UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The Design of a Software Architectural Framework for Tunnelling Metering Protocols over TCP/IP and Low Bandwidth Packet Switched Networks with Support for Proprietary Addressing

by

Albert Fredrich Johannes von Gordon

Submitted in partial fulfilment of the requirements for the degree
Master of Engineering (Computer Engineering)
in the
Faculty of Engineering, the Built Environment and Information Technology
UNIVERSITY OF PRETORIA

March 2007

ACKNOWLEDGEMENTS

A special word of thanks to my supervisor, Prof. Gerhard Hancke of the Department of Electrical, Electronic and Computer Engineering at the University of Pretoria for his support and guidance throughout this research. I would also like to thank him for providing me with the opportunity to visit the Vienna University of Technology in Austria where I was able to conduct a large part of the research presented in this dissertation.

I would like to thank Albert Treytl and Aleksey Bratoukhine of the Institute of Computer Technology at the Vienna University of Technology for providing guidance and advice during my research there.

I would also like to thank e-Logics and Spoornet, two companies that gave me access to the metering hardware required to implement the proposed research in this dissertation and conduct experiments.

An extra special thanks to my beloved wife Susan, who supported and motivated me throughout the research process. I would not have been able to complete this research without her constant support.

The Design of a Software Architectural Framework for Tunnelling Metering Protocols over TCP/IP and Low Bandwidth Packet Switched Networks with Support for Proprietary Addressing

by

Albert Fredrich Johannes von Gordon

Supervisor:	Prof. G.P. Hancke
Department:	Electrical, Electronic and Computer Engineering
Degree:	Master of Engineering (Computer Engineering)

KEY TERMS

protocol tunnelling; Fieldbus systems; power line communication; distributed driver architecture; application protocol data units; metering protocols; remote meters; quality of service; measurement values; protocol driver

ABSTRACT

This document discusses the concept of drivers implemented within the context of the REMPLI[1] network. The process image approach and the tunnelling approach are presented and reasoning is given why the tunnelling approach is preferred. Each of the drivers implemented is associated with a specific metering protocol. This document further discusses the general architecture of such a driver structure. The generic software architecture serves as a framework for integrating serial communication based metering protocols over packet-orientated remote networks and meters, by tunnelling the protocol data units to the remote meters. Principally each Protocol Driver consists of three parts, one part situated at the Application Server, one at the Access Point and one at the Node. This document then gives a description of the general driver structure within the REMPLI network and briefly explains the functions of all the modules contained within the driver

[1] REMPLI (Real-Time Energy Management over Power line and Internet) see section 1.8.

structure. An example is used to show how these modules, which make up the software architecture of the Protocol Driver, are used to send an application generated request from the Application Server to the Metering Equipment and sending the response back from the remote Metering Equipment to the Application Server. This dissertation further discusses the need for address translation within the REMPLI network and the need to restrict access to meters by using these addresses and an access control list. This document also discusses the need for a "Keep-alive" signalling scheme, if supported by the underlying protocol and gives a general concept as to how it should be implemented. The role of an Optimization Module is also discussed for low bandwidth networks by means of an M-Bus example. Finally the M-Bus protocol driver implementation is discussed. The results achieved are presented, showing that the driver architecture can successfully be used to tunnel the M-Bus protocol to remote meters, provided the underlying network conforms to the quality of service requirements determined by the implemented metering protocol.

The work proposed in this document started off as part of the REMPLI project by the REMPLI team but was completed independently.

Die Ontwerp van ‘n Sagteware Argitektuur vir die Tonnel van Meterprotokolle oor TCP/IP en Lae-Bandwydte Pakket-Geskakelde Netwerke wat Eiesoortige Addresse Ondersteun

deur

Albert Fredrich Johannes von Gordon

Studieleier:	Prof. G.P. Hancke
Departement:	Elektriese, Elektroniese en Rekenaar-Ingenieurswese
Graad:	Meester van Ingenieurswese (Rekenaar-Ingenieurswese)

SLEUTELTERME

tonnel van protokol; Fieldbus stelsel; kraglyn-kommunikasie; verspreide aandrywer argitektuur; protokol data eenhede; meter-protokolle; verspreide meters; kwaliteit van diens; meetwaardes; protokolaandrywer

OPSOMMING

Hierdie dokument bespreek die konsep van sagteware-aandrywers binne die konteks van die REMPLI[2] netwerk. Die sistematiese berging van lesings asook die tonnel van data deur middel van meterprotokolle as benadering vir die opname van lesings word bespreek. ‘n Verduideliking word verskaf waarom die tonnel-benadering verkies word. Elk van die geïmplementeerde sagteware-aandrywers word gekoppel aan ‘n spesifieke meterprotokol. Voorts word ook die algemene sagteware argitektuur van so ‘n aandrywer bespreek. Die generiese sagteware argitektuur dien as ‘n raamwerk vir die integrasie van seriekommunikasie-gebaseerde meterprotokolle met meters wat oor pakket-geskakelde wye-area netwerke lesings neem. Dit word bewerkstellig deur die meter se protokol-data-eenhede te tonnel na meters wat oor lang afstande geleë is. In hoofsaak bestaan elke Protokolaandrywer uit drie dele, een module geplaas by die Toepassingsbediener, een

[2] REMPLI (Real-Time Energy Management over Power line and Internet) sien afdeling 1.8.

module by die Toegangsbediener en een module by die Nodus. Die dokument beskryf voorts die algemene sagteware aandrywerstruktuur binne die konteks van die REMPLI netwerk. Dit verduidelik ook kortliks die funksionele eenhede waaruit die aandrywer bestaan. By wyse van ‘n voorbeeld word die kommunikasieproses beskryf. Dit word gedoen deur van die funksionele eenhede waaruit die Protokolaandrywer bestaan gebruik te maak. Die voorbeeld beskryf hoe ‘n Toepassingsbediener ‘n versoek wat deur aanvraag gegenereer word, na meters stuur deur van die Protokolaandrywer gebruik te maak. Die meter stuur dan die lesings terug na die Toepassingsbediener. Hierdie verhandeling bespreek ook verder die noodsaaklikheid om voorsiening te maak vir adres-omskakeling binne die REMPLI konteks. Toegangsbeheer deur die gebruik van hierdie adresse sowel as ‘n toegangsbeheer lys word bespreek. Die noodsaaklikheid van ‘n meganisme om die kommunikasiekanaal oop te hou word ook bespreek om te verseker dat die Toepassingsbediener die geleentheid het om die verlangde waardes vanaf die meter-toerusting te verkry. Die rol wat ‘n optimeringsmodule speel in die gebruik van ‘n lae-bandwydte netwerk aan die hand van ‘n M-Bus voorbeeld word ook bespreek. Ten slotte word die M-Bus sagteware-aandrywer se implementering bespreek. Die resultate van die geïmplementeerde stelsel toon aan dat die sagteware-argitektuur suksesvol is om die M-Bus protokol na M-Bus meters oor ‘n wye-area netwerk te tonnel. Verder word ook getoon dat die sukses van die implementering afhanklik is van die kwaliteit wat verskaf word deur die onderliggende netwerk.

Die voorgestelde navorsing het begin as deel van die REMPLI projek en REMPLI span, maar is later onafhanklik voltooi.

List of Abbreviations

A	Address Field
APDU	Application Protocol Data Unit
C	Control Field
CI	Control Information Field
CS	Checksum
De/Mux	Multiplexer/De-multiplexer
FAN	Field Area Networks
GAN	Global Area Network
GSM	Global System for Mobile Communication
GPRS	General Packet Radio Service
HVAC	Heating, Ventilation, Air Conditioning
IDT	Inter-Domain Transactions
IP	Internet Protocol
L	Length Field
LAN	Local Area Network
LSB	Least Significant Bit
MAC	Media Access Control
M-Bus	Meter Bus
MLR	Multi-Logical Ring
OSI	Open Systems Interconnection
OSI-RM	Open Systems Interconnection – Reference Model
PDU	Protocol Data Unit
PLC	Power line Communication
QoS	Quality of Service
REMPLI	Real-time Energy Management via Power lines and Internet
REQ/RES	Request/Response
RNA	REMPLI Node Address
RTT	Round Trip Time
SCADA	Supervisory Control And Data Acquisition
SND/CON	Send/Confirm
SNMP	Simple Network Management Protocol
SNMP-MIB	Simple Network Management Protocol Management Information Base

TCP	Transmission Control Protocol
TCP/IP	Internet Protocol over Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
XML	Extensible Markup Language

Table of Contents

Chapter 1

1 INTRODUCTION

In the REMPLI (Real-time Energy Management via Power lines and Internet) network, protocol tunnelling is implemented in order to transfer data between Application Servers and Metering Equipment, [1]. This protocol tunnelling scheme requires the use of protocol drivers in order to transfer the data from the Application Server to the Metering Equipment. For each protocol supported by the REMPLI system, a corresponding driver is required to tunnel the data contained in this protocol over the IP (Internet Protocol) network to the access point and then tunnel the data over the PLC/GPRS (Power line Communication/General Packet Radio Service) network to the Metering Equipment situated at a REMPLI Node. To make this tunnelling process transparent to the Application Server, a triplet of distributed drivers are required, one situated at the Application Server, one situated at the Access Point and one at the REMPLI Node. These drivers are known as REMPLI Protocol Drivers and they should satisfy the requirement of supporting existing Application Server software and existing metering and control equipment. Another goal of the driver design is to support future applications and allow for future scalability, which is currently lacking in internet protocols for fieldbus systems, [2].

The intention of this document is to give an overview of the functionality offered by the general REMPLI Protocol Driver and to give a general structure with basic functions that each protocol will require, to implement effective and efficient transmission of the protocol data over the REMPLI network. Further more the distributed REMPLI Protocol Driver is implemented for the M-Bus (Meter Bus) metering protocol, [3] with results obtained as to its effectiveness.

The background of fieldbus and metering systems is given and the contribution that the work done in this document is given within the context of the current research being conducted within this field of fieldbus systems and metering networks.

1.1 SCOPE

The scope of this research is the development of a software architectural framework for transparent tunnelling of metering protocols over remote hybrid networks. The hybrid networks are limited to two different addressing schemes and include proprietary communication systems. This research focused on TCP/IP and PLC/GPRS networks for gathering of readings by remote meters.

Many fieldbus systems originated from proprietary development projects, due to a lack of international standardisation at the time of development, in order to satisfy specific needs, [4]. The relevant fieldbus systems available and in use on the market at present have been standardized on an international level. Some of the fieldbus systems were specifically developed to acquire data over remote networks including TCP/IP based networks. However these fieldbus systems can often only be used in very specific environments for limited purposes, [4]. There is still a great need in fieldbus systems, and metering systems for unification. Unification can be accomplished by the implementation of a framework that is independent of fieldbus and metering systems to extend the functionality of both fieldbus and metering systems by allowing existing metering protocols to be integrated into proprietary fieldbus systems in a transparent manner.

1.2 MOTIVATION

Originally fieldbus systems were introduced as a means of implementing distributed control into process and manufacturing automation systems and later on for building automation. Fieldbus systems are industrial communication networks without any specific application context attached. Therefore it is not suitable to use the data representation model within a fieldbus system, but rather to concentrate on the model represented by the underlying standards used within the metering and control systems.

