example: on the left end side, a simple master/slave connection; in the middle M1 forwards a packet from S2 to S3; on the right end side S2 sends a packet to a standard TCP/IP Ethernet host (A).

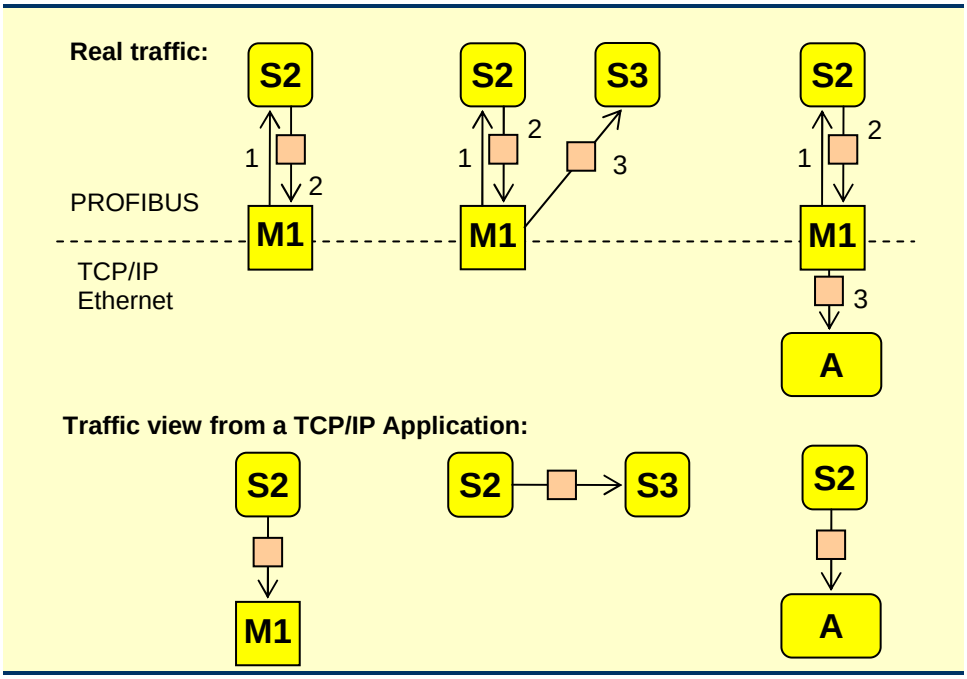



**Figure 4.2: Slave initiative examples in a symmetrical scheme**

Another issue requiring careful design concerned converting Profibus addresses to and from IP addresses, and the routing of IP packets.

Given the fact that Profibus has a limited 7-bit addressing space, a natural solution is to make the direct mapping of Profibus addresses into IP Class C Host addresses – with the higher bit set to zero. The remaining 3 bytes (see Figure 4.3), including the Class C prefix of the IP Class C address, are programmed in each station and known as *RFieldbus Network ID* (similar to the implicit network address programming in TCP/IP stations given an IP address and a network mask).

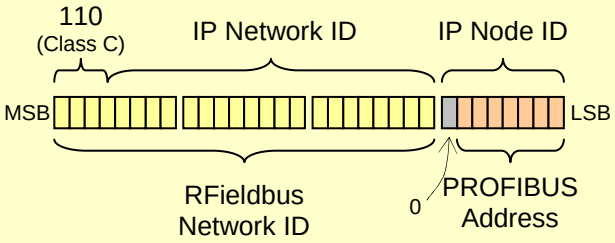



**Figure 4.3: RFieldbus Profibus-IP addressing scheme**

Concerning the IP routing, the solution is a complement of the slave routing via masters. When a station receives an IP packet from the Profibus network, the IP Mapper performs the following algorithm (regardless of being a master or slave station):

1. Check if the *IP Destination Address* matches its own address. If so, the packet is delivered to the local TCP/IP stack.
2. Check if the three higher bytes of the *IP Destination Address* match its own *RFieldbus Network ID*. If so, it forwards the packet to the Profibus network using the *Host ID* of the *IP Destination Address* as *Profibus Destination Station* removing the most significant bit.
3. Check if a Gateway station is configured. If it is the case, the packet is forwarded to the Gateway station.
4. If everything else fails, the packet is delivered to the local TCP/IP stack. The local TCP/IP stack makes the decision to discard the packet or to send it to another host or router in the TCP/IP network.

On a correctly configured network, Steps 2 and 3 should only be relevant for master stations.

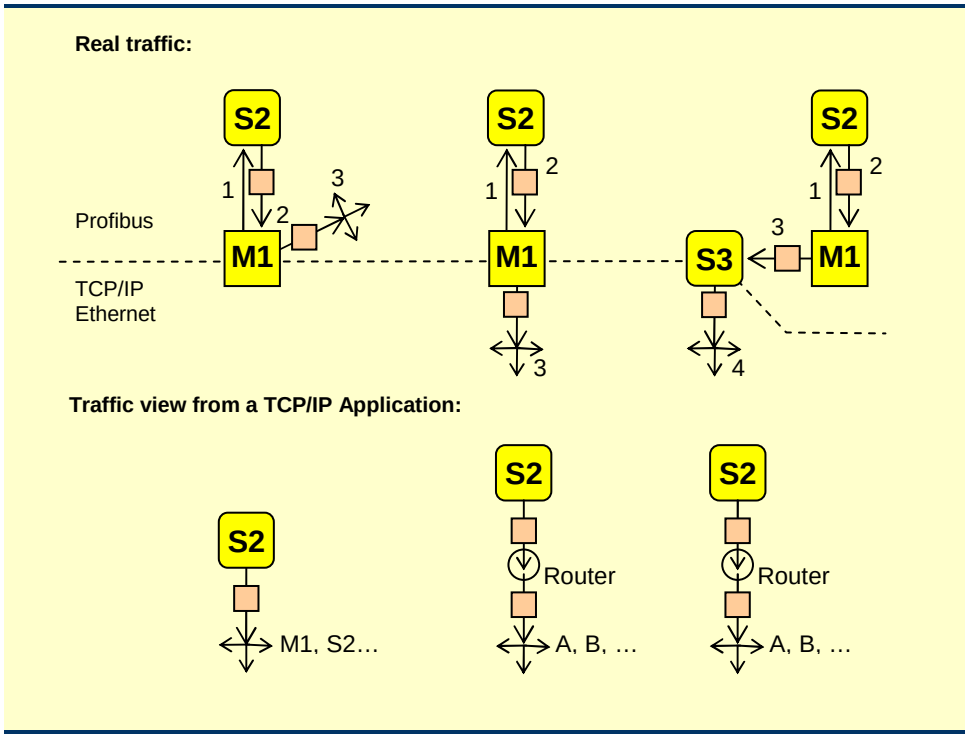



**Figure 4.4: Multicast/Broadcast scenarios in RFieldbus**

For broadcast packets, the processing will be the same. Note that the IP Broadcast Host ID (255) is translated to the Profibus Broadcast Address (127) by the above algorithm. Multicast IP address translation is not supported inside the RFieldbus system.

However, the above address-processing algorithm makes it possible for an RFieldbus station to send a multicast stream to stations *outside* the RFieldbus network.

In Figure 4.4, three examples of broadcast initiated by a slave station are presented: on the left, same IP network (sharing the same Class C network ID); in the middle, different IP networks using TCP/IP stack routing; on the right, different IP networks using RFieldbus Gateway.

The details on each of the components of the dual-layer stack are described in the next sub-sections.

## *4.2 IP Mapper*

The IP Mapper sub-layer is located directly below the standard TCP/IP protocol stack, converting TCP/IP services into Profibus DLL services (and vice-versa). It performs the identification, fragmentation and re-assembly of the IP packets to/from Profibus DLL frames. In master stations, the IP Mapper is also responsible for routing slave TCP/IP packets to other stations. It also estimates the network usage of each fragment sent.

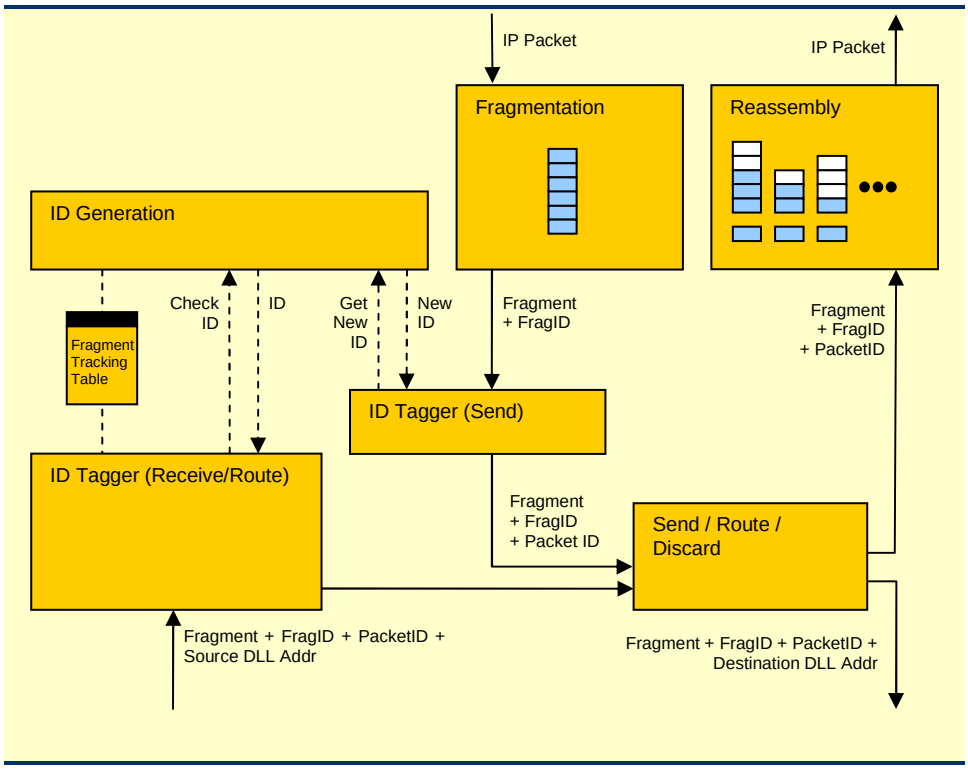



**Figure 4.5: IP Mapper Internal Architecture**

Traffic class assignment, e.g. IP Best Effort (IPBE) or IP High Priority (IPH) is done in the IP ACS sub-layer, as presented in Section 4.4.

The IP Mapper includes several entities and functionalities, briefly illustrated in Figure 4.5 and described next.

The *Fragmentation* function receives an IP packet from the TCP/IP stack and divides it in fragments of appropriate size for the Profibus network; each fragment is also marked with a *Fragment ID*. IP datagrams that do not need fragmentation are marked with *Fragment ID* zero. Note that the IP Fragments that the IP Mapper passes to its lower layers take into account the limitations that are imposed by the underlying (Profibus) network. In this context, the IP Mapper may receive from the TCP/IP Stack an already fragmented IP packet and re-fragment it according to these limitations.

For local fragments, the *ID Generator* assigns new *Packet ID*s for each IP Packet from a pool of 256 possible values. For remote (routed) fragments, the ID Generator first checks if the remote *Packet ID* is in use. If not, then it is returned unchanged, while if the ID is used then a new ID is generated and the *Fragment Tracking Table* (FTT) entry is updated accordingly. The *Release ID* function (not represented in Figure 4.5) is called every time a fragment is discarded or when a packet is completely sent or received.

The *Send/Route/Discard* functionality first checks if the *IP Network Address* of the packet matches the *IP Network Address* of the station and the *IP Host Address*. If both addresses match, then the packet is delivered to the *Reassembly* entity. If only the *IP Network Address* matches, then the packet is delivered to the IP ACS using the *IP Host Address* as the *Profibus Destination DLL Address*. If they do not match, it means that the destination is not in the local network but in a remote IP network and therefore a gateway station must be used. If there is a gateway for this station, then its *Destination DLL Address* is used, if not the fragment is discarded.

All fragments of a particular IP packet received from the fragmentation module are assigned a new *Packet ID* (at transmission time) by the *ID Tagger (Send)*.

For received fragments, the *ID Tagger (Receive/Route)* uses the FTT. The first fragment generates a new entry in the FTT with the *Source DLL Address*, *Original Packet ID* and a locally generated *Packet ID*. When receiving other fragments, the *ID Tagger* fetches a matching *Source DLL Address* and *Original Packet ID* from the FTT. If no entry is found then the fragment is discarded, while if a match is found than the *ID Tagger* replaces the remote *Packet ID* by the local *Packet ID*.

The *Reassembly* function rebuilds IP packets to be delivered to the TCP/IP stack. When the first fragment is received, a memory buffer of the total IP packet size is reserved (the packet size information is in the IP header that is always available in the first fragment). Subsequent fragments are concatenated as they are received. Since low error rates are assumed, there are no special provisions for data retransmission, and an out-of-order reception voids the entire packet (it is assumed that a fragment was lost).

### *4.3 DP Mapper*

The *DP Mapper* is the Profibus-DP equivalent to the *IP Mapper* in TCP/IP (refer to Figure 4.1), and it is specified in detail in the RFieldbus Data Link Layer Specification (RFieldbus Project, 2001a).

The *DP Mapper* is located below the standard Profibus-DP Application Layer. It incorporates the already existing mapping functionality of the Profibus Data Link Layer Management entity (DDLM) while enabling new features relevant to the integration of DP Traffic and IP Traffic. The DP Mapper takes care of traffic identification of DP traffic. Based on relevant System Management MIB (Management Information Base) Objects, DP Traffic is classified into DP High Priority (DPH) Traffic, DP Low Priority (DPL) Traffic and DP Best Effort (DPBE) Traffic. It also calculates the maximum transaction time of each PDU. Finally, it passes the PDU to the appropriate queue of the underlying DP/IP Dispatcher layer. DP Traffic fills three of the five queues of the DP/IP Dispatcher Layer – DPH, DPL and DPBE.

### *4.4 IP ACS*

The *IP Admission Control and Scheduling* (ACS) sub-layer is responsible for the control/limitation of the use of network resources by the TCP/IP applications. Each IP packet is classified according to the *IP Header* fields, such as destination address and port. Given this classification, the corresponding fragments are placed in a specific queue. Moreover, this sub-layer implements the appropriate scheduling policies, in order to provide the required QoS for multimedia applications.

In each master, the available IPH slots must be used to convey the IP traffic imposing QoS requirements. The ACS sub-layer (see Figure 4.6) is composed of several *Relationship Entities* (REs) and a *Scheduler*. Essentially each RE relates to a particular TCP/IP stream flow (with a particular QoS service level) identified by the IP Mapper. Each RE includes a First-In-First-Out (FIFO) queue, used to store the IP fragments coming from the IP Mapper. Fragments pending in these queues are passed to the Dispatcher sub-layer by the Scheduler. Each RE has a configurable maximum queue size; when this value is reached, requests are discarded. When a fragment is discarded due to queue overflow, all pending fragments of the same packet are also discarded.

The *Scheduler* is responsible for the appropriate emptying of the different Relationship Entity Queues so that all different QoS requirements are fulfilled. The Scheduler uses a service interface, internal to the ACS, for the emptying of the different Relationship Entity Queues or the acquisition of information relevant to their contents. When a request for a fragment is issued by the Scheduler to an empty Relationship Entity Queue, then the Relationship Entity generates a special frame if the *Slave Poll Option* is chosen. This feature of the Relationship Entities is used by slave stations to support multimedia capabilities. In practice, it guarantees slave-to-master TCP/IP bandwidth by ensuring that the slave station has the chance to send a packet to the master at programmed intervals, even when no master-to-slave traffic exists.

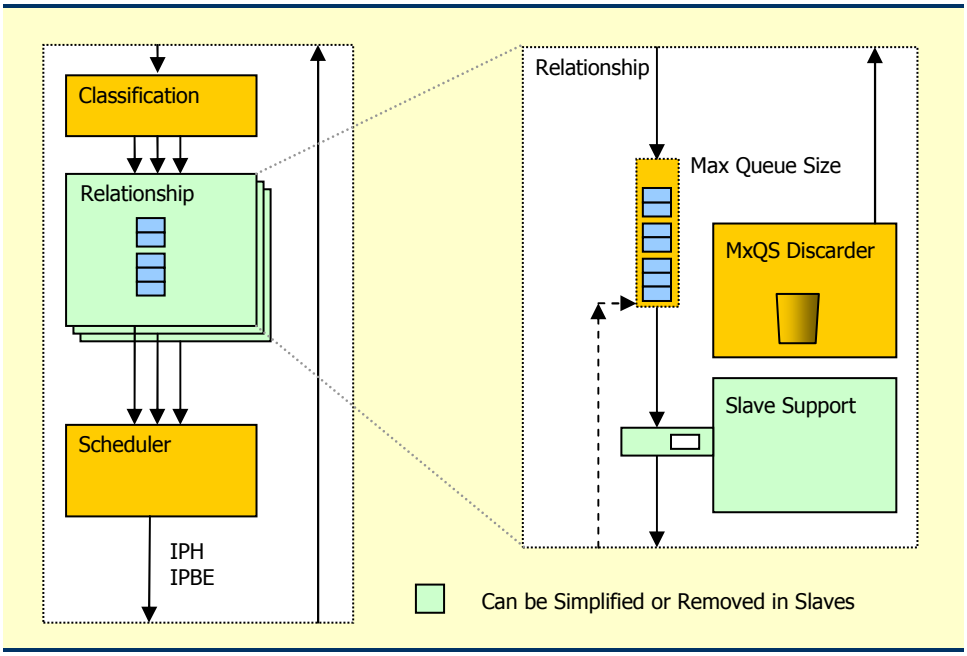



**Figure 4.6: IP ACS Architecture**

As previously mentioned, the multimedia traffic can be of two types: traffic that does not impose stringent timing requirements (denoted as IP Best Effort traffic – IPBE); and multimedia traffic characterised by specific QoS characteristics, namely bandwidth and jitter (referred to as IP High Priority – IPH). At the ACS sub-layer, there is one RE (for both IPH and IPBE traffic) for each TCP/IP stream flow. Each RE has also associated timing parameters that are used by the Scheduler.

The Scheduler uses an interface to the DP/IP Dispatcher layer in order to determine whether it is allowed to fill the DP/IP Dispatcher queues or not. The queues that may be fed by the ACS layer are the IPH queue and the IPBE queue.

The Scheduler is executed periodically with an interval defined as $T_{DCY}$. There are two main strategies to perform the Scheduling of the Relationship Entity queues; off-line or on-line, as outlined next.

If scheduling is done off-line, a table in the Station Management gives the actual schedule of the different Relationship Entity queues. This schedule is created *a priori*, taking into account all needed information so that the diverse QoS requirements of the different Relationship Entity queues are met. In this case, the Scheduler actually works as a dispatcher for IP Traffic. In case the scheduling is done on-line, a table in the Station Management provides the parameters needed by the Scheduling Algorithm so that the actual schedule is determined every time the Scheduler is executed. In addition to these QoS-specific parameters, the Scheduling algorithm has to take into account the time allocated for IP HP traffic and the remaining time for BE traffic.

The above procedures apply to the IPH traffic. On the other hand, IP BE Traffic Queues are delivered to the dispatcher in a round robin fashion. Fragments delivered by

the Scheduler attach information about the traffic class (IPH or IPBE) and about network usage estimation.

The following example illustrates some of these principles. The scheduler micro-cycle is equal to $T_{DCY}$, the worse rotation time of the token. For this example $T_{DCY}$ = 10 ms and $T_{IPH}$ = 2 ms for a particular master station. We consider five TCP/IP data flows with 200-byte fragments, characterized as described in Table 4.1.

**Table 4.1: Example configuration**

| *Flow* | *At every "n" $T_{DCY}$* | *Transaction duration* | *Multimedia data throughput* |
|---|---|---|---|
| IPH1 | 1 | 100 µs | 100 · 1600 = 160 kbps |
| IPH2 | 3 | 200 µs | 33.3 · 1600 = 27 kbps |
| IPH3 | 3 | 200 µs | 33.3 · 1600 = 27 kbps |
| IPH4 | 4 | 400 µs | 25 · 1600 = 40 kbps |
| IPH5 | 4 | 1000 µs | 25 · 1600 = 40 kbps |

The *Transaction Durations* depend on the locations of the stations and the data payloads. Spawning multiple domains (wired/wireless) results in additional delays. Also smaller data payloads result in smaller transaction times. However, for most TCP/IP traffic the maximum fragment size is used except in some particular applications, e.g. applications than send small User Datagram Protocol (UDP) packets.

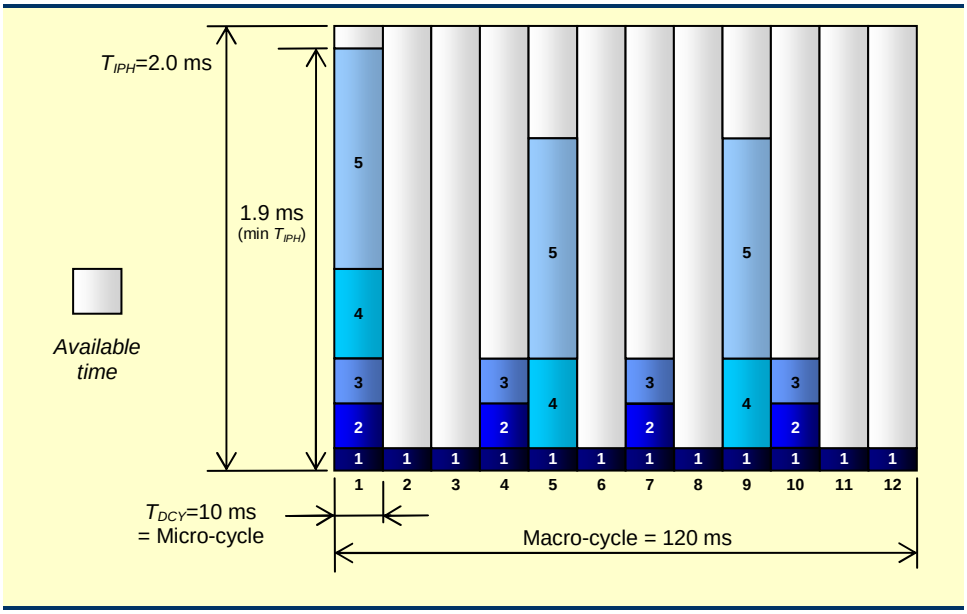



**Figure 4.7: Scheduling example at IP ACS level**

For this example configuration, a simple Rate Monotonic scheduling solution is presented, where each stream is scheduled according to its periodicity. This particular

solution is simple but needs a value of $T_{IPH}$ of at least 1.9 ms (see Figure 4.7); more complex scheduling algorithms can be used to reduce this value. Other implementation alternatives and design options will be discussed in Chapter 5, Section 5.1.

Besides the scheduling table (or a runtime implementation of the scheduling table algorithm), the IP ACS Scheduler can also include a runtime mechanism that compensates for empty IP queues in subsequent micro-cycles. The reader is referred to Section5.1 for more details.

The traffic scheduled by the IP ACS is then fed into the DP/IP Dispatcher where it is mixed with other traffic.

## *4.5 DP/IP Dispatcher*

The DP/IP Dispatcher layer (RFieldbus Project, 2001b) resides under the IP ACS Layer and the DP Mapper (see Figure 4.8). Both DP Traffic and IP Traffic pass through this sub-layer, which is responsible for transferring both kinds of traffic to the Profibus FDL.

The DP/IP Dispatcher sub-layer considers three traffic classes, which are supported by five different FIFO queues according to the traffic source (Figure 4.9).

The *Guaranteed High-Priority* traffic, which is Profibus high priority traffic that must be always scheduled on time. This traffic class is intended to support DP high priority traffic with real-time requirements (DPH).

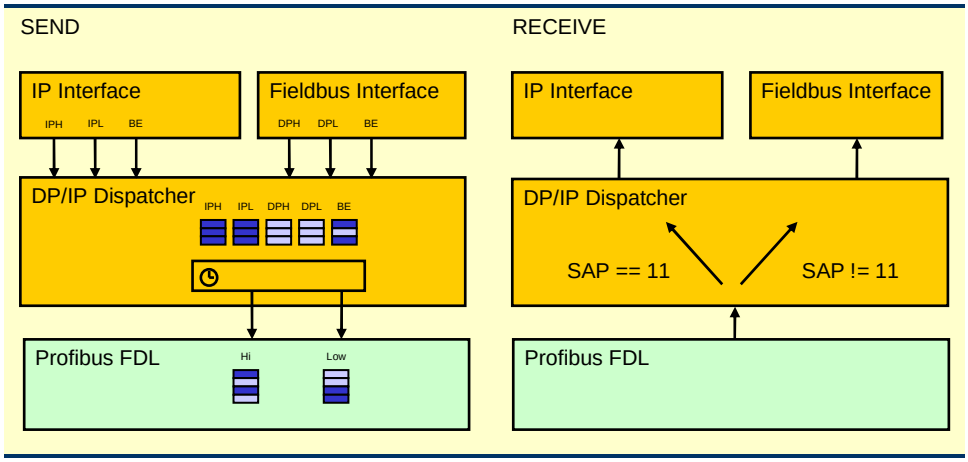



**Figure 4.8: Dispatcher functionality and interfaces**

The *Guaranteed Low-Priority* traffic, which is Profibus low priority traffic that is scheduled on time after the guaranteed high priority traffic. This class of traffic is intended to support the two sub-classes: DP low priority traffic with timing requirements (DPL) and IP traffic with QoS requirements (IPH).

The *Best-Effort* traffic, which is Profibus low priority traffic that is scheduled after the guaranteed traffic, without any guarantees of timely delivery. This traffic class is

intended to support two sub-classes: IP traffic without QoS requirements (IP-BE) and non real-time DP low priority traffic (DP-BE).

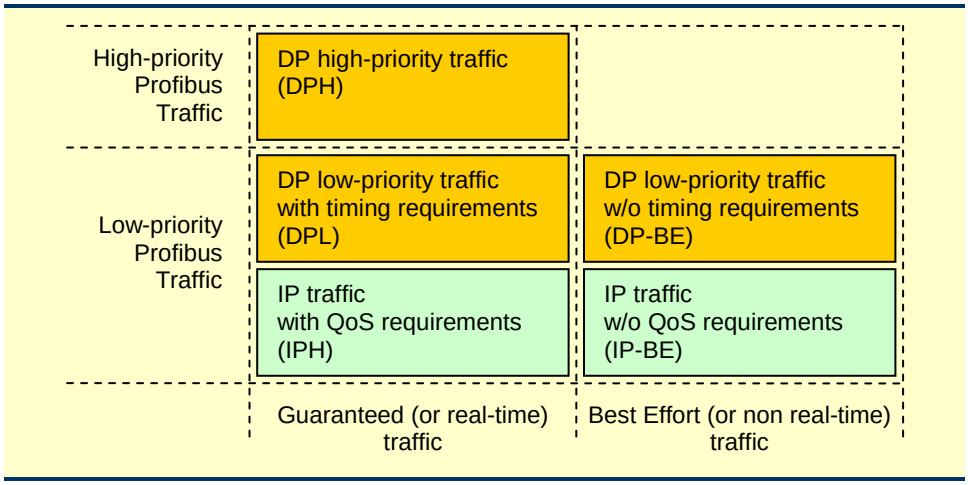



**Figure 4.9: Dispatcher traffic classes**

The dispatching algorithm is executed periodically every $T_{DCY}$, using an independent timer of the IP ACS Scheduler (that also runs every $T_{DCY}$). The algorithm uses bandwidth reservation parameters that are provided as different MIB (Management Information Database) objects and are determined pre-run-time (during System Planning). These objects define the time limits for the different kinds of traffic that are to be served at each cycle. DPH traffic may be constrained to an appropriate time value ($T_{DPH}$) calculated during system planning or left unconstrained so that all DPH traffic generated is fed to the FDL High Priority Queue in every dispatcher cycle. The DPH option is configurable for each station. The other traffic has time constrains which are the DPL Time Limit ($T_{DPL}$) for DPL Traffic, the *IPH Time Limit* ($T_{IPH}$) for IPH Traffic and the *BE Time Limit* ($T_{BE}$) for the combined DPBE and IPBE Traffic. The traffic is constrained in the above-defined order.

Upon setting these time constraint parameters, the Dispatching algorithm passes the appropriate amount of traffic from the five different queues to the FDL High and Low Priority queues. The IP ACS layer is responsible for providing an optimal filling of the guaranteed QoS IP dispatcher queues, but sometimes the available time may be reduced by higher priority DP traffic.

The dispatcher uses an estimate of the network usage time for each fragment to be sent. The IP Mapper and DP Mapper sub-layers calculate this time using several variables. These variables include: the *topology of the system*, since the communication between peers through a number of hopping stations (the complete system supports networks with multiple wired and wireless segments) poses greater delays to communications between peers in the same e.g. wired segment; the *DLL Service used* (confirmed requests take longer than non-confirmed requests); the *number of bytes* that

the fragment contains (to account for the transmission delays of the fragment itself); and the Delays due to hardware/software constrain (e.g. delays on module-to-module communication, hardware interface delays, etc.).

With reference to the target message cycle time, corresponding to a worst-case message cycle duration including the request and reply and taking into account idle time, the dispatcher algorithm may be configured to work according to three schemes: *Direct*, *Direct Tabular* and *Computing*. In the Direct schema, the target message cycle time is given as one object in station management for all kinds of fragments. In the Direct Tabular scheme, there is a table containing the calculated target message cycle time values for different options of peer connections. This data is stored in the station management database. The calculations are done at system planning, taking into account the different system aspects. Finally, in the Computing schema the target message cycle time is calculated by the Dispatcher for each different fragments, taking into account the system management parameters.

The target message cycle time is a significant parameter for the fieldbus system since it is used by both the DP/IP Dispatcher and the IP ACS Layers and is relevant to the selection of the appropriate fragments to be transmitted, so that all timing requirements imposed by the applications are met.

The Dispatcher sub-layer interfaces the DP Mapper and the IP ACS to the Profibus DLL. For transmission, it provides several queues concerning the priority of service requests. The Dispatcher transfers requests from these queues to the DLL, limited by the master allocation time. The requests are transferred at least within the Dispatcher Cycle Time and according to the queue priority.

According to the fragment model, a pre-defined set of dispatching rules imposes that, at each master station, the Dispatcher cyclically transfers to the FDL layer a number DPH PDUs depending on the DPH processing option. These PDUs are followed by DPL PDUs up to configurable $T_{DPL}$ usage estimation limit and then IPH fragments up to configurable $T_{IPH}$ usage estimation limit. If station time is available, DP-BE PDUs and IP-BE fragments are sent.

Such dispatching strategy generates predictable traffic scenarios, where the token holding time ($T_{TH}$) is never overran (provided that $T_{TR}$ is set according to the rules of the *constrained low priority traffic* profile).

Each queue of the Dispatcher must hold the traffic needed for one dispatcher cycle. The Dispatcher is implemented on a cyclic basis, and the dispatching algorithm is triggered every $T_{DCY}$. At each dispatcher cycle, the Dispatcher serves its queues and transfers traffic to the FDL queues. When the dispatcher algorithm is triggered, it starts by processing DPH. After processing DPH traffic, the dispatcher serves DPL traffic until the $T_{DPL}$ is consumed *or* there is no more available time for the current dispatcher cycle. After processing DPL traffic, the dispatcher serves IPH traffic until the $T_{IPH}$ is consumed *or* there is no more available time for the current dispatcher cycle. Finally, after processing the IPH traffic, the dispatcher serves Best Effort traffic: one PDU from DP-BE queue if it fits the remaining time of the dispatcher cycle; one fragment from IP-BE queue if it fits the remaining time of the dispatcher cycle; this is repeated until the queues are empty or no traffic from the queues fit the remaining time of the dispatcher cycle. This process is illustrated and exemplified in Figure 4.10.

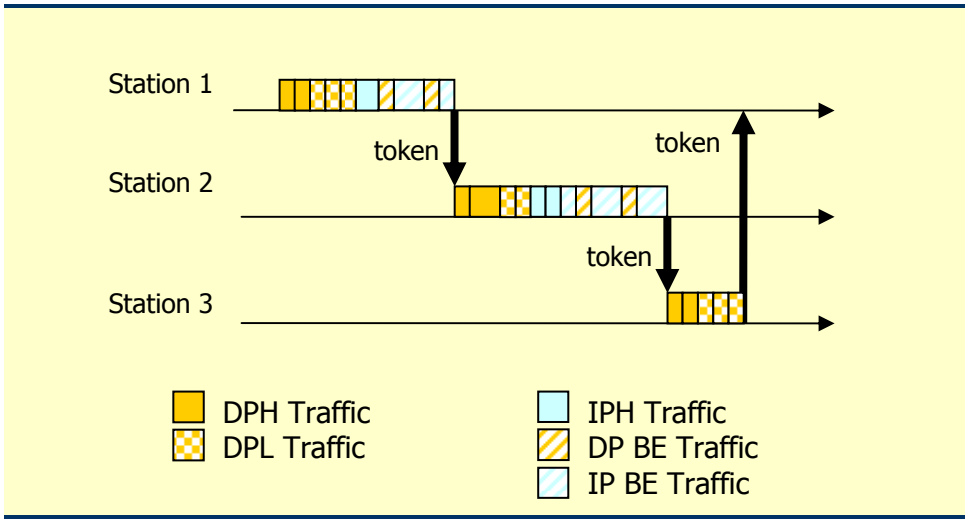



**Figure 4.10: Dispatcher traffic classes and timing concepts**

In each dispatcher cycle, the DPH Traffic that is forward to the FDL may be processed in three alternative ways:

1. all DPH PDUs are sent (results in standard Profibus processing of this kind of PDUs);
2. DPH PDUs up to the dispatcher cycle time are sent (guarantees that this station does not delay the token):
3. DPH PDU up to $T_{DPH}$ usage estimation limit are sent (guarantees that DPH traffic does not starve IPH traffic).

The alternative to be used is a configuration parameter of the station.

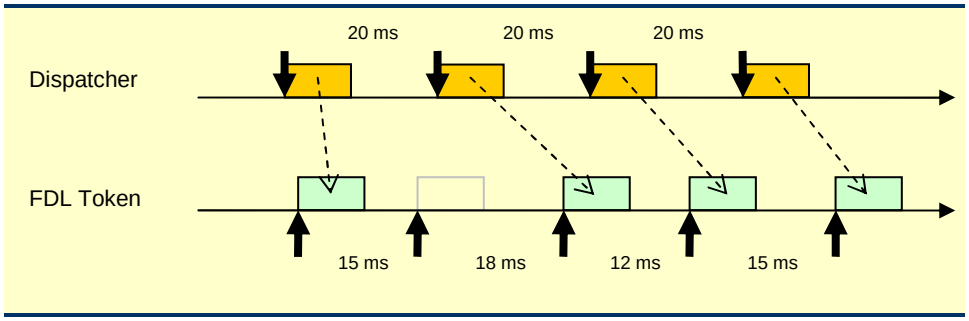



**Figure 4.11: Dispatcher and token timing**

Ideally, the dispatching activities should be synchronised with the token arrivals at the FDL layer, maximising the available throughput, since at each token arrival there would be, at most, the agreed number of PDUs to be transferred. However, such synchronisation is not trivial, since it would imply modifications to the Profibus FDL.

Then, in order to guarantee that the assumptions of the constrained low priority traffic profile are always satisfied, it is considered that the token arrives at the station at the same rate that the Dispatcher is executed, i.e. every $T_{DCY}$.

Consequently, the traffic throughput cannot be maximised, since there are some token arrivals when there is no traffic to be transferred at the FDL layer. For example, if $T_{DCY}$=20 ms and $T_{cycleaverage}$=15 ms, the traffic at station *k* would be processed as presented in Figure 4.11. However, the scheduling guarantees are always meet if the token is never late, this can be achieved using options 1 or 2 of the DPH processing at the dispatcher in all stations or guaranteeing that this condition is fulfilled by the application.

# Chapter 5
# Other Design and Implementation Issues

This chapter addresses a few details on how to implement the mechanims proposed in Chapter 5, namely concerning the Admission Control and Scheduling (ACS) mechanism, how packet fragmentation works and how to implement and configure a network scenario using commercial technology.