The widely used standards contained within fieldbus systems such as Profibus, Lonworks, CAN or Modbus offer solutions for metering and control systems, but none of these systems have been specifically designed for implementing control and metering systems over large networks, spanning great distances, for example Wide Area Networks. In order to implement these systems for remote metering or control, additional data conversion is required in order to manipulate the data. This could lead to complex data management, depending on the system being implemented.

Standard metering and control protocols on the other hand often deal with simplistic network configurations, containing a single bus with a master and multiple slaves on the bus. This configuration poses a challenge in implementing more complex systems without retranslating the data packets on each level of communication from the metering or control equipment right up to data capturing devices like SCADA (Supervisory Control And Data Acquisition) capturing systems. This is one of the problems faced within the REMPLI system, [1]. The addressing schemes implemented by these metering and control systems are not adequate for the multi-structural addressing required within the REMPLI system.

Metering and SCADA system's devices on the lower level, supported by applications on the top layer use metering and control protocols for communication and data transmission. This provides the second reason for focussing on the data representation model used in these systems. Thus it is often a requirement of the field-test environment to support IEC 60870-5-101, EN 62056-21 and M-Bus, because the low level devices communicate to the upper level devices using these three common standards. Because this document will focus specifically on the M-Bus protocol it will also only focus on the data representation model of the M-Bus protocol.

All of the protocols implemented in the REMPLI system are closely related to each other, because they are often derived from the same companion standard. In particular most of these protocols share the same communication mechanisms for metering and control devices on different levels of the OSI (Open Systems Interconnection) model, in particular the Physical and Data link levels. M-Bus also define the Application layer. Some of the OSI layers are the same for the standards and some are more specific for each protocol. As an example the M-Bus protocol uses the Data link layer mechanisms specified in IEC 60870-5-101.

Thus in order to implement a remote metering and or control system, the addressing mechanism has to be defined and the underlying protocol specific data representation model supported for communicating between the application and the metering equipment. This will allow the REMPLI system to provide for the demand side requirements while also supporting existing metering and control systems by transparently communicating between the upper level applications and the lower level metering equipment.

The first part of the problem addressed by this research is the lack of metering implementations that allow for the acquisition of measurements from remote meters over hybrid networks, in particular TCP/IP and power line/GPRS networks. The second part of the problem addressed in this research is the lack in transparency when implementing these metering protocols in existing fieldbus systems, causing limited scalability.

1.3 OBJECTIVES

The objective of this research is to develop a software architectural framework that tunnels existing metering protocols over hybrid networks with a focus on TCP/IP and PLC/GPRS networks, with the following characteristics:

1. The framework must enable access and acquisition of data from remote meters over a TPC/IP and PLC/GPRS based network.
2. The framework must provide a distributed driver architecture for containing additional information and management functionality required to tunnel metering protocols to remote metering equipment.
3. The framework must enable for the integration of existing connection-oriented standard metering protocols into a remote packet orientated network.
4. The framework must integrate metering protocols in a packet oriented network in a transparent manner.
5. The framework must enable multiple concurrent meter protocols to be tunnelled to remote meters.
6. The framework must provide for the management of the Data link layer protocol data units of the metering protocols.

7. The framework must make optimal use of the low bandwidth provided by the power line communication.
8. The framework must provide a mechanism for address translation.
9. The framework must provide access control functionality.

The objectives for the software architectural framework to be developed can be summarized as follows: *transparency*, *scalability*, *interoperability*, *data integrity* and *security*.

1.4 CONTRIBUTION

This research provides two significant contributions to the body of knowledge in the field of remote metering and fieldbus systems. The first is a software architectural framework for transparently tunnelling existing connection-orientated metering protocols over remote packet orientated networks. The suggested framework also makes provision for the addressing requirements and the Data link layer communication management of the metering protocols. Provided that the underlying network QoS (Quality of Service) is provided the functionality of metering systems and fieldbus systems can be extended by integrating existing metering systems into fieldbus systems, without the need to develop an entirely new metering system. Many proposals have been made on how to extend existing fieldbus systems to integrate IP or GPRS into the fieldbus system. These suggestions however are focused on the existing fieldbus systems, where as the framework suggested in this research focuses on the existing metering equipment and how to extend them for use in fieldbus systems.

The second contribution to the body of knowledge is the successful implementation of the suggested software framework on the M-Bus protocol and successful expanding the current capability of the M-bus protocol to retrieve measurement values over a wide area network by traversing a TCP/IP and GPRS/PLC based network. Currently many proposals focus on wired/wireless extensions of fieldbus systems, [5], [6], [7]. The framework suggested in this research allows for the extension of fieldbus systems by using power line communication and or any proprietary communication system, by supplying the required addressing needs.

Another significant contribution by this research is the scalability provided by the framework. If a new metering system with a new metering protocol is required in the fieldbus system, then a protocol specific distributed driver is developed for the new protocol and integrated into the existing fieldbus system. This expandability of the framework significantly reduce implementation costs once the fieldbus system is in place, and uses the distributed driver architecture to tunnel metering protocols over packet oriented networks. The suggested framework is also not limited to fieldbus systems and can be implemented independently to access remote meters.

By applying the distributed driver architecture given in this document it is possible to convert current serial based metering systems into remote metering systems. Provided that the metering protocol meet the addressing requirements of the underlying network and the underlying network provides the required QoS demands for the metering protocol.

The proposed research combines the fields of remote metering with the area of fieldbus systems to enable the reading of values obtained from standard metering implementations. The application software for the metering equipment is able to communicate transparently with remote meters over TCP/IP, GPRS or PLC based networks, through the use of distributed protocol drivers.

1.5 RESEARCH METHODOLOGY

A key component in the design of a software framework for tunnelling connection-orientated metering protocols over packet-orientated networks is a thorough understanding of existing connection-orientated metering protocols and fieldbus systems as well as the underlying communication systems that each of them implement. Therefore a thorough literature study was done to understand current systems and the limitations of currently existing systems. The literature study was also used to obtain the key factors and requirements that would influence the design of a software architecture for tunnelling existing metering protocols over packet-oriented remote networks. The developed software architecture was implemented using the M-Bus metering protocol stack and deployed for acquiring measurement values. The implementation was used to verify its design and to determine key requirements for implementing the developed architecture, which include the QoS and the number of remote meters that can be accessed using the suggested framework. The limitations of the developed architecture are given and key factors to consider when implementing it. Finally future developments are suggested for improving the suggested software architecture and further analysis required on implementing the developed software architecture in a full-scale fieldbus systems.

1.6 BACKGROUND

The proposed distributed driver given as a solution for tunnelling existing fieldbus metering protocols across different networks, results in the use of a field-level network on which the drivers operate. On a physical level the combination of fieldbus networks with packet switched networks used to implement remote data acquisition using existing fieldbus devices results in the term field-level networks. This section gives the definition of the different networks used in a field-level network as well as the different communication mechanisms and limitations associated with each of the specified networks. The final design of the distributed driver must incorporate all of the features and mechanisms associated with the different communication networks in order to transparently communicate across the different types of networks within the field-level network.

Communication networks are classified by the areas that they span. These include:

- Global Area Networks (GAN’s), which span over multiple continents around the world and are known as world-wide networks;
- Wide Area Networks (WAN’s) , which cover large parts of continents or large land masses;
- Local Area Networks (LAN’s), which are limited to specific geographical areas within several kilometres.

Networks are used for connecting distributed devices, which can then be used to share information and automate several processes. Certain network types and topologies are best suited to certain processes and distributed devices. As an example, devices utilizing LAN’s include: Computers, Terminals, Micro Controllers and Measuring Equipment. These communicating devices can best utilize the functionality provided by a network, due to the following reasons:

- Protocols that are supported by the communicating devices and supported by the network topologies.
- The bandwidth provided by the network topology.
- The computing ability of the communicating devices.
- The volumes and type of data that needs to be transmitted between different distributed devices.

LAN's are configured in different topologies. The main topologies used for local area networks are: Bus Topologies, Ring Topology and the Star Topology as shown in figure 1.1.

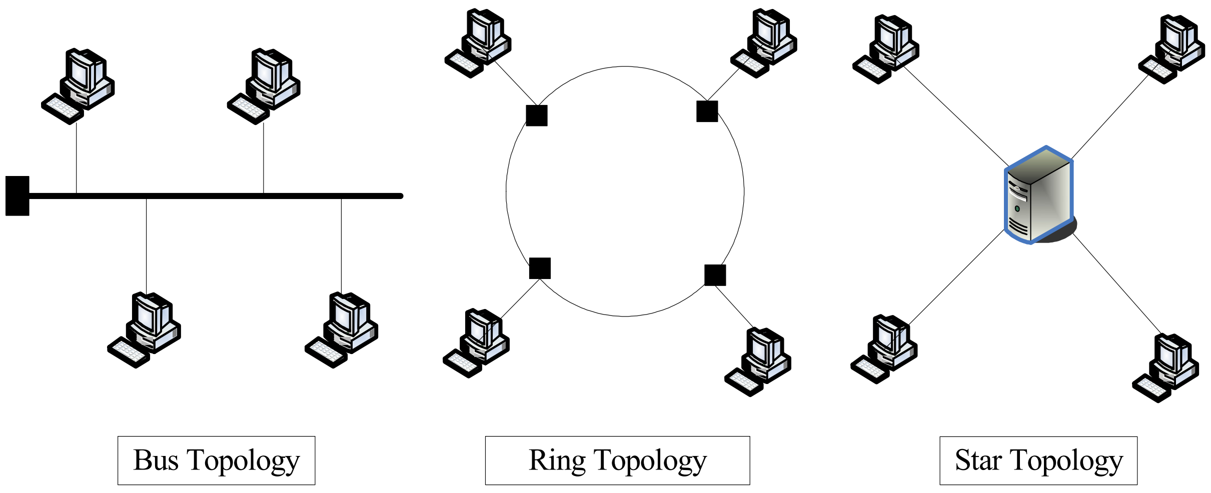


Figure 1.1. Different Local Area Network Topologies.

In a bus topology, every device is connected to every other device via an appropriate hardware interface through a common line known as a bus. In full-duplex mode data is allowed to be transmitted onto the bus and received from the bus simultaneously. In half-duplex mode data is only allowed to be transmitted onto the bus, or only allowed to be received from the bus. A transmission from any device propagates the entire length of the bus and can be received by all devices connected to the bus. Terminators at the end of the bus absorb any signal that reaches it and thus removes the signal from the bus.

Transmission of data takes place, either serially (one bit after the other) or parallel where multiple data bits are transmitted simultaneously, each on a separate data line.

In the ring topology repeaters receive and retransmit data in a closed loop. Transmission takes place in a unilateral manner, in one direction only (clockwise or counter clockwise). As the data is passed from one device to the other the device for which the data is intended detects it and copies it. Once the data has completed the circular transmission and arrives back at the original source it is removed from the network. Only one station may send data at a time and a medium access control mechanism is required to manage this.

In the star topology, each device is directly connected to a common central device. Either the central device broadcasts data to all connected devices, which results in a logical bus topology, or all incoming data is buffered and then redirected to the intended recipient through the outgoing link.

Many forms of serial bus systems exist. The following diagram gives an overview of the most prominent serial bus system's medium access and transmission techniques, [3].