## *5.1 IP ACS Scheduler*

While in Section 4.4 a simple scheduling algorithm was proposed, the RFieldbus implementation can use any other scheduling policies that might eventually be more adequate for each specific application scenario. As an example, we present a Deferred-Release scheduling algorithm adapted from (Tovar and Vasques, 2001), that aims to minimize the minimum $T_{IPH}$ value, thus allowing the fulfilment of traffic with more stringent time requirements.

```
function deferred_release;

inputs:
   niph     /* IPH data flows */
   k[i]     /* array with number of fragments per period for each data flow */
            /* ordered ascendingly */
            /* i goes from 1 to niph */
   Ttmc [i] /* transaction duration */
            /* (TMC paramenter )*/
   Tdcy     /* TDCY value, the scheduler cycle */
   Mcy      /* number of micro-cycles in a macro-cycle */

outputs:
   sched[i,cycle] /* scheduling table */
                  /* cycle goes from 1 to n_µcy */
   offset[i]      /* offset relative to first micro-cycle */
   Tiph           /* TIPH value */

begin
1: /* offset calculation */
2: for i = 1 to niph do
3:    min_load = MAXINT;
4:       for cycle = 1 to (Ttmc[i] div Tdcy) do
5:          cycle1 = cycle;
6:          max_load = 0;
7:          repeat
8:          if load[cycle1] > max_load then
9:             max_load = load[cycle1];
10:         end if;
11:         cycle1 = cycle1 + (Ttmc[i] div Tdcy)
12:      until cycle1 > Mcy;
13:      if max_load < min_load then
14:         cycle_min = cycle;
15:         min_load = max_load;
16:      end if;
17:   end for;
18: end for;
19: cycle = cycle_min;
20: offset[i] = cycle_min - 1;
21:
22: /* update each cycle workload */
23: /* build scheduling table */
24: repeat
25:   load[cycle] = load[cycle] + Ttmc[i];
26:   sched[i,cycle] = 1;
27:   cycle = cycle + (Ttmc[i] div Tdcy);
28: until cycle > Mcy;
29:
30: /* get TIPH value */
31: tiph = 0;
32: for i = 1 to Mcy do
33:   if load[i] > Tiph then
34:      Tiph = load[i];
35:   end if;
36: end for;
return sched, offset, Tiph;
```

**Figure 5.1: IP ACS Deferred-Release algorithm**

Decreasing the value of $T_{IPH}$ results in more network bandwidth available for other traffic in other stations, or other traffic in the same station, but there is less "headroom" for compensating missed IP fragments (i.e. fragments that were expected in a given period from the TCP/IP stack but that for some reason were slightly delayed). The algorithm delays some fragments by one or two micro-cycles but never more than the data flow period so the final QoS of the flow is unaffected.

The algorithm presented in Figure 5.1, results in the scheduling scenario as illustrated in Figure 5.2.

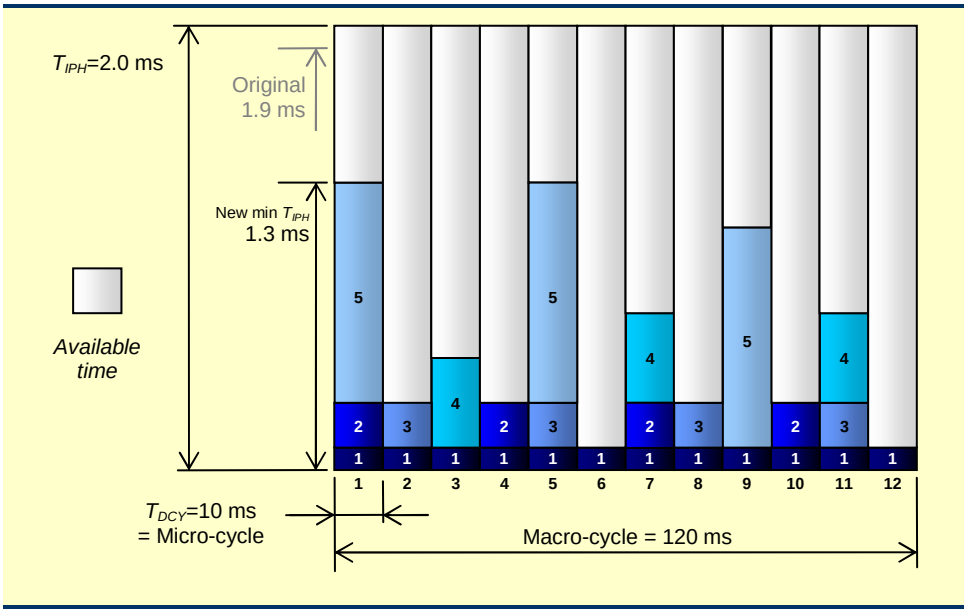



**Figure 5.2: IP ACS Deferred-Release scheduling example**

Another condition to take into account is when the scheduler has a slot available in a micro-cycle for a particular data flow but, for some reason, no fragment is available. This can occur due to the way applications generate data flow (that can be slightly bursty and not a perfect constant rate), due to TCP connection control mechanisms, Operating System and TCP/IP stack delays and even IP Mapper fragmentation delays.

An on-line compensation mechanism can be implemented to overcome this situation. The concept is that when there is a "miss" on the scheduler slot this can be compensated in the next micro-cycles if there is enough free $T_{IPH}$ time. In order to avoid a burst of compensation cycles in the future, a limit on the maximum number of compensated fragments must be set. In (Ferreira *et al.*, 2001) the value of this limit is defined according to the maximum acceptable jitter ($J_i$) for the application with period ($T_i$) and presented in Eq. (6.1).

$$N_i = \left\lceil \frac{J_i}{T_i} \right\rceil \quad \textbf{(5.1)}$$

The result of this compensation for an example scenario is presented in Figure 5.3. IPH1 generates fragments in a variable way, from zero to three fragments in each micro-cycle. However, the average for the full macro-cycle is one fragment per micro-cycle.

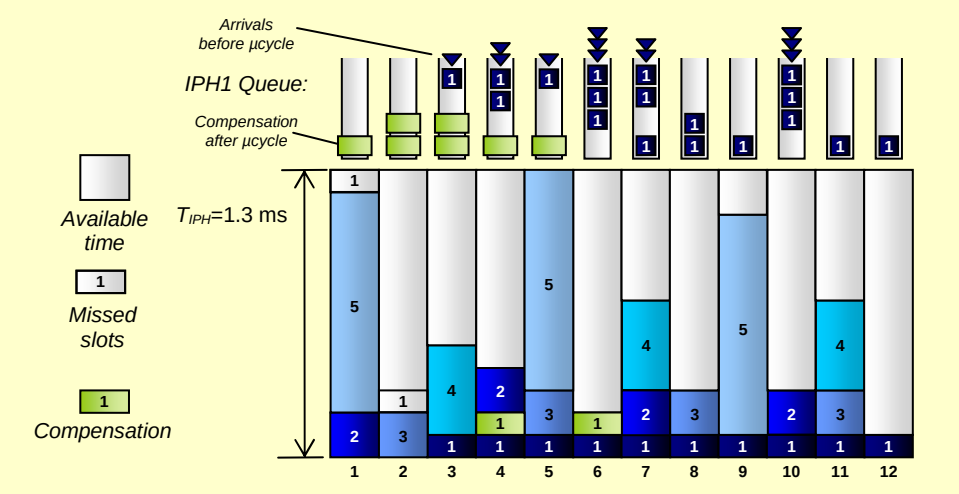



**Figure 5.3: IP ACS Deferred-Release Scheduling with Jitter compensation**

This compensation mechanism can be implemented over an off-line scheduling table, using an enhanced dispatcher, or it can be integrated in an on-line scheduling algorithm.

## *5.2 Configuring the RFieldbus Network*

As seen in Section 2.4, configuring a Profibus network with wired and wireless segments involves setting correct timing parameters such as $T_{TR}$, $T_{SL}$, $T_{ID1}$ or $T_{ID2}$. While this configuration is usually done in an intuitive way, this approach is inadequate when tackling more complex networks such as the ones supported by the RFieldbus architecture (multiple wired and wireless segments). A systematic presentation of this configuration was performed The main aspects of the approach are described next.

The starting point for the approach consists in summing up all the required master allocations and guarantee that $T_{DCY}$ is greater than that value. $T_{TR}$ is then set accordingly. Figure 5.4 illustrates the reasoning.

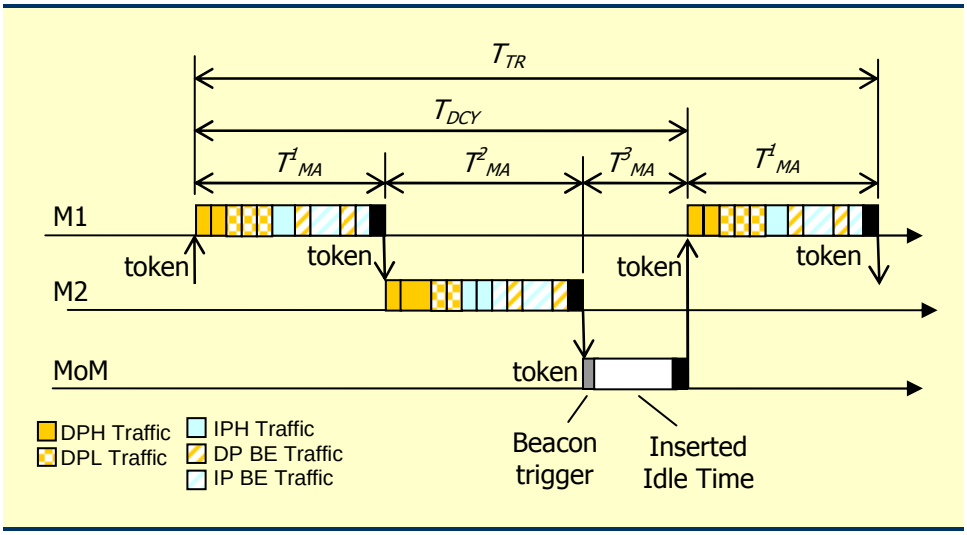



**Figure 5.4: Basic temporal parameters for network configuration**

In a more formal way:

$$T_{DCY} = \sum_{i=1}^{n} T_{MA}^{i} \leq T_{MIN} \quad \textbf{(5.2)}$$

Since $T_{DCY}$ is the maximum time that the token can take between visits to a particular master, this value must be smaller than the requested network response time ($T_{MIN}$), an application-dependent parameter. $T^i_{MA}$, the allocation time for each master can be calculated by:

$$T_{MA}^{i} = T_{DPH}^{i} + T_{DPL}^{i} + T_{IPH}^{i} + T_{BE}^{i} + T_{token}^{i} \quad \textbf{(5.3)}$$

In Figure 5.4, the $T_{TR}$ value is also illustrated. It is fundamental that $T_{TR}$ is high enough so that the token is never late. This means that $T_{TR}$ must allow for a complete $T_{DCY}$ cycle plus the transmission of all PDUs (including token passing) in the master with the largest $T_{MA}$.

$$T_{TR} = T_{DCY} + \max_{i}\{T_{MA}^{i}\} = \sum_{i=1}^{n} T_{MA}^{i} + \max_{i}\{T_{MA}^{i}\} \quad \textbf{(5.4)}$$

All these values are calculated depending on the transaction duration for each scenario. In case of an SRD message (confirmed request) this duration is:

$$T_{MCSRD} = C_{req} + T_{ST} + C_{resp} + T_{ID1} \quad \textbf{(5.5)}$$

In addition, for the simpler SDN message (unconfirmed request) this duration is:

$$T_{MCSDN} = C_{req} + T_{ID2} \quad \textbf{(5.6)}$$

In these last two equations, $C_{req}$ and $C_{resp}$ can be computed using the message size (in bits) and the bit rate. For simplicity, we use the largest possible value for these variables.

Regarding $T_{ID1}$, this parameter must be carefully calculated on RFieldbus systems due to the PDUs being relayed between mediums with different bit rates and frame formats. If $T_{ID1}$ is too small this can lead to buffer overflow in the relaying stations and thus to unpredictable PDU end-to-end delays. Setting $T_{ID2}$ has a similar impact, but now regarding SDN transactions. Finally, $T_{ST}$, the "timeout" value for a response of a slave must take into account delays due to queuing and bit rates.

Lastly, for the MoM (Mobility Master) station, an additional idle time $T_{ID2}$ must be inserted such that the wireless stations have enough time to perform radio channel assessment and hand-off, regardless of their location in the RFieldbus network (some mobile stations can be closer to the MoM than other). When the MoM sends the Beacon Trigger it is forwarded by all relaying stations in the network and detected by wireless slaves. After forwarding the Beacon Trigger, the base stations start sending a series of beacons spanning all pre-defined radio channels, during the mobility management period. In parallel, the mobile stations start probing each available radio channel and select (hand-off) the best available channel. The number of beacons each base station must send sufficient must take into account the delay in the beacon trigger relaying (eventually involving multi-hop) and the worst-case duration of the channel assessment by the wireless stations.

The complete reasoning for the computation of $T_{ID1}$, $T_{ID2}$ and $T_{SL}$ and of the mobility management parameters is available in (Alves, 2003)

## *5.3 Profibus Fragmentation Needs*

As presented in Chapter 2, Profibus supports up to 246 bytes of data per PDU. However, small stations can be limited to 32-64 bytes of data.

In contrast, a standard TCP/IP packet can have up to 64 Kbytes of data, which can be fragmented automatically by IP to match the *Maximum Transmission Unit* (*MTU*) of the forwarding networks. The minimum *MTU* is limited by the need to send one full IP Header per fragment, and has a theoretical minimum value of 68 (Postel, 1981). Most applications expect networks that can send fragments of around 1500 bytes – e.g. Ethernet – and up to more than 9000 bytes – e.g. ATM using AAL5 (Atkinson, 1994).

For instance, in a network with a *MTU* of 1500 bytes, a standard (i.e. with a 20 byte header) IP packet with 4000 bytes of data is divided into three fragments: two fragments with 1500 bytes (1480 bytes of data) and one fragment with 1060 bytes (1040 bytes of data). This means that there will be two extra IP headers due to fragmentation, and out of the total 4060 bytes, 40 are used to that purpose alone, meaning an overhead of 1% of the original data.

The same example in a network with very small *MTU* results in an excessive overhead. In a network with a *MTU* of 128 bytes, an IP Packet with 4000 bytes of data is divided in 38 fragments: 37 fragments of 128 bytes (108 bytes of data) and 1 fragment with 24 bytes (4 bytes of data). In this situation, the additional 37 IP headers due to fragmentation will now be 740 bytes, i.e. 18.5% of the original data. For a *MTU* of 64 bytes the overhead of the example packet goes up to 83% (!), and in the best-case Profibus scenario (*MTU* of 246) we have an overhead of 4.5%.

As a result, an RFieldbus-specific fragmentation scheme was devised. Each RFieldbus PDU has a 2-byte header (Figure 5.5). The first byte identifies the original IP packet (and is unsurprisingly named *Packet ID*), and can have values between 0 and 255. Each RFieldbus station generates its own *Packet IDs*, and so, *Packet IDs* are unique for each source. This circumstance limits the maximum number of concurrent (i.e. with interleaved fragments) IP packets that can be sent from a station to a maximum of 256. The second byte is used to identify the order of the fragment (*Fragment ID*) and can range from 0 up to 127. Value 0 is used to identify non-fragmented packets. Values higher that 127 are reserved for future use.

On reception, the actual length of the Packet is obtained from the IP Header in the first fragment. This limits the current protocol to be used on IP (version 4) packets, but it can be extended using the reserved range of the *Fragment ID*.

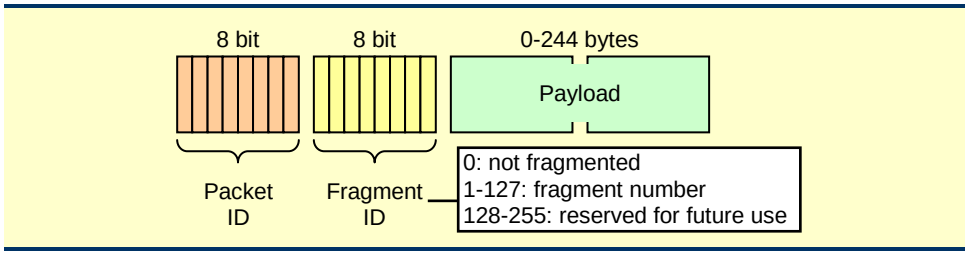



**Figure 5.5: RFieldbus packet format**

This solution has a minimum PDU data size of 22 bytes if IP packets without IP options are used, or 62 bytes to support all possible IP packets. Since we have a maximum of 127 fragments per packet, the maximum IP packet size (i.e. the *MTU* seen from the TCP/IP layers) will be around 8000 bytes even for Profibus networks with limited PDU data length of 60 bytes.

The overhead of the previous examples is now reduced significantly: a 4000-bytes IP packet has an overhead due to fragmentation of only 3,25% in a 64-byte Profibus network; and in the best-case scenario of a 246-byte Profibus network this goes down to 0,85%.

It should be noted that the current implementation is PC-based and the PDU length limit is not actually an issue, but in future implementations at more resource-limited stations, the problem may arise.

The maximum IP packet used in a network can also be limited due to the time needed to transmit the packet that can be relevant in time-critical applications. However, a Profibus-DP network at 12 Mbps takes only about 8 ms to send 8000 bytes of

unconfirmed data ((8000 data bytes + 33 * 11 header bytes) * 11 bits on the network per usable byte / 12 Mbps), a delay that might be acceptable for most TCP/IP applications.

Other side-effects that may limit the usage of very large *MTUs* in IP include Cyclic Redundancy Check (CRC) resiliency with *MTUs* greater than 10000 bytes (Jain, 1990), but Profibus has its own error-control scheme and our fragmentation mechanism discards the full packet if there is an error in any fragment.

The usage of large IP packets is possible due to the high reliability of the Profibus network itself. Since the network has a low error rate and IP has its own methods to overcome errors (using automatic resend for TCP Streams, UDP applications are aware that packets may be lost) no mechanism was implemented to resend lost fragments at the RFieldbus network. A simple timeout is implemented so that a lost fragment results in a discarded packet; with very large packets this may have a significant impact on the network performance. It is viable to implement such a mechanism in the future.

## *5.4 Windows NT Network Drivers*

The IP Mapper and Dispatcher were integrated on the Windows NT Network Drivers architecture, which is described next.

The Microsoft TCP/IP protocol suite is comprised of *core protocol elements*, *services*, and the *interfaces* between them, as illustrated in Figure 5.6. The *Transport Driver Interface* (TDI) and the *Network Device Interface* (NDIS) are public and their specifications are available from Microsoft. In addition, there are a number of higher-level interfaces available to user-mode applications. The two most commonly used are Windows Sockets and NetBIOS.

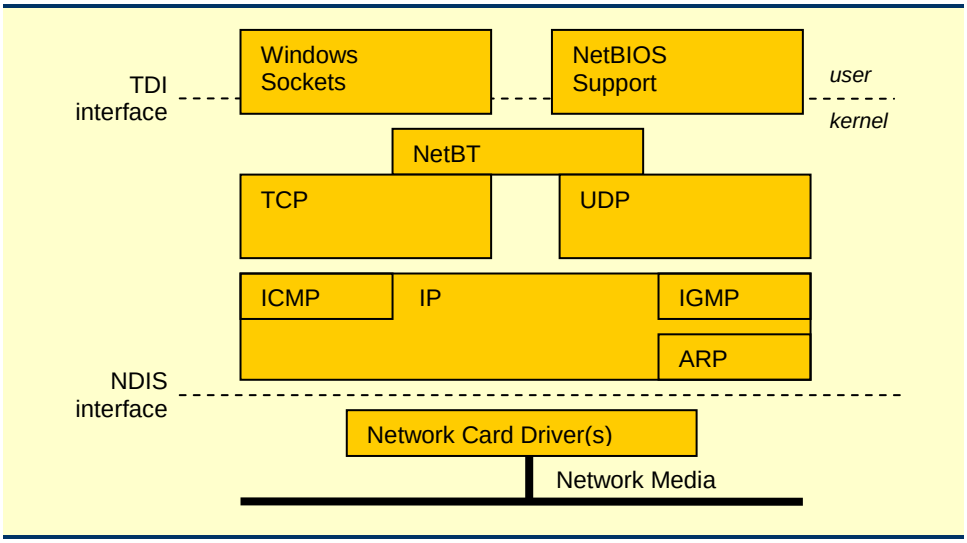



**Figure 5.6: Windows NT TCP/IP network model overview**

### 5.4.1 The NDIS interface and below

Microsoft networking protocols communicate with network card drivers using the NDIS. NDIS defines a fully abstracted environment for *Network Interface Card* (NIC) driver development. For every external function that a NIC driver needs to perform, it can rely on NDIS routines to perform the operation. This includes the entire range of tasks performed by a NIC driver, from communicating with protocol drivers, to registering and intercepting NIC hardware interrupts, and communicating with underlying NICs through manipulating registers, port I/O, and so forth. Therefore, NIC drivers can be written entirely in platform-independent high-level languages such as C. These drivers can then be recompiled with a system-compatible compiler to run in any NDIS environment.

NDIS includes features that simplify the driver development and integration including: single driver instance used to control all network adapters supported; a fully abstracted interface (ndis.sys); symmetric multiprocessor support; loopback support; multiprotocol support (protocols can be bounded to NDIS NIC drivers independently of implementations, including native Windows ARCNET and WAN Support); simplified administration; single or multiple packet per send request interfaces; additional information can be attached to packets (like QoS parameters) and full duplex operation on SMP machines.

### 5.4.2 Intermediate drivers

A NDIS intermediate driver usually exports *MiniportXxx* functions at its upper edge and *ProtocolXxx* functions at its lower edge. Less commonly, an intermediate driver can export *MiniportXxx* functions at its upper edge and a private interface to an underlying non-NDIS driver at its lower edge (Figure 5.7).

An intermediate driver is typically layered over one or more NDIS NIC drivers and under a transport driver (possibly multilayered), that supports TDI at its upper edge. Theoretically, an intermediate driver could be layered above or below another intermediate driver, although such an arrangement is unlikely to exhibit good performance.

An example of intermediate drivers is a LAN-emulator intermediate driver layered below a legacy transport driver and above a miniport NIC driver for a non-LAN medium. Such a driver receives packets in a LAN format at its upper edge, translates them to another NIC-native medium format and sends them on to an NDIS miniport for that NIC. On receives, this intermediate driver translates packets indicated up from the underlying NIC driver to a LAN-compatible format and indicates these converted packets to the upper level transport driver.

An intermediate driver can also be deployed below NDIS, when the Intermediate Driver depends on an underlying driver of a device other than a NIC. For example, an intermediate driver might handle network I/O requests for a device connected to a serial port. Such an intermediate driver would export a set of *MiniportXxx* functions to communicate with NDIS at its upper edge and use standard Windows NT I/O Request Packets (IRPs) to communicate with the underlying serial device driver at its lower edge.

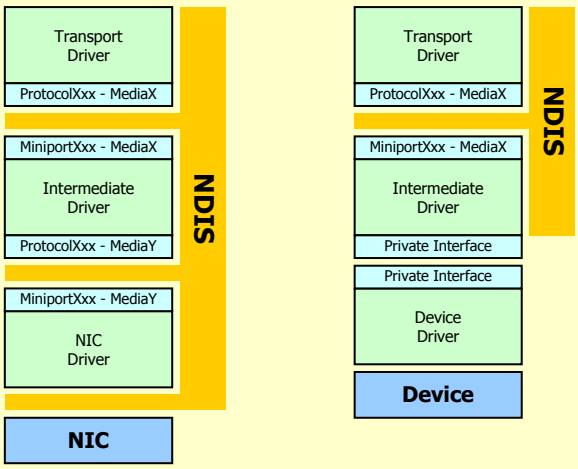



**Figure 5.7: Supported intermediate driver configurations**

## *5.5 RFieldbus Prototype Implementation*

The implementation of the RFieldbus prototypes is based on existing software, which supports Profibus master and slave functionalities. It consists of three main parts: the Profibus firmware; the NDIS miniport driver and the NDIS intermediate driver (Figure 5.8). In addition, a card driver *Dynamic Link Library* (DLL) is necessary for the Profibus control application.

Since Windows NT4.0 was used in the RFieldbus field trials, an NDIS miniport driver was needed in order to support the adequate interface to the TCP/IP stack or to underlying intermediate layers. On the other hand, an interface to the Profibus applications was necessary. These applications run in the user mode, wherefore the NDIS interface is not usable. That is why a *Windows Driver Model* (WDM) interface was also implemented, to support interfaces to both the Profibus application and the TCP/IP stack.

The device driver has to perform two main tasks: to set up the hardware access according to the different board types and to manage the exchange of service primitives between the TCP/IP protocol and the Profibus firmware.

The Device Driver is started with the Windows boot process. However, it rejects all send packet requests from the network protocols until the Profibus firmware is initialized. It is the task of the Profibus application to start-up the hardware via the Hardware Management features of the Miniport Driver. The initialization is performed in three steps. First, the Profibus application prompts the device driver to make a hardware reset to the board, maps the DPRAM (hardware interface) into the user mode

address space and returns the virtual DPRAM address. Then, the application initializes the required firmware protocol (master or slave) and forwards the offset of the IP command area of the DPRAM interface to the driver. The driver initializes its private interface to the Profibus firmware. Lastly, the Profibus application loads the network parameters into the firmware to activate the connection to the Profibus system.

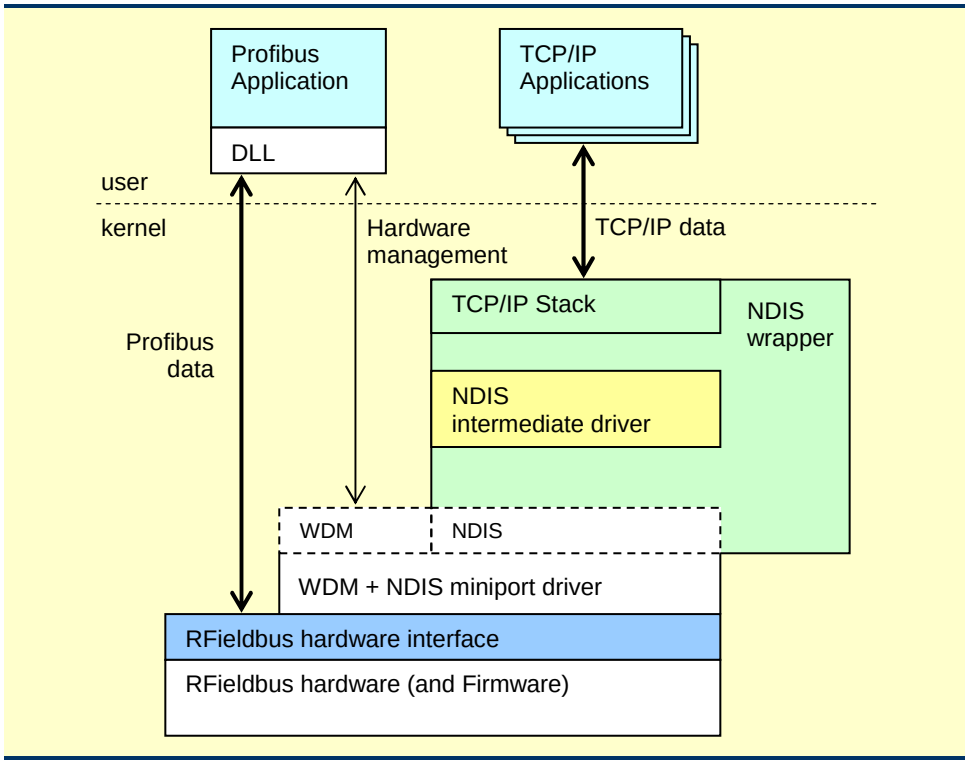



**Figure 5.8: RFieldbus NDIS implementation architecture**

After a successful firmware initialization, the device driver forwards send packet requests from the network protocols to the Profibus firmware and receive packet indications from the firmware to the protocols. Send packets are returned to the network protocols together with state information, when receiving the related send confirmations.

Receiving service primitives from the Profibus firmware is done by polling the report area of the DPRAM interface.

The Intermediate Driver is responsible for interfacing with upper level protocols (i.e. TCP/IP) on its upper edge, and with the lower Miniport. Mainly, the Intermediate Driver implements IP Mapper and IP ACS functionalities.

Figure 5.9 depicts the NDIS Intermediate Driver internal interface. Both IP Mapper and IP ACS modules rely on a common driver support facility, and interact using a defined function interface. The Intermediate Driver's entry/exit functions are NDIS standard calls.

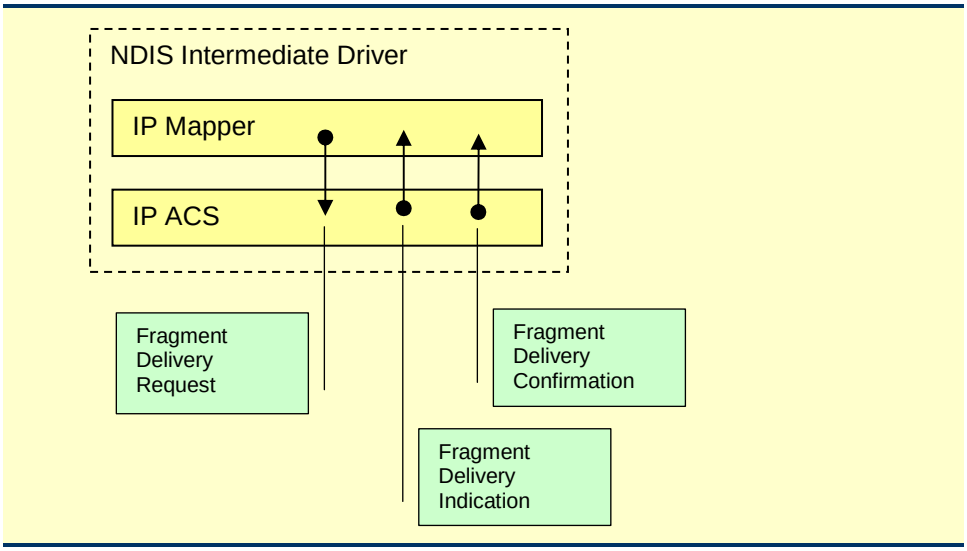



**Figure 5.9: RFieldbus NDIS Intermediate Driver Interfaces**

To send packets, TCP/IP indicates to the Intermediate Driver "send" function the data (represented by a NDIS defined structure) to be transmitted, using a standard NDIS call. In the opposite case, where a reassembled IP packet is ready to be forwarded to TCP/IP, another standard NDIS call is used to forward the data to the TCP/IP stack. To send and receive fragments, the IP ACS uses standard NDIS calls to communicate with the lower Miniport in a similar manner.

As for the IP Mapper/IP ACS interface, the three service functions described next are utilized. Upon reception of fragments from the lower layers, the IP ACS uses the *Fragment Delivery Indication* service function to forward them to the IP Mapper. By calling this function, the IP ACS passes to the IP Mapper the pointer to the fragment as well as its source address. This extra information is necessary for the identification of the fragment.

On the other hand, for fragments destined to the lower layers, the *Fragment Delivery Request* service function is used to send them to the IP ACS. For every fragment sent to the IP ACS, the IP Mapper expects a confirmation indicated by the service function *Fragment Delivery Confirmation*. This confirmation, sent by the IP ACS, depends on the delivery status of the fragment to the lower layers.

The Dispatching functionality is implemented in firmware.

### 5.5.1 NDIS Intermediate Driver details

The Profibus NDIS Driver implements most of the functionalities of the IP Mapper and ACS sub-layers, as illustrated in Figure 5.10.

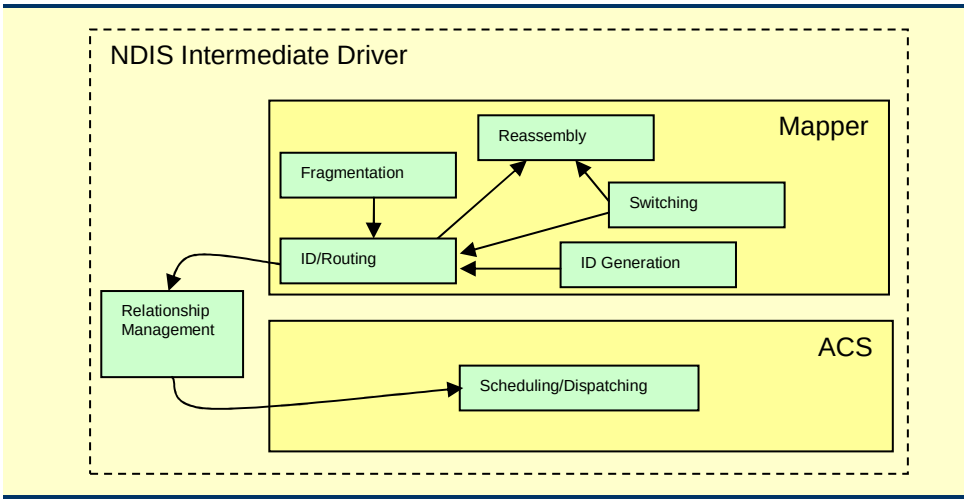



**Figure 5.10: RFieldbus NDIS intermediate driver functionality**

### 5.5.2 Sending packets

When the transport layer has a packet to send down to the network, it indicates this to NDIS that, in turn, calls the appropriate function registered during initialization. This was set to the "send" function of the Profibus NDIS Driver. This function is also responsible for the packet identification, making decisions about whether to send the packet or discard it, its fragmentation, and queuing in the proper Relationship.

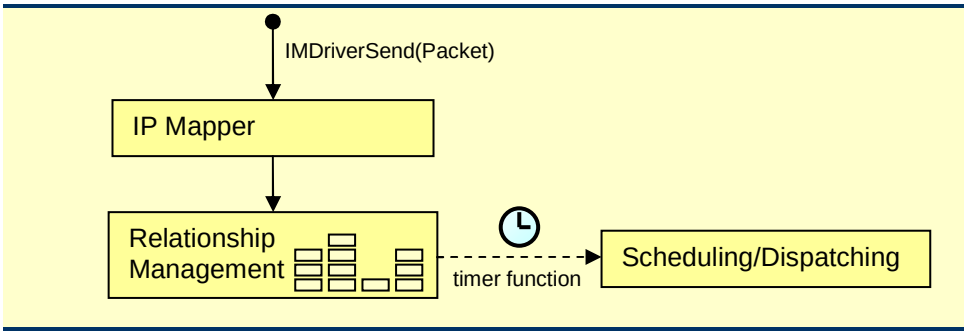



**Figure 5.11: Modules acting in the task of sending a packet**

A timer function is triggered at predetermined intervals. When the timer function triggers, the state of the Relationships is checked, and the scheduling of the fragments to send is defined. The fragments are then sent according to this scheduling. The

scheduling takes into account several parameters of the different IP streams to serve, according to a proper algorithm as described in Section 5.1.

In Figure 5.11 the functionality modules impacting the task of sending a packet are described.

The diagram in Figure 5.12 shows the processes involved when a packet is sent from the transport layer, until its fragments are stored in the appropriate relationship.

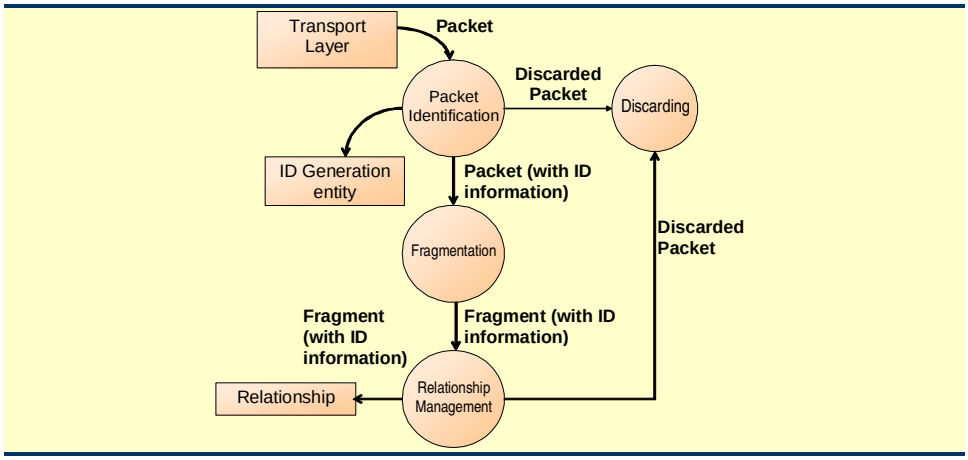



**Figure 5.12: DFD - Store packets to send in appropriate relationship**

When the timer function triggers, the relationships are checked, and emptied according to a scheduling algorithm as briefly illustrated in Figure 5.13.

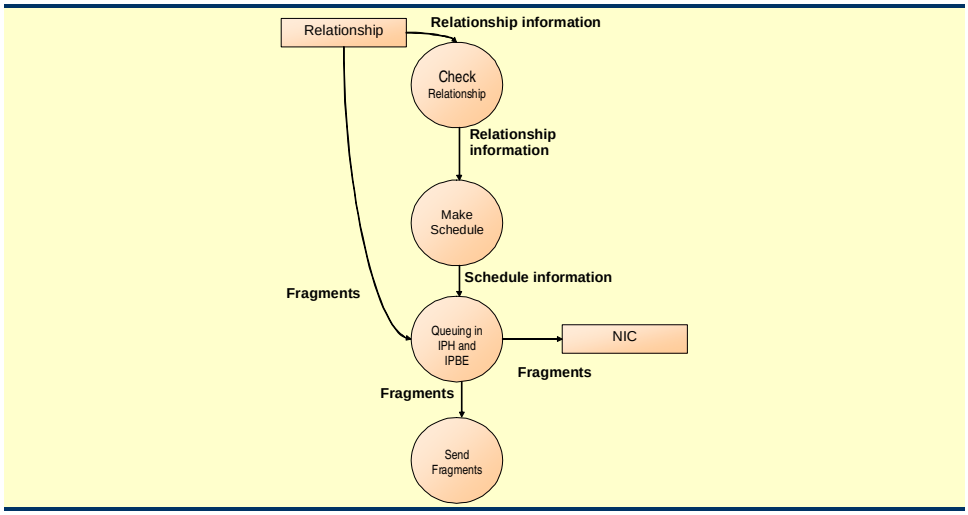



**Figure 5.13: DFD - Emptying of the relationships**

### 5.5.3 Receiving data

The reception of fragments is a simple task, as the NDIS Miniport passes up to the NDIS Intermediate Driver the received fragments. Upon this, the several fragments are buffered until the whole packet is received; at this time, the data is delivered to the upper layer, as presented in Figure 5.14.

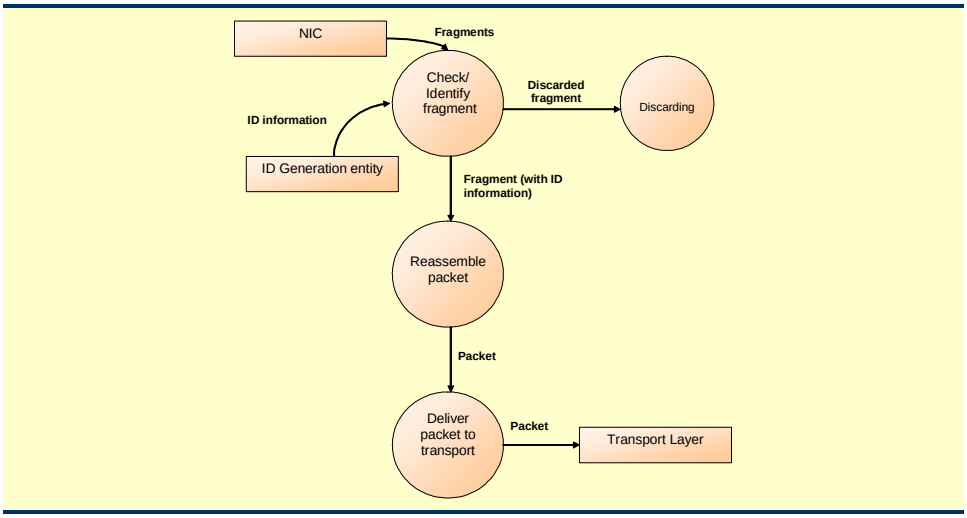



**Figure 5.14: DFD - Receive packets and deliver to upper layer**

# Chapter 6
# Validation

This chapter addresses the experimental validation of the mechanisms proposed in Chapter 5 and for which some implementation details were provided in Chapter 6. The Manufacturing Automation field trial of the RFieldbus European Project was used as a testbed. This enabled us to test and validate the feasibility and correcteness of the proposed mechanisms in a real and application-rich scenario.

## *6.1 Introduction*

The RFieldbus features presented in this thesis were tested in the RFieldbus Manufacturing Field Trial, which is described in Sections 6.2 and 6.3. Configuration, tests and discussion of results are described in subsequent section of this chapter.

## *6.2 The Manufacturing Automation Field trial*

The manufacturing automation field trial (IPP Hurray, 2002) involved the use of traditional *Distributed Computer Control Systems* (DCCS) and 'factory-floor-oriented' multimedia (e.g. voice, video) application services, supporting both wired and wireless/mobile communicating stations (mobile vehicles, for example). It was also a major goal that the manufacturing automation field trial would provide a suitable platform for RFieldbus timing (e.g. guaranteeing deadlines for time-critical tasks) and dependability (e.g. reliability) requirements to be assessed.

RFieldbus mobility requirements impose the use of wireless stations such as transportation vehicles and handheld terminals for supervision and maintenance. The manufacturing automation field trial also involved the use of wired segments, i.e. a hybrid wired/wireless fieldbus network.

One very important issue that was addressed in the manufacturing automation field trial was bringing multimedia applications into the factory-floor. Applications such as (mobile) on-line help for maintenance purposes and hazardous or inaccessible location monitoring are examples. The manufacturing automation field trial was designed to be an adequate test-bed to assess the suitability of the RFieldbus system to support both real-time control data and multimedia data in the same transmission medium, as deeply addressed in this Thesis.

To have an application gathering all the previously referred characteristics, an industrial (sub) system that transports, classifies and distributes parts according to a

certain criterion was specified (RFieldbus Project, 2002). The mechanical system imposes stringent timing and fault-tolerance requirements for the communication network supporting the diverse I/O points (sensors/actuators/servos).

### 6.2.1 Layout and Components

The manufacturing automation field trial implements a system that transports, classifies and distributes parts according to their type. Roller belts and different pneumatic equipment are used to transport and distribute parts to output buffers, according to their type. When output buffers are full, they are moved to the respective unload station, in order to be emptied. This operation is done either by an *Automatic Guided Vehicle* (AGV), a robot arm and an operator or just by an operator. Considering the classification criteria, we assume, at this moment and for the sake of simplicity, that a part type is distinguished by its colour. The physical layout is presented in Figure 6.1.

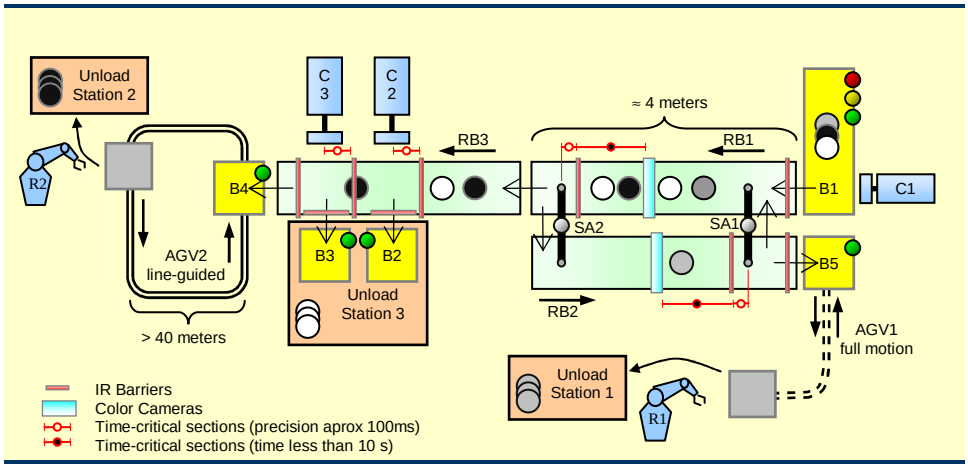



**Figure 6.1: Manufacturing field trial mechanical system layout**

The input buffer (B1) stores black, white and grey (defective) parts, which are sequentially pushed into the roller belt (RB1). SA2 (a swivelling double arm with suction cups) pushes grey parts to RB2. Grey parts go into B5. If this buffer is full or in transit grey parts must circulate around RB1-RB2. When B5 is full, AGV1 moves to U1, for unload operation carried out by a robot arm (R1) and an operator, and then returns to the initial position. White and black parts go into RB3, and white parts are pushed into output buffer (B2). When B2 is full, an operator is warned, in order to unload it. Meanwhile B3 must be used to receive white parts. If both B2 and B3 are non-operational (full or in transit), white parts must circulate in RB1-RB2. Black parts go into B4, until it is full or if it is in transit. When B4 is full, AGV2 moves to U2, for unload operation carried out by R2. Black parts must circulate around RB1-RB2, if B4 is unavailable.

### 6.2.2 RFieldbus Communication Subsystem

The RFieldbus Communication Subsystem includes all the RFieldbus equipment necessary to interconnect all the wired and wireless components of the distributed system. In order to test, validate and demonstrate the technical capabilities of the RFieldbus approach, a network infrastructure (Figure 6.2 - Left) including a wired segment and two radio cells was devised, forcing communication between wired and wireless stations and the handoff between radio cells.

In order to have a structured wireless network supporting mobility, the RFieldbus network infrastructure is composed of two Link Base Stations (LBS1, LBS2) that interconnect the two wireless domains (WL1, WL2) and the wired segment (WR). All stations are Profibus slaves (PC2-6, I/O1-2, PLC1 and Drive1-2), except PC1 and MoM, which are Profibus master stations.

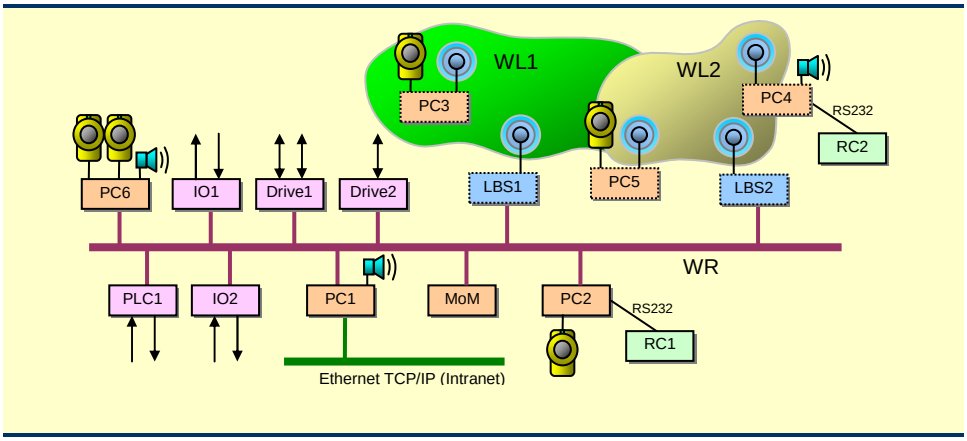



**Figure 6.2: Manufacturing field trial network topology**

### 6.2.3 Multimedia Streams

Several multimedia applications are used for control, monitoring and interpersonal communication. The correspondent data streams on the RFieldbus network are presented in Figure 6.3 and described next.

***TCP/IP Remote Part Classification (MM1, MM2):*** Two cameras in PC6 acquire images of the moving parts at a predefined rate. These images are down sampled and compressed to greyscale JPEG files. This data is then sent using TCP/IP connection to the remote machine (PC1). On the monitoring side (PC1), each received image is decompressed, processed to identify the presence of a piece and classify it.

***TCP/IP Remote Video Monitoring (MM3, MM4):*** This application enables the operator in the central control PC (PC1) to visually monitor the area in the trajectory of

the AGVs (AGV1 and AGV2). It must also have basic control facilities like start and stop the video stream.

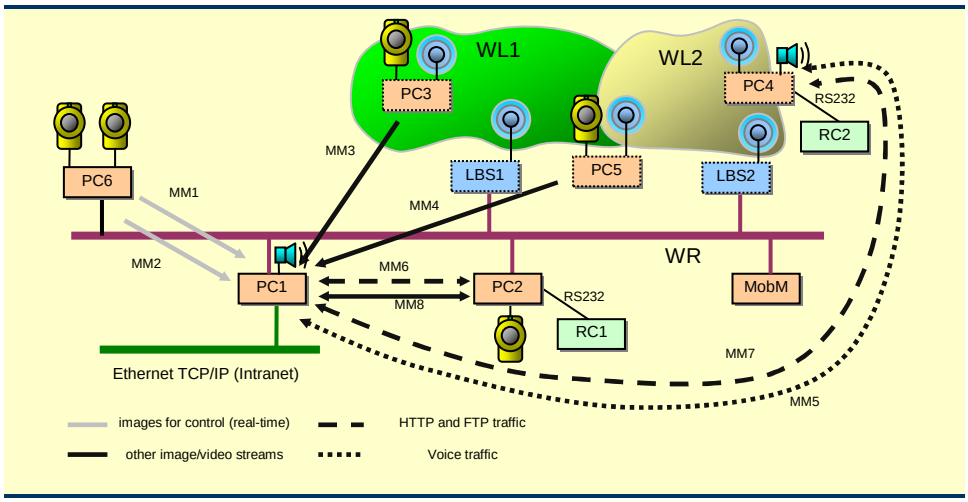



**Figure 6.3: Manufacturing field trial multimedia streams**

***TCP/IP Voice Connection (MM5):*** This is a simple point-to-point TCP/IP bi-directional voice application connecting PC1 and PC4. The only controls needed to the operator are "dial", "answer" and "hang-up".

***TCP/IP Remote Position Detection (MM6):*** The autonomous vehicle (AGV1) may slightly deviate from the ideal loading/unloading position. Therefore, a visual position detection mechanism was implemented in order to make the appropriate position corrections for the robot arm to manipulate the buffer. On the capture side (PC2), images are captured by request of the monitoring machine (PC1) and sent using TCP/IP connection to PC1. Each received image is decompressed and processed to identify the presence and location of the buffer in the Robot 3D coordinate system.

***Remote Robot Control Services (MM7, MM8):*** In order to be able to remotely control the two robots of the field trial, support to FTP and HTTP is provided (in PC2, PC4). The FTP servers are configured to enable the transference of program files to a specific directory on the computer. The WWW application enables the transfer of these files to the robot and interact with the Robot system itself.

***Intranet Interface Services:*** Several services are available for system monitoring and control using standard TCP/IP stations in the Intranet attached to PC1. Two ways to access this information where deployed. The **WWW Server** provides several HTML pages and forms where that the user can browse to check the current system status and interact (give the proper credentials) with the system. Any WWW browser can access this information. The **UDP Server** supports efficient broadcasting of information to several stations on the network. Specific clients where developed to interact with this server.

### 6.2.4 Supporting Technologies

In order to exploit the multimedia characteristics of the RFieldbus field trial, several additional technologies were integrated in the industrial automation system. Some of these technologies are not common in the factory floor now but there is a clear eagerness to start their widespread use with clear benefits (Pacheco *et al.*, 2002).

***Wireless Network:*** Despite the fact that RFieldbus wireless modems share the same unlicensed spectrum of the IEEE 802.11b wireless network standard, a Windows Laptop PC and a Pocket PC where integrated in the field trial with connections speed of up to 11Mbps and significant mobility with transparent connection to the field trial Ethernet segment using a IEEE 802.11b bridge.


**Figure 6.4: Using a HMD in the manufacturing field trial**

***Head Mounted Display:*** the HMD technology opens a new level in the way information is presented to the user. The display is in front of the user's eye giving (due to the lenses used) the sensation of a big monitor. A simple monocular gray-scale monitor was used, presenting basic information about the system status, alarms or need of user intervention, as illustrated in Figure 6.4.

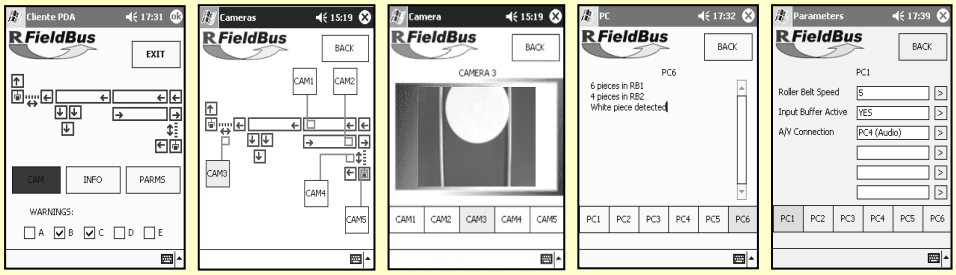



**Figure 6.5: Manufacturing field trial Pocket PC Client Application snapshots**

Available since more than a decade, Personal Data Assistants (PDAs) have been used almost exclusively for they main purpose: as an electronic version of the traditional

pocket agenda. In the last years many new applications and several operating systems have appeared. The latest generation of Pocket PCs has advanced features like: fast processors, full-colour displays, TCP/IP and WWW support, connectivity (irDA, 802.11b, Bluetooth), expandability (using PCMCIA, SecureDigital, etc.), GPS and more. In the manufacturing automation field trial, a PDA was used to fine-control the system and to get information about all stages of the trial (see Figure 6.5).

The system featured also a simple GSM SMS (Shot Message Service) gateway, and selected alarm classes were automatically forwarded to a particular phone with full text descriptions.

## *6.3 Low-level communication flows characteristics*

This sub-section presents a detailed view on all the expected traffic flows of the manufacturing automation field trial.

### 6.3.1 Cyclic DPH Traffic

Figure 6.6 depicts the traffic flows for the DP High Priority Traffic, used for real-time control functions.

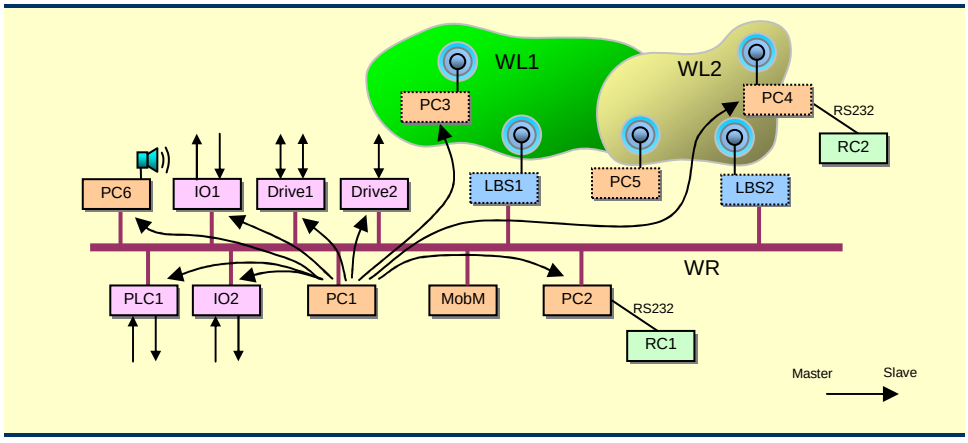



**Figure 6.6: Manufacturing field trial cyclic DPH traffic**

PC1 sends cyclic DPH PDUs to 9 slaves in the system, to update its outputs and to read its inputs. The swivel arms functionality is controlled by PLC1, a PLC with a Profibus DP module. It is possible to start a rotation operation on either swivel arms and to get information when the operation is complete and upon error events.

The roller belts are controlled by Drive 1 and Drive 2, two variable speed motor drives with Profibus DP modules. It is possible to start or stop each drive and to select the speed or get status information (like current speed).

Robotic arms R1 and R2 are controlled via PC2 and PC4, two PCs with Profibus DP network cards. The Profibus DP application enables remote start of a transfer operation and informs on the status of the operation (stopped, running, error). PC4 is also used to control AGV2 stop/start at the remote load/unload station and to detect its presence there.

AGV1 operation is controlled via PC3, a mobile PC with Profibus DP interface with wireless capabilities that is carried by the vehicle. AGV1 has full motion capabilities but the DP interface simply tells it to go to load/unload station 1 or 2 and gets information on current status (moving to 1, moving to 2, stopped, error).

IO1 and IO2 are Profibus DP Digital I/O modules used to control pneumatic cylinders, indicators and to get information on parts sensors and buffer status. IO2 is also used to control AGV2 stop/start and arrival at the local load station.

PC6 is used as a Profibus DP sound generator: specific DP commands activate a different sound like soft bell, telephone ring or warning message. It is a PC with Profibus DP interface, a sound card and speakers, it is an output-only station from the DP point of view. Finally, PC5 has Profibus DP capabilities and bandwidth reservation but these where not used by applications in the field trial.

### 6.3.2 Non-Cyclic DPL Traffic

The non-cyclic DPL Traffic is not related to the field trial application control, but only to "internal" Profibus DP generated traffic: logical ring maintenance, live list, etc. This is taken into account in the configuration of the master.

### 6.3.3 TCP/IP Traffic over Profibus DP

There are two main classes of TCP/IP traffic in the system:

- Guaranteed service (IPH): traffic that is essential for the correct functionality of the system as a whole. These include images used for part classification that must be transferred from PC6 to PC1 in a limited time, the voice-link application between PC4 and PC1 that must have adequate QoS parameters to be useful; video feeds from cameras at AGV1 and AGV2 (UDP traffic from PC3 and PC5 to PC1); and finally a frame capture application that is used to detect the precise position of AGV2 in UL1 and adjust the robotic arm operation accordingly.
- Best-effort service (IPBE): traffic that can be served after the critical parts of the system. This includes HTTP and FTP traffic used to manage the programs in the robotic arms (PC1 to PC2 and PC4).

### *6.4 System configuration*

The system is configured using a *System Planning Application* (SPA) developed in the RFieldbus project (Alves *et al.*, 2003). Given the temporal characteristics of the base networks and the characterization of the several information flows and endpoints the SPA calculates de system parameters for correct operation or informs the user that it cannot guarantee the system performance for the given scenario.

The basis for the SPA application are described in detail in (Alves, 2003) and where introduced in Section 5.2. The first parameters to be used in the configuration are the DLL characteristics summarized in the following table.

**Table 6.1: Field trial network configuration parameters**

| ***Parameter*** | ***Value*** |
|---|---|
| Bits per DLL character | 8 bits |
| Maximum PDU size | 255 chars |
| Minimum PDU size | 6 chars |
| Token size | 3 chars |
| Interconnection delay | 25 µs |
| $T_{IDmin}$ | 100 bits |
| Minimum $T_{SDR}$ | 10 µs |
| Maximum $T_{SDR}$ | 50 µs |

The Mobility-related parameters were considered as illustrated in Table 6.2:

**Table 6.2: Field trial mobility management parameters**

| ***Parameter*** | ***Value*** |
|---|---|
| Beacon Trigger PDU size | 10 chars |
| Number of radio channels | 2 |
| Beacon duration | 200 µs |
| Beacon interval | 25 µs |
| Radio channel switching delay | 700 µs |
| Buffering delay | 25 µs |

The parameters for wired/wireless interoperability support as described in Table 6.3.

**Table 6.3: Field trial wired/wireless parameters**

| ***Parameter*** | ***Wired*** | ***Wireless*** |
|---|---|---|
| Transmission rate | 1.5 Mbps | 2 Mbps |
| Header size | 0 bits | 180 bits |
| Trailer size | 0 bits | 32 bits |
| UART character size | 11 bits | 8 bits |
| Offset | 33 bits | 148 bits |

Provided this data, the SPA can calculate $T_{MC}$ for all information flows in the system. The value depends also on the data payload length, the type of requests and the

path between stations. Although the system has multiple link stations, the architecture can be simplified for timing calculations and reduced to a wired bus (with master, slaves and MoM) and a wireless domain (with slaves). In this simplified scenario, there are only exchanges between the master and a generic slave on the wired segment and exchanges between the master and a generic slave on the wireless domain.

**Table 6.4: Data flows for field trial configuration**

| *SPA Flow* | *Master* | *Slave* | $C_{req}$ | $C_{resp}$ |
|---|---|---|---|---|
| A1 | Wired | Wired | 11 chars | 11 chars |
| A2 | Wired | Wired | 6 chars | 255 chars |
| A3 | Wired | Wired | 12 chars | 13 chars |
| A4 | Wired | Wired | 21 chars | 21 chars |
| A5 | Wired | Wired | 25 chars | 25 chars |
| A6 | Wired | Wireless | 6 chars | 255 chars |
| A7 | Wired | Wireless | 11 chars | 11 chars |
| A8 | Wired | Wireless | 39 chars | 255 chars |
| A9 | Wired | Wireless | 255 chars | 255 chars |

Other system parameters that had to be calculated include the $T_{DCY}$ value, which is equal to the maximum period between requests so the application can work correctly. On the manufacturing automation field trial case, the time-critical events occur between the detection of a part using the infrared sensors and the time to activate one of the actuators (pneumatic cylinders or swivel arms). If the delay between requests is too high, there is the risk that the actuators do not handle the part correctly; a value too small overburdens the network with no direct benefit.

The system was put into operation with several $T_{DCY}$ values and we concluded that a value smaller or equal to 100 ms resulted in adequate system performance. A more formal approach could take into account the roller belt speed (0.15 m/s), the acceptable positioning error (about 20 mm), the sensor delay (2 ms), the combined valve and actuator delay before hitting the part (25 ms). Some of these values, in particular the actuator delay and the position error, are only available as an estimation. The final $T_{DCY}$ value is then:

$$\max T_{DCY} = T_{Position} - T_{Sensor} - T_{Actuator}$$

$$\max T_{DCY} = \frac{\varepsilon_{Pos}}{v_{RB}} - 2ms - 25ms = \frac{0.02}{0.15} - 27ms = 106ms \qquad \textbf{(6.1)}$$

The values for $T_{ID1}$ and $T_{ID2}$ are calculated by SPA for each master in the system and for the token. This resulted in $T_{ID1}$ = 293 bits, $T_{ID2}$ = 3442 bits and $T_{ID1token}$ = 393 bits. The values for Profibus configuration are min{$T_{SDR}$} and max{$T_{SDR}$}, and given the fact that other values that could affect this sets are two small when compared to the results we can set min{$T_{SDR}$} = 393 bits (the greater of $T_{ID1}$ and $T_{ID1token}$) and max{$T_{SDR}$} = 3442 bits.

**Table 6.5: Transaction and token delays for the field trial**

| *Flow* | *Type* | *Stations* | $T_{ST}$ | *Transaction Duration* |
|---|---|---|---|---|
| S1 | DPH | PC1↔PC6 | | 710 bits |
| S2 | DPH | PC1↔PC3 | | 2205 bits |
| S3 | DPH | PC1↔PC5 | | 2205 bits |
| S4 | DPH | PC1↔PC4 | | 2205 bits |
| S5 | DPH | PC1↔PC2 | | 710 bits |
| S6 | DPH | PC1↔Drive1 | | 930 bits |
| S7 | DPH | PC1↔Drive2 | | 930 bits |
| S8 | DPH | PC1↔IO1 | | 743 bits |
| S9 | DPH | PC1↔IO2 | | 643 bits |
| S10 | DPH | PC1↔PLC1 | | 1018 bits |
| S11 | IPH | PC1↔PC6 | | 3427 bits |
| S12 | IPH | PC1↔PC3 | | 6023 bits |
| S13 | IPH | PC1↔PC5 | | 6023 bits |
| S14 | IPH | PC1↔PC2 | | 3713 bits |
| S15 | IPH | PC1↔PC4 | | 8778 bits |
| Token | | PC1↔MoM | 393 bits | 443 bits |

Transaction delays are used to set the $T_{MC}$ parameter of Profibus network, the maximum transaction duration is 2205 bits and so this is the $T_{MC}$ value.

For the IP Mapper, the transaction delay is also used to estimate the *Target Message Cycle Time* ($T_{TMC}$) of each IP flow. The estimation can be done in three ways: (i) using the same value for all data flows; (ii) using a table with flow IDs and the value; and (iii) calculated on a PDU-by-PDU basis. On the manufacturing automation field-trial, the second option was used, providing a balance between accuracy and system complexity.

The SPA also calculated that 10 beacons were necessary for the given configuration and $T_{BT}$ to be 180 bits. This results in an overhead for mobility management $T_{MAMoM} \approx 4100$ bits. Combined with the aggregate intermediate allocation AIM ≈ 25000 bits needed for the data flows we conclude that the network can easily handle the load since $T_{MAMoM}$ + AIM (4100 bits + 25000 bits) is still much less then $T_{DCY}$ (150000 bits).

## *6.5 Scheduling of TCP/IP Traffic*

The micro-cycle to be used is the worst token rotation time, i.e. 150000 bits. While the overall system involves six IPH flows, we reduced it down to five considering the traffic from the two identical applications as S11 (but using them separately).

To determine the number of fragments per second used by each application, a TCP/IP capture application was used testing the data flows during an adequate period. For each capture, the number of fragments was calculated considering that IP packets larger than 240 bytes are fragmented.

**Table 6.6: Transaction and token delays for the field trial**

| *Flow* | *Frags/s* | $T_{frag}$ | *TCP/IP Application* | *Stations* |
|---|---|---|---|---|
| S11 (x2) | 5 | 2 | Color Detection Application | PC1↔PC6 |
| S12 | 10 | 1 | Image Stream | PC1↔PC3 |
| S13 | 10 | 1 | Image Stream | PC1↔PC5 |
| S14 | 1.67 | 6 | Image Position Application | PC1↔PC2 |
| S15 | * | 0.5 | Bidirectional Voice Call | PC1↔PC4 |

Since the voice application (S15) generates a variable data flow, the correct value was obtained via experimentation: several scheduling tables were used and the system tested. The conclusion was that 2 fragments per each $T_{DCY}$ resulted in adequate application behaviour, with lower values leading to a degradation of the voice quality.

Given this data and using the algorithm described in Section 5.1 we got the scheduling table of IPH fragments at PC1.

**Table 6.7: Scheduling table for IPH data flows in PC1**

| *Micro-cycle* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| S15 | 2 | 2 | 2 | 2 | 2 | 2 |
| S12 | 1 | 1 | 1 | 1 | 1 | 1 |
| S13 | 1 | 1 | 1 | 1 | 1 | 1 |
| S11 | | 2 | | 2 | | 2 |
| S14 | 1 | | | | | |
| $T_{IPH}$ (bits) | 33315 | 36456 | 29602 | 36456 | 29602 | 36456 |

The maximum $T_{IPH}$ is aprox. 36500 and the network is still viable since *AIM* + $T_{MoM}$ + $T_{IPH}$ (25000 + 4100 + 36500 = 65600 bits) is still much less than $T_{DCY}$ (150000 bits).

Finally, the $T_{TR}$ and $T_{SL}$ Profibus parameters must be adequately set. For the $T_{TR}$ calculation there is the need to estimate the traffic allocation for PC1. Considering just the IPH, DPL and IPH traffic we have $T_{MAPC1}$= 2250·10+2205+36500+443 ≈ 61200 bits. We can round up this value to 70000 bits and the remaining is used for BE traffic. This results in a $T_{TR}$ of 150000+70000 = 220000 bits. Finally, for $T_{SL}$, we look to the greatest of $T_{SL1}$ and $T_{SL2}$ that takes the greatest value of $T_{ST}$ and $T_{STtoken}$ respectively. Since $T_{SL1}$ is 2775 bits and $T_{SL2}$ is 393 bits, $T_{SL}$ is set to 2775 bits for PC1.

## *6.6 Manufacturing automation field trial results*

The manufacturing automation field trial applications behaved as planned (Machado, 2006), (Van Nieuwenhuyse and Behaeghel, 2003) with no interference between multimedia (TCP/IP) traffic and control (Profibus) traffic. Both DP and TCP/IP applications performed as expected and a Profibus network/protocol analyser enabled to confirm that the traffic in the network was as expected, given the pre-defined data stream scheduling. The only exception was the image identification application where sporadically some pictures were not grabbed by the system.

The system planning was done in a way to leave some extra bandwidth available for encompassing new applications or modifications to the existing ones. Thus, after the first tests phase the system was readjusted. Two of the applications that could most benefit from additional network resources were the image position application (due to the scheduling policy, an image would take about 30 seconds to be transferred over the network) and the voice-call application (that had a start-up delay of about 1 second).

The first step was to solve the issues with the S11 stream. The solution was simply to give more bandwidth to the application, as shown in Table 6.8.

**Table 6.8: Revised scheduling table to improve S11**

| ***Micro-cycle*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** |
|---|---|---|---|---|---|---|
| S11 | 2 | 2 | 2 | 2 | 2 | 2 |
| S15 | 2 | 2 | 2 | 2 | 2 | 2 |
| S12 | 1 | 1 | 1 | 1 | 1 | 1 |
| S13 | 1 | 1 | 1 | 1 | 1 | 1 |
| S14 | 1 | | | | | |
| $T_{IPH}$ (bits) | 40851 | 37028 | 37028 | 37028 | 37028 | 37028 |

Afterwards, the scheduling parameters where changed and the image position application response time was decreased from 30 to 10 seconds, by using a macro-cycle of 2 instead of 6, resulting in 1 fragment every 2 micro-cycles for S14, instead of the original 1 fragment every 6 micro-cycles. Another change was to use 3 fragments (instead of 2) per micro-cycle for S15, the voice-call application, but this time this adjustment resulted in a marginal start-up delay reduction and no noticeable voice quality improvement.

All these changes where cumulative and did not affect in any way other applications in the system. At this moment we had a $T_{IPH}$ that was about half of the maximum $T_{IPH}$ so there was still room for further application bandwidth upgrading.

Things got more complicated when larger bandwidth was allocated for S11, resulting in the following scheduling table:

The system responded with a fast degradation of transmission capabilities in all applications until it completely crashed. This was unexpected since the allocated bandwidth was still far from the maximum possible for TCP/IP traffic. Testing a similar change with S15 (Voice) stream got similar results: system-wide degradation and lack of TCP/IP functionality after a minute of operation.

**Table 6.9: Revised scheduling table to 2nd improvement on S11**

| ***Micro-cycle*** | ***1*** | ***2*** |
|---|---|---|
| S11 | 5 | 5 |
| S15 | 3 | 3 |
| S12 | 1 | 1 |
| S13 | 1 | 1 |
| S14 | 1 | |
| $T_{IPH}$ (bits) | 60768 | 56945 |

After low-level scrutiny (using Windows NT kernel level debugging capabilities) we detected that the problem was related to the way Profibus hardware (IFAK _isPRO ISA) used in the PCs handled large bursts of fragments. The system would not work correctly when the dispatcher was configured to send more than 9 fragments per micro-cycle. The problem was overcome by decreasing (if possible) cycle times.

Another problem detected was that $T_{IPH}$ cannot be too large, or the TCP/IP stack stops working correctly due to timeouts. At the time, it was not investigated if Windows NT parameters (of the stack) could be changed to avoid this situation.

Finally, a test was done to check the result of overflowing the IP ACS queues: using an UDP application we sent 1 fragment per micro-cycle when in reality the scheduler only handles 1 fragment per 2 micro-cycles. As expected, the fragments started suffering long delays and some were lost (as seen from the application). Unfortunately, it was not possible to test alternative configurations during the field trial, like limiting the queue size at the REs and time-stamping of UDP data payload, so the only conclusion is that the system does not crash when a queue overflows.

# Part III
# Power-Line Communication System

# Chapter 7
# Proposed Architecture

Power-line communication (PLC) provides a natural medium for electrical energy distribution applications like metering and grid control. However, the medium itself is a harsh one when considering long distances, large number of stations, and wildly varied physical configurations used by each energy provider over the world. This leads to the availability of a basic master-slave network, with resilient service but limited capabilities. Taking advantage of the dual-level voltage used in the end leafs of the distribution grid, the proposed Energy Management System connects two master-slave networks and provides complete bi-directional, end-to-end, services over this two-level system. This chapter provides both the system-wide architecture details, to particular solutions found to ease the development of the embedded software

## *7.1 System Objectives*

As presented in Chapter 3, within the REMPLI system, the Transport Layer (TL) is the fundamental communication layer dealing with setting up bi-directional, end-to-end, communication in the energy management system. This layer allows a direct link between the AP and Node devices, on top of the basic master-slave network provided by the power-line communication subsystem.