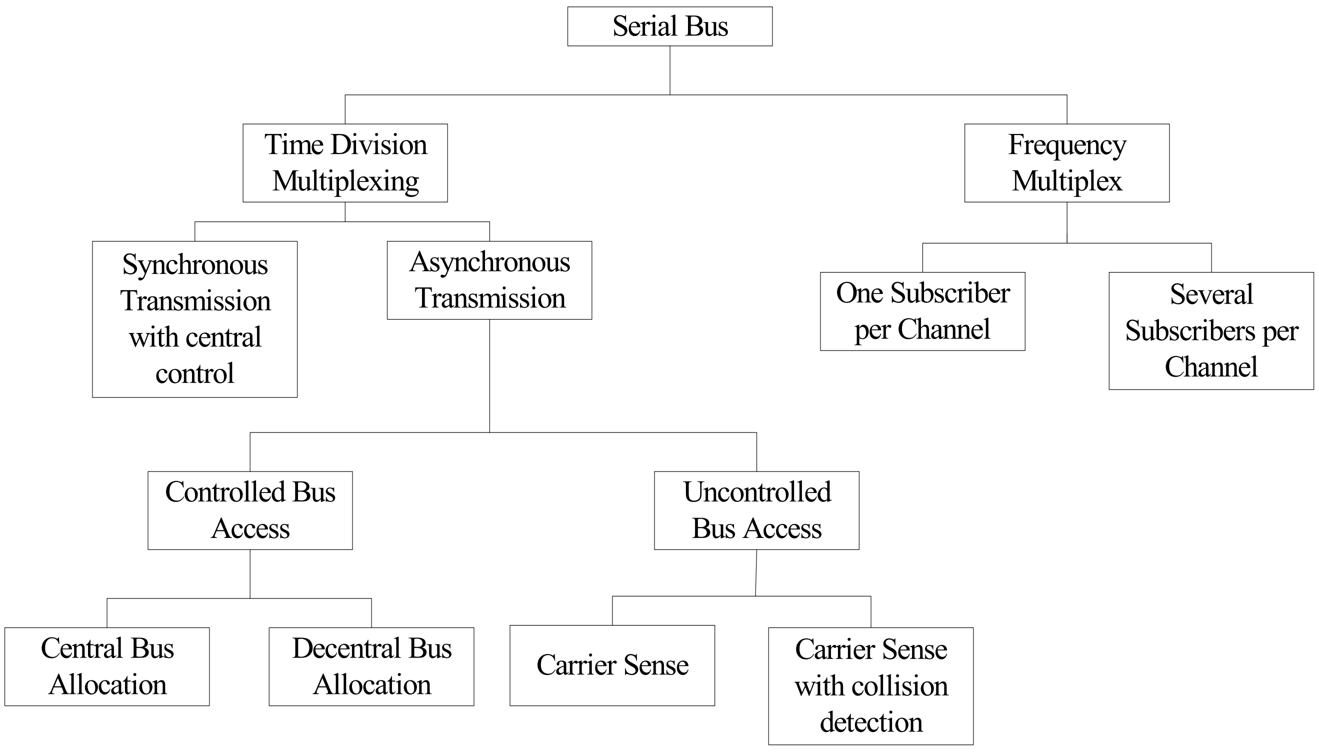

Serial Bus
Time Division Multiplexing
Frequency Multiplex
Synchronous Transmission with central control
Asynchronous Transmission
One Subscriber per Channel
Several Subscribers per Channel
Controlled Bus Access
Uncontrolled Bus Access
Central Bus Allocation
Decentral Bus Allocation
Carrier Sense
Carrier Sense with collision detection

Figure 1.2. The classification of serial bus systems according to medium access and transmission techniques.

The nature of a bus system and the fact that multiple devices share the same transmission medium and capacity requires a means of controlling access to the transmission medium. This is achieved through access techniques such as Medium Access Control (MAC). These techniques must ensure that only a single station transmits at any given time, thus avoiding collisions. These access techniques must ensure that each station is given a minimum timeframe in which it can transmit.

A fundamental requirement for communicating on a serial bus architecture is synchronization. The receiver of data must know at what rate to sample data and the sender of data must know at what rate to send data in order for the receiver to be able to receive the data correctly, [8]. Synchronization can thus be seen as the coordination, in time, between the communication devices on a transmission line. In order to achieve synchronous communication two techniques are used: Synchronous transmission and Asynchronous transmission.

During synchronous transmission a block of bits is transmitted in a steady stream. There are no start or stop bits within this stream, [8]. Over long distances these bits may drift. In order to allow the receiver to correctly receive the stream of bits the streams between sender and receiver must be synchronized. By using a clock signal, either on a separate line, or by sending regular clock pulses on the line as a clock this synchronization can be achieved. Another alternative is to encode the data stream to contain the clock signal within the data stream sent over the transmission line if the signal is of digital nature. If an analog signal is transmitted the carrier frequency can be used to synchronize the send and receiver, based on the phase of the signal. Once the individual bits in each stream are synchronized each block of data also needs to be synchronized. This is achieved by preceding each data block with a predetermined, fixed bit pattern (preamble) and following each data block by another predetermined fixed bit pattern (postamble). This data stream with its preamble and postamble and control information is known as a frame, [8].

During asynchronous transmission the problem of maintaining a shared clock signal is bypassed by only allowing for short characters. Timing only needs to be maintained within each character. When no data is transmitted on the line, the line is in an idle state which is usually binary 1. A start character 0 is used to indicate the start of a character. The character is transmitted with the Least Significant Bit (LSB) first, followed by a parity bit. The parity bit is set so that the number of ones in the character is either even or odd. Finally a binary 1 is used after the character to indicate the end of the current character.

The main difference between these two synchronization techniques is that asynchronous transmission is easier to achieve, as no clock signal is required, but more overhead is generated. To implement synchronous transmission, less overhead is generated and utilizes the transmission line more effectively. However, the cost of implementing a synchronous transmission system is more costly due to the complexities involved in implementing this scheme.

A protocol is used to facilitate the communication between different entities connected to each other on a network. The protocol determines what is communicated and how it is communicated and must conform to a mutual convention. A protocol consists of three main elements, [8]:

- Syntax: Data format and signal strength.
- Semantics: Control information for coordinating and error handling.
- Timing: Transmission speed matching and sequencing.

Due to its characteristics the serial bus LAN topology is often the most cost effective LAN topology to implement when implementing a distributed network. Metering devices that are used to read data from distributed metering devices is one example for which the serial bus LAN topology is best suited due to the following:

- The low computing power that is often present at such metering devices, limiting these devices to simplistic protocols and very limited data management control.
- Low cost of implementation. Due to the simplistic nature of a serial bus network the cost of implementing such a system is less costly.
- Adequate bandwidth provided by the serial bus network to transmit the required loads of data over the network.
- High degree of transmission integrity.

- Capable of transmitting data over long distances.

1.7 FIELDBUS SYSTEMS

The term fieldbus is used to describe a digital communications network. This network is used to connect remote field devices, such as metering equipment, controllers and sensors. The fieldbus network supports digital, bi-directional communication on a serial-bus LAN topology.

Each field device has limited computing capabilities, enabling it to perform certain tasks on its own, such as:

- Control
- Communication Functions
- Diagnostic functions
- Maintenance

The computing ability at the field device level allows for distributed control networks, which allows for the remote accessing of field devices and inter-device communication.

The advantages of implementing a fieldbus network include the following:

- Improved quality of service, digital is more accurate than the analog networks it replaces.
- Lower cost of implementation.
- Improved efficiency.

The advantages mentioned above are due to the digital nature of the network, computing capability of the field devices, communication capabilities of the field devices and the self maintenance and diagnostic capabilities situated at the field device, allowing for better maintenance and control of the field devices while at the same time reducing the amount of resources required to perform these maintenance and control tasks. Another advantage of the implementing field devices with computing capability is that these devices are more flexible and can be utilized to perform several tasks; one device could be used to perform metering tasks, as well as control tasks etc.

Fieldbus systems originated during the early 1970’s and the development of fieldbus systems rapidly increased during the 1980’s as the potential of remote automation by means of fieldbus systems became apparent [4]. These fieldbus systems were primarily developed by corporations within the automation industry, but the acceptance rate of these commercially developed fieldbus systems were not high enough and international standardization was required for higher levels of uptake within the automotive industry. During the late 1980’s the IEC made an effort to draw up international standards for fieldbus systems. Development of fieldbus systems continued, mainly in Europe, with two main fieldbus systems came to prevalence during this time: FIP from France and PROFIBUS from Germany. Both these systems were standardized at national level and both were submitted to the IEC for international standardization. These two fieldbus systems are fundamentally different in their approach: PROFIBUS focused on distributed control and was based on an object-oriented vertical communication client/server model. FIP on the other hand implemented a real-time control scheme within a centrally base producer-consumer horizontal communication model. Even though these fieldbus systems were different they complemented each other and a single standard was required combining the advantages of both systems, which was proposed in [9].

The next development in fieldbus systems included the development of an international standard for fieldbus systems at the hardware level as IEC 61158-2. However at the Data link layer no consensus could be reached, which lead to the CENELEC group of standards being adopted which took up all the national fieldbus standards as given in [4], and included all the protocols given in table 2.1.

Table 1.1. The national protocols taken up in the CENELEC group of fieldbus protocols as given in [4].

CELENEC Standard	IEC Standard	Commercial Fieldbus System
EN 50170 – 1	IS 61158 Type 4	P-Net
EN 50170 – 2	IS 61158 Type 1/3/10	PROFIBUS
EN 50170 – 3	IS 61158 Type 1/7	WorldFIP
EN 50170 – A1	IS 61158 Type 1/9	Foundation Fieldbus
EN 50170 – A2	IS 61158 Type 1/3	PROFIBUS-PA
EN 50170 – A3	IS 61158 Type 2	ControlNet
EN 50254 – 2	IS 61158 Type 8	INTERBUS
EN 50254 – 3	IS 61158 Type 3	PROFIBUS-DP
EN 50254 – 4	IS 61158 Type 7	WorldFIP (FIPIO)
EN 50325 – 2	IS 62026-3	DeviceNet
EN 50325 – 3	IS 62026-5	SDS
EN 50325 – 4		CANOpen
EN 50295 – 2	IS 62026-2	AS-Interface

The IEC 61158 standard provided the international standard that allowed for devices to be interconnected at the hardware level. However many of the differences between these protocols are contained within the user layer.

Originally fieldbus systems were incorporated within factory environments containing only a couple of field devices. These devices were connected to serial bus networks.

The increased use of Ethernet networks in the automation environment lead to the development of fieldbus process automation and control systems that utilize these Ethernet network topologies. PROFINET was developed as a industrial Ethernet communication suite, to take advantage of the widely available Ethernet networks. One of the most widely used protocols based on the RS485 physical layer standard is PROFIBUS. The PROFIBUS fieldbus network standard developed into 3 different variants [10], these variants are listed below:

- PROFIBUS-DP : PROFIBUS-DP is designed for communication between automation and control systems within a distributed environment of I/O devices. It is designed to be optimized for speed at a low cost, and focuses on the devices level.

- PROFIBUS-PA : PROFIBUS-PA is designed for process automation. In this configuration of the PROFIBUS network, communication takes place over 2 lines. These two lines are also used to provide power to the distributed devices.