It does so fulfilling the following main objectives:

1. implement high/level services like confirmed unicast packets, response request service and alarms;
2. support of unlimited[4] packet size in the above services;
3. fast reaction times for small requests;
4. enable usage of medium voltage and low voltage power distribution networks as a single data network with a flat address space;
5. be resource conscious in terms of network usage, processing power and memory needs;
6. provide a simple priority-based scheduler than can be updated to other alternatives

Since the underlying REMPLI Network Layer is designed to be used directly by a single application there is no direct support for application multiplexing at this level: the

[4] Limited only by available memory and the impossibility of processing blocks larger than $2^{32}$ bytes in the target software/hardware system.

REMPLI Transport Layer is the only client of the Network Layer; and the DeMux is the only customer of the Transport Layer (see Figure 7.1[5]). The services provided to the DeMux by the TL were designed to be used by Metering and SCADA applications, also enabling the deployment of new solutions in these areas. The Unicast service is focused on commands, the Request/Response on gathering information and the Alarm service on reverse direction transmission of events.

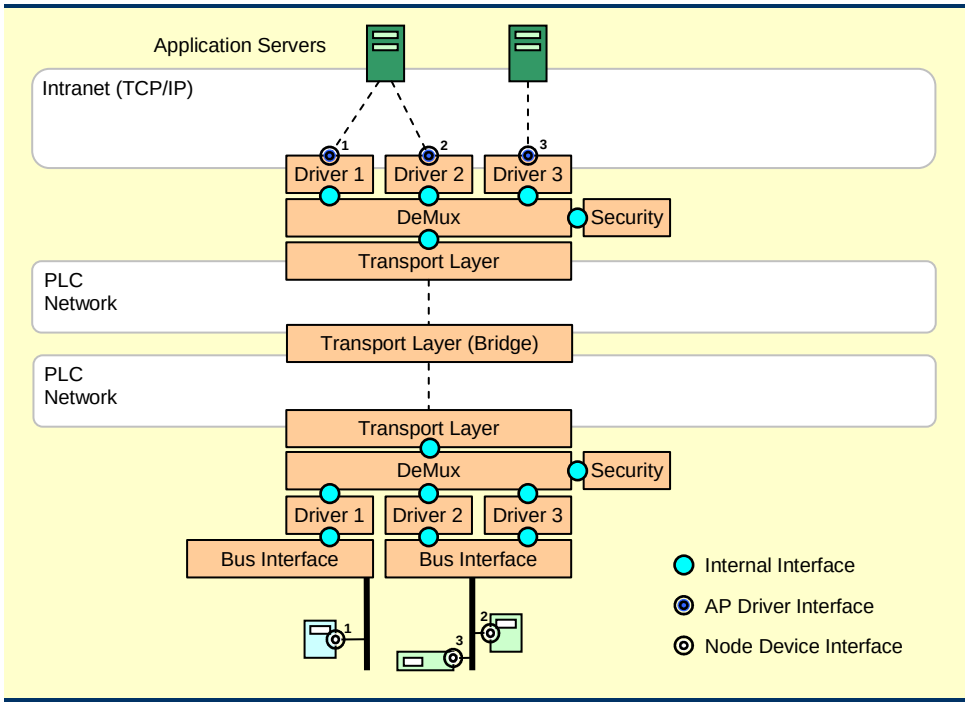



**Figure 7.1: REMPLI Upper Layer Functionality ("inside" view)**

SCADA and Metering applications have variable Quality of Service needs. However, one common mandatory feature is that short packets (e.g. smaller than one hundred bytes) are typically issued frequently and needing to be processed quickly by the system. This is particularly valuable not only for remote control of devices but also beneficial to an adequate scalability of the network. Also important is the possibility of enabling some traffic, like urgent control commands, to "overpass" background traffic like daily meter readings. The Transport Layer provides such services with the priority based scheduler and diminutive network overheads. Nevertheless, it is also open allowing the exploration of distributed scheduling mechanisms for new and improved services (these are out of the scope of this work).

[5] This is Figure 3.4 repeated here for completeness

The target applications also imply the deployment of a large base of end-user stations, meaning that the system must be cost-aware. To make this possible, the end-user stations have to be inexpensive and efficient (as presented in point e) above) in the list above. To put the objective in perspective one of the open paths of the project is to implement the Nodes on low-budget 8051-class processors in the future. Also of paramount importance is the possibility of updating the software (firmware, other program files or data) in these stations in an efficient and simple form. This is simplified considering that the Transport Layer services can be used unchanged for large packets. In fact, even on the current version with "limited" 24-bit lengths (16 MiB) it is unlikely that a Node station has enough memory to process the largest sized packet.

The usage of the power distribution grid as a communication medium eases some typical deployment problems like placing new cables and providing power to stations. However, stringent regulatory limitations restrict the usable bandwidth and the extreme geographic distribution of some layouts implies error resilient coding at the cost of bandwidth. The Transport Layer must take all this limitations in account and be aware that both physical and logical network topologies can change over time. Power grids are not a static arrangement of links and in normal operation new connections are dynamically created and others are removed. This effect is also present in other areas of the power grid in terms of propagation of the communication signals: activation and deactivation of noise sources can occur in an unpredictable fashion. The Transport Layer capability of connecting the Medium Voltage and Low Voltage networks in multiple points makes it possible to overcome these drawbacks efficiently.

On the other hand, dynamic network configurations should not be an issue for the end-user of the system (the utility companies) and so a flat address space is provided that effectively hides the system hierarchy and topology. It also enables simple field station replacement: the *Node Address* used by application is maintained and only the table that maps addresses to serial numbers has to be updated.

The remaining of this chapter presents the main features of the Transport Layer, starting with network layer login/logout and address conversion, needed for providing a flat address space to drivers. Afterwards the routing and distributed link quality mechanisms are explained, followed by the slave-to-master communication capabilities. Traffic priority schemes and the Alarm service functionality conclude the architecture overview.

## *7.2 Login/Logout processing and Address conversion*

To gather base routing information about the system, the Transport Layer keeps track of Login and Logout events. When a slave station connects to a master station at the Network Layer level, the Transport Layer in each side receives information on the events including the *Unique Serial Number* and the *Network Address* (*NLAddr*) assigned at the moment to the slave station. This information is used for address conversion from the "flat" address space seen by the Applications to the temporary login/logout addresses (*NLAddr*) used by the Network Layer. A configuration table includes *Unique Serial Number* and corresponding *Node Address*. This information is appended with the

*NLAddr* provided by the Network Layer when a station logins, and in order to build the route tables, the Transport Layer uses NL Login/Logout information and link quality information forwarded from remote nodes. Figure 7.2 presents the complete Login processing steps, which are managed by the *Transport Route Manager* (TRM) a sub-module of the Transport Layer.

When a new Bridge is connected by the Network Layer to an AP, two Login events arise: one at the AP and the other at the Bridge. The Bridge Login event includes information on the newly active Network Unit (this information is stored by the TL in a table with a fresh *BridgeID*). This *BridgeID* is used to inform the Nodes of the original packet source when needed.

Meanwhile in the AP side the Login event includes not only NL addressing information (*NLUnit* and *NLAddr*) but also the *Unique Serial Number* of the Bridge. All this information is stored in the local routing table with the matching *Node Address* a *BridgeID* of 0 (to signal that this is a direct connection). This data is needed not only for routing, but also to access the Node functionality of the Bridge itself.

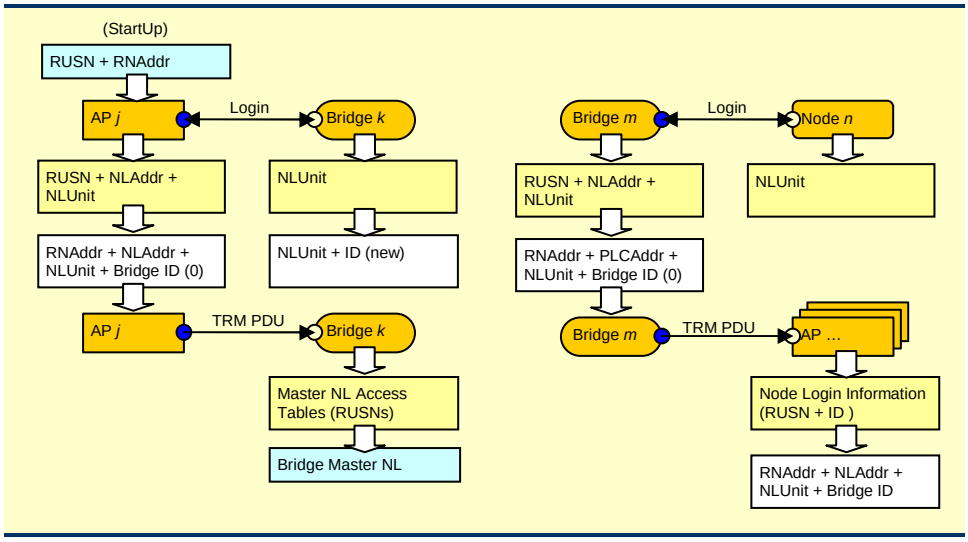



**Figure 7.2: REMPLI login processing**

The AP then sends a list with the authorized Nodes that can be connected to the Bridge's Network Layer. The configuration of these tables depends on the dimension of the network: in simpler networks can be an "allow-all" list, a list shared by all bridges, or a per-Bridge list on large systems. When a Bridge and a Node are newly connected, another set of events occurs. At the Login event, the Node simply stores the activated *NLUnit* identifier. This information is used to generate Alarm packets.

On the Bridge side, the Transport Layer starts by generating a new *BridgeID* and storing the route information of the new node. Afterwards the Bridge forwards the new route information (*Unique Serial Number* + *BridgeID*) to all the available APs. A Bridge with attached Nodes that connects to a new AP sends this information to the newly

connected AP. On the AP side, the received information is included in the routing tables that contain information on all the possible paths from this AP to a particular Node.

The process for a Node directly connected to the AP the new connection information is stored at each side and no TRM PDUs are exchanged.For the Logout processing (Figure 7.3) the main tasks is to clear the obsolete tables and to discard any pending requests that where using the disconnected path.

If a Node logs out from a Bridge then the Bridge informs all the connected Access Points of the event using a TRM PDU and they react accordingly.

When a Bridge disconnects from a particular AP, it does not inform the Nodes of this event. The Bridge ignores any pending responses that the node tries to send back to the disconnected AP. If needed, it is the task of Node Drivers and Applications to generate traffic to guarantee that the link is still active.

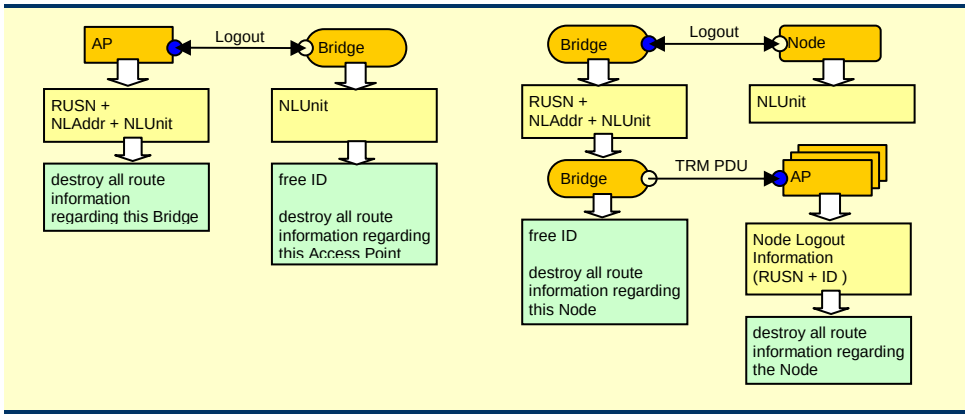



**Figure 7.3: REMPLI logout Processing**

## *7.3 Routing and Link Quality information*

Apart from the Login/Logout events, the routing tables on the Transport Layer are updated periodically with link quality information and remote queue information to enable more accurate scheduling/routing decisions at the AP (also a task of the Transport Route Manager sub-module).

The *Link Information* provided by the Network Layer is the average number of slots used to transmit PDUs to a particular station in the past. This value varies depending on the number of retries needed for a successful delivery (the NL has a basic retry mechanism) and the number of repeaters needed to reach a station. Hence, lower values reflect better quality. On the other hand, this reflects the actual “quality” of the link between the master and the slave.

Link information quality is gathered in Bridges and APs by periodically pooling the NL for link quality data. The Bridge forwards the data to the AP when needed together with the data queue depths. At the AP, the Transport Layer estimates the time

that a non-confirmed fragment takes from the AP to a particular Node given not only the link-quality of the connections but also the pending fragments on the intervening queues.

$$d^{TRM}_{APi \to Bj \to Nk} = q^{AP}_{APi \to NLU(Bj)} \cdot d^{NL}_{APi \to Bj} + (q'^{B}_{Bj \to NLU(Nk)} + 1) \cdot d'^{NL}_{Bj \to Nk} \qquad \textbf{(7.1)}$$

Where $d$ are estimated delays in time slots, $q$ are queue sizes in fragments for the Network Unit that connects to station, $q'$ and $d'$ is based on forwarded information (from Bridge to AP) and $d^{TRM}$ is the estimated delay for a particular path.

The forwarded information is updated regularly based on the network conditions, e.g. if the Bridge sent a queue size of 4 to the AP, then this value is decremented automatically by timed operation in the AP depending on the link quality information of the Bridge itself. This reduces the need to update the "real" information frequently.

After calculating the delays, the TRM at the AP simply selects the fastest route available to a particular Node. Since this calculation includes the queued fragments, it is the natural behaviour of the TRM to distribute a sequence of big packets over all available links.

Routing decisions are taken only at the AP and per request: all fragments of a request follow the same path, and if there is a response, it also follows the same path as the request. The *BridgeID* field is used in the fragments to transmit this information over the network.

This solution has the following features: good use of the available network resources; very small overhead on the network for data transmission; simple implementation; and some additional resource are needed on the stations to keep track of address conversions (*BridgeIDs*).

Link Information is also provided to higher levels giving an estimate of the delay that takes a single fragment to be sent from a particular AP to a specific Node.

$$d^{TL}_{APi \to Nk} = \frac{1}{\left| S_{APi \to Nk} \right|^2} \sum_{j \in S_{APi \to Nk}} q^{AP}_{APi \to NLU(Bj)} \cdot d^{NL}_{APi \to Bj} + (q'^{B}_{Bj \to NLU(Nk)} + 1) \cdot d'^{NL}_{Bj \to Nk} \qquad \textbf{(7.2)}$$

In Eq. (7.2) $S_{APi \to Nk}$ is the set of Bridges that connect AP ($AP_i$) and Node ($N_k$). The square operation reflects the fact that if more than one path is available then, in average, the packets (but not fragments of packets) are delivered faster since they can be sent in two parallel channels.

If multiple APs have connection to a Node, it is the task of upper layers (e.g. DeMux or Application Servers) to manage that redundancy eventually using this information (Figure 7.4). Since this estimation includes queue information, it is highly dynamic and periodically updated by the Transport Layer.

In similar fashion to the Link Quality Information service, the Transport Layer also handles Node Status information transfers from Nodes to APs via Bridges.

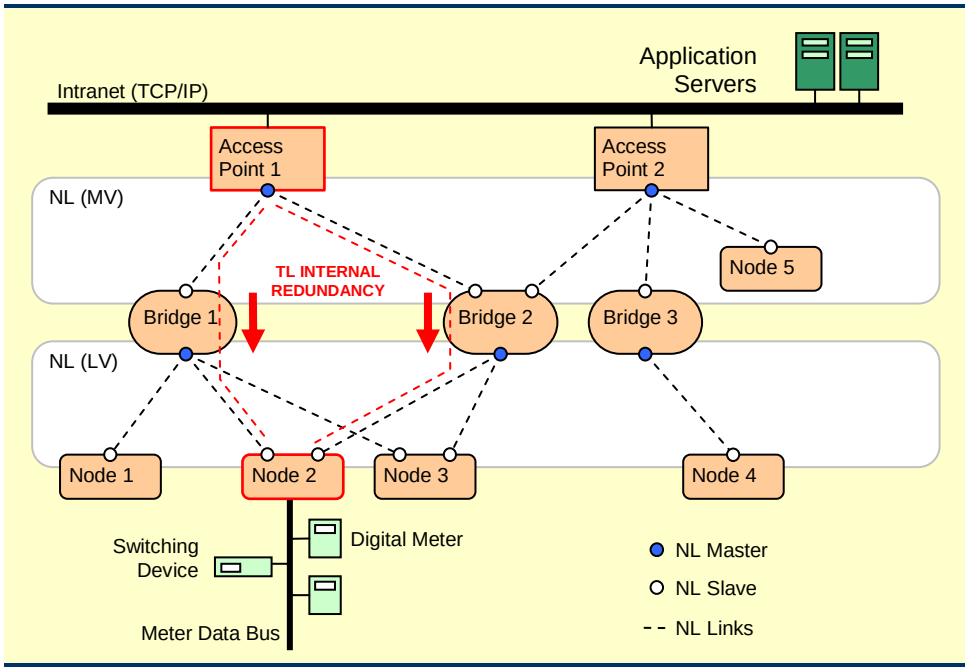



**Figure 7.4: REMPLI Network Layer example layout**

## *7.4 Sending fragments from slaves to masters*

As presented before, the base communication channel on the REMPLI system is a master/slave network. However, there are two specific higher-level services where slaves have the initiative: in the *Alarm* service slaves send data spontaneously; and in the *Request/Response* service the slave can send a single Response in a rather large time window overcoming the usual "reserved response slot" paradigm of master/slave systems.

These services are tightly integrated with two Network Layer specific features. The first feature is that the NL guarantees that a particular slave station is visited (i.e. a request-response is sent) regularly with a maximum run-time configurable delay. A second specific feature is that after visiting a slave the master tries to fetch all the data in the slave station's queue. These features were developed inside the REMPLI project itself and enable the "spontaneous" transmission of packets from the slave to the master with timing parameters controlled on the fly by the Transport Layer.

Given this scenario, the task of the Transport Layer is to configure the timing parameters correctly (a task of the TRM) and to put the adequate data on the Network Layer queues at the slave side (a task of the QM). For the Alarm service, a system-wide maximum delay is configured that guarantees a minimum QoS. For the Response service the system-wide maximum delay is used by default but the Node applications can indicate a smaller delay if needed.

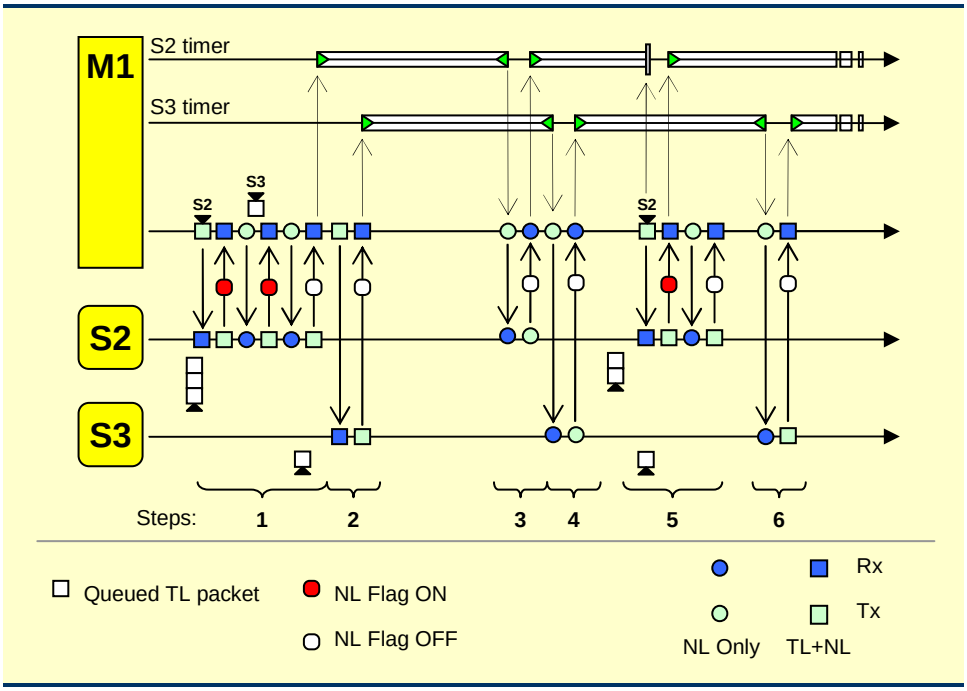



**Figure 7.5: Slave timer concept in REMPLI system**

Figure 7.5 presents the main timing mechanism of the Network Layer. In step **1** the NL queues on master station 1 (M1) have one packet for slave station 2 (S2) and one packet for S3; the NL queues on S2 have three packets and the NL queues on S3 have one packet. When the NL sends the first packet from M1 to S2, the reply has not only TL data but also an NL flag that signals that more data is available on the slave. The NL in M1 automatically issues further confirmed requests until no more data is available. The NL in M1 then moves on to the next slave (step **2**) and sends a confirmed request that has an immediate response, no more data is in the queues. In M1 separate timers are used for S1 and S2. When they expire the NL issues empty requests automatically (steps **3** and **4**). If no data is available then the timers are restarted. When TL sends a confirmed request to a station (step **5**) the NL resets the timer for that station. On step **6** the timer for S2 has expired and the NL has retrieved one packet from the NL.

To set the slave-specific delay, the Transport Layer on the slave side keeps track of open transactions and respective expected delays in response. The expected delay for a response is a service then can be used by a Node driver to give a hint on when the response will be available. To simplify implementation the TL uses half this value as the ideal periodic visits needed to serve the response, and selects the minimum value of all open requests to set the NL parameter. The NL automatically forwards this value to the master in the next empty NL response. Only then, the new parameter is effective, since it is the master's side task to handle the timers.

## *7.5 Traffic prioritization and queuing*

Traffic differentiation is provided by an 8-bit priority identifier that can be used by applications to signal different importance. The Transport Layer uses a simple "serve all higher-priority" mechanism with round-robin service for same-priority traffic. In order to provide priority on the responses and over bridges, priority information is encapsulated in some Transport Layer headers.

The Network Layer supports up to three priority classes (only two are usable at the slave side). All Transport Layer traffic is sent using the lowest Network Layer priority, except if the application chooses one of the two special priority identifiers (-1 and -2) that are mapped directly to the two higher priority queues of the Network Layer.

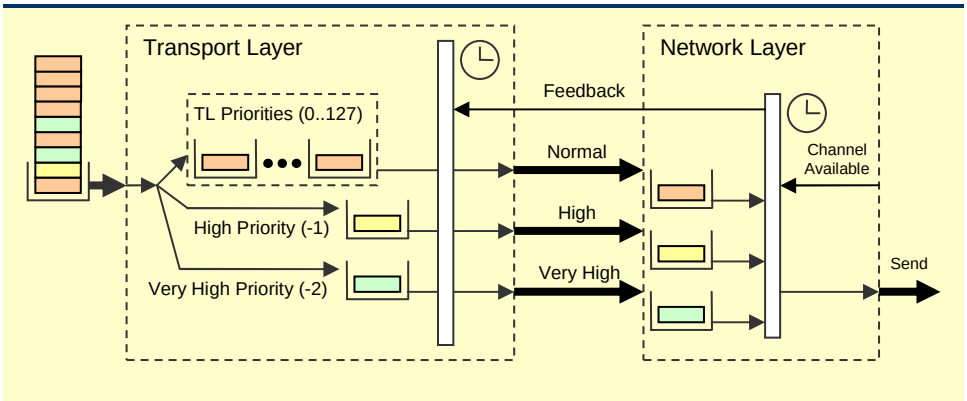



**Figure 7.6: REMPLI priority queues processing**

To make sure that no time slots are lost due to Network Layer queues starvation, the Transport Layer feeds a programmable number of fragments to the Network Layer queues even before the queues are empty. The Network Layer has a feedback channel (see Figure 7.6) to inform Transport Layer that a fragment was removed from the queues.

The disadvantage of this scheme is that when a fragment arrives in the Transport Layer it may be delivered to the network later than lower-priority fragments already queued into the Network Layer. The two "special" priorities overcome this problem since these are delivered directly to specific queues on the Network Layer and can pass in front of normal priority fragments.

On slave stations the Network Layer has only two queues and requests for High and Very High priorities are treated as a single queue. Nevertheless, this is transparent to the Transport Layer, which delivers the two special priorities to the Network Layer.

### *7.6 The Alarm Service*

The Alarm Service makes it possible for any Node to send a packet to, at least, one AP. The Node cannot choose the destination station, and it is possible that more than one AP receives the generated alarm. The Node can set the Priority and the relative Timeout of the request.

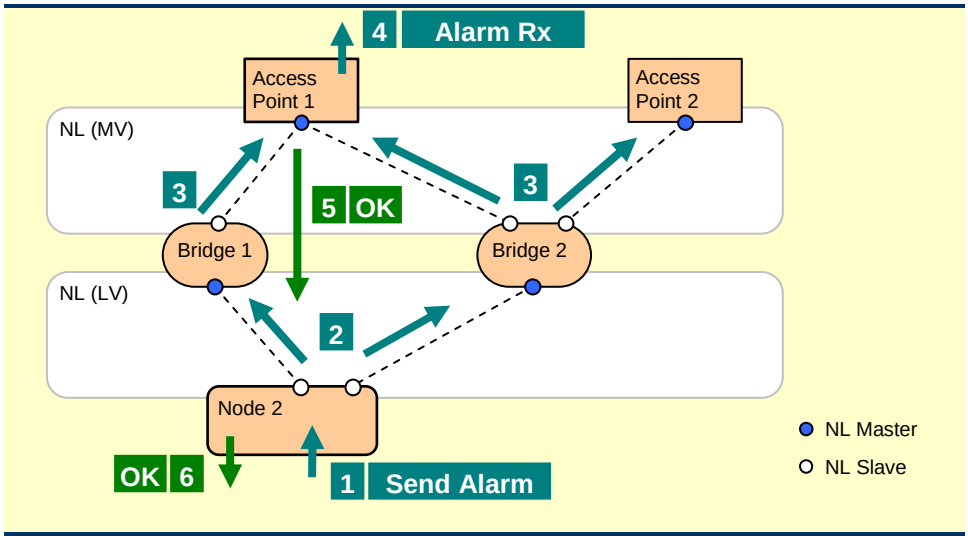



**Figure 7.7: REMPLI Alarm service**

The implemented algorithm is based in the possibility of multiple paths for delivering the fragments (Figure 7.7). It is possible for AP 1 to have the complete data receiving some fragments from either bridge. Another feature is that when each station has confirmed the delivery of a fragment to all network units it can safely discard the data block preserving memory in the stations. At the state presented in the picture, the AP 1 would start to inform the other stations on the network that the Alarm delivery was successful. Since this is a distributed mechanism, it is possible that other APs gather all the fragments of the packet while this finishing process is ongoing.

# Chapter 8
# Implementation Issues

This chapter presents further details on how to implement the mechanims proposed in Chapter 7. It starts by presenting an overview of the software architecture of the Transport Layer, providing afterwards the main implementation details of each service.

## *8.1 Transport Layer Software Architecture*

The REMPLI Transport Layer architecture was designed and tested using OMNeT++ (OMNeT++, 2007), a public-source software suite. OMNeT++ is a discrete event simulation environment with focus on the simulation of communication networks. Since it has a generic and flexible architecture, it is also used in other areas like the simulation of complex IT systems, queuing networks and hardware architectures as well.

Programming of components (modules) is done in C++. Modules can be nested and inter-connected into larger components using the NED high-level language. OMNeT++ runs on Linux and Windows and has full GUI support.

The base OMNeT++ code does not include any models. There are several simulation models and frameworks available directly at omnetpp.org website, these include Mobility Framework (focused on OSI layers 1 and 2) and INET Framework (focused on higher OSI layers). For example, INET Framework includes not only protocols like IP and UDP/TCP but also models of IEEE 802.11, PPP, IPv6 and others.

To run a simulation in OMNeT++, it is necessary to implement the components and interconnections, and specify the simulation parameters. The simulation results can be recorded using OMNeT++'s tools or the users records.

Transport Layer code was built in order to be used unchanged in both OMNeT++ and the end-system embedded Linux easing the deployment and testing of the system (Marques and Pacheco, 2007).

The Transport Layer was designed from the start to be compatible between the simulation environment under Windows or Linux and the deployment in the field on embedded Linux using the same source code (Marques and Pacheco, 2007). The main blocks of the Transport Layer are presented in Figure 8.1**Error! Reference source not found.**. It is divided into four modules, the *RCI Manager* (RCIM), the *Transport Route Manager* (TRM), the *Queue Manager* (QM) and the *NL Interface* (NLI). Interface with the higher layers of the system is done through the *Rempli Communication Interface* (RCI), with TCP/IP based streams, whilst the interface to the lower layer is done through

a Linux character driver (for efficiency reasons, the lower Network Layer is within the Linux kernel).

The higher-level connection with the DeMux is controlled by the RCI Manager. This thin module does routing of messages from the *Rempli Communication Interface* (RCI) to the QM or TRM depending on the message type. It also forwards messages from QM and TRM to the RCI. Some not implemented RCI functionality (like *Access Point Connect*) results in an immediate response from the RCIM without interference from other TL modules.

The RCI uses an *IPC Transaction Identifier* and a *Thread Identifier* (the later helps DeMux internal tasks) for each RCI Request. These identifiers are recorded by the RCIM for all messages received from the RCI. Responses from the internal TL modules only have the *IPC Transaction Identifier*, and the RCIM adds the matching *Thread Identifier* to the response. Events generated by the TL do not use either identifier. The RCIM distinguishes Events – that do not used identifiers – from Responses – that use *Transaction Identifiers* and *Thread Identifiers* – by the message type.

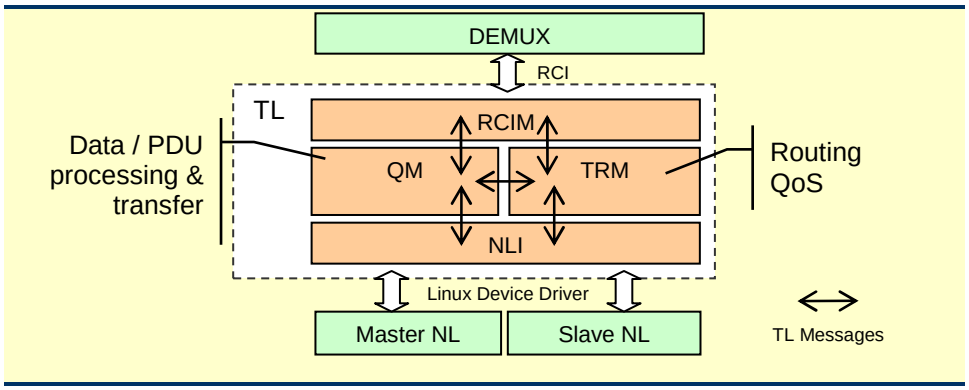



**Figure 8.1: REMPLI Transport Layer internal architecture**

Similarly to the RCIM, the NLI function is to route messages from QM and TRM to the Master NL or to the Slave NL. Again, routing is done using the message type. Some messages from the TL to the NL are of a request/response nature. For example: a *TL_MASTER_SEND_CONFIRMED* message is eventually followed by a matching response from the NL. The NL pairs these messages using the *NL Transaction ID*. However, for the internal TL modules, the *NL Transaction ID* is not used and the *Queue ID* is used instead since it maps directly to multi-fragment data. The NLI handles conversions from *TL Queue IDs* and *NL Transaction IDs* and the automatic generation/disposal of *NL Transaction IDs*.

To ease the task of the QM, some additional information like *Fragment ID* and *Queue Type* is also stored with the *Queue ID*.

The QM is the larger TL module, and it handles most of the data processing and transfer functions of the system including fragmentation, forwarding and most of the request adaptation tasks (RCI to NL and vice-versa). In addition, it provides data

communication services for the TRM. On the other hand, the QM relies on the TRM for routing information (i.e. address conversion) and scheduling of transmission tasks.

The TRM has a global view about the network status, keeping track not only of login and logout events but also on link quality information and queue sizes in a distributed fashion. In the current implementation the TRM uses this information to make route selection based on fragment delay estimation However, it supports the addition of more advanced scheduling policies. The TRM also handles some accessory functions like *Link Status* information.

The main tasks of QM and TRM are presented in the next paragraphs. Most of the code of the Transport Layer is used (Figure 8.2) in both the OMNeT++ simulation and the final-system HyNet (Hyperstone, 2007) board; the main difference is the addition of a Message processing system that handles the interface between the Transport Layer blocks and the "outside world".

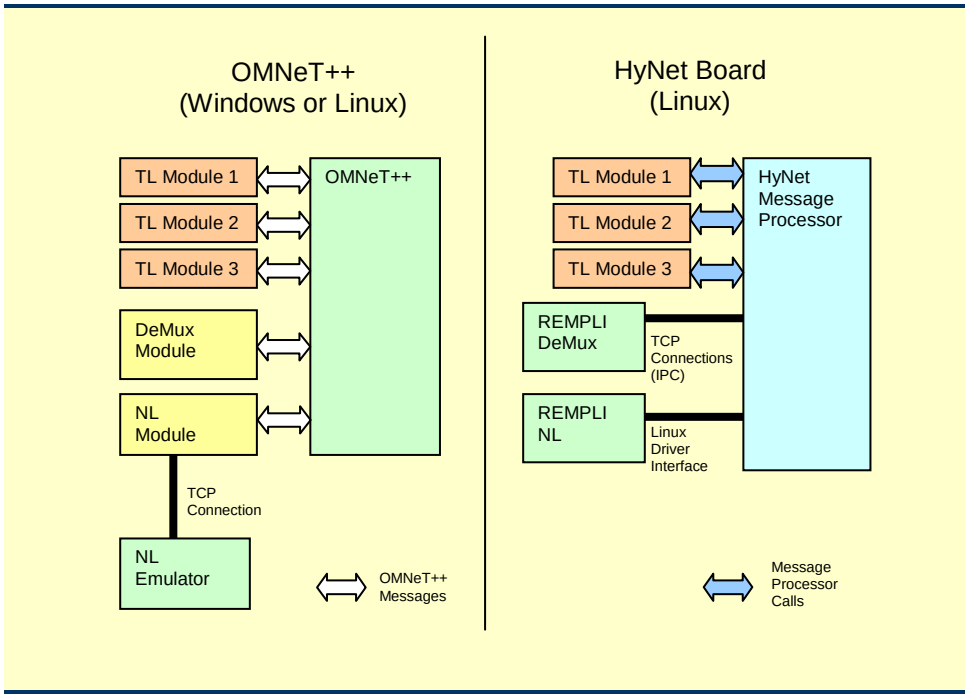



**Figure 8.2: OMNeT++ simulation and HyNet implementation**

The code sharing is possible because the Transport Layer code was written from the start having this objective in mind. At a first stage of the project, a C Object-Based implementation was considered but some analysis of the code architecture made us switch to a more efficient implementation. In particular, the REMPLI code had the following specific features: there is no direct C++ inter-module communication, i.e. all inter-module communication is done via messages not via C++ inter-object calls; there is

only one instance of each module object in each HyNet target machine; OMNeT++ message communication functionality is implemented by a C module specific for HyNet.

The REMPLI code of each module was inserted into a simulation using OMNeT++ network design tools and some simulation-specific modules. A main network layout (Figure 8.3) was used with a dual PLC network that was used for both bridged services simulation with several Nodes and for direct services simulation using the integrated Node functionality the Bridges.

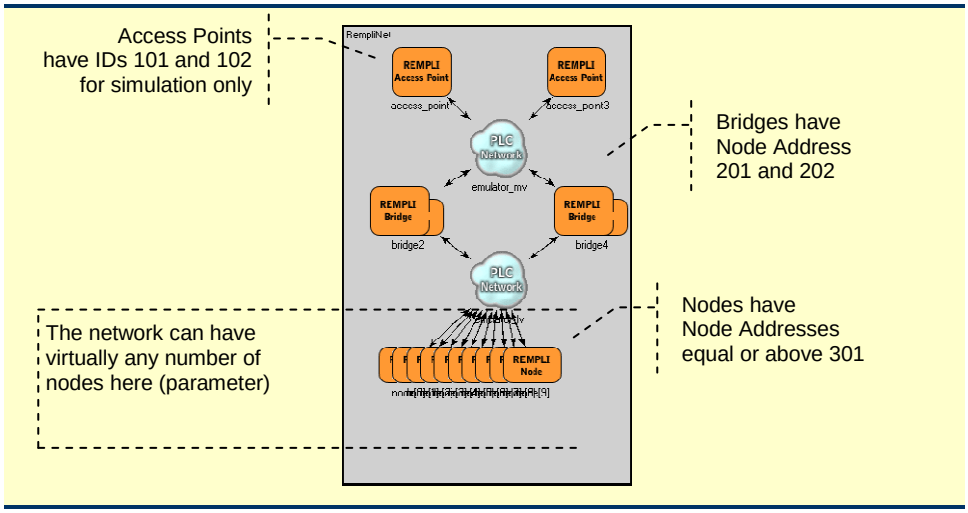



**Figure 8.3: OMNeT++ simulation network layout**

The *PLC Network* module is used to replace the Network Layer functionality, thus the messages on the network connections in the Figure are at the TL/NL interface level. The OMNeT++ module itself only handles serialization/de-serialization of these messages to the TCP connection for the separate Network Layer emulator application. The current Network Layer emulator only supports single-master networks, therefore, in order to provide a simulation environment with multi-master capability, each *PLC Network* module connects to multiple TCP server ports (and corresponding applications) using the network unit identifiers to map the traffic between the connections (Figure 8.4). In practice, for the presented network layout this means that for each simulation we have four Network Layer emulator applications running.

When a message is received by the *PLC Network* module from other OMNeT++ modules, it routes the request depending on the following rules:

- for master requests (OMNeT++ ports 0 and 1) the destination TCP socket index is the same of the request and the station identifier on the TCP message is zero;
- for slave requests (up to 8 slaves are supported on the current configuration), the destination TCP socket is selected depending on the *Net Unit* identifier and

the station identifier of the TCP message is equal to the OMNeT++ port minus one.

On the reverse direction, the following rules are applied to messages received on the TCP interface:

- master related messages are routed to the OMNeT++ port with the same identifier of the TCP socket, the *Net Unit* identifier is also set to the identifier of the TCP socket – master-related messages from the *NL Emulator* have always *Station Identifier* equal to zero;
- slave related messages are routed to the OMNeT++ port with the identifier of the *Station Identifier* in the TCP message plus one.

The *PLC Network* module also handles timed self-messages to trigger the simulation process on the *NL Emulator*. Since we are using two *NL Emulators* to simulate two masters each simulation is triggered every two slot time intervals.

There is one exception on the above message processing: multicast requests are handled by the *PLC Network* module by sending a unicast request to a pre-defined slave station. When the *PLC Network* module receives a TCP message with the predefined station identifier, it duplicates the message to all OMNeT++ slave ports. This exception was needed since multicast is not supported on the available *NL Emulator*. The drawback of this approach is that the timing behaviour of the multicast service is not reflected into the simulation.

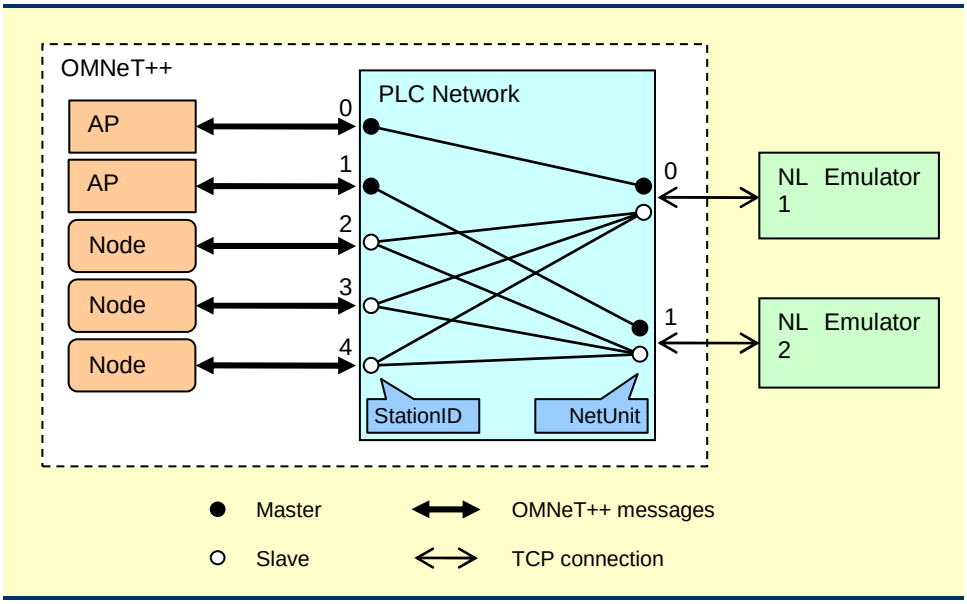



**Figure 8.4: Implementation of multi-master simulation**

Finally, Figure 8.5 presents the internal modules of the Transport Layer in the OMNeT++ simulation. On the left, there is the Bridge module, which simply uses a

Bridge Transport Layer that has connections to the Master NL port and the Slave NL. The Access Point module and the Node module are similar but a special *Dummy* module shunts the missing ports. Despite the label, the *Bridge Transport Layer* is exactly the same module that is used inside the Access Point and the Node modules.

On the right side of Figure 8.5 the same modules and connections that were conceptually presented in **Error! Reference source not found.** are now portrayed in the OMNeT++ simulation. There are also three connections to the "outside": Driver De/Mux port; Master NL port and Slave NL port.

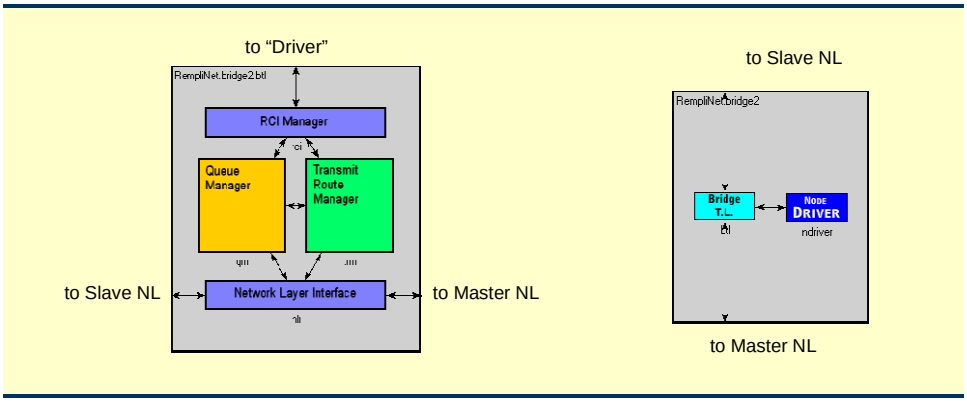



**Figure 8.5: OMNeT++ Simulation "Bridge" and "TL" modules**

## *8.2 Message Processor*

After developing the functional modules of the Transport Layer on OMNeT++, as presented in the previous section, an important part of the layer was developed to enable inter-module and inter-layer communication outside of OMNeT++.

Figure 8.6 depicts the main blocks and characteristics of the Message Processor.

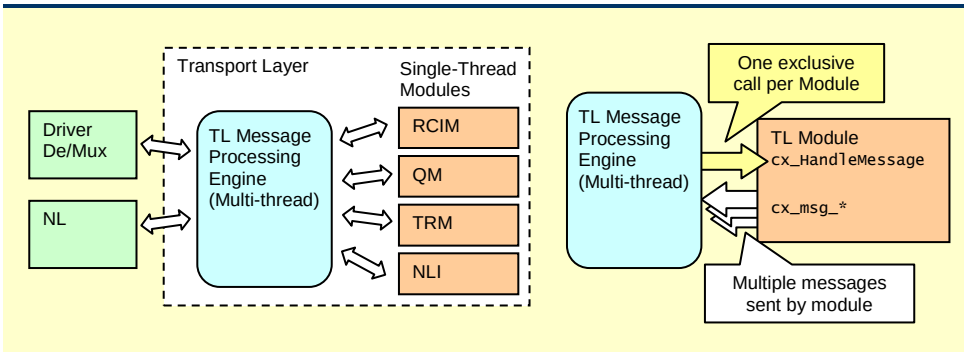



**Figure 8.6: REMPLI message processor concepts**

This “Message Processor” is in fact the main process of the Transport Layer when running on the target devices. Although the modules were developed to be compatible with both Windows and Linux systems (and were tested in the two on OMNet++), the Message Processor was developed specifically for the HyNet board.

The Message Processor is a multi-thread module, which manages TCP/IP connections to the Driver De/Mux and Linux Driver interface to the Master NL and Slave NL, adapting internal TL messages to these channels. The module also stores internal messages that will be later delivered to internal modules or one of the outside interfaces, including both event- and time-triggered messages. Each instance of the module is guaranteed to run in single-thread fashion, but different modules may be running at the same time.

## *8.3 Inter-module messages*

In terms of message exchange, one of the more active internal TL connections is the QM/TRM link. The main tasks of the TRM are (i) to inform the QM of the destination path for a request/response, and (ii) to trigger the transmission of particular fragments. This also means that any change on QM queues must be forwarded to the TRM so the later has an up-to-date view of the pending requests.

Figure 8.7 presents the main messages exchanged by these two modules. There are messages for the QM to signal new queues (with *Node Addresses*) and respective route responses from the TRM. Route information includes *NLAddr*, *Bridge IDs* and *NL Units* depending on the situation. Each queue type has a different message type since each type has its own set of parameters. Requests and responses are associated by QM’s *Queue ID*, being this association unique for the “creation” messages. Other message types do not require this unique mapping, and the QM may issues several messages with the same *Queue ID* to the TRM before receiving the matching responses.

When the TRM wants to delete a QM queue, or when a QM queue does not have a viable path, it sends one of the *QueueRoute* messages with the *NLUnit* set to zero. After a first valid route message the TRM can send a delete route message afterwards if the connection to a station is lost.

There is a simple protocol to TRM signal to QM when new fragments should be delivered to the NL (*ServeSlot*) and respective results (*ServedSlot*, *UnusedSlot*). The QM can also signal that a queue was updated or destroyed (*QueueUpdate*, *DestroyQueue*). The update messages are always with relative values, i.e. increment or decrement any of the queues characteristics.

Although some messages are specific for certain station types, (e.g. *NewAPQueue* can only be issued in an Access Point) the code is the same in all stations to simplify development. Different configurations are supported through specific annotations in the code that can be later pre-processed (for instance to create a “Node-only” version of the Transport Layer in the future).

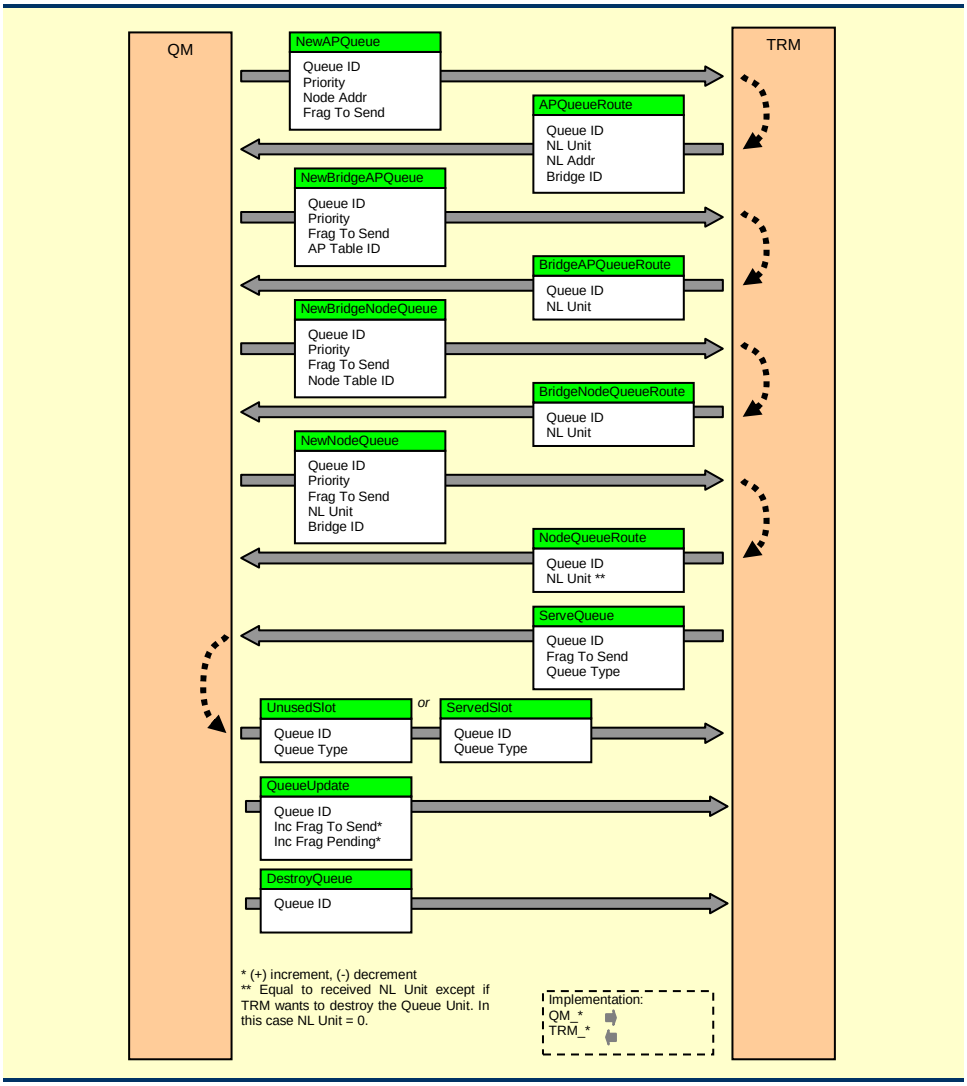



**Figure 8.7: REMPLI TRM/QM messages**

A specific set of messages is also used for communication to and from the Network Layer Interface (Figure 8.8). The first distinguishing point is that above this interface (i.e. to QM and TRM) the requests and responses are matched using the QM's *Queue ID*, while below the interface (i.e. to the Network Layer) the *NL Indication IDs* are used (in this context, "requests" are defined as the messages sent from the TL to the NL direction). In fact, it is the main task of the interface to handle the creation and translation of theses IDs when needed. Nevertheless, a small sub-set of the messages, like *SlaveStatusUpdate*, do not use IDs.

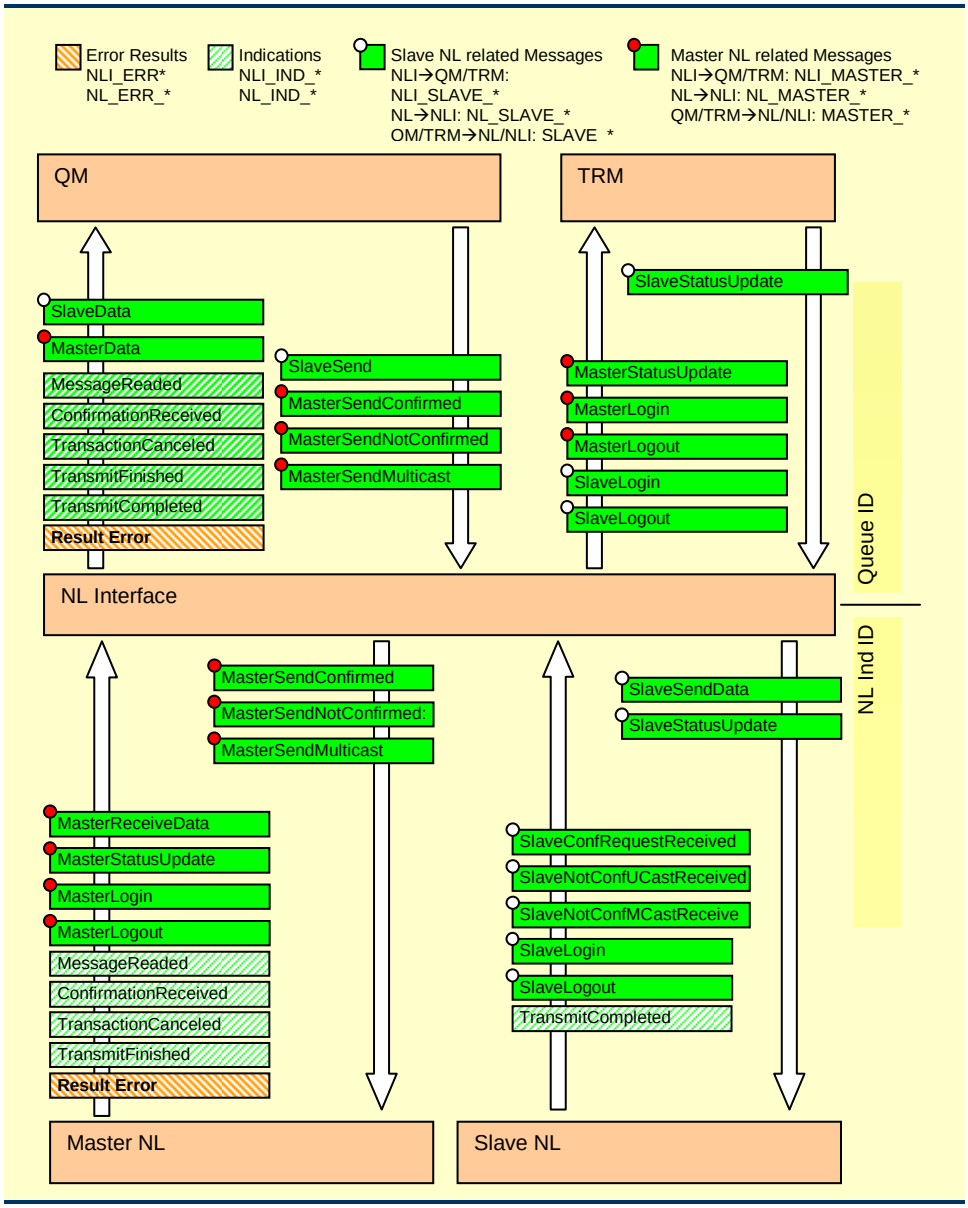



**Figure 8.8: REMPLI NLI messages**

The NLI also routes messages between master and slave NL and the QM and TRM modules according to the message type as presented in the figure. For example, all *MasterReceiveData* messages are delivered to the QM after being morphed to *MasterData* messages. This is the case even if these messages include TRM data: it is

the QM task to extract the TL header and from this header distinguish the final destination of the data.

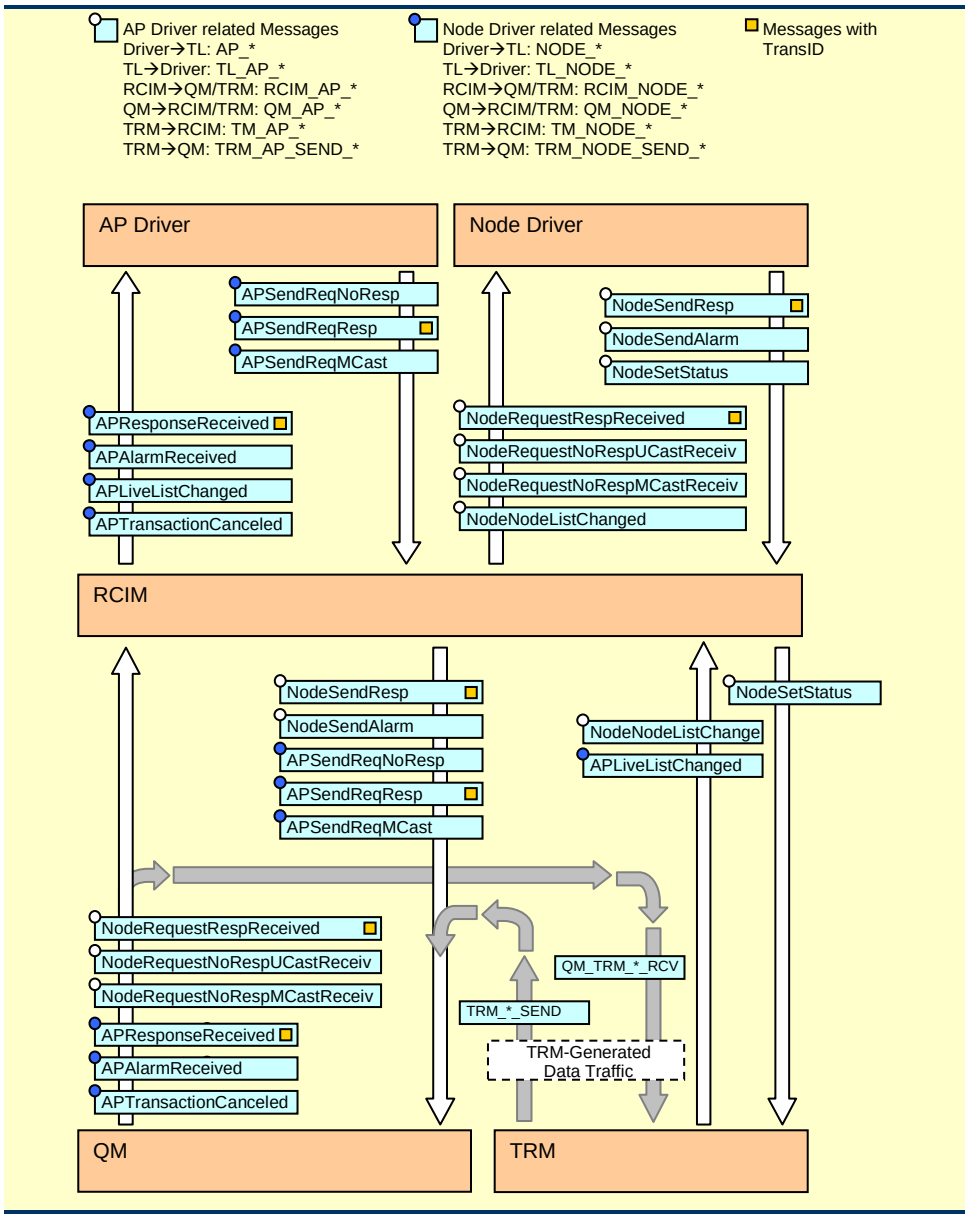



**Figure 8.9: REMPLI RCI messages**

As expected, the messages in the NLI are a faithful representation of the available Network Layer services including master/slave separation, confirmed and non-confirmed requests, login/logout events and status update information. In these services, stations are identified by (run-time) *NLAddr* and *Unit IDs* and the data payload sizes are very limited.

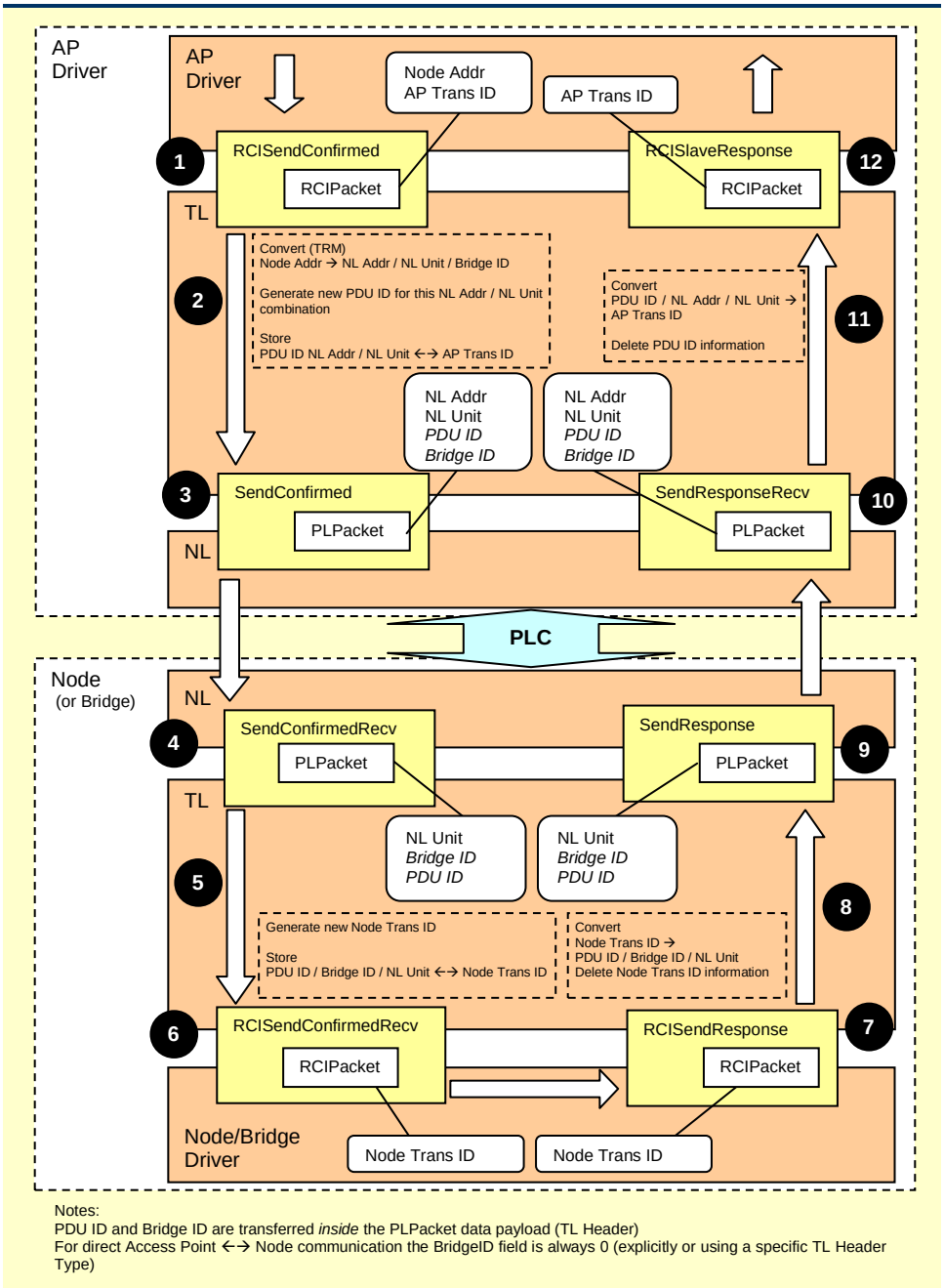
Figure 8.9 provides the flow of messages at the RCI interface, between the QM/TRM and the Access Point and Node Drivers. The services are the ones provided by the RCI: Request with Response, Request With No Response, Multicast Data, Alarm Service, Status Update and Live List information. All the data-related services support very large data payloads and destinations are identified by Node Addresses.

Importantly, the TRM can also use the data-related services of the QM, as a Driver would. The only limitation is that TRM services are never fragmented and therefore the data payload is always restricted in size.

## *8.4 Processing Requests*

In order to understand the behaviour of the Transport Layer, it is important to understand how requests are handled. Figure 8.10 depicts the processing of a confirmed request, where:

1. The TL receives an *RCISendConfirmed* request, with the *RCIPacket* data, and related *Node Address* and Access Point *TL Transaction ID*.
2. The TL converts the *Node Address* to {*NLAddr*; *NLUnit*; *Bridge ID*}. Since the depicted example is for a direct connection, *Bridge ID* is always 0. Then the TL generates a new *PDU ID* to group the fragments of the request on the PLC network. PDU IDs are unique for each *NLAddr* and *NLUnit*. Finally, the TL saves the AP *TL Transaction ID* for this request.
3. The TL sends fragments using the NL for the *NLAddr*, *NLUnit* destination adding its own header with *PDU ID* and *Bridge ID*.
4. The Node NL receives the fragment
5. The TL of the Node rebuilds the fragments of the request (the current implementation supports selective acknowledge mechanism to complete this task), generates a new Node *Trans ID* and stores corresponding *PDU ID* / *Bridge ID* / *NLUnit*.
6. The TL delivers the complete request with the attached Node *TL Transaction ID* to the DeMux, which eventually delivers it to the Node Driver.
7. The Node Driver processes the request and prepares the adequate response. The answer is delivered via the DeMux to the TL. The Node *TL Transaction ID* is used to match request and response.
8. The TL now fragments the response and sends it back to the AP. The fragments of a response have the same *PDU ID* and *Bridge ID* of the original request, so the first task is to retrieve this information, saved in step 5.
9. Fragments are sent to the AP using the same *NLUnit* of the request. The *Bridge ID* and *PDU ID* are included in the TL header information.




Notes:
PDU ID and Bridge ID are transferred *inside* the PLPacket data payload (TL Header)
For direct Access Point ←→ Node communication the BridgeID field is always 0 (explicitly or using a specific TL Header Type)

**Figure 8.10: REMPLI PDU processing (direct connection)**

10. The AP receives the fragments

11. After receiving all fragments of the response the TL matches the response to the original AP Driver request
12. The response is delivered to the AP Driver. The AP *TL Transaction ID* is used to match the request and the response.

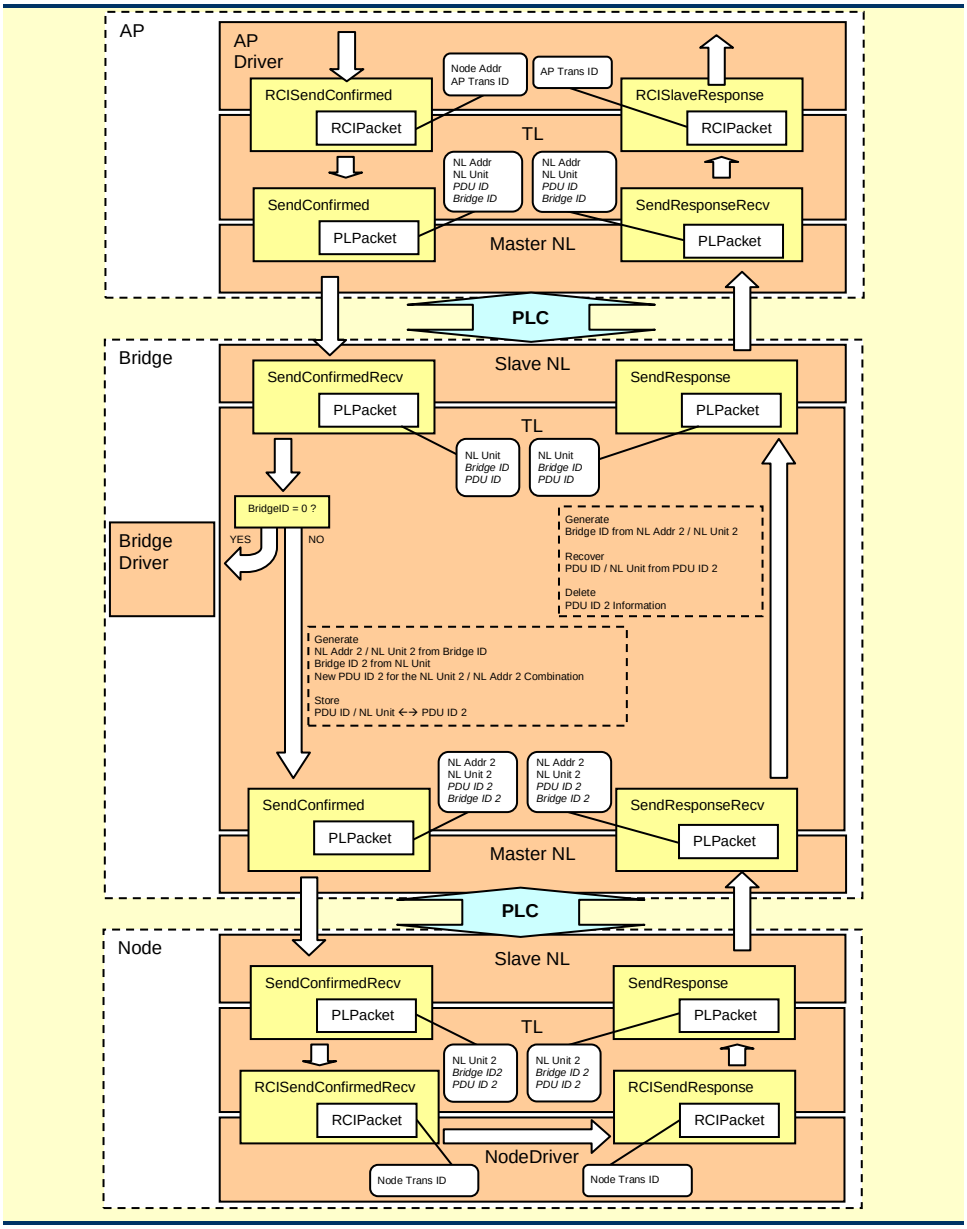



**Figure 8.11: REMPLI PDU processing (via Bridge)**

For a bridged request most of the processing is similar (Figure 8.11), with some exceptions:

- When a Node connects to a Bridge, the Bridge assigns this Node a Bridge ID. When the Access Point wants to communicate with the Node it uses this Bridge ID to address it at one particular Bridge. On the reverse direction (Node to Access Point via Bridge) a similar mechanism is used. The advantage of this method is that TL header space is reduced significantly without compromising scalability of the system.
- If the Access Point wants to communicate directly to the Bridge (i.e. using the Node functionality of the Bridge itself) it uses the reserved Bridge ID of zero.
- Each network segment has its own PDU IDs, Bridge IDs, etc.
- All the fragments of a request and matching response (when applicable) follow the same route.

## *8.5 Fragmentation and Headers*

Like in Profibus networks, the REMPLI NL is also limited to small PDU size in order to improve system responsiveness. The TL is built over the NL layer to provide very large data payload services to applications.

The need to combine fast response services with large data lengths (up to 16 MiB on the current configuration) on the same system lead to solution with three different headers (Figure 8.12):

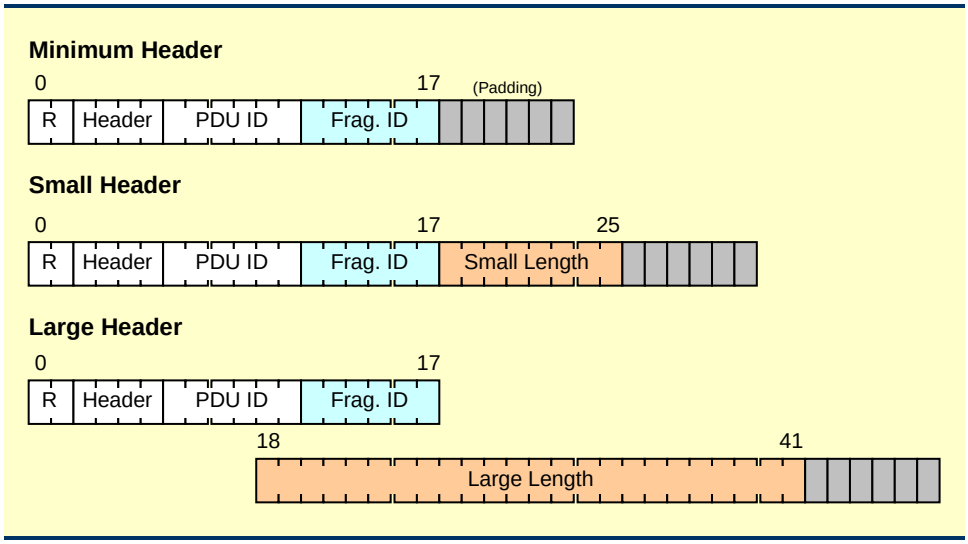



**Figure 8.12: REMPLI Fragmentation Headers for Unicast Data Services**

- a *Minimum Header* with *Request Type* (2 bits), *Header Type* (4 bits), *PDU ID* (6 bits) and *Offset* (6 bits), with a total of 18 bits or 3 bytes;
- a *Small PDU Header* with the same information fields of the *Minimum Header* plus a 8-bit *Data Length* field, with a total of 26 bits or 4 bytes;
- a *Large PDU Header* with the same information fields of the *Minimum Header* plus a 24-bit *Data Length* field, with a total of 42 bits or 6 bytes.

The first field identifies the type of PDU, *Request/Response*, *Unicast*, *Response* and *TRM Data*. The second field *Header Type* is used to distinguish between the 15 available header configurations at the current version. The *PDU ID* field is unique per source (i.e. the master or slave of a particular station) and identifies a group of fragments as belonging to a packet. In the current header architecture it would possible to share *PDU IDs* between some groups of header types but to reduce complexity this is not implemented at the moment. The 6-bit *Frag. ID* identifies the order (starting at 0) of each fragment in the set of fragments of a given packet.