- PROFIBUS-FMS : The PROFIBUS-FMS protocol is designed for more general purpose use at all levels of the communication stack. The most flexible protocol of the three variants.

These extensions on the original PROFIBUS standard catered for the ever widening fields, in which these fieldbus systems were used.

The PROFIBUS-DP standard describes the Physical, Data link and Application layers of the protocol. A framework for designing and implementing PROFIBUS-DP slave devices is given in [10] where a main communication board is used by the DP Master to communicate to multiple slave devices. However the PROFIBUS-DP protocol is limited in its expandability and the distance between field devices. These limitation lead to the development of Fieldbus networks that were capable of communicating over much greater distances utilizing network topologies other than the traditional serial bus networks used in the PROFIBUS-DP networks.

The PROFIBUS fieldbus standards have been tested and scrutinized in many studies. At the fieldbus level the use of a multi-ring scheduling strategy for PROFIBUS is proposed in [11]. This approach utilizes two virtually separate logical rings on the same physical channel. The first ring is a fast traversing ring, called Ring A and a second slower ring called Ring B. This approach does not penalise faster stations compared to slower stations on the PROFIBUS system. Another advantage of this approach is that the QoS (Quality of Service) of the PROFIBUS network is improved, because all the traffic in the PROFIBUS system is better served, both the high- and low-priority data. In order to achieve the same fault tolerant capabilities of the standard PROFIBUS system, all the master devices on both the fast and slow rings need to maintain an alive-list, which keeps an updated list of devices which the master device can communicate with. This approach improves the performance of the PROFIBUS system that communicates over a serial bus network.

Serial bus systems are however very limited, both in the number of devices that they support and in the distances over which communication can take place with these devices. Thus the next field of research focussed on improving fieldbus systems, so that they could incorporate a large number of fieldbus nodes over vast distances. In order to achieve these goals fieldbus systems must be able to utilize a large variety of networks and support a globally accepted and widely used protocol.

1.8 REMPLI SYSTEM BACKGROUND

The Real-time Energy Management via Power lines and Internet (REMPLI) system is a field-level network, and as the name implies operates over the Internet and a power line communication (PLC) network. On a logical level the REMPLI system can be viewed as a fieldbus network, however on a physical level the REMPLI system consist of multiple different networks which are not based on a serial bus network, hence the term field-level network. The aim of the REMPLI project is to create a distributed infrastructure, which is suited for real-time data acquisition and statistical data processing for planning and tariff management purposes, [12]. The acquired data can also be used for management of the distribution network, which include supervision, control, quality, energy loss detection and fault control. Provision is also made for ad-on services such as domotic control and security within the REMPLI infrastructure. An overall structure of the REMPLI network is given in figure 1.3. Application Servers are responsible for sending and receiving data to and from metering equipment situated at remote locations. The requests and responses generated by the application servers traverses the Internet and a power line network to the serial bus network connected to the metering equipment situated at the remote site. The primary function of the REMPLI network is to enable application servers (SCADA or metering software), connected to the REMPLI Intranet, to retrieve/send data to/from nodes connected to the REMPLI communication network. This is achieved by connecting to an access point which then connects to the REMPLI Node, which in turn connects to the metering devices.

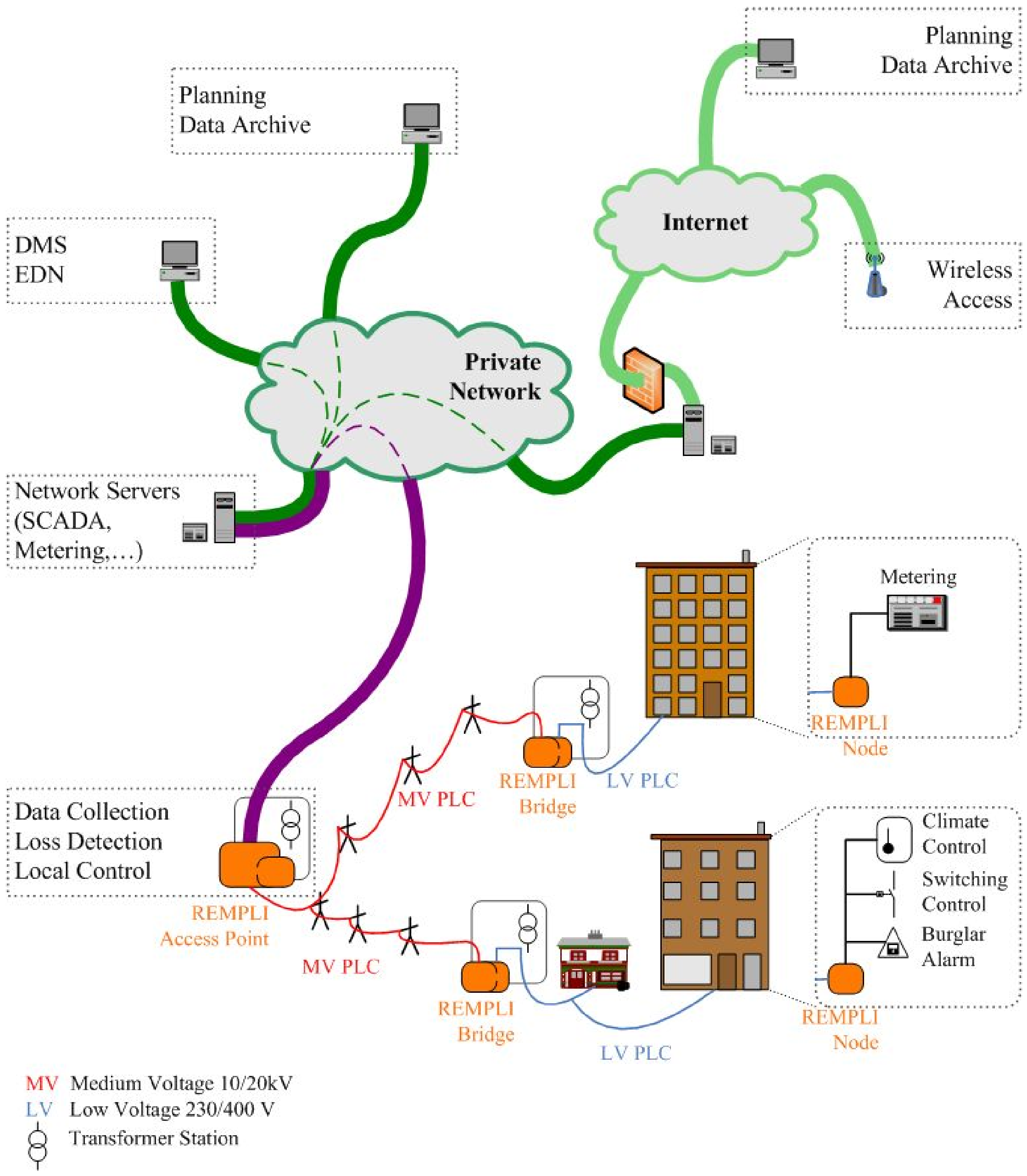


Figure 1.3. This figure shows the structure of the REMPLI system, [12].

The classical communication model used for metering or control devices in a fieldbus system defines two components: the application device and the metering or control device.

The communication model between the application device and the metering or control device is configured in a master-slave model in the majority of fieldbus implementations. In the master-slave communication model, the application device serves as the master device, while the metering or control device serves as the slave device. A direct connection exists between the application device and the metering or control device, which is implemented by establishing a direct physical channel e.g. RS232 or RS232 over GSM (Global System for Mobile Communication). Multiple slave devices is connected to the application device through this single media, which has no network segment (communication not defined for OSI layer 3).

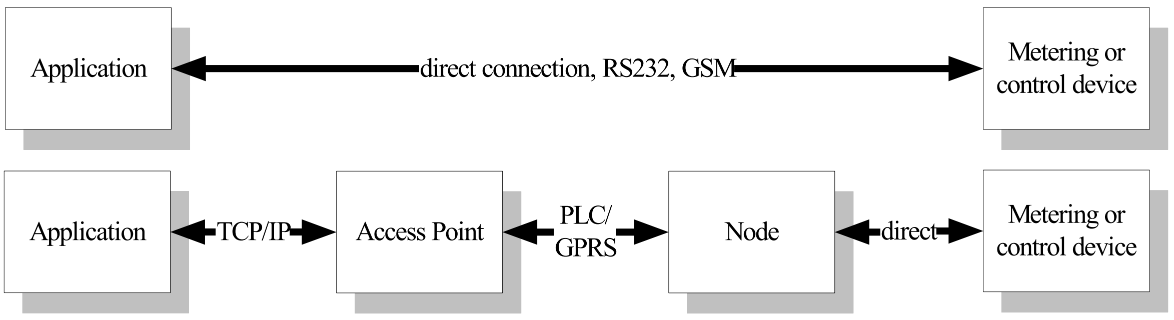

Application
direct connection, RS232, GSM
Metering or control device
Application
TCP/IP
Access Point
PLC/ GPRS
Node
direct
Metering or control device

Figure 1.4 Classical (top) and REMPLI Data transmission models (bottom), [1].

The REMPLI fieldbus system introduces a network segment with redundant paths and routing. A challenge arises when trying to integrate the current metering and control standards, which are only specified and defined for a direct connection, into a packet orientated communication system. The EN 62056-1, IEC 60870-5-101/4 and M-Bus protocols do not contain enough information to fulfil the requirements of the REMPLI system. Even IEC 60870-5-101/4 which is defined for a packet orientated system does not provide an adequate addressing scheme for the REMPLI system.

In order to make the REMPLI system versatile and interoperable and to use the standard applications as well as standard metering and control devices, REMPLI introduces the concept of Access Points and Nodes, which simulate a direct connection between the application and the metering or control devices. As is shown in Figure 1.5 REMPLI components are transparent for the applications: metering or control device is not aware of other system components other than the Node. The Node behaves like a directly connected application for the metering or control device. The situation for the application is identical in that the Access Point hides the REMPLI system from the application. Within the PLC network there are no connections, traffic is strictly packet orientated.

There are two possibilities to implement such a system: either tunnelling of the protocol data units from the application to the metering and control devices or creating a process image (located at the Access Point), which mirrors data points from metering and control equipment and directly answers requests from the application as an intermediary.

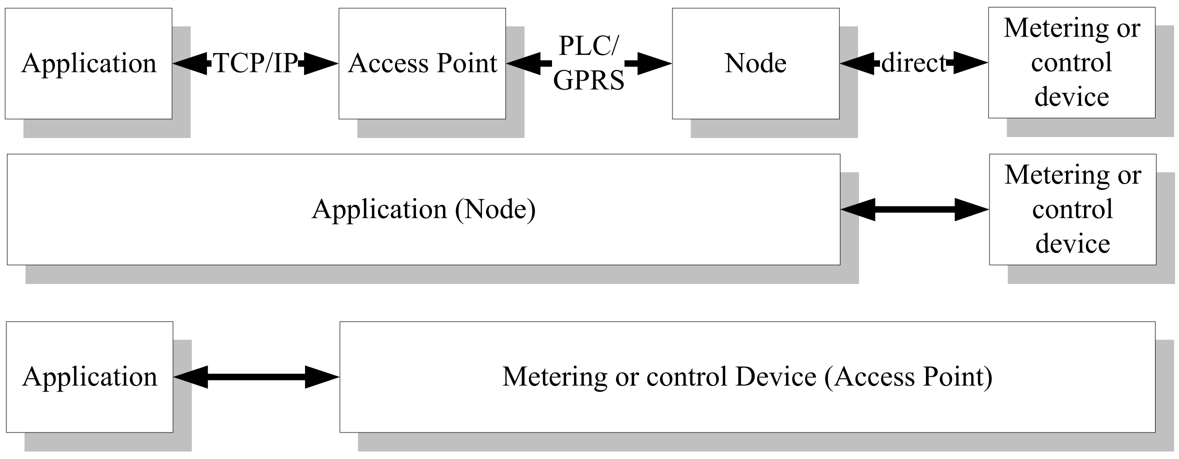

Application
TCP/IP
Access Point
PLC/ GPRS
Node
direct
Metering or control device
Application (Node)
Metering or control device
Application
Metering or control Device (Access Point)

Figure 1.5 Transparency of REMPLI Communication model, [1].

Data representation is based on common metering and control standards (SCADA), which are used to connect Application Servers with metering and control devices. Current fieldbus systems support various protocols which offer data representation models for control information and metering values. These fieldbus systems are however limited to proprietary protocols designed specifically for the tasks of acquiring metering data and controlling the communication process. The REMPLI system has as its goal the ability to offer a communication system that facilitates the use of existing metering and control equipment, supporting existing protocols and application servers. This allows for the use of currently commercially available equipment, saving on the costs of implementing a fieldbus system.

The most prominent protocols in use throughout Europe today include the IEC-870-5-101/104 protocol, EN-62056-21 protocol and the M-Bus protocol. That is why these three protocols have been chosen as the protocols to be supported by the REMPLI network. Another reason for choosing these three protocols is that they meet the communication needs for the deployment environment of the REMPLI system. These three protocols drastically differ in the way that they represent data. The EN-62056-21 protocol does not support data representation for data values. The IEC-870-5-101/104 protocol has definitions for ordinal data types like real, integer and string. The M-Bus protocol goes one step further and defines data types such as dates, type of metering data (gas, water, heating, electricity etc.) and consumption, depending on the application.

Therefore a generic method of sending and receiving these protocols over the REMPLI system is required. Due to the fact that the original protocols are designed for simple serial bus point-to-point systems, a distributed driver architecture is used, which contains additional information needed by the REMPLI system to successfully transmit the protocol data units (PDU’s) contained within the protocols from source to destination within the REMPLI network. The driver will be responsible for managing the Data link layer data contained within the protocols mentioned above, which include transmission parameters, telegram formats, addressing and data integrity. This data contains the information required to transmit PDU’s between metering or control equipment and applications.

1.9 M-BUS PROTOCOL AND DRIVER

1.9.1 M-Bus Driver Overview

The M-Bus protocol is one of the standardized protocols that are implemented within the REMPLI system. This section gives a brief description of the M-Bus protocol.

The M-Bus protocol is used for the remote monitoring and control of electricity (or other) meters and is derived from the IEC 870-5 standard but does not implement all of the functions described in the IEC 870-5 standard. This protocol allows for the interconnection of many devices over long distances, while maintaining a high degree of transmission integrity. The M-Bus protocol consists of a master-slave hierarchy and the master device is the only device that is allowed to initiate communication with slave devices. At the Data link layer four different frame formats are used to perform various functions such as initialization of slaves, sending of user data, requesting data and response to requested data. The Data link layer provides send/confirm and request/response transmission services.

Data transmission takes place in half-duplex asynchronous mode over a serial line with data rates between 300 and 9600 baud supported. The M-bus protocol uses the master-slave structure to communicate because slave devices are not allowed to communicate with each other. In this case the master station (master) is the primary station that initiates all message transfers while outstations are secondary stations (slaves) that may transmit only when they are polled.

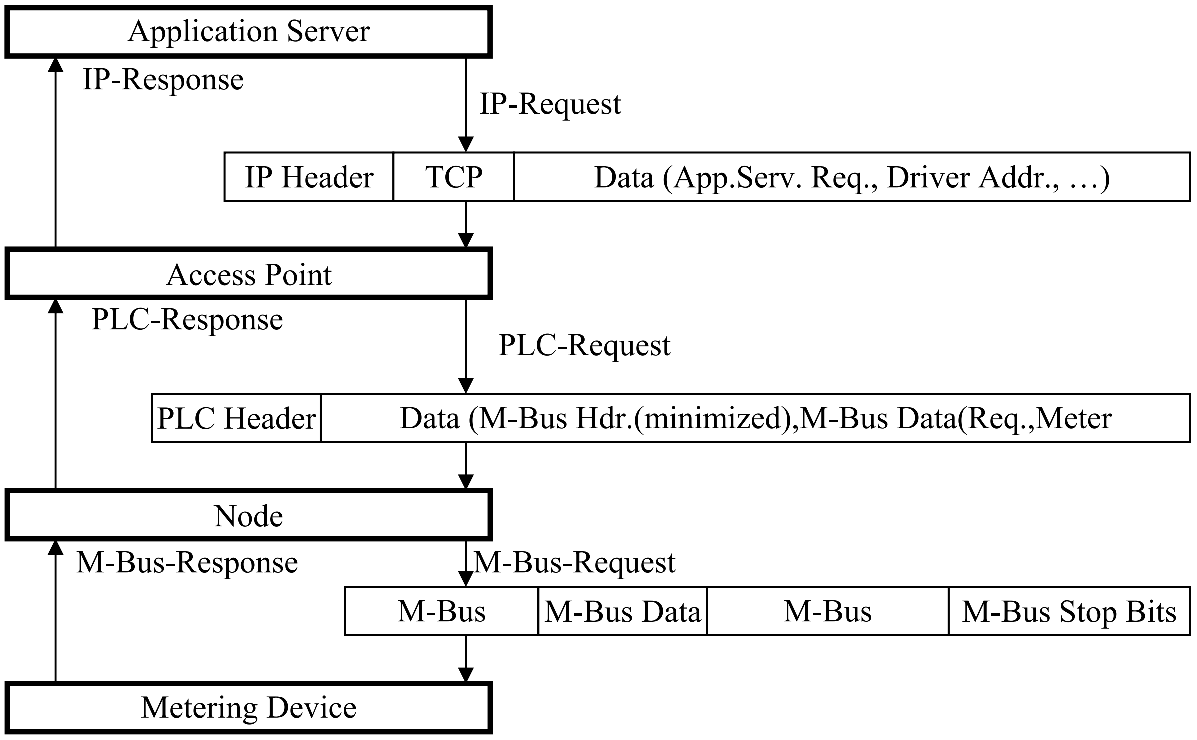


Figure 1.6 Communications process in REMPLI network with simplified packet structure.

The general communications structure for the REMPLI network is shown in figure 1.6. The Application server initiates the communication by sending a request to the REMPLI Access Point over a TCP/IP link. The general structure of this packet is also shown in figure 6.5. Once the TCP/IP packet is received at the REMPLI Access Point the relevant data is extracted from the IP frame and processed for further use. The resultant data is sent over the REMPLI power-line communications protocol to the REMPLI Node. The “twin” driver at the Node extracts the data received from the PLC interface and builds the original headers in order to transmit the request to the Metering Equipment. After the request was successfully received by the M-Bus Metering Equipment the response is sent to the Node-Side Driver. The response is then optimized to be sent across the PLC network where the “twin” driver situate on the Access Point will process the data in the frame in order to transmit it via IP to the Application Server. This in broad terms explains the general communication structure of the REMPLI project. It is also important that the communications chain is master-slave driven and therefore only the master device can initiate communication. However the communication between Access Point-Side Driver and the Node-Side Driver is not master-slave, they are equal and can send APDUs

(Application Protocol Data Units) at any given time. There is also no time constrains which would allow the drivers to manage the communication process. This holds even for the alarm protocol, which have to be polled by the Node-Side Driver at regular intervals for an alarm state. If an alarm is detected, the adequate response is returned.

1.10 DOCUMENT OVERVIEW

Chapter 1 provides an introduction to the research presented in this document. A brief summary is given of the scope of the research and the motivation is given with introductory objectives of the research, the methodology used in the research and background is given in the relevant fields.

Chapter 2 gives an overview of the current literature published in the relevant fields required in completing the research presented.

Chapter 3 Gives the Objectives of the research presented.

Chapter 4 presents the research and the approach used in the final software architecture design.

Chapter 5 presents the concept of the distributed driver architecture with the functional requirements of each component presented.

Chapter 6 presents the protocol tunnelling architecture as implemented for the M-Bus protocol. The functions of all the components are given and explained through the use of examples, with sequence diagrams. Required functionality such as address translation, access control and optimization is discussed.

Chapter 7 gives a brief outline of the implementation solutions used in the M-Bus distributed protocol driver.

Chapter 8 presents the results obtained from the implemented software architecture using the M-Bus protocol stack and discusses future work for the developed software architectural framework suggested in this research.

Chapter 2

2 OVERVIEW OF CURRENT LITERATURE

The REMPLI system is unique in fieldbus domain due to the communication that takes place over power-line communication and the tunnelling architecture that enables currently available fieldbus protocols to be used to communicate to remote field devices over an IP/PLC or IP/GPRS network. The REMPLI system combines the field of power-line communication with IP based routing in a field-level network. The relevant fields of research mainly comprise power-line communication and fieldbus systems. Figure 2.1, below gives a tree view of the main areas of research involved in fieldbus systems, relevant to this thesis.