When sending fragments, the first fragments (a compilation-time constant, *TLH_NumberBigHeaders*, with a typical value of 3) are with length fields, and the following fragments use only minimum headers. Since the system can only store data fragments after a successful reception of a header with a length field, the number of fragments with packet length information is configurable depending on the expected error rates of the network and also on the probability of out-of-order delivery.

In addition, the bit lengths for “small” and “large” PDUs can be easily pre-configured and can have up to 32 bits (4 GiB). Decision on the length sizes of headers is dependent on the specific system and the predicted traffic patterns.

The fragmentation functions handle most of the data reception and transmission tasks for typical services. For transmissions, the fragmentation starts after the QM is aware of the maximum length available for a particular request. It creates (*NodeCreateFragments*, *AlarmCreateFragments*, etc) a linked list with fragment information and ready-to-use PDUs data blocks, complete with headers. If applicable (see below), the fragmentation process handles the different headers used in one request, that is, only the first fragments have “complete headers” thus the remaining fragments have more space for data payload. Fragment information varies with the type of queue but typically includes TTL counter, PDU final size, absolute fragment number (starting at 0) and delivery status (e.g, *DataToBeSent*, *Sent*, *Confirmed*).

After creating the fragments, the original data block is discarded. This means that there is a temporary duplication of the data payload on the station, but the advantage is that as soon as each fragment is confirmed the data can be discarded and memory released gradually. The other reason to choose ready-to-use PDUs is to guarantee that when a queue is scheduled, fragments are delivered as fast as possible to the Network Layer. The disadvantage is a higher worst-case memory footprint.

This is also dependent on the possibility of simultaneous use of different sized network layer units. If this was not foreseen, fragmentation could be done immediately when the data is received at the DeMux interface moving part of this functionality from the QM to the RCIM. The QM would later add the headers to the fragments.

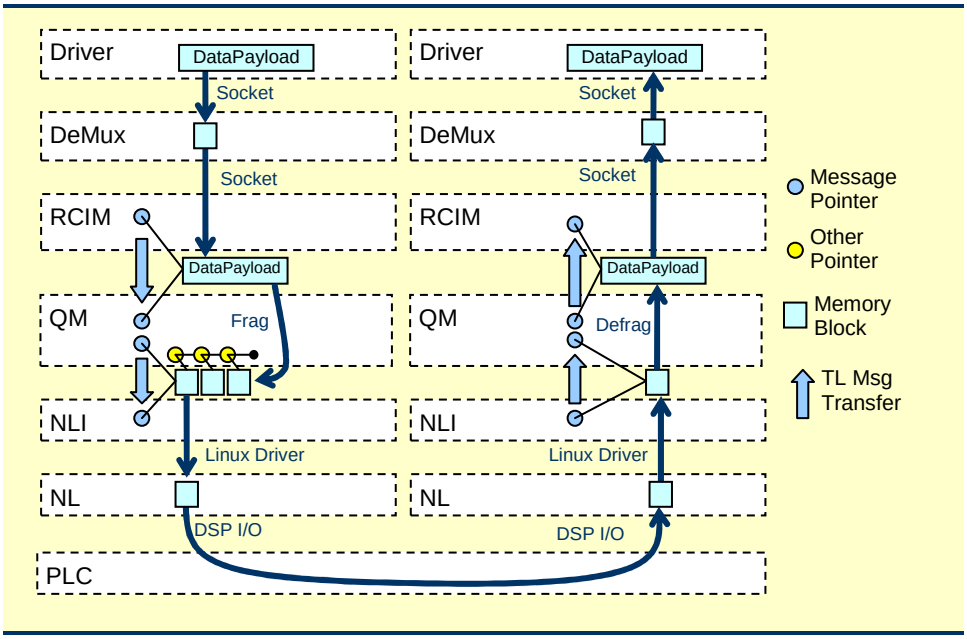



**Figure 8.13: Memory Blocks in a Unicast service (Linux HyNet System)**

When receiving, the first fragment triggers the allocation of the full packet. Afterwards, it is simply a task of placing each fragment in the correct position. The only caveat is positioning the fragments: the smaller payload of the first fragments has to be taken into account when calculating the byte offset of each fragment in the reception buffer. In order to control delivery of fragments, at every 16 fragments (or when the packet is complete) a confirmation packet is sent back to the source of the PDU with fragment status information. This is done creating a new *TxQueue* that is scheduled by the TRM eventually.

Since the system may experience out-of-order delivery and bandwidth is scarse, a bit-oriented fragmentation confirmation mechanism is used enhancing the standard sliding window mechanism. The ideia is that instead of using one PDU to confirm each fragment (or a group of consecutive confirmed fragments) we can set the status of a group of fragments at a time. This data is sent in a *QMStatus* structure that has a *PDU ID*, an absolute 32-bit offset of the first missing fragment (like a standard sliding window mechanism), and 24 bits with the status of 24 fragments *after* the first one. There are two reasons for this enhancement: (i) on transmission fragment blocks can freed earlier and (ii) on bridge forwarding it is possible to implement a forwarding service oriented to the fragment (and not to the sliding window) optimizing memory usage.

A straightforward algorithm to build this status information is presented in Figure 8.14. On the other station, the process of detecting if a fragment is confirmed or not is adapted accordingly. Please note that this process eventually generates a PDU from the station that is receiving the Data PDU to the station that is transmitting the Data PDU.

On the TL, the offset is signed, and if lower than zero then all fragments are confirmed[6]. On the side that is sending the Data PDU several tasks have to be fulfilled. If all fragments are confirmed the transmission is complete. Depending on the station and queue type, this can originate an event for the driver (e.g. *NodeOkAlarm*) and/or the destruction of the queue itself. For partial lists, all the fragments with offsets lower than the first fragment are released (since they are confirmed implicitly), the bit-by-bit confirmed fragments are also released. For the "not-confirmed" bits a simple retry mechanism was implemented: after the reception of two "not-confirmed" bits for the same fragment, the fragment is placed back in the "to send" state (and is scheduled in due course). For unicast requests, the first 64 (*QueueSegmentSize*) fragments are numbered in sequence starting at 0 up to 63, the next ones from 0 to 63 and so on. A sliding window mechanism is used to keep track of which section of the fragment list are being dealt with. The first fragment value of the *QMStatus* structure sets the first fragment of the "window" and only up to 32 fragments ahead of it are sent. Like "traditional" sliding window implementations, the window can only move when a new *QMStatus* structure is received.

```
// STATION RECEIVING THE DATA PDU

// Assume that:

// a. when a fragment is received a structure is placed in an ordered linked
// list with the absolute fragment offset.
// b. block->frags{0} reads or sets the first bit, {1} the second and so on
// c. block->frags{0..4}=true sets the fist 5 bits, etc


// function GenerateQMStatusBlock

// Return a pointer to Status Block for
// one Reception Queue, or NULL if all fragments
// are received

(StatusBlock *) GenerateQMStatusBlock(rxQueue) {
   // temporary var to store the block
   StatusBlock * block;

   // 1. find first missing fragment or unconfirmed fragment

   curFrag = 0;
   aux = rxQueue->firstFragment;
   while (aux!=null) {
      if (aux->offset != curFrag)
         break;
      aux = aux->next;
      curFrag++;
   }

   if ((aux == null) && (curFrag == rxQueue->numFrags)
      return null; // we have all fragments

   // 2. build the status block and set the first fragment value

   block = new StatusBlock();
   block->firstFragment = curFrag;
   block->frags{0..23} = true;

   // 3. build the bit list of other fragments
   curFrag=0;
   aux = aux->next;
   while (aux!=null || curFrag > 24) {
      deltaFrag = aux->offset - block->firstFragment + 1
      if (deltaFrag != curFrag) {
         if (deltaFrag >= 24) {
            block->frags{curFrag..23} = false;
            break;
         } else {
            block->frags{curFrag..deltaFrag-1} = false;
            curFrag += deltaFrag-1;
         }
      }
      curFrag++;
      aux = aux->next;
   }

   // 4. we are done!
   return block;
}
```

**Figure 8.14: REMPLI Status Information Algorithm**

[6] In terms of programming this is almost the same as checking if the offset is equal (or greater) to the number of fragments in the PDU, but it eases debugging tasks. Another difference is that the number of bits used system-wide is configurable at compilation time.

For non-unicast services, the main difference is that no "minimum headers" are used (Figure 8.15) and the headers have the absolute order number of the fragment. The benefit is that all fragments have the necessary information to start the data reception.

In order to conserve bandwidth, the "internal" TL traffic "piggybacks" several data blocks inside a Network Layer PDU, aligned at byte boundaries.

For the internal traffic between TRM units in different stations, the system fills the available data space with a series of TRM data blocks, each with its own function. It is possible for example to send several link quality update notifications mixed with status service updates in one single NL packet. QM uses a similar procedure for its own packets, e.g. the QMStatus described in the Confirmation process above.

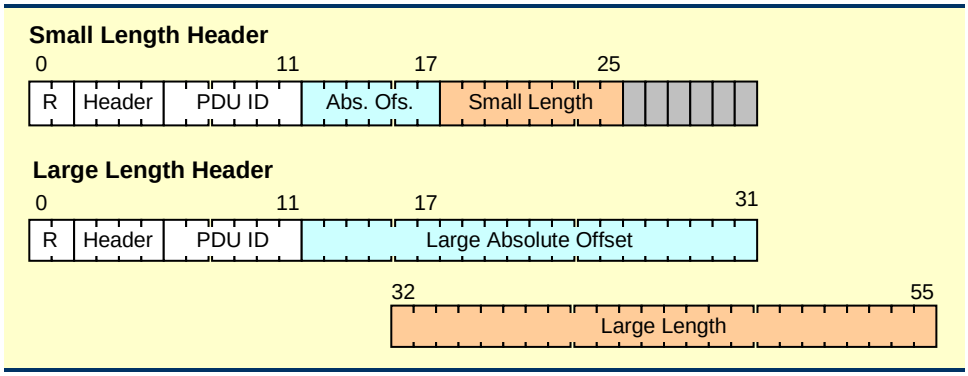



**Figure 8.15: REMPLI Fragmentation Headers for Non-Unicast Data Services**

## *8.6 Direct Unicast Service*

The TL unicast service aims to provide reliable transmission of packets from the Access Point to a Node over the two-level network. The base network does not guarantee the order of delivery of packets and has a very limited payload per PDU so this service is a major improvement over the existing system. However, it was important for the implementation that small packets could be delivered fast (but not necessarily *confirmed* as fast) and the service has this in consideration.

The algorithm of the unicast service is summarized in the next paragraphs for the non-bridged version; the bridged version is very similar to the alarm processing that is presented in the next sub-chapter.

**1. Access Point Driver issues a Unicast Request directed to a Node**

When a request is issued by the driver the packet data is delivered first to the DeMux and then to the RCIM and finally to the Queue Manager (*CreateApTxQueueUnit*).

The Queue Manager saves the PDU data pointer for later processing. It also assigns a *Queue ID* (unique for this station) to the request and saves the queue information in the list of Transmission Queues (*InsertTxQueueUnit*). It then waits for the Transport Route Manager for an available route to the given *Node Address* destination.

The Transport Route Manager then calls the Fragment Scheduler and sends either a route to the QM or an message signalling than no route is available. In the latter, the QM destroys the queue information and issues an error message that is later delivered to the driver.

If a route is available, (*UpdateApTxQueueUnitInfo*) the QM assigns a new *PDU ID* (unique for each Network Layer Unit). The route information also includes the maximum data length for the path and it can now create (*APCreateFragments*) the fragment information. The PDUs created have the request type of *ReqNoResp*. If for some reason fragments cannot be created, the QM informs TRM to remove the Queue Unit information.

It then sends an update message back to the TRM with information on how many fragments are to be sent. The QM uses a sliding window in order to support very large packets. Only fragments up to the sliding window size are marked as "available to be sent" to the TRM. The queue is identified in these messages by the *Queue ID*.

This message starts the cycle of fragments transmission: when the network is available the TRM informs the QM to send (*ServeTxQueue*) one or more fragments to the Network Layer. When the Network Layer processes the fragment (e.g. when the fragment is about to be sent to the physical network) the QM informs the TRM that a slot is available and the process is repeated until there are no more fragments to send.

When a fragment is processed by the Network Layer it does not guarantee that it is correctly delivered. The fragment is kept as "pending" on the fragment list. When a confirmation is received from the NL then the fragment is considered to be "delivered". If an error is received from the NL then the fragment is set as "available to be sent" again.

The Fragmentation process controls the delivery of all fragments (*ReceiveQmMasterData*) and delivers the OK message to the DeMux when the process is complete.

**2. Node Transport Layer Receives Data**

When the Node receives a fragment with *ReqNoResp* request type it starts by building the new queue information (*CreateRxQueueUnit*). This process first checks if the queue already exists, e.g. if there is any *RxQueueUnit* with the same *PDU ID* and *NLUnit*.

If it does not exist then it creates it. It saves the base queue unit parameters (like priority, length, *PDU ID* and *NLUnit*), and the event type that is used to transfer the data do the DeMux (in this case *NodeUcastReceived*). Since this is a reception queue it does not have a *Queue ID*.

After creating the queue, or if the queue already existed, the fragment reception process (*ReceiveRxFragment*) is run. If all the fragments are received the data pointer is transferred to the DeMux using the previously stored event type. Finally, the last step is deleting the queue unit information (*DeleteRxQueueUnit*). This function simply removes the structure from the linked list and frees the used memory.

**3. Timeout and error processing**

For simplicity timeout, processing was not included in this description. The main concept is that the Queue Manager calculates the deadline for the packet and forwards this value to both the Bridge and the Node. This solution was chosen since the

synchronization of the clocks in all stations is guaranteed by other REMPLI components (Gaderer *et al.*, 2006).

The processing of typical errors, including retry operations are handled by the *ReceivedRxFragment* function.

Except for timeout checks in the Queue Manager and the *Live List Cost* updates in the Transport Route Manager, all the processing inside the Transport Layer is driven from external events: TL progression is only activated when there is an external input (from the DeMux or from the NL).

### *8.7 Bridged Alarm Service*

The Alarm Service makes it possible for any Node to send a confirmed packet to, at least, one Access Point. The Node cannot choose the destination station, and it is possible that more than one Access Point receives the generated alarm. Therefore it can be classified as an "anycast" service originated in the Nodes.

The Node can set the *Priority* and the relative *Timeout* of the request.

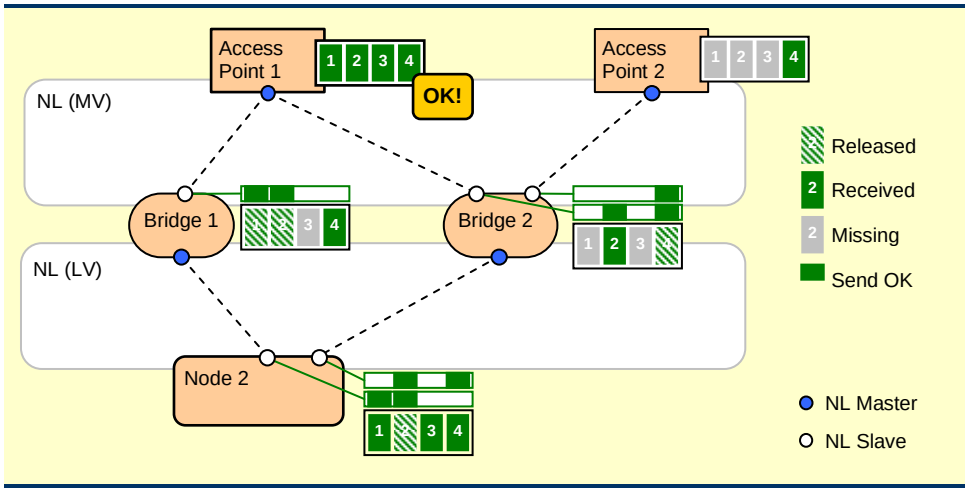



**Figure 8.16: The Alarm Service Fragment Status – Simplified Concept**

The concept is based in the possibility of multiple paths for delivering the fragments (Figure 8.16). It is possible for Access Point 1 to have the complete data receiving some fragments from either bridge. Another feature is that when each station has confirmed the delivery of a fragment to all network units it can safely discard the data block preserving memory in the stations. At the state presented in the picture, the Access Point 1 would start to inform the other stations on the network that the Alarm delivery was successful. Since this is a distributed mechanism, it is possible that other Access Points gather all the fragments of the packet while this finishing process is ongoing.

In the REMPLI Transport Layer the process is more complex due to the support of multiple sized Network Layers: the received fragments can have a different size of the transmit fragments. This is possible since the network layers can be configured differently depending on the physical medium characteristics.

Consequently, a dual-queue architecture was designed to enable support of different data lengths in master and slave side of the Bridge: a collection of *QueueSegments* gathers blocks of received information from the Node; and fragments ready-to-send to the Access Point are attached when possible to a transmit *QueueUnit*.

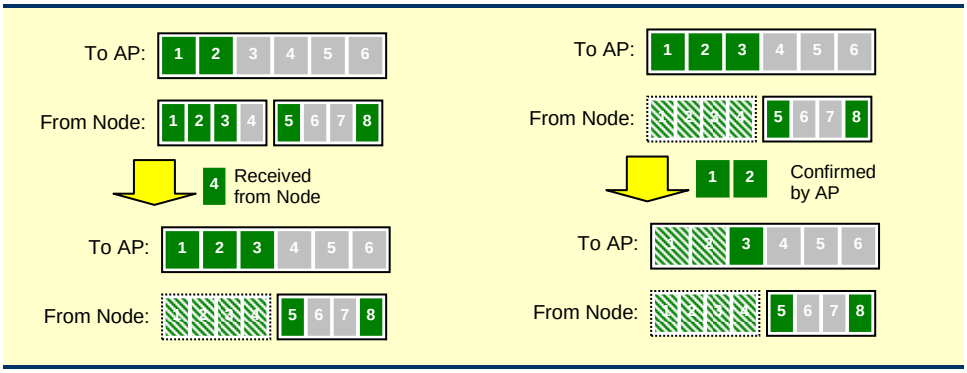



**Figure 8.17: The Alarm Service Bridge Dual-Queue Architecture**

Received data is released in a segment block when all the data of that block is completely received and has been transferred to the transmission queue. Transmission data is released on a fragment-by-fragment basis like any other transmission queue.

This process is summarized in Figure 8.17. Nevertheless, a clarification must be made in terms of memory usage. At the reception, each *SegmentQueues* is allocated at once to store user data of several fragments and released when possible. In transmission memory is only allocated when needed by a fragment, and released when the fragment is confirmed. In terms of the figure, this means that the gray boxes at the “To AP:” side are not allocated yet, but the gray boxes on the “From Node” side are already allocated but not filled up. However, *SegmentQueues* are only allocated when needed: in the example above the second segment was only allocated when the first of fragments 5 or 8 arrived.

The choice for *SegmentQueues* was done to mimic as far as possible the normal non-bridged reception without wasting too many resources.

**1. Node Driver sends Alarm request**

At the Node side the process begins with a *NodeSendAlarm* request from the Node Driver. As usual this request is passed to the Driver and to the RCI and, finally, to the Queue Manager.

The Queue Manager starts by (*CreateAlarmQueueUnit*) saving in a list the basic information about the request, priority, size, data payload, etc. It also appends to the data payload an absolute timeout value. It then passes scheduling information to the TRM.

The TRM stores the information and issues a *AlarmAddNetUnit* message for the QM regarding each available network unit on the station. This enables the "reverse" broadcast nature of the Alarm service.

Back on the QM side, the first message triggers the fragmentation process (*AlarmCreateFragments*). In the *Alarm Queues* each fragment has status information regarding each network unit so it can track the delivery independently. Only after confirmation in all units is the fragment is released. The QM issues an update message to the TRM with the number of fragments pending for the given net unit.

At this point, the TRM Fragment Scheduler can select one of the alarm units to be served depending on the current pool of pending requests. In this, Alarm priorities are treated like any other queue priority, but advanced Schedulers could differentiate queue types.

When the Fragment Scheduler selects a particular Alarm queue and unit the QM scans (*ServeAlarmQueue*) the fragment list for fragments to be sent. If found they are marked as "not confirmed" and a copy of the fragment PDU delivered to the slave NL (via NLI). Multiple fragments can be scheduled with a single TRM request.

**2. Bridge receives fragments from Node**

The first fragment originates a new Queue Unit (*CreateBridgeAlarmQueueUnit*), other fragments are "added" to the queue unit data (*ReceiveAlarmBridgeFragment*).

When creating the *QueueUnit* the QM sends a *BridgeGetBridgeID* message to the TRM, with network unit and address of the Node. The TRM eventually sends back a *BridgeGetBridgeIDResponse* message with the matching *BridgeID* that is used to identify the original node in the fragments to the Access Points, transmission fragments are only created after receiving this information since it is needed for the fragment headers. The QM also issues a *NewAlarmQueue* for TRM scheduling purposes. The last step in creating the queue is calling *CreateAlarmBridgeQueue*. This allocates memory for one *AlarmBridgeQueue* structure and the first *AlarmBridgeQueueSegment* structure.

The received fragment data is saved in the data block of one of the *AlarmBridgeQueueSegment* structures (if needed a new structure is allocated). As referred above, when possible, e.g. when contiguous memory is available, received fragments are rebuilt into fragments (*AlarmBridgeReconstructFragments*) ready to be sent in the slave side of the Bridge stored in the *AlarmQueueUnit* structure like other transmit queues. When all the data in a particular *AlarmBridgeQueueSegment* is transferred to the *AlarmQueueUnit* the *AlarmBridgeQueueSegment* is released. The new fragments originate *UpdateQueue* messages to the TRM.

If an *AlarmOK* PDU is received a confirmation is sent back. The need for this "re-confirmation" is that the Node is not able to confirm via the Network Layer if the PDU was actually delivered or not. This PDU is sent back even if there is no matching queue unit in the Bridge. All the remaining *AlarmBridgeQueueSegment* data, and the *AlarmBridgeQueueSegment* structure, is released at this point.

**3. Node receives fragments confirmation**

Eventually the slave station receives special QM PDUs with Alarm information. Each PDU includes information about the *AlarmID*, if the Alarm was delivered to one destination (*AlarmOK*), and fragment confirmation data.

If the *AlarmOK* bit is set, the QM releases (*ReceiveQmAlarmHeader*) all the fragments and set the status as "finished". If this was the first *AlarmOK* PDU for this queue then the Node Driver receives the confirmation that the alarm was sent.

## *8.8 Request with Response Service*

The processing of Request with Response service is similar to the Unicast service on the AP-to-Node direction followed by a "single-route" Alarm service on the Node-to-AP direction.

One of the differences it that at the Node side a special queue is kept in "open" state while the Node driver processes the Request. The Node driver can opt to send a Response with any data length (including 0) or a special *NoResp* command. The Response is matched to the Request by the *TL Transaction ID*.

If the queue timeout expires before a response is issued, a *NoResp* PDU is sent to the AP Driver by the Transport Layer and the *TL Transaction ID* is invalidated. There is no way for the AP Driver to know if the *NoResp* was due to a timeout or really issued by the Node Driver. The AP Driver only receive timeout errors for the *transmission* phase of the request.

An a additional feature is that the Node Driver can send a *RespTimes* command for a particular *TL Transaction ID* with information on the expected delay till the Response command. Periodically the Transport Layer scans all open requests in the Node and selects the smallest of these delays to setup the timeout sub-system of the Network Layer (as presented in Section 7.4). The timeout is set to half the minimum value indicated by the Node drivers.

# Chapter 9
# Validation

This chapter addresses the experimental validation of the mechanisms proposed in Chapter 7 and for which some implementation detail was provided in Chapter 8. This validation was based in extensive test cases in a simulation environment, which allowed to build an application-rich scenario and comparison with performance in actual field tests.

## *9.1 Introduction*

With the test scenarios devised to validate the REMPLI Transport Layer, particular attention was given to the advanced features which were the focus of the previous chapters. These consisted in the new end-to-end services, routing in a two-level network, scheduler performance and resource usage. Extensive test cases were implemented in a simulation environment, whilst field test results allowed to confirm the adequateness of the approach.

## *9.2 Simulation Environment*

By design, the Transport Layer was prepared to be tested in the OMNeT++ simulation environment. However, this environment did not support the REMPLI physical and network layers, thus a new emulator was designed by a group within one of the project partners (iAd). This application emulates the physical layer behaviour of the PLC system in a single time slot using a simplified (and fast) mathematical model of the network, which deals with successful or successful data delivery between several network points at the same time. For example, if two stations in different network positions send a PDU to the physical medium, the emulator calculates all the positions that received each PDU and delivers the respective data considering possible mutual interferences. These network models where obtained from a much more complex (and time-consuming) simulation system of the physical layer, including channel encoding and synchronisation, and estimated response of the physical layer considering the signal responses of actual power lines.

The emulator included data files for topologies like *Ring*, *Open Ring* and *Random Area* (a central point with several “trees” radiating). Depending on the models, the number of positions varied from 10 to 200 stations.

On top of this emulator, a version of the Network Layer was developed by another partner (Loria), therefore building a complete environment for testing. This was integrated with the OMNeT++ simulation tool via a TCP channel. The simulation sends new data packets to the emulator where they are queued. In each "time slot". an emulation cycle is requested to the emulator that sends back the resulting data packets and the respective stations. Since the emulator supported only single-master networks, multi-master networks were supported using two (or more) emulators in parallel, and the number of used slots adjusted accordingly. Nevertheless, multiple master networks sharing the same TDMA slots (e.g. in "distant" points of the network) are not supported in the current version of the simulation system. This would require changes to support multiple network layers in parallel and move each master to a programmed position in the grid (currently they are always in position 1).

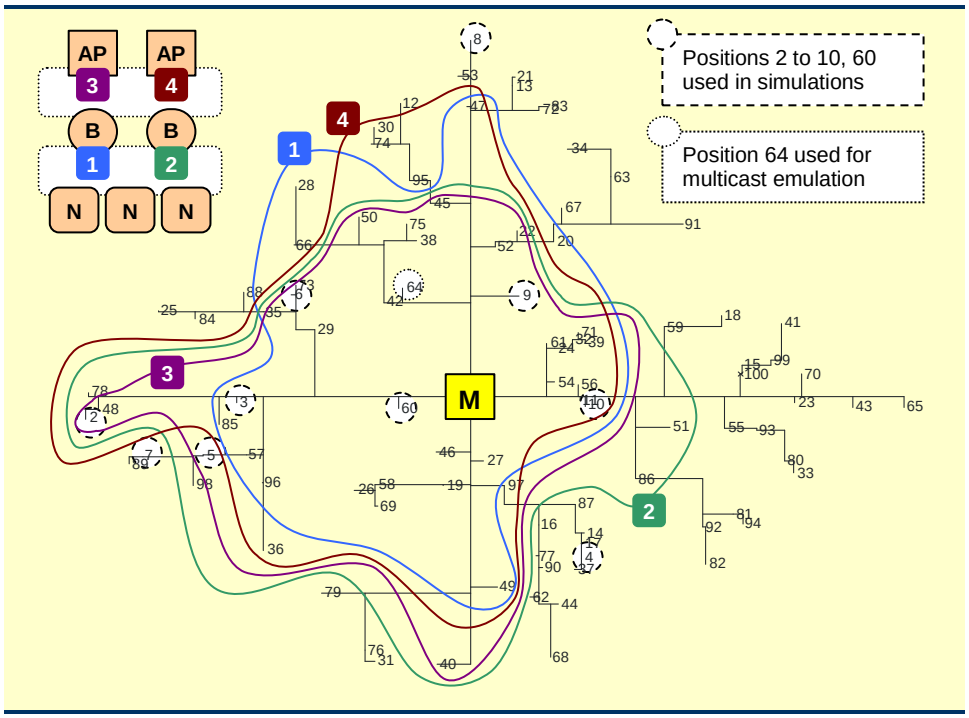



**Figure 9.1: Nodes connected in REMPLI Network Emulators**

Figure 9.1 presents the physical layout of the nodes of the physical layer emulator; the curves surround the slaves that connected to the master in the centre of the network. The difference between the simulations is only on the seed of the random number generator used for the physical layer emulator. In these physical layer simulation scenarios, all 100 stations were connected after 140 simulation seconds. Also presented are the selected node positions for simulation: 2 to 10 and 60. Looking at the map it is possible to conclude that positions 4, 7 and 8 never receive data, and positions 3, 6, 9, 10

and 60 always connect successfully. Other positions connect only in some of the emulators. Position 60 has the best reception possible on the network in terms of delays with a high probability of not using repeaters at all; other positions may be subject to repeater delays, especially the ones nearer the borderline.

Emulators 1 and 2 are used in the low voltage segment and Emulator 3 and 4 in the high voltage segment. Table 9.1 presents the map positions with corresponding Node Address and connection capability. On Emulators 3 and 4 only Node Addresses 201 and 202 are presented since only two bridges were used in the simulations; Node Addresses 301 to 310 are for Nodes. These values are only for the Transport Layer and Drivers. The Network Layer and Physical Layer still behave according to the scenario of 100 stations in each network, including internal management activity to keep track of all logged stations.

**Table 9.1: Channel conditions in each REMPLI emulator position**

| ***Position →*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** | ***60*** |
|---|---|---|---|---|---|---|---|---|---|---|
| Node Address | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 310 |
| Emulator 1 | | OK | | | OK | | | OK | OK | OK |
| Emulator 2 | OK | OK | | OK | OK | | | OK | OK | OK |
| Node Address | 201 | 202 | --- | --- | --- | --- | --- | --- | --- | --- |
| Emulator 3 | OK | OK | OK | OK | | | | OK | OK | OK |
| Emulator 4 | OK | OK | | OK | | | | OK | OK | OK |

Apart the Network Layer Module, simulation-specific Access Point and Node Driver Modules were designed for OMNeT++. Other modules like TRM, QM, NLI and RCIM use the same source code as the target implementation.

The Access Point Driver module is specific for the simulation scenarios. In a real station, the drivers would receive requests from external applications and translate them into protocol requests to the DeMux. In the simulation there are no applications making requests, as the Access Point Driver module emulates these requests in each simulation station depending on the features to be tested. The Access Point Driver can generate confirmed requests, unconfirmed unicast requests, unconfirmed multicast requests and status requests. Since we have two different PDU headers in the system depending on data lengths, the simulation also takes this fact into account.

On the other end of the network are the Node Drivers, which are also simulation specific. In this case the only requests are the Alarm Service and Status Service Updates. Depending on the station, the Node Driver can also issue automatic responses (eventually with a delay) to confirmed requests of the Access Point.

In the end system there is a DeMux layer between the Transport Layer and the (multiple) Drivers per station, but for simulation this was not critical since the service interface and functionality is comparable for modeling purposes. Although the name, the “Drivers” in these simulations emulate the DeMux functionality including additional fields in the transfers with the TL when appropriate.

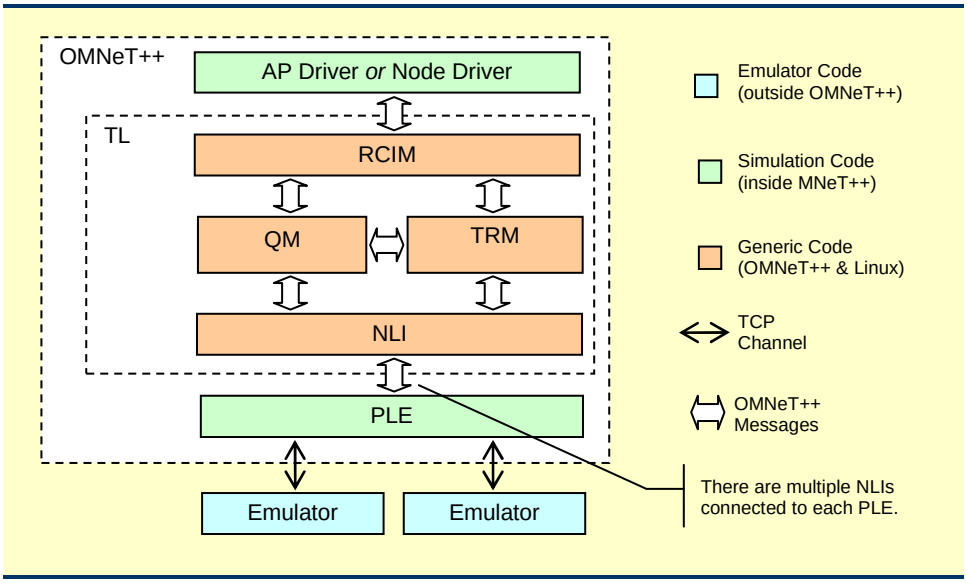



**Figure 9.2: Transport Layer Simulator Architecture**

### *9.3 Base Network Layer Characteristics*

The base network emulator was configured matching the laboratory test bed configuration that was also successfully in the field trial: 64-byte packets (51 available after Network Layer) and a slot time of 9.5 ms.

Since we are using a dual master network sharing equal parts of the time slots with a normal interleave of four, the parameters for the emulators were adjusted accordingly. Each emulator was configured with an interleave-value of two and each pair is called in alternating slots. In reality, each emulator is activated every 19 ms. Apart this configuration parameters the network emulator is treated in a black box fashion in this section.

In order to be able to obtain higher sensitivity regarding the Transport Layer performance, it is interesting to use information about the underlying Network Layer. With this data, it is possible to compare not only the end-to-end additional delays included by the additional Transport Layer mechanisms, but also to have some information on the issues create by lower layers delays and errors (also having the base average bandwidth). In order to be a faithful representation of the future tests base network, this test only sends PDUs to slaves that logged in and it is used in the other tests (i.e. Node Addresses 301-310 and 201-202).

The first test was to produce the maximum allowed load in the network with unconfirmed requests from all masters, registering the delays until the PDU was delivered to the slaves and/or possible error indications. For this test, the NLI was adapted to provide the desired scenario, including careful synchronization with the emulator slot timer, while remaining TL functionality was disabled. This

synchronization was possible since the NL generates an event every time it reads a packet from an output queue. This process guarantees that an emulator slot occurs immediately after the Master Network Layer generates a new request with an effective null time difference and with minimal queuing. In the normal operation of the Transport Layer, there is no such synchronization since the timing is controlled by external entities (the Drivers) that do not have the need for a precise synchronization with the Network Layer. In practice this means that "real-life" results should have in average an additional delay of half a slot in a mono-master network (in the worse case the PDU has to wait almost an entire time slot), or in a dual-master network with interleaved slots (e.g. M1-M2-M1-M2 and so on…) the average delay increases to a full time slot.

The tests results are summarized in tables 9.2 to 9.4.