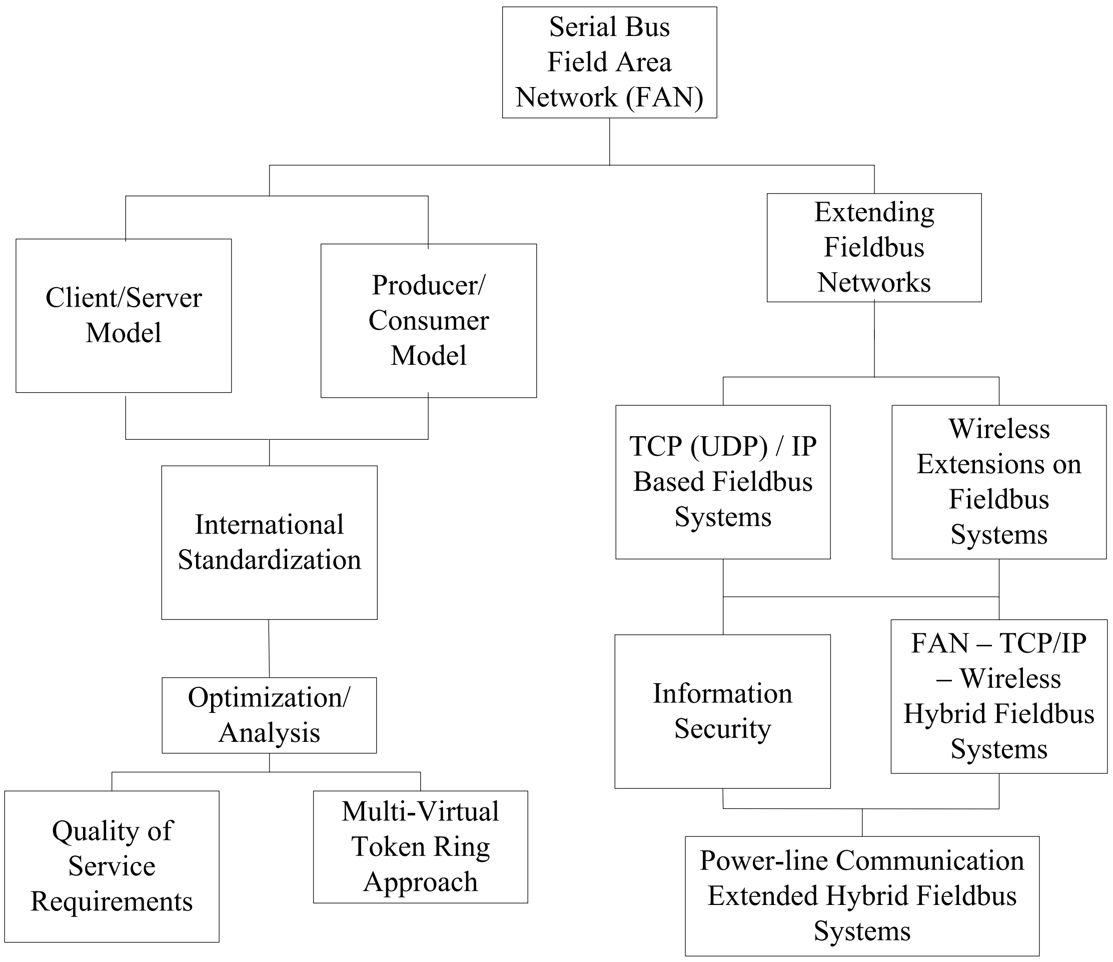

Serial Bus Field Area Network (FAN)
Client/Server Model
Producer/ Consumer Model
Extending Fieldbus Networks
International Standardization
TCP (UDP) / IP Based Fieldbus Systems
Wireless Extensions on Fieldbus Systems
Optimization/ Analysis
Information Security
FAN – TCP/IP – Wireless Hybrid Fieldbus Systems
Quality of Service Requirements
Multi-Virtual Token Ring Approach
Power-line Communication Extended Hybrid Fieldbus Systems

Figure 2.1 Tree view of main research areas relevant to the work proposed in this research.

2.1 FIELDBUS SYSTEMS

The integration of IP traffic into fieldbus systems would allow the use of IP based networks for fieldbus systems, allowing for communication and control of field devices over a global network. Several challenges must be overcome to integrate IP traffic into fieldbus systems given in [13] as listed below:

- A mismatch exists between the fieldbus protocols and the TCP/IP protocol stack. Fieldbus systems are only defined for the Physical, Data link and Application layer of the OSI while the TCP/IP protocol stack resides between the Network and Transport layers of the OSI-RM (Open Systems Interconnection – Reference Model).
- IP traffic should remain intact and not interfere with the control data.
- Fieldbus systems were not originally designed to host a large number of field devices, thus an addressing scheme has to be developed to uniquely identify all the field devices within the fieldbus system.
- The IP based network must adhere to strict Quality of Service (QoS) requirements in order to guarantee successful communication and control.

By filling the gap in the OSI layers filled by the TCP/IP protocol stack with an IP Scheduler, IP Mapper and a TCP/IP stack [13] it is possible to route fieldbus PDU's over an IP based network. The IP Scheduler is responsible for scheduling the IP traffic in such a way that the Quality of Service requirements needed for the successful communication is achieved. The IP Mapper is responsible for translating the IP traffic into fieldbus application datagrams and converting the fieldbus application datagrams back into IP traffic. The IP Mapper is also responsible for facilitating any architectural issues when integrating the IP traffic client-server communication into the resident fieldbus communication model.

The approach to implement a fieldbus over an IP network mentioned above has one major short coming: The IP Scheduler, IP Mapper and TCP/IP stack are dependent on the fieldbus protocol being implemented and thus needs to be specifically implemented for each different fieldbus system/protocol.

Another approach to implementing fieldbus systems over IP based networks that do not require proprietary Web technology or fieldbus-dependent tools, is to utilize the Simple Network Management Protocol (SNMP) as given in [14], which allows for largely fieldbus independent integration into IP based networks. The two main advantages of utilizing network interconnection to access remote field devices as given in [14] are as follows:

- Provides Access to FAN's (Field Area Networks) and field devices on FAN's from anywhere that the internet is accessible. This allows for remote control and automation. This includes the ability to automate buildings remotely. It is also useful for utility companies to utilize existing network infrastructure to gain access to information.
- The Internet is based on global standards for communicating with remote systems. Many tools and function libraries exist and are readily available, which is platform independent, making the use of the Internet an ideal and very versatile communication platform for implementing a distributed fieldbus system.

One of the major drawbacks of utilizing the Internet as a communication medium in fieldbus systems is that the Internet is a best effort communication system, which restricts IP-based fieldbus systems, to be restricted to fieldbus systems where the data rate and response times are not of predominant interest [14].

A notable characteristic about the SNMP approach is that it is interoperable with current LAN networks and technologies [15]. The SNMP approach requires the use of an interface device in order to provide abstract access to fieldbus systems, called a gateway. The approach followed in [15] uses the gateway to represent several fieldbus nodes, which belong to one or more FAN's (Field Area Networks) within a single SNMP MIB (Simple Network Management Protocol Management Information Base).

The required functions for distributed processing for application layer functions are given in [16]. The application functions are divided into two main categories: control functions and management functions.

The role of the gateway is to provide a conversion mechanism between the SNMP protocol data and the fieldbus device. The MIB on the proxy agent is responsible for representing the information on all the fieldbus nodes. The protocol mapping between the SNMP PDU's and the fieldbus PDU's translates the SNMP PDU's into fieldbus commands. It is however impossible to setup a bijective relationship between the two systems, because SNMP provide only 5 SNMP messages and there are normally more, and a larger variety of commands used by fieldbus devices.

This approach however has some disadvantages:

- The adoption rate in the future and the levels of implementation of the SNMP protocol is in unsure, due to its dynamic nature.
- The lack of security provided by the SNMP protocol.
- Simultaneous FAN access of the proxy, along with other applications is not possible, unless the API of the FAN supports multiplexing multiple requests from multiple applications.

Another important communication network that fieldbus systems utilize is wireless networks. This is the other focus of research done to diversify and improve fieldbus systems. Wireless networks have the advantage that no physical cables have to be installed. The focus of the research in fieldbus systems thus extended to designing hybrid, wired/wireless networks.

A tag-based tree approach can be used, similar to XML (Extensible Markup Language) to give fieldbus systems a plug-and-play interoperability mechanism, with a simple addressing mechanism, [17]. This data representation format is independent of the fieldbus system. One major drawback for low bandwidth fieldbus systems is that this approach increases the overhead and data sent over the network. The overall complexity of field devices and the communication system is also increased, at the expense of greater flexibility.

An approach to achieve this hybrid wired/wireless is given in [5]. This approach suggests the use of a bridge interface between the wired network and the wireless network. A Multi Logical Ring (MLR) token passing mechanism is used to communicate between the initiator and the recipient. A logical ring is allocated to each bridging protocol extension called the Inter-Domain Protocol [6]. In the MLR approach each logical ring serves stations on a unique communication medium. As an example a wired domain comprises only stations that are connected via a wired network interface, while all stations connected via a wireless network interface belongs to the wireless domain. IDT (Inter-Domain Transactions) take place between an initiator and a responder that belong between different domains, i.e. one or more bridges in the communication path. Bridges can be located between different wireless domains. This approach does however have a disadvantage, in that it does not efficiently utilize the available wireless bandwidth. A packet loss rate of 7% combined with independent packet losses results in the logical ring being incomplete for more than 50% of the allocated token time. Due to the erratic nature of wireless networks in industrial environments results in stations being dropped and then they have to be re-included into the ring, which is time consuming [6]

By further extending the virtual ring extension approach to utilize specific wireless protocols [6], the efficiency of the MLR wired/wireless approach can be increased. The wireless domain utilizes a specific MAC and link-layer protocol that offers the same link layer interface to the upper layers, as the wired domain, which will result in portability of application layer software. A polling-based protocol is suggested for this task in combination with the original PROFIBUS protocol.

As an extension of the current research done thus far the work done in this paper proposes a transparent tunnelling approach to communicate to field devices on a fieldbus system over a hybrid wired/wireless PLC network [1] and gives a generic framework for the driver structure that is needed to accommodate the integration of current fieldbus protocols into the REMPLI system. An earlier implementation suggested implementing a process image, which required the protocol data received from application servers to be converted in order to be transported across an IP-based network [18]. This approach is not used in the REMPLI system due to certification requirements prohibiting the translation of APDU's generated in the REMPLI system.

2.2 POWER LINE COMMUNICATION

The challenges faced in using power lines as a communication is highlighted in [19], where the prevailing noise sources on a power line channel is investigated and characterised. The noise on power line networks are well understood but remain hard to predict. Due to the wide geographical coverage of reticulation networks [19], bandwidth and power levels have been limited on communication signals.

Even though the latest fields of interest in power line communication is to apply internet broadband techniques to power line communication, there is still a need for low-cost low-data-rate telemetry or control applications, such as remote meter readings [20]. These communication systems require low-cost transceivers thus narrow-band half duplex communication is used.

Many difficulties exist within a power line network if it is used for communication purposes. Power lines exhibit multi-paths caused by reflections caused by the discontinuities of the network. Different sources of noise exist on power line networks for example: coloured background noise, narrowband interference and impulse noise. Robust and frequency efficient transmission techniques have thus to be used to ensure data integrity [21].

The use of power lines as a communication medium to acquire metering values remotely is suggested by [22], [23], [24], [25], [26], [27], [28], [29] and is very useful in performing load balancing in the power grid. Various local area network strategies and implementations are proposed and discussed. It is clearly shown in these publications that power line communication is both an effective and efficient means of communication used to acquire remote readings from metering equipment.

The requirements for the REMPLI power line communication network are provided in [30] and include the following: high reliability, automatic handling of redundancies, high coverage and distances, support for a large number of communication nodes, provide the appropriate delay and system responsiveness, security and ease of deployment and maintenance.

The communication stack for the power line network implemented in REMPLI is presented in [31], and is focused on the Network and Transport layers. The Network layer provides a master-slave time division network with basic error recovery and short distance routing mechanisms. The Transport layer supports the network layer with inter-network routing, fragmentation, request/response pairing and address translation and alarm signal support [31].

Chapter 3

3 OBJECTIVE

The goal of the research proposed in this document is to provide a software framework for utilizing standardized fieldbus protocols and systems within the REMPLI context, which will guarantee versatility, interchangeability and scalability in ultimately transparently communicating over multi-tier networks in which any of a combination of TCP/IP, GPRS or PLC communication can be used. A triplet of software drivers called the REMPLI Protocol Driver is used to achieve this.

Due to the diverse nature of the Application layer data contained within the protocols used within the REMPLI context the Application layer data can not be used by the protocol drivers. It is however the requirement of the protocol drivers to facilitate the use of application or manufacturer specific information. The protocol driver is however not allowed to use any information contained within the application layer data of the protocol for data management purposes.