**Table 9.2: Network Layer Performance Tests**

| ***Global paramenters*** | | ***Slots*** |
|---|---|---|
| Slot time ("real") | 9.5 ms | |
| Interleave factor ("real") | | 4 |
| Number of networks ("real") | 2 | |
| Number of masters per network | 2 | |
| Number of slots per interleave cycle for one Master | | 2 |
| Physical layer raw data rate (both networks) | 107786 bps | |
| Physical layer raw data rate per master | 26947 bps | |

**Table 9.3: Network Layer Unconfirmed Requests performance test**

| ***Unconfirmed Requests Results*** | | ***Slots*** |
|---|---|---|
| Total possible slots | | 31578 |
| Slots used / Packets issued | | 20597 |
| Network availability | 65% | |
| Average Tx data rate (TL Payload) per master | 14006 bps | |
| Minimum network access delay | 0 ms | 0.0 |
| Average network access delay | 11 ms | 1.1 |
| Maximum network access delay (99.5% best) | 57 ms | 6.0 |
| Maximum network access delay | 3382 ms | 356.0 |
| Packets delivered | | 18945 |
| Packet Error Rate | 8% | |
| Average correct Tx data rate (TL Payload) per master | 12883 bps | |
| Minimum transmission delay | 38 ms | 4.0 |
| Average transmission delay | 64 ms | 6.7 |
| Maximum transmission delay (99.5% best): | 152 ms | 16.0 |
| Maximum transmission delay | 3420 ms | 360.0 |

**Table 9.4: Network Layer Confirmed Request performance test**

| ***Confirmed Requests Results*** | | ***Slots*** |
|---|---|---|
| Total possible slots | | 42104 |
| Slots used / Packets issued | | 13024 |
| Network availability | 31% | |
| Average Tx data rate (TL Payload) per master | 6642 bps | |
| Minimum network access delay | 0 ms | 0.0 |
| Average network access delay | 89 ms | 4.5 |
| Maximum network access delay (99.5% best) | 285 ms | 30.0 |
| Maximum network access delay | 5890 ms | 342.0 |
| Packets delivered | | 13020 |
| Packet error rate | 0.03% | |
| Average correct Tx data rate (TL Payload) per master | 6640 bps | |
| Minimum transmission delay | 38 ms | 4.0 |
| Average transmission delay | 102 ms | 10.8 |
| Maximum transmission delay (99.5% best) | 437 ms | 46.0 |
| Maximum transmission delay | 3287 ms | 346.0 |
| Confirmations received | | 13017 |
| Deliveries non confirmed | 0.02% | |
| Minimum confirmation delay | 76 ms | 8.0 |
| Average confirmation delay | 154 ms | 16.3 |
| Maximum confirmation delay (99.5% best) | 513 ms | 54.0 |
| Maximum confirmation delay | 3325 ms | 350.0 |

Analysing these results, the first conclusion is that a large part of the network raw capability is “lost” in the current setup. A part of it is lost in headers (64 bytes turn into 51 bytes usable by the Transport Layer: a 20% drop), but most of the “lost” slots are used by the network layer repeater mechanism to increase the coverage area (some are lost in network layer’s management). Although the fact that we are connecting only a few stations to the OMNeT++ simulation, the emulator, which includes the network layer functionality, uses the complete network of 100 stations. Since the selected nodes are distributed in the “connection” area, some have the minimum delay (4 slots) but in average they need at least two repeaters (8 slots) and some need three (12 slots). However, less than 0.5% of the requests show extremely large delays, much larger than the repeating slots and more in the order of 3 seconds. A quick review of the test logs shows that the network did not generate any traffic during these delays, but data stays on the Network Layer queues, therefore it is possible to assume they reflect internal network maintenance cycles (also possible that the network detects a fault and stopps sending packets temporarily). These outages have a great impact in the network. Considering the unconfirmed request case, the 0.5% worse cases (101 requests) occupy 5662 slots. If we deduct from this value the time used in average for the other 99.5% requests, in order to process 101 requests we have 5056 slots “wasted”. This is around 20% of the usable slots.

Concerning error rates, for unconfirmed PDUs, 8% of the requests did not reach their destination. Nevertheless, this results vary wildly depending on the link studied, for example the connection between 101 and 201 is consistently worse (minimum 8.0, average 12.1 slots) than the connection between 101 and 202 (minimum 4.0, average 5.6 slots). This result is expected from the mere observation of the connection map where position 2 (station 201) is in the borderline of the connection area and position 3 (station 202) much nearer the centre of the map.

For confirmed PDUs, there is a residual packet error rate (4 errors in 13024 requests); in the test we also reporter very few missing confirmations for correctly delivered packet (just 3 missing confirmations in 13020 delivered packets). Therefore, the simulated network is extremely reliable when using confirmed requests when compared to the unconfirmed request performance in this parameter.

In terms of delays, it takes at least 4 time slots for a PDU to go from A to B. For confirmed requests, the average is 6.7 slots for data to be delivered from the master to the slave, and this value increases to 10.8 slots for confirmed requests (see Figure 9.3). The additional delay in confirmed requests is due to the automatic retry mechanism supported by the Network Layer: if an error is detected, additional time slots are used to retry the request and, as seen in the final error rates, the Network Layer behaves well in this aspect. These delays also influence the queuing time before PDUs are sent.

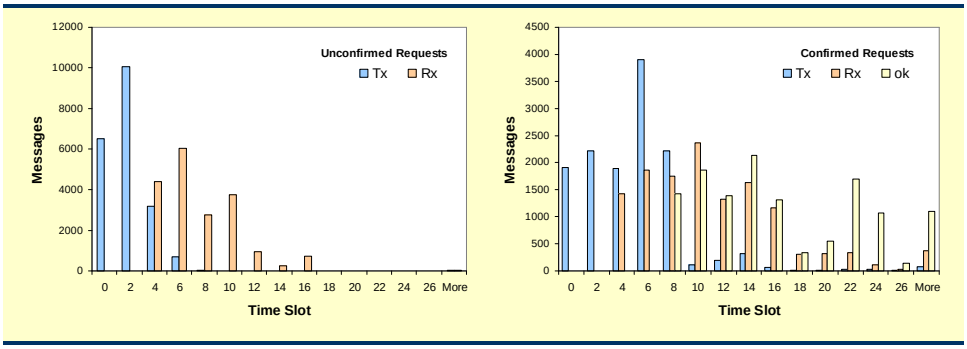



**Figure 9.3: Network Layer performance histograms**

It was also possible to repeat the confirmed request tests with echo functionality from the slave station. The results were similar to the table above and most of the confirmations at the master were followed by the slave data in the same slot. 11859 slave responses arrived to the masters, which means a loss of about 9%. Slave to master data payloads are typically sent by the Network Layer in the confirmation of master request, but they are not re-confirmed. For a particular master to slave confirmed request a PDU can be generated by the slave with a data payload and not make it all the way to the master. The network layer at the master side reissues the original request if needed but the slave data is lost anyway.

Another effect was that approximately 20 PDUs were queued for delivery after the last confirmation at master 201 and 202, while in master 101 and 102 only 3 PDUs where delivered late. Most of these “queued” PDUs were delivery to the master in a

burst with only 2 to 6 slots intervals between PDUs. Since the slave-to-master channel was being used at this time, the network data rate for correctly delivery data went up to 12679 bps - that is almost the same as the results for non-confirmed data.

Another test was setup with duplicated echo PDUs. In this scenario, the Network Layer tries to empty the slave's queues as soon as possible and uses more bandwidth for the slave-to-master channel. This resulted in only 5256 confirmed requests generated, one of them being lost during transmission and 9556 PDUs being received back in the master (again 9% less than the generated).

In global 14812 slots where used for this last test of the around 42104 possible. In the tests with responses a lot of additional slots where used since the slave replies continued after the last transmission for more 5 seconds in bridge 202 and around 2.5 seconds for the other masters. On non-echo tests this additional slots where much more limited (maximum 288 slots) and were not taken into account.

### *9.4 Unicast Test and the TL Queued Requests Parameter*

Apart the basic address conversion tables that assign *Unique Serial Numbers* to *Node Addresses*, the Transport Layer has multiple configurable parameters concerning not only fragmentation-related information but also timeouts, thresholds, and queue sizes.

The optimal set of parameters is highly application dependent, but for all applications one parameter in particular has a likely direct impact on the timing performance of the system. This parameter is the number of pending Network Layer requests that the TL leaves open before stopping sending new requests.

In theory, a higher queue parameter guarantees that the NL queues are never unnecessarily starved (e.g. a transmission slot is available but the Transport Layer failed to fill it up in time); on the other hand smaller queues guarantee faster response to priority packets.

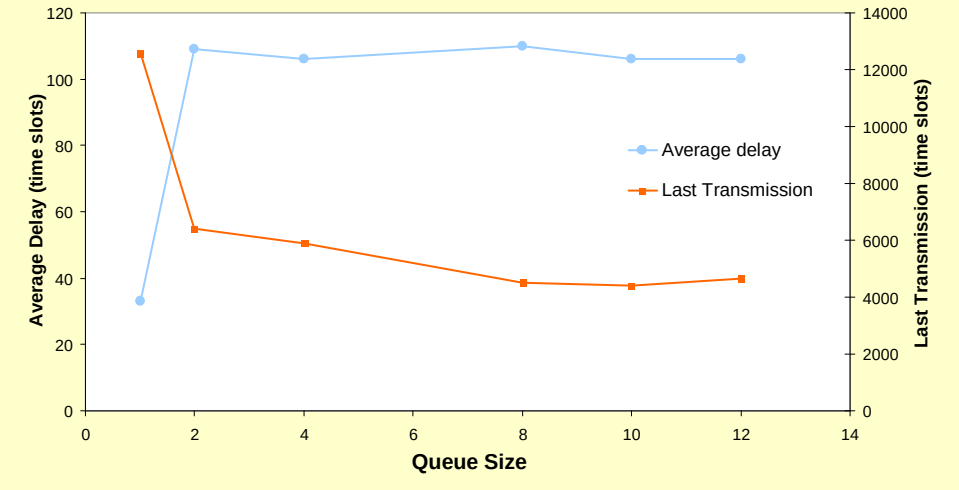



**Figure 9.4: Queue size average delay and last transmission**

For this test, small sized packets (that fit in a single fragment) were used. The tests sent 500 packets from each master to all stations logged in the network, including the Bridges. A total of 2000 packets were injected in the network (one for each master, and one for each reply). Up to 10 packets were sent with 1 ms interval. After this, new packets where generated when the TL replied to a previous request.

The node echoes the packet back in the same simulation instant. Since the AP Driver is not synchronized with the NL, there may be an additional delay up one time slot, which is almost irrelevant in this scenario.

For a queue size of one, the test ended before all the packets where processed since the delay was extremely larger than in other tests (see Figure 9.4). For the other queue sizes, the average is almost the same (around 105) and the last transmission happens later (6400 for queue size 2) for smaller queue sizes, but is practically the same for queue sizes of 8 to 12 (around 4500).

The histogram in Figure 9.5 allows a slightly different analysis: the queue size of 1 behaved well and delivered more than 300 packets with a delay smaller than 40 time slots between transmission and delivery; however, this effect was destroyed by the amount of time the confirmation response took to go back to the AP Driver.

Analysing the other queue sizes, size 2 delivers most packets in the 40-60 slots interval and other intervals follow an almost linear response but with a long tail. Size 8 and 12 show two irregular bell curves with queue 8 having the norm at 100 and queue 12 at 120.

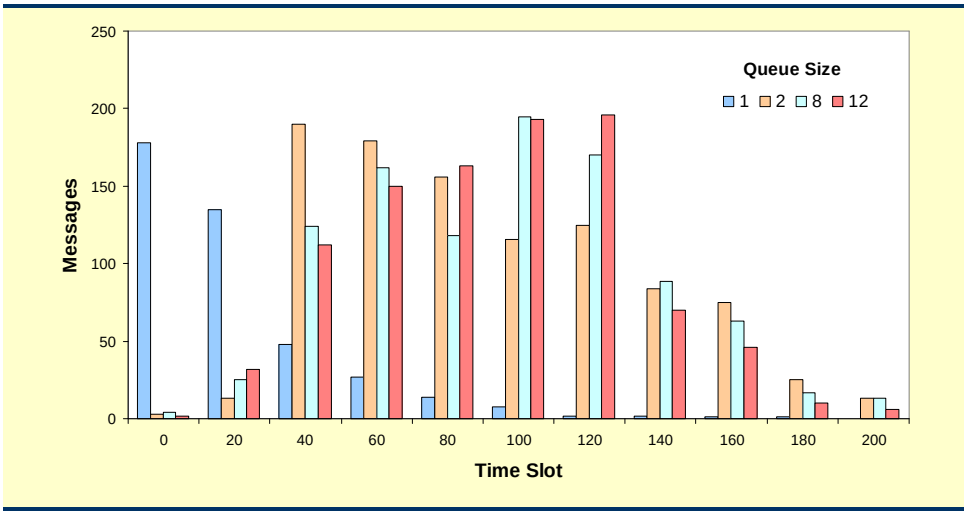



**Figure 9.5: Delivery delays histogram**

The main conclusion from this analysis is that if there are no stringent requirements to deliver higher priority packets in front of other traffic, a queue value between 8 and 12 is a reasonable balance between delivery times and faster confirmations. Therefore, a value of 8 was selected for the next tests.

The following histogram in Figure 9.6 shows the side-result of this first test set: the temporal behaviour of the Unicast service for very small data payloads (i.e. single-fragment).

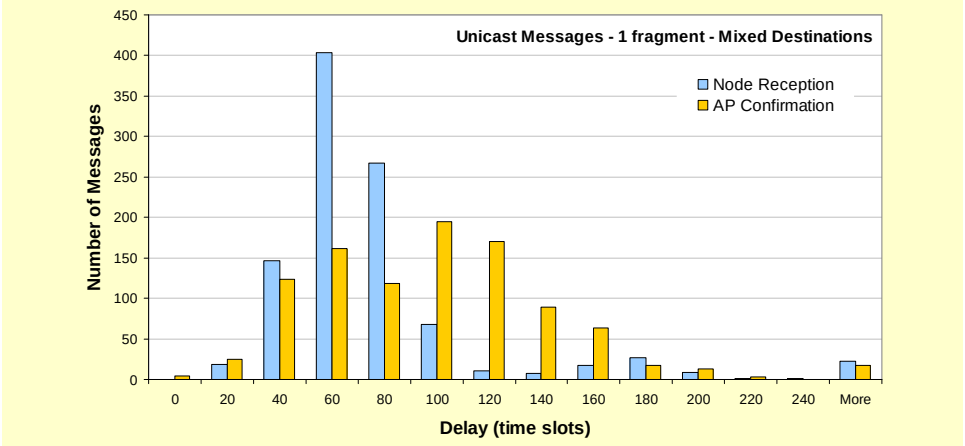



**Figure 9.6: Unicast Requests - 1 fragment – Mixed destinations**

This test was performed sending packets to all stations. Since the majority of stations are nodes behind the bridges, it is natural that the final performance is not ideal. The next test set shows the requests only for direct-connection destinations 201 and 202. Here the variability is much limited and, even with 1000 packets generated in each AP, the system experiences very small variations.

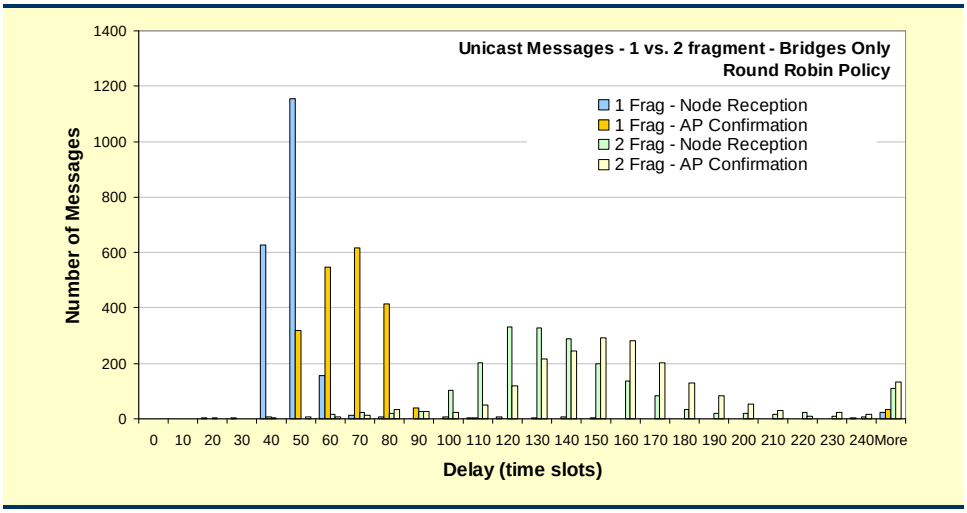



**Figure 9.7: Unicast Requests – Bridges Only – Round Robin Policy**

However, the fragmentation performance could be considered unexpected (see Figure 9.7). The average delay for the Node Reception with one fragment was 48 time slots, but with two fragments, this average went to 147 time slots. The issue here is the round-robin scheme of the Fragment Scheduler: since up to 10 packets are scheduled in parallel, the second fragment has to wait for the round robin cycle to be complete.

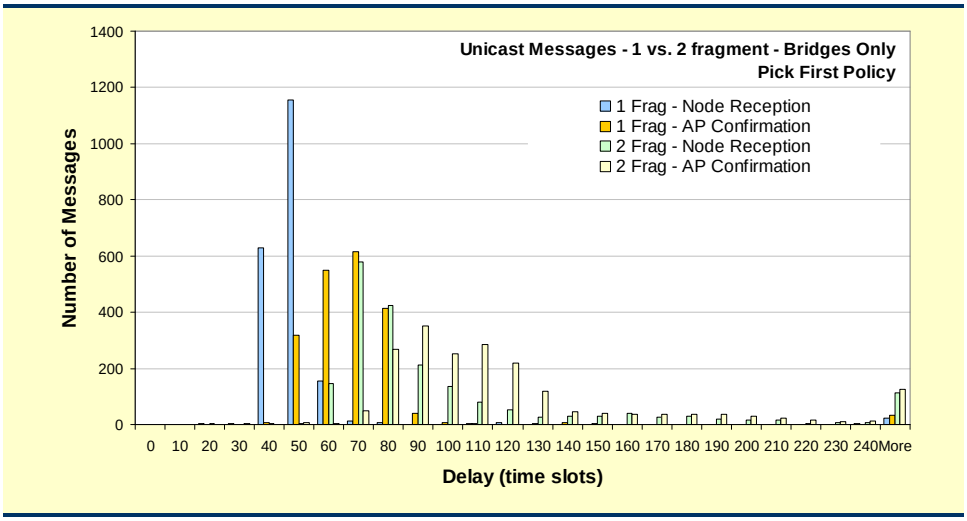



**Figure 9.8: Unicast Requests – Bridges Only – Pick First Policy**

In theory, this method shares the available slot more evenly between the queues. On the other hand, the pick first policy guarantees fast delivery for most packets, but if there is a network problem then a small group of packets suffers long delays (see Figure 9.8). The principle of round robin applies well to multiple sized packets, but the pick first policy induces additional delays for small packets, that have to wait for big packets to be completely delivered.

## *9.5 Request/Response Service*

In the Request/Response service, an Access Point Driver can send a packet to a particular node and the Node Driver responds with another packet (that can be empty). In the first test, each AP Driver sends a uniform distribution of packets with 1 to 4 fragments to all connected Nodes. The Node Drivers respond immediately with the same data block. The histogram in Figure 9.9, shows the delays until the Node receives the request packet, and the AP receives the response packet (based on the original message transmission time).

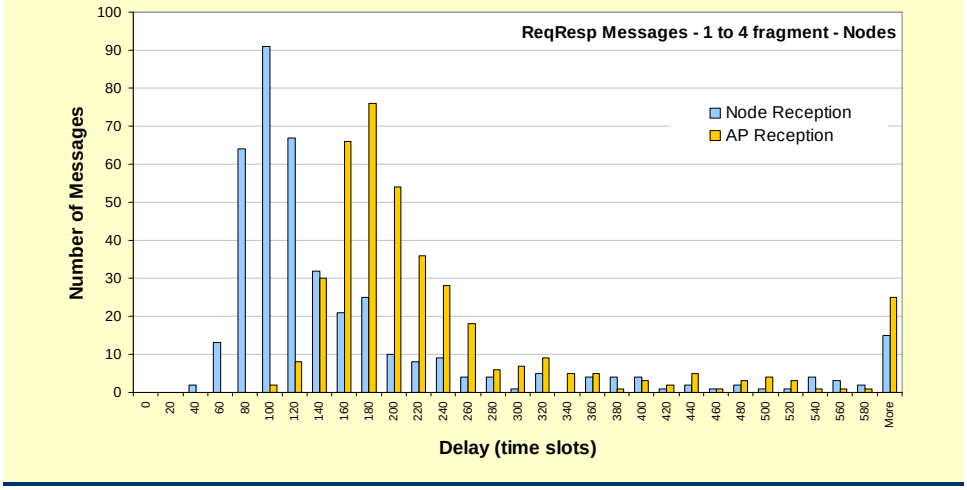



**Figure 9.9: Request/Response, 1 to 4 fragments, Nodes only**

## *9.6 Alarm Service*

The Alarm service enables a Node station (or the Node functionality in a Bridge) to send a confirmed packet to at lest one of the available Access Points. The Transport Layer manages the reliability and data management features, but it benefits from the Network Layer capability of "picking" fragments from slaves. The Network Layer task is eased by the fact that not only confirmation traffic from masters to slaves but also internal TRM-to-TRM traffic "opens" slots for data transport in the reverse channel.

The first test of this service uses both Bridges, with 150 requests sent sequentially one at a time with a small delay between the OK and the new request at each bridge. In the current implementation, the TRM does not try to select better paths for slaves and it simply serves the queues based on priorities.

In the one fragment test, 66% of the packets were received first by AP 101 and 34% by AP 102. The minimum delay of an AP event was just 4 slots and the average 79 slots. On the other hand, the average delay for confirmations on the Node was 95 slots since the start of the node request. On this test the requests were generated at the Node two seconds after the confirmation, and of the 300 packets generated, 20 overrun this delay and might had their performance affected by the new request on the network.

For the two fragments test, the minimum delay for delivery was 10 slots with an average 105 slots, an increase of 26 slots compared to the one-fragment test. For the confirmations the average was 121 slots, again an increase of 26 slots compared to the one-fragment test. The distribution of the first alarm between AP 101 and AP 102 was very similar to the one-fragment test (see Figure 9.10). The requests were generated 4 seconds after the confirmation and none was overrun. In the one-fragment test, all alarms were received by both APs, while on the two-fragment test only 2 out of the possible 600 were not received.

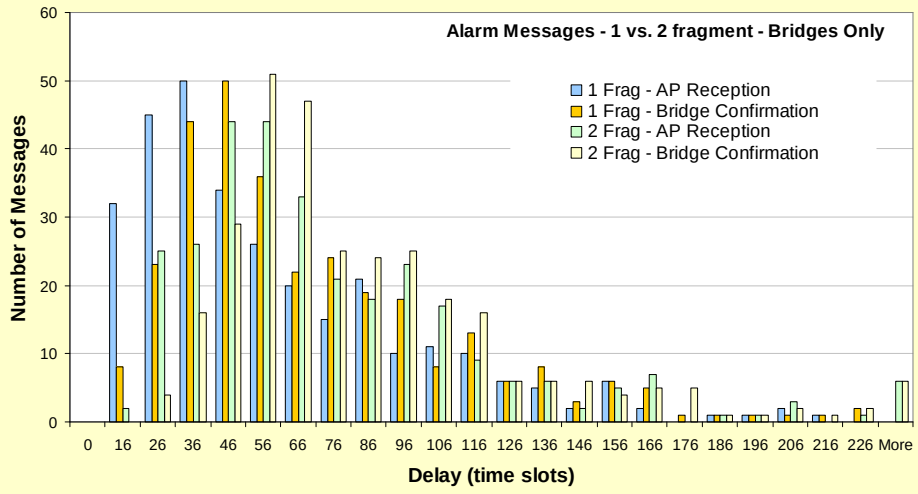



**Figure 9.10: Alarm Service, 1 vs 2 Fragments, Bridges 201 and 202**

For the Nodes, 50 requests were generated in active Node 310, the "best" connected station in the network. The average delay for the first AP delivery was 188 slots, and the confirmations were received at 212 slots average. 60% of the requests were received first at AP 101 and 40% at AP 102.

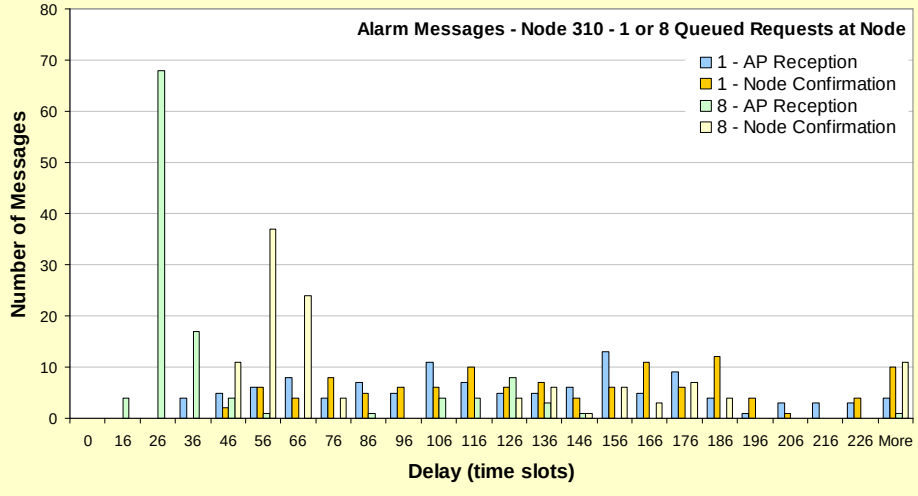



**Figure 9.11: Alarm Service, 1 Fragment, Node 310**

The test was repeated with eight fragments queued at the Node side. The performance improvement was significant. The average delay for the first AP delivery was 80 slots, and the confirmation received at 193 average slots. Looking at the histogram in Figure 9.11 it is clear that a significant part of the deliveries are in the 26 to 35 slot range when they were much more scattered in the one-fragment scenario. However, the largest delay increased to 2086 slots, when it was around 400 slots for the tests with only one-fragment queues.

The main point on these two tests is that when a single request with no traffic on the network is present, it is up for the network layer polling cycle to fetch the data, and this operation can take several slots. When several requests are queued together, the network polling cycle fetches all the pending requests in a fast sequence reducing the average delay.

This conclusion led to another test: to issue Alarms and Unicast services in the same network. A small alarm test was performed with 50 Unicast requests and 50 Alarm requests. The average delay for the Unicast delivery was 153 slots, but the average delay for the Alarm packets was only 24 slots, showing a significant improvement on the Alarm performance. The results can be compared in Figure 9.12, where the results without AP-to-Node traffic are the same as presented previously but scaled down to match the 50-packet scenario.

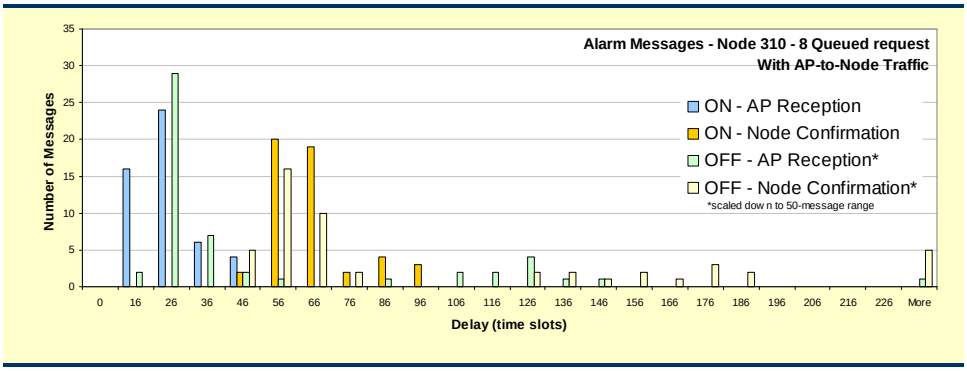



**Figure 9.12: Alarm Service with AP-to-Node traffic**

These results show that a good performance may be obtained from the Alarm service but there is a trade-off between "wasting" network bandwidth at the Network Layer level, for more frequent pooling requests, and limiting the expected delay for the Alarm service. However, if the network is used periodically by the application then this already used bandwidth is beneficial for decreasing the Alarm delays, differently from the "traditional" way network bandwidth is characterized.

Given the connectivity losses experienced by the physical layer emulator, the maximum delay of the Alarm is dependent on the network interruptions and not on the Transport Layer capabilities.

## *9.7 Unlimited packet size and fragmentation*

One of the design features of the REMPLI Transport Layer is to use multiple headers depending not only on the size of the packet but also on the number of fragments. In particular for the Unicast service there is a "Small Header" that takes 4 bytes and handles PDU lengths up to 255 bytes ($2^8$-1); a "Large Header" that takes 6 bytes and handles PDU lengths up to 16 MiB (to be precise $2^{24}$-1); and finally a "Minimum Header" that takes only 3 bytes. All this length limits are compile time options and can be easily changed depending on a particular deployment.

The objective is to have very large packets on the system without compromising the real-time performance of the very small packets.

On the current test set, the TL can issue fragments with up to 51 bytes to the Network Layer. This means that, in practice, the usable data payload per fragment is 48 bytes for minimum headers, 47 bytes for small headers and 45 bytes for large headers.

With these parameters the number of fragments per packet is:

$$n(L,S) = \begin{cases} \lceil L/S \rceil & ,if\ L < 47 \cdot N_1 \\ \lceil (L - N_1 \cdot S)/48 \rceil + N_1 & ,if\ L \geq 47 \cdot N_1 \end{cases} \quad \textbf{(9.1)}$$

Effective usable bit rate $B_{app}$ available for applications in bits per time slot is:

$$B_{app} = \begin{cases} (L/n(L,47)) \cdot 8 & ,if\ L \leq L_1 \\ (L/n(L,45)) \cdot 8 & ,if\ L > L_1 \end{cases} \quad \textbf{(9.2)}$$

where $L$ is the packet size in bytes, $N_1$ is the number of starting headers, before switching to minimum headers (3). $L_1$ is the length limit for using small length headers (255).

The graphics in Figure 9.13 present the usable bit payload per packet and give some insight on the consequences on choosing the length parameters.

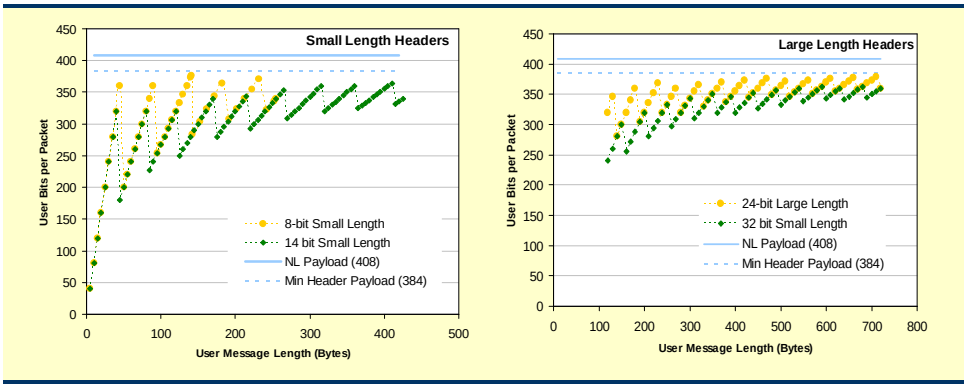



**Figure 9.13: Headers choices and effective usable payload**

For the small length headers it is clear that using 8-bits results in better bandwidth usage than using 14-bits. However, this 8-bit choice means that we can only use small headers for packets up to 255 bytes; after that, the large length headers start to be better.

In addition, when it comes to large headers it is obvious that a header with 32-bit length takes more space than one with a 24-bit length field, even if we consider that they are only used in the first three fragments. This difference starts to fade with bigger packets, but it is very significant for smaller packets and a large portion of the bandwidth can be wasted in a system that issues many small packets.

For systems that really need 32-bit length packets, it may be necessary to use small length headers larger than 8-bits, in order to avoid the performance hit on packets larger than 256 bytes. However, for systems where the time-critical packets are very small and there are not that many packets in the 256+ borderline, the 8-bits could be the best solution. The current setup of 8-bit small length and 24-bit large length provides a smooth transition and does not waste too much bandwidth in the smaller packets.

For system configurations with larger Network Layer payloads, the predominant factor for very small packets is not the header sizes but the fact a single fragment is sent in each slot resulting in poorer real-time performance.

Finally, another issue to take into account is the number of start fragments: in the current setup, with an interleave factor of 4, it is very unlikely that after sending 3 start headers and a minimum header the minimum header fragment arrives before all the other fragments. If this happens, the minimum fragment is discarded, but the other fragments are received correctly. In systems with low repetition rates at the network layer level, it is possible to send even less start headers without loosing packets, thus this can be an interesting option (depending on the traffic characteristics of the network).

### *9.8 Small size packets delivered quickly over bi-level network*

Most of the previous tests have “occupied” the network by issuing new requests when the previous ones were completed. The following test issues packets separated by one-second interval, basically “isolating” each request. The idea is to measure the time it takes to complete a unicast request in the network when there is no other traffic.

A total of 50 requests were issued from Access Point 101 to Node 310. The TRM Scheduler used only Bridge 202 in this test. The average delays for reception at the Node were 18 slots and 26 slots for 1 and 2 fragments respectively. The best case took only 10 and 18 slots, which are near the best theoretical values possible. For one fragment case, it takes 4 slots for the Access Point to Bridge transmission, plus 4 slots for the Bridge to Node transmission. The two other slots are lost due to probabilities: 1 since the probability of having 4 transmission slots in both networks is very low, and the other since the probability of having 0 slots of network access in both networks is also very low. With a larger data set some occasional values of 9 or 8 slots might occur. For the average case, the network delays are larger (due to NL’s internal traffic and also possible repetitions), and the access delays are more random, again due to NL’s internal traffic.

The histogram in Figure 9.14 also presents the delays for one fragment for unicast directly to Bridge 202, where it is possible to conclude that the one fragment unicast for

node 310 is almost a copy more "spreaded" (due to additional jitter) and shifted four slots.

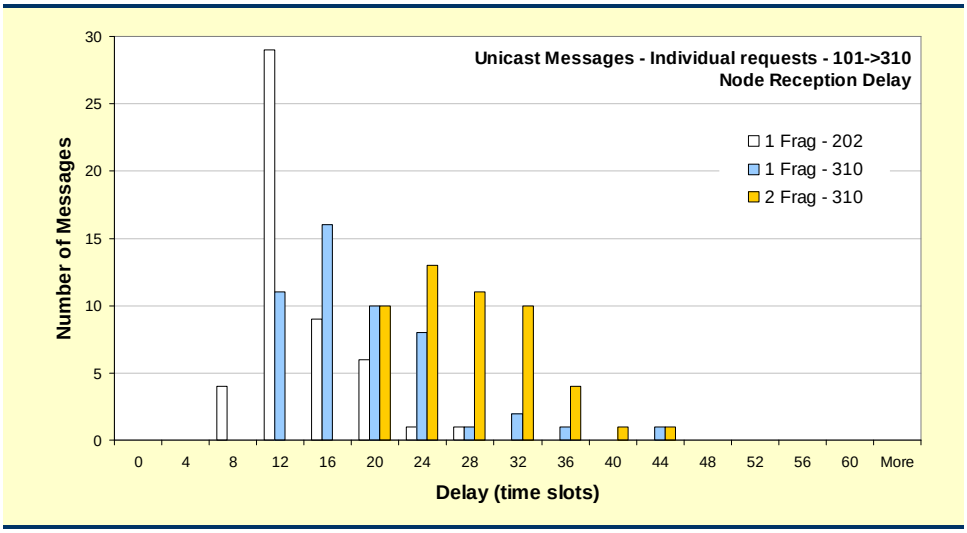



**Figure 9.14: Unicast Service, Individual Requests (101→ 310)**

## *9.9 Priority processing*

To test the traffic priority mechanism, the network was flooded with up to 10 queued unicast 1-fragment packets of priority 10 from AP 101 to Bridge 202. In parallel, at every 500 ms a different priority unicast packet is sent from AP 101 to Bridge 202. 200 packets where generated in total, including 15 of different priority. For the bridged request from AP 101 to Node 310, 50 packets were generated in total, including 10 to 12 of different priority generated every 200ms.

The results are displayed in Table 9.5.

The effect of the priority values is clear in these tests. The (large) difference between the "normal" TL Priority of 5 and the "special" priority of -1 is due to the queuing in the NL Layer. In this test set (like in most of the other tests in this section), the maximum number of queued requests by the Transport Layer to the Network Layer was set to 8. This means that when a request with priority 5 appears it is likely that it already has 7 requests "in front" at the Network Layer queue with priority 10 and it cannot overpass them. If the special priority is used, then the Network Layer handles the priority and the request with priority -1 is processed in front of the 7 pending "priority 10" requests.

**Table 9.5: Transport Layer unicast priority test**

| | ***Average Slots*** | ***Maximum Slots*** |
|---|---|---|
| Same Priority Test – 101→ 202 | | |
| Flood Priority = 10 | 58 | 84 |
| Timed with Priority = 10 | 58 | 85 |
| Higher Priority Test – 101→ 202 | | |
| Flood with Priority = 10 | 60 | 90 |
| Timed with Priority = 5 | 37 | 42 |
| NL Higher Priority Test – 101→ 202 | | |
| Flood with Priority = 10 | 62 | 90 |
| Timed with Priority = -1 | 10 | 19 |
| Same Priority Test – 101→ 310 | | |
| Flood Priority = 10 | 84 | 190 |
| Timed with Priority = 10 | 84 | 135 |
| Higher Priority Test – 101→ 310 | | |
| Flood with Priority = 10 | 89 | 185 |
| Timed with Priority = 5 | 66 | 89 |
| NL Higher Priority Test – 101→ 310 | | |
| Flood with Priority = 10 | 145 | 361 |
| Timed with Priority = -1 | 37 | 63 |

The duration of the test is limited by the number of packets. In the last test set from AP 101 to Node 310 with priority -1, there was a side-effect of this policy: the flooding requests were significantly delayed by the presence of the higher-priority traffic and so 12 high-priority packets were generated instead of the 10 in the other tests.

Since the higher priority requests were generated every 200 ms, i.e. every 21 time slots, this may be the result of a rush-in effect. If two higher priority requests overlap the lower priority traffic requests, they delay *all* the pending lower priority request. This effect is minor in the test with priority 5 due to the damping effect of the Network Layer queues, whichlimits the number of affected pending lower priority requests. To test the theory, a test re-run was done with the Driver queued requests increased to 15 (instead of the original 10) and keeping the 8 requests at the NL queues. The tests were done with same-priority and with priority 5 requests. There was a 31% increase in the delay of the normal traffic when the priority 5 requests were included in this scenario.

### *9.10 Best route selection*

The objective of this test was to analysed the behaviour of Transport Layer routing in dynamic traffic situations. The test is based in the fact that only Bridge 202 connects AP 102 to Node 301 due to the emulator network seeds selected. On the other hand, both Bridges are available by AP 101 to reach Node 310.

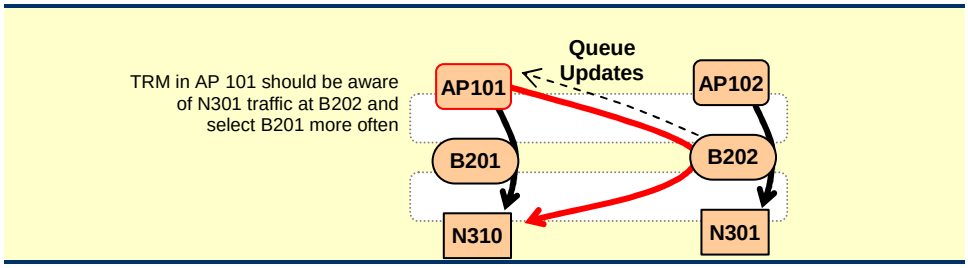



**Figure 9.15: Bridge routing test concept**

To test this application the TRM internal link quality values were updated every 200 ms. In the first test, the queue updates from Bridges to AP where used as normal; in the second test the code was changed to remove the fragment count update at the AP side. 40 packets were generated by AP 101 and sent to Node 310, AP 102 sent 40 packets to Node 301. Both bridges could route packets from AP 101, and Bridge 202 was the only route for packets from AP 102 (see Figure 9.15). The results are summarized in Table 9.6.

**Table 9.6: Transport Layer bridge routing test**

| | ***Average Slots*** | ***Maximum Slots*** |
|---|---|---|
| With Fragment Information Active – Global: | 90 | 167 |
| 101 → 310 | 77 | 167 |
| 102 → 301 | 104 | 162 |
| Bridge 201 used 7 times, 202 used 73 times | | |
| With Fragment Information Inactive – Global: | 110 | 202 |
| 101 → 310 | 111 | 191 |
| 102 → 301 | 108 | 202 |
| Bridge 201 used 2 times, 202 used 78 times | | |

The average delay improved significantly for packets from AP 101 to Node 310, and the overall system performance was superior in the system with fragment feedback enabled. This makes sense, since without this information the system tries to send more fragments to Bridge 202 only for them to be waiting in queues.

However, looking at the log files it was possible to determine that other improvements could be performed. The fragmentation feedback system uses a sampling method, thus when a timer expires or a link quality changes too much, a sample is taken at the Bridge and sent to the AP. During this particular test, despite the fact that the network was being slightly flooded with requests (we only used a Driver queue of 5 pending requests), many of the values transmitted from Bridge 202 were equal to zero. Therefore, averaging the number of fragments between the samples could improve the results.

# Part IV
# Conclusions & Future Work

# Chapter 10
# Conclusions

This chapter concludes this Thesis, by providing an overview of the main achivements.

## *10.1 Overview of research objectives*

Industrial communication networks following the master/slave paradigm usually provide limited-length data transfers between stations with guaranteed timings. These limitations are a natural consequence of their target market: control systems focused on low-latency and high reliability transmission of many small packets. However, there are several situations where these constraints make the integration of broader services very difficult or even impossible.

This Thesis addressed the design, implementation, test and validation of additional services over master/slave networks without loosing their native control timing characteristics. This was instantiated in factory automation and power-line communication networks.

## *10.2 TCP/IP integration with Profibus*

In the first context, multimedia applications over TCP/IP and Profibus-DP control applications were merged into a single Profibus network.

Traditionally TCP/IP over fieldbus has been seen with suspicious eyes from both the control-oriented experts, that do not expect much from TCP/IP, and from the multimedia-oriented field, where the modest bandwidth capabilities of the fieldbus networks look unpromising.

Nevertheless, by implementing a trouble-free dispatching method and a dual-stack architecture it was possible to run TCP/IP applications over a Profibus network preserving full backward compatibility with existing fieldbus stations and with advantages for both multimedia and control applications. With this architecture, multimedia applications can be placed directly in the factory floor without additional wiring, on the other hand, control applications can use TCP/IP as a flexible data communication method. The presented architecture enables end-user network-wide traffic isolation between the protocols.

The dispatcher system can be used like a traditional Profibus system where all high-priority packets are transmitted if so required by a particular deployment, but it can

also be configured to guarantee that additional traffic in one station does not have detrimental effects on the other stations deadlines, in fact improving the original Profibus protocol characteristics.

On the other hand, the implemented QoS mechanism not only guarantees network-wide traffic parameters but also enables to transparently in Profibus slave stations, overcoming the "lack of initiative" problem, as reported in this Thesis.

The proposed fragmentation mechanism overcomes the packet size limits of Profibus so that TCP/IP applications can use typical packet sizes available on Ethernet networks. Services like World Wide Web, File Transfer Protocol and even Voice connections are now easily deployed using traditional (unmodified) TCP/IP applications.

The system was validated via a factory automation field trial involving not only the merging of TCP/IP traffic and native control-oriented Profibus traffic, but also successfully including TCP/IP traffic in the real-time control-loop.

### *10.3 QoS aware end-to-end system over dual-level power-line communication network*

While the network context previously addressed focused on factory-floor networks, this second target domain, a power-line communication system, widens the geographic span of the network stations.

End-to-end geographically dispersed low-delay metering-oriented services are of paramount importance to Utility companies, however this deployment has been delayed due to lack of adequate technology combining performance and low cost. We believe that the REMPLI solution, based on a two-level master/slave PLC network, is a further step into the dissemination of such large-scale metering systems.

The REMPLI system overcomes the base master/slave network limitations and allows the transmission of very large information blocks between utility servers and metering devices at customers' premises, with QoS guarantees.

The support of large data blocks usually implies larger PDU headers that have a detrimental impact on small-length packet delays. However, the use of different headers for large and small packets guarantees a minimal impact on the fast response capabilities of the network for small packets. On the other end, the sliding-window mechanism enables forwarding of large packets via bridging stations with limited memory.

Bi-directional services are also available and the system can overcome the transmission path changes due to electromagnetic interferences and physical network reconfiguration. This was achieved through an efficient distributed-scheduling mechanism.

Due to the transmission medium, the system was designed with electrical energy as the main starting point, but can be used directly to measure any other utility product like water, heating, gas, etc. User-oriented services like security and remote control can also be integrated in the system.

These objectives are viable using nodes (the more numerous stations in the network) with modest capabilities that are effortlessly deployed and remotely configurable.

The tests (simulation-based) demonstrated that the proposed services are efficient and use limited resources, as initially envisaged.

# Chapter 11
# Future Work

This section gives an overview on future developments beyond this Thesis..

Both REMPLI and RFieldbus projects represent a significant integration work of very diverse technologies and solutions, aiming at extending existing communication infrastructures with additional functionalities and guaranteeing end-to-end quality-of-service (QoS) requirements.

While clearly developed in synergy with these two European-level efforts, this Thesis explored and instantiated several scientific and technological contributions on what we dubbed “Intermediate-Level Protocols”, that in some way were common to both projects. In this context, relevant inputs to the scientific and industrial communities resulting from this Thesis were clearly recognized.

Nevertheless, even more important than the direct benefits resulting from the research findings and engineering solutions provided within this Thesis for the projects stakeholders (e.g. companies involved in the two projects) is to identify how these can be leveraged for a wider spread use in the context of emerging paradigms in Information and Communication Technologies.

As we witness computers being increasingly embedded in the physical environments, scaling down in size and up in number, new research challenges emerge. The dawn of large-scale networked embedded systems will bring an undefined number of new cyber-physical applications that will improve our quality of life, some of them yet to be unveiled. What we can forecast is that the trend is for most of these applications to be largely geographically distributed and computing devices to be tightly embedded in their physical environments. Ambient intelligence, assisted living, home and building automation, monitoring/controlling large physical infrastructures such as roads, electrical and gas grids are just examples.

It is also accepted that the underlying large-scale network infrastructures will likely support many applications and services, most probably each of them with different quality-of-service requirements, e.g. depending on spatiotemporal issues. In this context, the research findings in this Thesis may be of extreme importance. Issues such as the provision of admission control and scheduling mechanisms can be used for achieving traffic differentiation, assigning packet priorities according to each application/task requirements. Packet fragmentation/defragmentation strategies will be of paramount importance when we think, for instance, on achieving a more pervasive Internet running into “smart objects” with limited processing, memory, energy and communication capabilities. The 6loWPAN protocol is just an example highlighting the need for subdividing (longer) IP packets to fit (smaller) IEEE 802.15.4 packets.

Conceiving QoS-aware Transport and Network level protocols for these large-scale networked embedded systems is also an enormous challenge. Could we apply the REMPLI Powerline Communications methodologies that were proposed in this Thesis to large-scale networked embedded systems? In fact, powerline communication systems and wireless sensor networks seem to have several commonalities, such as the unreliability of the links, low bandwidth, and network dynamics.

# References

ABB. (2008). "ABB tops Process Automation System market share worldwide (Press Release)."

Alves, M. (2003). "Real-Time Communications over Hybrid Wired/Wireless PROFIBUS-Based Networks."

Alves, M., Tovar, E., Marques, L., Behaeghel, S., and Nieuwenhuyse, K. (2003). "Engineering Hybrid Wired/Wireless Fieldbus Networks - a case study." In: *nd International Workshop on Real-Time LANs in the Internet Age (RTLIA03)*, Porto, Portugal.

AMRA. (2008). "AMRA - Automatic Meter Reading Association." *2008-02-12*, http://www.amra-intl.org.

Atkinson, R. (1994). "RFC 1626 - Default IP MTU for use over ATM AAL5."

Bumiller, G. (2001). "Network Management System for Telecommunication and Internet Application." In: *Power-Line Communications and Its Applications, 5th International Symposium on*, Malmö, Sweden, 129-134.

Burmann, C., Matthias, J., Bent, R., and Beine, K.-J. (2004). "Automation system and connecting apparatus for communication between two networks that use two different protocols with conversion between TCP/IP and PCP." U. Patent, ed., Phoenix Contact GmbH & Co. KG, USA.

Calandrini, D. (2003). "The Benefits of Fieldbus Technology in Power Plants." *InTech Protocol*.

DS 21906. (1990). "P-Net, MultiMaster, MultiNet Fieldbus for Sensor, Actuator and Controller Communications."

EtherCAT Technology Group. (2007). "EtherCAT - Ethernet for Control Automation Technology." *2007-08-29*, http://www.ethercat.org/.

Ferreira, L. (2005). "A Multiple Logical Ring Approach to Real time Wireless-enabled PROFIBUS Networks."

Ferreira, L., Machado, S., and Tovar, E. (2001). "Scheduling IP TRaffic in Multimedia Enabled PROFIBUS Networks." In: *8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2001)*, Antibes - Juan les Pins, France.

Fieldbus Foundation. (2005). "Study shows strong global growth for Fieldbus Foundation technology." In: *Fieldbus Facts Online*.

Fieldbus Foundation. (2006). "2005: Banner year for Fieldbus Foundation." In: *Fieldbus Facts Online*.

Fieldbus Foundation. (2007a). "Fieldbus Foundation Website." *2007-08-29*, http://www.fieldbus.org.

Fieldbus Foundation. (2007b). "Study results: FOUNDATION dominates process fieldbus protocols." In: *Fieldbus Facts Online*.

Gaderer, G., Loschmidt, P., Sauter, T., and Bumiller, G. (2006). "Investigations on fault tolerant clock synchronization within a powerline communication structure." In: *IEEE International Symposium on Power Line Communications and Its Applications (ISPLC 2006)*, Orlando, Florida, U.S.A.

Hoon, P. S., Huan, Y. R., Berge, J., and Sim, B. (2002). "Foundation/spl trade/ Fieldbus high speed Ethernet (HSE) " In: *International Symposium on Intelligent Control*, 777-782.

Hyperstone. (2007). "hyNet XS - Network Communication Controller." *2007-09-20*, http://www.hyperstone.com/products_hynet_xs_en,15474.html.

IDA. (2007). "IDA - Interface for Distributed Automation." *2007-08-29*, http://www2.modbus-ida.org/idagroup/.

Instrument Society of America. (1999). "IEC 61158 survives as technical specification." In: *InTech*.

Interbus. (2007). "INTERBUS Homepage " *2007-08-29*, http://www.interbusclub.com/.

IPP Hurray. (2002). "RFieldbus Manufacturing Field Trial Website." *2002-01-20*, http://www.hurray.isep.ipp.pt/rfpilot/.

Jain, R. (1990). "Error characteristics of fiber distributed data interface (FDDI)." *Communications, IEEE Transactions on*, 38(8), 1244-1252.

Kopetz, H. (1997). *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers.

Le Phu Do, Hrasnica, H., and Bumiller, G. (2005). "Investigation of MAC protocols for single frequency network technique applied in powerline communications." In: *Power Line Communications and Its Applications, 2005 International Symposium on*, Vancouver, Canada, 22-26.

Machado, S. (2006). "Parametrização e Análise de Funcionamento de Redes de Comunicação RFieldbus," Faculdade de Engenharia, Universidade do Porto, Porto.

Malcom, N., and Zhao, W. (1994). "The timed-token protocol for real-time communications." In: *Computer*, 45-41.

Marques, L., and Pacheco, F. (2007). "REMPLI Discrete Event Simulation System."

Modbus IDA. (2007). "Modbus IDA Website." *2007-08-29*, http://www.modbus.org.

Monforte, S., Alves, M., Tovar, E., and Vasques, F. (2000). "Designing Real-Time Systems Based on Mono-Master Profibus-DP Networks." In: *16th IFAC Workshop on Distributed Computer Control Systems (DCCS'2000)*, Sydney, Australia, 36-43.

MOST Cooperation. (2003). "MAMAC Specification - Rev 1.1 - 12/2003."

MOST Cooperation. (2006). "MOST Specification - Rev 2.5 - 10/2006."

MOST Cooperation. (2007). "MOST Cooperation Website." *2007-08-30*, http://www.mostcooperation.com.

ODVA. (2007). "ODVA Website." *2007-08-29*, http://www.odva.org/.

OMNeT++. (2007). "OMNeT++ Community Site." *2007-08-31*, http://www.omnetpp.org.

Pacheco, F., Lobashov, M., Pinho, M., and Pratl, G. (2005a). "A power line communication stack for metering, SCADA and large-scale domotic applications." In: *Power Line Communications and Its Applications, 2005 International Symposium on*, Vancouver, Canada, 61-65.

Pacheco, F., Pereira, N., Marques, B., Machado, S., Marques, L., Pinho, L. M., and Tovar, E. (2002). "Industrial Multimedia put into Practice." In: *7th CaberNet Radicals Workshop*, Bertinoro, Forlì, Italy.

Pacheco, F., Pinho, L. M., and Tovar, E. (2005b). "Queuing and routing in a hierarchical powerline communication system." In: *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, Catania, Italy, 59-66.

Pacheco, F., and Tovar, E. (2002). "User-interface Technologies for the Industrial Environment: Towards the Cyber-factory." In: *6th CaberNet Radicals Workshop*, Madeira Island.

Pereira, N., Kalogeras, A., Pacheco, F., and Tovar, E. (2001). "Supporting Internet Protocols in Master-Slave Fieldbus Networks." In: *4th IFAC FET Conference*, Nancy, France, 285-291.

Pokam, M. R., Guillaud, J.-F., and Michel, G. (1995). "Integrate multimedia in manufacturing networks using ATM." In: *20th Conference on Local Computer Networks*, Nancy, France, 280-288.

Postel, J. (1981). "RFC 791 - Internet Protocol."

Process Engineering. (2008). "Fieldbus needs skills to grow." In: *Process Engineering*.

PROFIBUS & PROFINET International. (2008a). "Number of PROFIBUS and PROFINET nodes very impressive (Press Release)."

PROFIBUS & PROFINET International. (2008b). "Profibus-DP." *2007-08-28*, http://www.profibus.com/pb/profibus/factory/.

PROFIBUS Nutzerorganisation e.V. (1992). "PROFIBUS Standard DIN 19245 part I and II."

PROFIBUS Nutzerorganisation e.V. (2008). "ProfiNET." *2007-08-29*, http://www.profinet.com/.

Prytz, G. (2008). "A performance analysis of EtherCAT and PROFINET IRT." In: *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008)*.

REMPLI Project. (2008). "REMPLI Deliverable 9.2 - Publishable Final Project Report."

RFieldbus Project. (2000). "D1.1 Requirements for the RFieldbus System."

RFieldbus Project. (2001a). "RFieldbus - Data Link Layer Specification." IST-1999-11316.

RFieldbus Project. (2001b). "RFieldbus - Higher Layer Specification." In: *IST-1999-11316*.

RFieldbus Project. (2002). "D1.5 Specification of Pilot Applications and Field Trials."

RFieldbus Project. (2008). "IST-1999-11316 - R-Fieldbus Project Website (Archived)." *2007-02-21*, http://www.hurray.isep.ipp.pt/rfieldbus.

Robert Bosch GmbH. (1991). "Bosch CAN Specification—Version 2.0, 1991."

Rowlands, I. (2007). "Demand Response in Ontario: Context and Research." In: *Spring Conference of the Peak Load Management Alliance*, Toronto, Ontario.

Schulz, V. (2008). "Cover Story - The Year In Question." In: *Control Engineering Asia*.

Sebeck, M., and Bumiller, G. (2000). "A Network Management System for Power-Line Communications and its Verification by Simulation." In: *4th International Symposium on Power Line Communications and Its Applications*, Limerik, England, 225-232.

Sempere, V. M., and Silvestre, J. (2003). "Multimedia Applications in Industrial Networks: Integration of Image Processing in Profibus." *IEEE Transactions on Industrial Electronics*, 50(3), 440-448.

Silvestre, J., Sempere, V. M., and Montava, M. A. (2002). "Optimization of the capacity of Profibus for the transmission of images and control traffic." In: *4th IEEE*

*International Workshop on Factory Communication Systems*, Västerås, Sweden, 133 - 140.

Simpson, W. (1994). "RFC 1661 - The Point-to-Point Protocol (PPP)."

SMSC. (2006). "Japan: MOST Interconnectivity Conference - MOST 50 - Double Bandwidth and Electrical Physical Layer."

Thomesse, J.-P. (2005). "Fieldbus Technology in Industrial Automation." *Proceedings of the IEEE*, 93(6), 1073-1101.

Tindell, K., and Burns, A. (1994). "Guaranteed Message Latencies For Distributed Safety-Critical Hard Real-Time Control Networks ", Dept. of Computer Science, University of York.

Tovar, E., and Vasques, F. (1999a). "Cycle time properties of the PROFIBUS timed-token protocol." *Computer Communications, Elsevier Science*, 22(13), 1206-1216.

Tovar, E., and Vasques, F. (1999b). "Real-Time Fieldbus Communications Using Profibus Networks." *IEEE Transactions on Industrial Electronics*, 46(6), 1241-1251.

Tovar, E., and Vasques, F. (2001). "Distributed computing for the factory-floor: a real-time approach using WorldFIP networks." *Computers in Industry*, 44(1), 11-31.

Tovar, E., Vasques, F., Pacheco, F., and Ferreira, L. (2001). "Industrial Multimedia over Factory-Floor Networks." In: *10th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2001)*, Vienna, Austria.

Treytl, A., Roberts, N., and Hancke, G. P. (2004). "Security Architecture for Power-line Metering System." In: *IEEE International Workshop on Factory Communication Systems, 2004*, Vienna, Austria.

Van Nieuwenhuyse, K., and Behaeghel, S. (2003). "Timing performance of a hybrid wired/wireless PROFIBUS-based network," Polytechnic Institute of Porto (ISEP/IPP)-Portugal, Porto.

Volz, M. (2001). "Technical Article: Quo vadis Layer 7?" In: *Industrial Ethernet Book*.

# List of Publications

During the time-span of this Thesis.

**Journals**

1: "*A Scalable and Efficient Approach to Obtain Measurements in CAN-based Control Systems*" (HURRAY-TR-061102) Andersson, B., Pereira, N., Elmenreich, N., Tovar, E., **Pacheco, F.**, Cruz, N. Published in IEEE Transactions on Industrial Informatics (TII), Volume 4, Issue 2, May 2008, pages 80-91

**Conference Proceedings**

2: "*A Complex Protocol Layer as a linux User-Space Process*" (HURRAY-TR-061006) Barros, A., **Pacheco, F.**, Pinho, L. Published in the Work in Progress session of the IEEE Symposium on Industrial Embedded Systems – IES'2006, Antibes Juan-Les-Pins, France, October 2006.

3: "*Queuing and Routing in a Hierarchical Powerline Communication System*" (HURRAY-TR-050901) **Pacheco, F.**, Pinho, L., Tovar, E. in 10th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA'05, Catania, Italy, September 2005, Volume 2, pp 59-66

4: "*A Power Line Communication Stack for Metering, SCADA and Large-scale Domotic Applications*" (HURRAY-TR-050402) **Pacheco, F.**, Lobashov, M., Pinho, L., Pratl, G. Published in the 2005 International Symposium on Power Line Communications and its Applications, Vancouver, Canadá, pp 61-65.

5: "*Characterizing the Timing Behaviour of Power-Line Communications by Means of Simulation*" (HURRAY-TR-0429) Marques, L., **Pacheco, F.**, Pinho, L. Published in Proceedings of the 2004 International Workshop on Real-Time Networks, Catania, Italy, pp. 81-84

6: "*Bringing Industrial Multimedia to the Factory-Floor: What is at stake with RFieldbus*" (HURRAY-TR-0311) Tovar, E., Pinho, L., **Pacheco, F.**, Alves, M. in Proceedings of the 5th IFAC International Conference on Fieldbus Systems and their Applications - FET03, Aveiro, Portugal, 7-8 July 2003, pp 131-138

7: "*Industrial Multimedia put into Practice*" (HURRAY-TR-0218) **Pacheco, F.**, Pereira, N., Marques, B., Machado, S., Marques, L., Pinho, L., Tovar, E. Published

in the Proceedings of the 7th CaberNet Radicals Workshop, Bertinoro, Forlì, Italy, October 2002.

8: "*Integration of TCP/IP and PROFIBUS Protocols*" (HURRAY-TR-0216) Pereira, N., Pinho, L., Prayati, A., Nikoloutsos, E., Kalogeras, A., Hintze, E., Adamczyk, H., Rauchhaupt, L., **Pacheco, F.** WIP Proceedings of the 4th IEEE International Workshop on Factory Communication Systems, Vasteras, Sweden, August 2002.

9: "*Workload Balancing in Distributed Virtual Reality Environments*" (HURRAY-TR-0212) Ditze, M., **Pacheco, F.**, Batista, B., Tovar, E., Altenbernd, P. Proceedings of the 1st Intl. Workshop on Real-Time LANs in the Internet Age, Satellite Event to the 14th Euromicro Conference on Real- Time Systems, Technical University of Vienna, Austria, June 18th, 2002, pp. 71-74

10: "*User-interface Technologies for the Industrial Environment: Towards the Cyber-factory*" (HURRAY-TR-0201) **Pacheco, F.**, Tovar, E. Published in the Proceedings of the 6th CaberNet Radicals Workshop, Funchal, Madeira Island, February 2002.

11: "*Supporting Internet Protocols in Master-Slave Fieldbus Networks*" (HURRAY-TR-0119), Kalogeras, A., Pereira, N., **Pacheco, F.**, Tovar, E. Published in the Proceedings of the 4th IFAC International Conference on Fieldbus Systems and Their Applications (FET'2001), Nancy, France, November 2001, pp. 260-266.

12: "*A High Performance Wireless Fieldbus in Industrial Multimedia-Related Environment*" (HURRAY-TR-0130) Tovar, E., Alves, M., **Pacheco, F.**, Ferreira, L., Pereira, N., Machado, S. Published in the Proceedings of the 4th CaberNet Plenary Workshop, Pisa, Italy, October 2001.

13: "*Industrial Multimedia over Factory-Floor Networks*" (HURRAY-TR-0124) Tovar, E., Vasques, F., **Pacheco, F.**, Ferreira, L. Published in the Proceedings of the 10th IFAC Symposium on Information Control Problems in Manufacturing (INCOM ’01), Vienna, Austria, September 20-22, 2001.

**Technical Reports (not published in conference proceedings or journals)**

14: "*Intranet UDP Message Format in the RFieldbus Multimedia Field Trial*" (HURRAY-BTR-0204) **Pacheco, F.**, Marques, B. R-Fieldbus Project Internal Report, September 2002

15: "*Manufacturing Automation Field Trial: October 2002 Report*" (HURRAY-BTR-0206) Pinho, L., **Pacheco, F.** R-Fieldbus Project Internal Report, October 2002

16: "*Specification of the Manufacturing Automation Field Trial*" (HURRAY-BTR-0131) Pinho, L., Francisco, V., **Pacheco, F.**, Alves, M., Tovar, E. R-Fieldbus Project Internal Report, December 2001

17: "*Data Link Layer Specification*" (HURRAY-TR-0117) Hähniche, J., Hammer, G., Heidel, R., Kalogeras, A., Adamczyk, H., Huang, T., Poschmann, A., Krogel, P., Rauchhaupt, L., Luttenbacher, J., Roether, K., **Pacheco, F.**, Alves, M., Tovar, E. R-Fieldbus Project Internal Report, April 2001

18: "*A Framework for Realistic Real-Time Walkthroughs in a VR Distributed Environment*" (HURRAY-TR-0125) **Pacheco, F.**, Tovar, E., Hansson, H., Altenbernd, P. White Paper, May 2001

19: "*Higher Layer Specification*" (HURRAY-TR-0116) Kalogeras, A., Huang, T., Rauchhaupt, L., Luttenbacher, J., Roether, K., **Pacheco, F.**, Alves, M., Tovar, E., Vasques, F. R-Fieldbus Project Internal Report, March 2001

20: "*A Survey of Techniques and Technologies for Web-Based Real-Time Interactive Rendering*" (HURRAY-TR-0108) Tovar, E., **Pacheco, F.** White Paper, March 2001

21: "*Basic Multimedia Functionalities for R-Fieldbus*" (HURRAY-BTR-0007) Kalogeras, A., Speckmeier, P., **Pacheco, F.**, Ferreira, L., Tovar, E. R-Fieldbus Project Internal Report, August 2000

22: "*D1.3 General System Architecture of the RFieldbus*" (HURRAY-TR-0016) Alves, M., Bangemann, T., Batista, B., , R., Ferreira, L., Hähniche, J., Hammer, G., Heidel, R., Kalivas, G., Kalogeras, A., Kapsalis, V., Koubias, S., , C., Kutschenreuter, M., Monforte, S., **Pacheco, F.**, Roether, K., Speckmeier, P., Tovar, E., Vasques, F. R-Fieldbus Project Deliverable, August 2000

23: "*D1.1 Requirements for the R-FIELDBUS system*" (HURRAY-TR-0002) Pipinis, H., Batista, B., Ferreira, L., **Pacheco, F.**, Hammer, G., Heidel, R., Kalivas, G., Kalogeras, A., Kapsalis, V., Karavasilis, C., Koubias, S., Koukourgiannis, A., Papadopoulos, G., Alves, M., Pinho, L., Rebakos, N., Speckmeier, P., Tovar, E., Vasques, F. R-Fieldbus Project Deliverable, April 2000

24: "*R-Fieldbus - Multimedia User Requirements*" (HURRAY-BTR-0004) **Pacheco, F.**, Tovar, E. R-Fieldbus Project Internal Report, March 2000

25: "*Comments on the R-Fieldbus User Requirements Questionnaire*" (HURRAY-TR-0008) Alves, M., Pinho, L., Ferreira, L., **Pacheco, F.**, Tovar, E., Vasques, F. R-Fieldbus Project Internal Report, March 2000

# Abbreviations & Clarification of Terms

| Abbreviation | Clarification |
|---|---|
| 802.11 | IEEE 802.11 Wireless Network Standards |
| AAL | ATM Adaptation Layer (Networks) |
| ACS | IP Admission Control and Scheduling (RFieldbus) |
| AGV | Automated Guided Vehicle |
| AL | Application Layer (Profibus) |
| AP | Access Point (REMPLI) |
| ARCNET | Attached Resource Computer Network (Networks) |
| ARP | Address Resolution Protocol (TCP/IP) |
| ASCII | American Standard Code for Information Interchange |
| ATM | Asynchronous Transfer Mode (Networks) |
| BE | Best Effort |
| Bridge | A kind of REMPLI station than interconnects LV and MV networks |
| COM | RS-232 Serial Communication Port (PC) |
| CRC | Cyclic Redundancy Check |
| CSRD | Cyclic Send and Request Data with Reply (Profibus) |
| *DA* | Destination Address (Profibus) |
| DCCS | Distributed Computer Control System |
| DeMux | De/Multiplexer (REMPLI) |
| DDLM | Direct Data Link Mapper (Profibus) |
| DFD | Data Flow Diagram |
| DLL (i) | Data Link Layer (Networks) |
| DLL (ii) | Dynamic Link Library (Microsoft Windows) |
| DMA | Direct Memory Access |
| DMS | Distribution Management System (Power distribution) |
| DP | Decentralized Peripherals (Profibus DP) |
| DPH | DP High Priority (RFieldbus) |
| DPL | DP Low Priority (RFieldbus) |
| DPRAM | Dual-Ported Random Access Memory |
| DSP | Digital Signal Processor |
| EDN | Electricity Distribution Network (Power distribution) |
| *FC* | Frame Control (Profibus) |
| *FCS* | Frame Checking Sequence (Profibus) |
| FDL | Fieldbus Data Link (Profibus) |
| FIFO | First In, First Out |

| Abbreviation | Clarification |
|---|---|
| FPGA | Field-Programmable Gate Array |
| FTP | File Transfer Protocol (TCP/IP) |
| FTT | Fragment Tracking Table (RFieldbus) |
| GB | Gigabyte: $10^9$ bytes |
| GiB | Gibibyte: $2^{30}$ bytes |
| GSM | Global System for Mobile communications (Mobile Telephony) |
| GUI | Graphical User Interface |
| HMD | Head Mounted Display |
| HP | High Priority |
| HSE | High Speed Ethernet (Fieldbus Network) |
| HTML | Hypertext Mark-up Language (Internet) |
| HTTP | Hypertext Transfer Protocol (Internet) |
| ICMP | Internet Control Message Protocol (TCP/IP) |
| IDA | Interface for Distributed Automation (Fieldbus Network) |
| IEEE | Institute of Electrical and Electronics Engineers |
| IGMP | Internet Group Management Protocol (TCP/IP) |
| I/O | Input/Output |
| IOCTL | Input/Output Control |
| IP | Internet Protocol (TCP/IP) |
| IPC | Inter-process Communication (Operating Systems) |
| IPH | IP High Priority (RFieldbus) |
| IPL | IP Low Priority (RFieldbus) |
| irDA | Infrared Data Association |
| IRP | I/O Request Packet (Microsoft Windows) |
| JPEG | Joint Photographic Experts Group image compression format |
| KB | Kilobyte: $10^3$ bytes, i.e. 1000 bytes |
| Kbps | Kilobit per second: $10^3$ bits per second |
| KiB | Kilibyte: $2^{10}$ bytes, i.e. 1024 bytes |
| LAN | Local Area Network |
| LBS | Link Base Station (RFiedlbus) |
| LE | Length of PDU (Profibus) |
| LLI | Lower Layer Interface (Profibus) |
| LP | Low Priority |
| LSB | Less Significant Bit |
| LV | Low Voltage (Power Distribution Grid) |
| MAC | Medium Access Control |
| MANET | Mobile Ad-hoc Network |
| MB | Megabyte: $10^6$ bytes |
| Mbps | Megabit per second: $10^6$ bits per second |
| MEER | Ministry of Energy and Energy Resources of Bulgaria |
| MiB | Mibibyte: $2^{20}$ bytes |
| MIB | Management Information Base (RFieldbus) |
| MoM | Mobility Master (RFieldbus) |
| MOST | Media Oriented Systems Transport (Multimedia Network) |
| MSB | Most Significant Bit |

| Abbreviation | Clarification |
|---|---|
| *MTU* | Maximum Transfer Unit length |
| MV | Medium Voltage (Power Distribution Grid) |
| NDIS | Network Device Interface (Microsoft Windows) |
| NetBIOS | Network Basic Input/Output System (Microsoft Windows) |
| NetBT | NetBIOS Over TCP/IP (Microsoft Windows) |
| NIC | Network Interface Card |
| NFS | Network File System (RFC 1094, RFC 1813, and RFC 3530) |
| NL | REMPLI Network Layer |
| NLI | REMPLI Network Layer Interface |
| NLIM | NLI Manager (REMPLI Transport Layer) |
| *NLAddr* | REMPLI Network Layer Address |
| *NLUnit* | REMPLI Network Layer Unit Identifier |
| NMS | Network Management System (REMPLI Network Layer) |
| Node | A kind of REMPLI station |
| *Node Address* | Used to identify nodes at RCI Interface at APs (REMPLI) |
| Packet | A block of data (used in higher-level protocols) |
| PC | Personal Computer |
| PCI | Peripheral Component Interconnect |
| PCMCIA | Personal Computer Memory Card International Association |
| PDA | Personal Data Assistant |
| PDU | Protocol Data Unit (used at lower-level protocols) |
| PHY | Physical Layer |
| PLC | Power Line Communication |
| PPP | Point to Point Protocol |
| PSTN | Plain Standard Telephone Network |
| QM | Queue Manager (REMPLI Transport Layer) |
| QoS | Quality of Service |
| RCI | REMPLI Communication Interface |
| RCIM | REMPLI TL RCI Manager |
| RE | RFieldbus Relationship Entity |
| RFC | Request for Comments |
| Rx | Receiving |
| *RUSN* | REMPLI Unique Serial Number |
| SCADA | Supervision Control and Data Acquisition. Can be local or distributed |
| *SA* | Source Address (Profibus) |
| *SAP* | Service Access Point (Profibus) |
| *SD* | Start Delimiter (Profibus) |
| SDA | Send Data With Acknowledge (Profibus) |
| SDN | Send Data with No Acknowledge (Profibus) |
| SFN | Single Frequency Network (REMPLI Network Layer) |
| SMP | Symmetrical Multiprocessing |
| SMS | Short Message Service (Mobile Telephony) |
| SRD | Send and Request Data with Reply (Profibus) |
| Station | A device with networking capabilities |
| TCP | Transmission Control Protocol (TCP/IP) |

| Abbreviation | Clarification |
|---|---|
| TCP/IP | The TCP/IP Protocol Stack, including UDP, IGMP and other protocols |
| $T_{DCY}$ | Dispatcher Cycle Time (RFieldbus) |
| TDI | Transport Driver Interface (Microsoft Windows) |
| $T_{DPL}$ | Usage Estimation Limit for RFieldbus DP Low Priority Traffic |
| $T_{IPH}$ | Usage Estimation Limit for RFieldbus IP High Priority Traffic |
| TL | Transport Layer (REMPLI) |
| TRM | Transport Route Manager (REMPLI Transport Layer) |
| $T_{TH}$ | Token Holding Time (Profibus) |
| $T_{TR}$ | Token Target Rotation Time (Profibus) |
| Tx | Transmission |
| UDP | User Datagram Protocol (TCP/IP) |
| USB | Universal Serial Bus |
| VPN | Virtual Private Network |
| WAN | Wide Area Network |
| WDM | Windows Driver Model (Microsoft Windows) |
| WWW | World Wide Web |