The framework proposed in this document implements a distributed driver structure in such a way that satisfies all the above mentioned requirements and enables the communication model required within the context of the REMPLI system. A further objective of the research done in this paper is to implement the proposed distributed driver framework on a multi-tier network with M-Bus meters and the M-Bus metering protocol as a test case scenario.

The REMPLI system has as its goal the ability to offer a communication system that facilitates the use of existing metering and control equipment, supporting existing protocols and application servers, by transparently tunnelling these protocols over TCP/IP and PLC/GPRS based networks. This allows for the use of currently commercially available equipment, saving on the costs of implementing a proprietary fieldbus system.

Chapter 4

4 PROPOSED RESEARCH

As mentioned in chapter 1, the REMPLI system enables current application servers to send and receive PDU's to the remote metering equipment over the Internet and a power line/GPRS network. Two possible solutions were considered for implementing such a system: either tunnelling of the protocol data units from the application to the metering and control devices or creating a process image (located at the Access Point), which mirrors data points from metering and control equipment and directly answers requests from the application server as an intermediary communication device. The communication process implemented in the REMPLI system and the data sources used has a major influence in the manner in which data is represented. Thus this section will introduce both models and give reasoning why REMPLI favours the tunnelling approach instead of using a process image.

4.1 PROCESS IMAGE

Due to low reliability and low performance in terms of bandwidth and RTT (Round Trip Time) of the PLC communication system within REMPLI, the concept of a process image was proposed. Due to the unstable nature of an energy distribution network, any one of a multitude of parameters within the communication network can spontaneously fluctuate, causing unexpected delays in return trip time or interruptions in the communication process. In order to overcome these inherit shortcomings in communicating over a power distribution line, cascaded loggers was proposed to manage, and store all sensitive data. The logging of data takes place in a two step process as given below.

The first step in the logging of data takes place at the Nodes: At each REMPLI Node the software driver is designed to store measurement values at the Node (data logger). These readings are taken by the metering equipment connected to the Node. Each measured value stored at the Node is given a timestamp, derived from a real-time clock which is synchronized between all Nodes within the REMPLI network. The stored measurement values, which are now associated with a timestamp is sent back to the Application Server, via the Bridges and Access Point. The Nodes would require enough resources to store all the measurement values taken over a time period of several days.

The second step in the logging process takes place when the Access Point periodically poll the Node side data loggers for stored values, not yet retrieved from the Nodes. The Access Point will then retrieve all the outstanding values up to the last measurement taken by the meters. At the Access Point the retrieved values are stored and a history log is maintained on the stored values. The Access Point would require enough resources to store the measured values taken by all the Nodes connected to it over several months. The Application Server then uses the Access Point as the source of the readings it requires. The Application Server can poll the Access Point for all the outstanding values in its local database. The Access Point can easily determine which values are still required by using the timestamp associated with each value.

This concept of data loggers at each level in the REMPLI network introduces redundancy. The main advantage of redundant data is that if communication breaks down at any level in the network no data will be lost, as long as the storage capacity of the lower level devices are not exhausted. This proposed system eliminates a single point of failure in the network and thus improves the overall system reliability, by ensuring that even if a device in the communication chain fails its nearest neighbours would contain enough information to restore the gap in the measurement history. As an example if an Access Point suffers complete data loss, the most recent data could be retrieved from the Nodes connected to it. The older values are already stored by the Application Server and the loss of this data at the Access Point has no negative impact on the measurement history.

Due to the limited bandwidth of the power line network and the master-slave communication model implemented by the power line communication system the need arises for redundancy in the communication network. The concept of data loggers provides this redundancy, however the prerequisites to implementing such a system of data loggers are:

- During a typical usage scenario, PLC channels are frequently idle and have no data traversing the PLC network. The available time slots where the PLC line is idle could be utilized to fetch data from the Nodes in the background.
- The performance of the power line communication system can significantly be increased by running multiple concurrent request/response procedures to multiple Nodes. When data transmission adheres to the allocation of fixed time slots and a

Node cannot immediately respond, some time slots will be available for communication.

The REMPLI system is the sole user of the PLC communication system and thus it has control over the allocated time slots for communication, this allows for the system to take advantage of the freely available time slots and exploit all available bandwidth. Thus apart from the Node-level data loggers collecting measurements from the meters, the Access Point also constantly polls the different Nodes for the stored values located at these Nodes. As explained previously these values are then stored at the Access Point and it is periodically updated.

Whenever the Application Server request a measurement value the Access Point responds immediately with the locally stored response. This eliminates the need for a time consuming, and often costly PLC transaction. The benefit of this method of communication is that the network load on the power line system is greatly reduced when different applications request measurement values concurrently. Because the Nodes don't have to be polled, thus the throughput that each individual user can achieve is increased.

4.2 TUNNELLING

A second approach was proposed in the form of tunnelling protocols over the entire network. This section presents the concept of a tunnelling approach and the advantages of using this approach. The shift from the data logger approach to the tunnelling approach was mad due to two problems encountered in the REMPLI system:

- The Application Servers using the REMPLI system will be required to poll the Node directly and refuse data from the process image much more often than assumed, leaving far less free timeslots available for communication.
- Many features of common metering and control protocols cannot be supported, because the common denominator would be read and write operations on single data points consisting of a structured value and a physical unit.

The first problem listed leads to the performance requirements of the PLC network increasing. This is due to the fact that the PLC network traffic cannot be scheduled according to the needs of a data logger approach. There is also a need to serve requests on demand. A fundamental short coming is listed in the second problem in that it is impractical to create a data representation that supports all features supported by current systems and future fieldbus/communication protocols. A further fundamental flaw of the data logging approach is that data transformation is required in the data logging system. In the context of the REMPLI system and for billing purposes protocol data units can not be manipulated. Measuring values can not be retrieved in the original data packet, manipulated and the later be retrieved into the original format, without certifying the whole REMPLLI system. The constraints on the system, listed above make the process image (data loggers) approach impractical.

In order to accommodate the constraints and problems listed above with the process image approach, a tunnelling approach was selected for each protocol implemented in the REMPLI system. Each protocol requires a specific driver which is responsible for tunnelling the respective protocol data units, associated with the protocol, between the Application Server and the Node via the Access Point. In order to optimally utilize the available bandwidth and performance, only application layer data units are tunnelled over the REMPLI system. Layer 2 functionality, such as a keep-alive signal that is described within the protocol is discarded and the respective driver is responsible for maintaining the communication link. This allows the driver to limit the amount of data sent across the PLC network, thus optimising the communication system. The Node side driver will inform the Access Point side driver to terminate a current session if for example a polled meter responds with an alarm signal, indicating that the specific meter is not functioning correctly.

4.3 REMPLI Tunnelling Scheme

The general requirements of the applications to the REMPLI project:

- Applications demand the current data from the metering and control devices according to their own schedules.
- Applications use features of specific common metering and control protocols. No common denominator of all protocols can be implemented with reasonable effort. Translation between different protocols cannot be done in a generic way.

- Another problem is certification, meaning that data from a metering device cannot be changed/transformed/touched during transfer from the device to the Application Servers; fulfilling this requirement is only possible when tunnelling a protocol.

The system requirements listed above implies that the tunnelling approach is more suited to the REMPLI system. Three different metering protocols are initially going to be implemented by the REMPLI system, requiring the use of three different tunnels. In the REMPLI systems drivers for the M-Bus, EN 62056-21 and IEC 60870-5-101(104) protocols must be developed. A pair/triplet of drivers is required by each protocol, situated at the Application Server side, Access Point side and the Node side. The Node side driver functions as a master to the metering equipment and a slave for the Access Point side driver. The Access Point side driver in turn functions as a slave to the Application Server side driver.

The Node side driver communicates with the meter and control equipment using the appropriate standard metering protocol, after completing this procedure the resultant data is forwarded to the Access Point side driver over the power line network. The Access Point side driver then forwards the received data packet to the Application Server over a TCP/IP network. If predefined protocols to transfer metering and control data to the Application Server are defined, the implemented drivers should utilize these protocols. E.g. IEC 68070-5 includes the 104 part that defines data transmission over TCP/IP networks. If no standard protocol is defined, e.g. M-Bus and EN 62056-21, the data packets will be tunnelled to the Application Servers over the TCP/IP network. This tunnelling requires the use of drivers at the Application Server side, to transmit the data to/from the Access Point side drivers.

Figure 4.1 shows the protocols scheme. It gives an overview of the protocols and entities from a driver point of view. Whereas the interface to the REMPLI network is common for both parts of the driver the interface to the applications entities is totally different. On the Node side it includes protocol stacks to interface the metering and control devices via the hardware interfaces of the Node. On the Access Point side the driver needs to interface the TCP/IP stack.

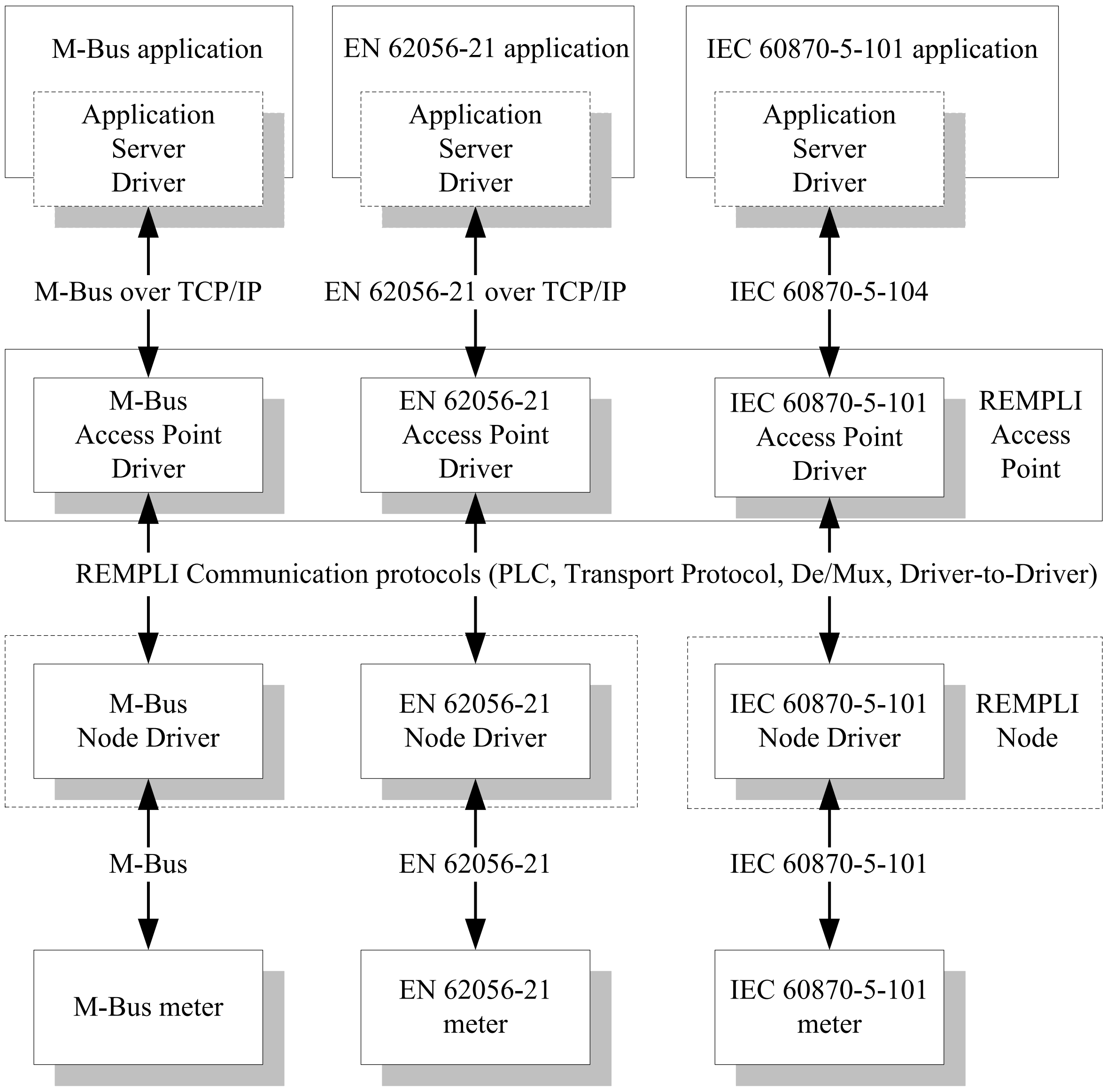

M-Bus application
Application Server Driver
EN 62056-21 application
Application Server Driver
IEC 60870-5-101 application
Application Server Driver
M-Bus over TCP/IP
EN 62056-21 over TCP/IP
IEC 60870-5-104
M-Bus Access Point Driver
EN 62056-21 Access Point Driver
IEC 60870-5-101 Access Point Driver
REMPLI Access Point
REMPLI Communication protocols (PLC, Transport Protocol, De/Mux, Driver-to-Driver)
M-Bus Node Driver
EN 62056-21 Node Driver
IEC 60870-5-101 Node Driver
REMPLI Node
M-Bus
EN 62056-21
IEC 60870-5-101
M-Bus meter
EN 62056-21 meter
IEC 60870-5-101 meter

Figure 4.1 REMPLI protocols communication scheme.

Chapter 5

5 REMPLI PROTOCOL DRIVER

5.1 GENERAL STRUCTURE OF THE REMPLI PROTOCOL DRIVER

The purpose of this section is to give a brief but complete general structure and functional analysis of the driver software for the REMPLI Protocol Driver to be implemented within the REMPLI project. The driver software will compose of three parts, distributed over the REMPLI network. The drivers are located at the Application Server, Access Point and Node side, as indicated by figure 5.1.

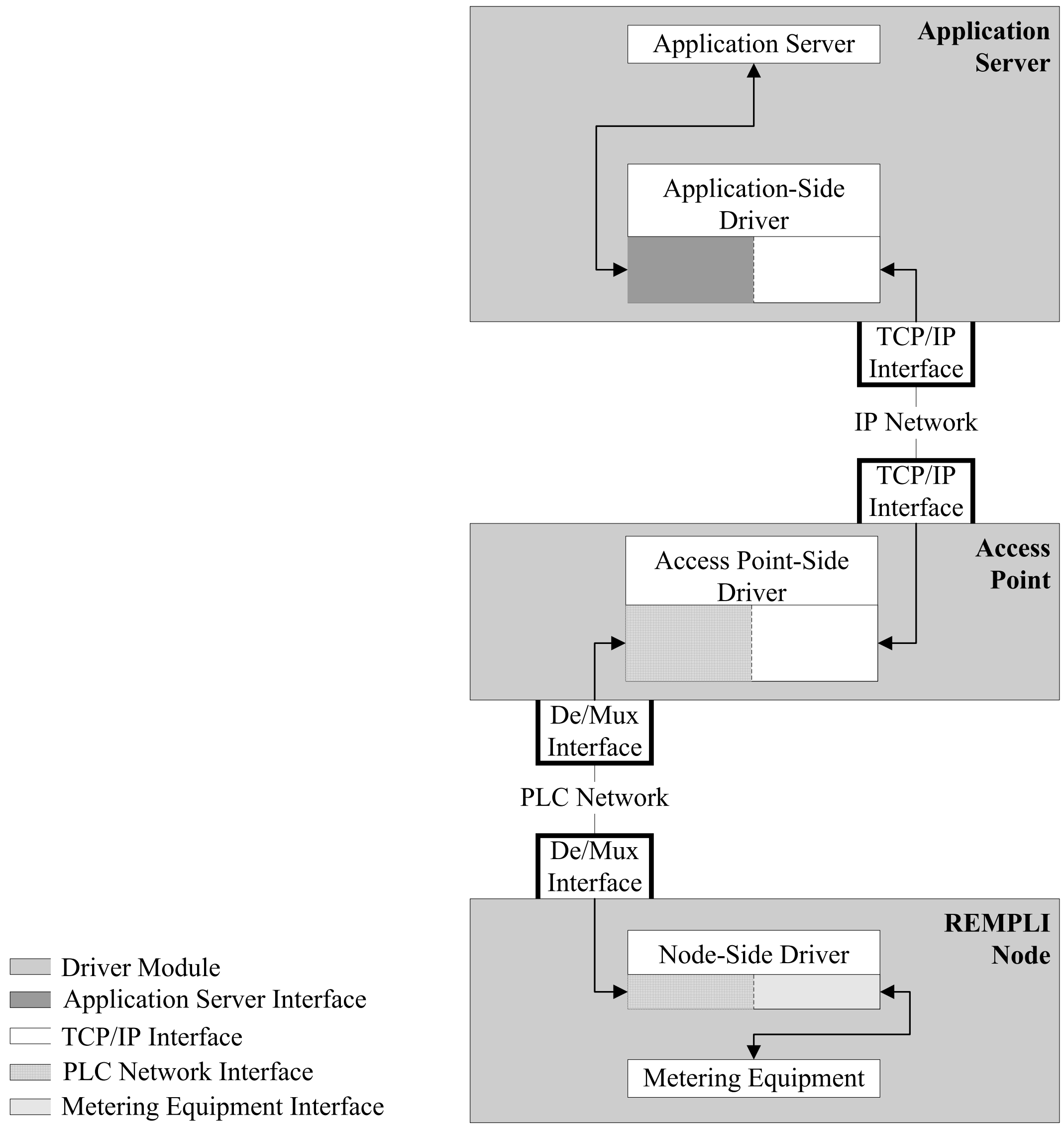


Figure 5.1. This figure gives the general driver structure for the REMPLI Protocol Driver within the REMPLI network.

The Application-Side Driver is located at the Application Server and it interfaces with the Application Server and the TCP/IP network. The Access Point-Side Driver is located at the Access Point and it interfaces with the TCP/IP network and the De/Mux (Multiplex/De-Multiplex) or GPRS Interface. The Node-Side Driver is located at the REMPLI Node and it interfaces with the De/Mux or GPRS Interface and the Metering Equipment connected to the Node.

5.2 APPLICATION SERVER-SIDE DRIVER

The Application Server-Side Driver is situated at the Application Server. The data could be received from the Application Server and sent to the TCP/IP interface, or received from the TCP/IP interface and sent to the Application Server. The functional block diagram of the Application Server-Side Driver is given in figure 5.2.

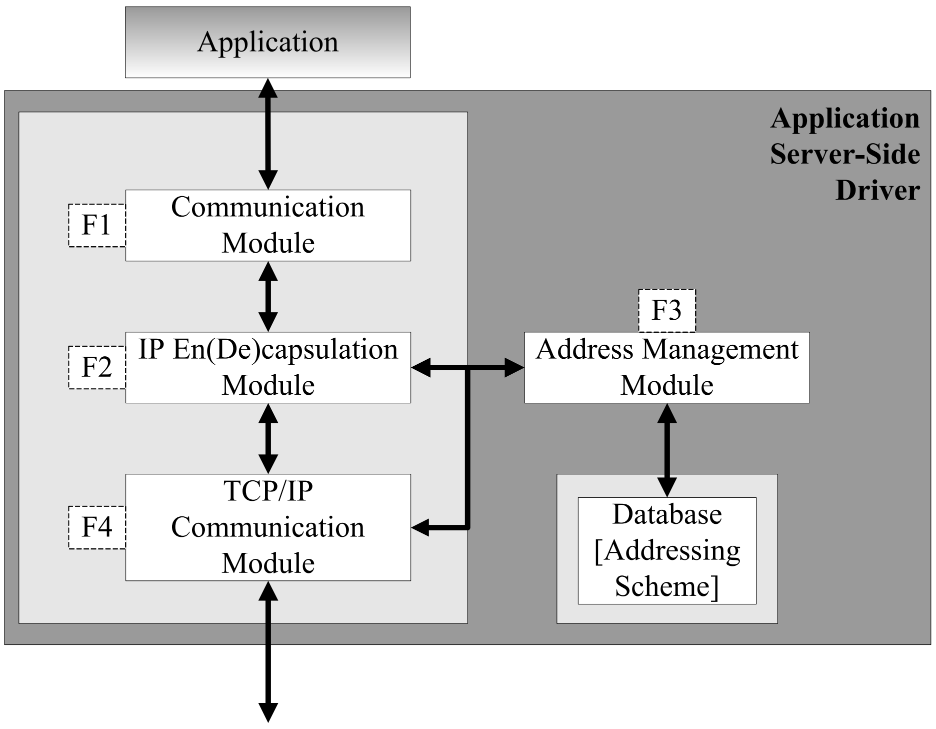

Application
Application Server-Side Driver
F1
Communication Module
F2
IP En(De)capsulation Module
F3
Address Management Module
F4
TCP/IP Communication Module
Database [Addressing Scheme]

Figure 5.2. The functional block diagram of the Application Server-Side Driver situated at the Application Server

All of the functional units in figure 5.2 are identified with tags [F1, F2,..F4]. The functions of each tagged module are given below:

F1: The function of the **Communication Module** is to communicate with the Application Server. Sending and receiving of APDUs to and from the Application Server. This is a communication system that is implemented in software and it communicates by sending and receiving data to/from the Application Server. Optimized “Keep-alive” signal management[3] might also be implemented at this module. This “Keep-alive” functionality is protocol specific and depends on

[3] The “Keep-alive” signalling concept is discussed in section 6.6.

whether or not the protocol implemented by the driver supports a “Keep-alive” signalling scheme.

F2: The function of the **IP En(De)capsulation Module** is to encapsulate the APDUs received from the Application Server into TCP/IP packets. It is also responsible for decapsulating the TCP/IP packets received from the IP network into the APDUs used by the Application Server. Before the APDUs can be fully encapsulated within an IP packet the destination Access Point IP address and the RNA (REMPLI Node Address) are required and it is obtained from the Address Management Module.

F3: The function of the **Address Management Module** is to obtain the necessary TCP/IP addresses in order to deliver the application generated PDUs (APDUs) to the corresponding Access Point destinations and to address the correct meter connected to the corresponding REMPLI Node[4]. These Access Point destination addresses consists of the Access Point IP address and the destination RNA supplied by the Application Server. If the appropriate destination addresses are not supplied by the Application Server, then these addresses will have to be derived from the Application Server data.

F4: The function of the **TCP/IP Communication Module** is to manage the sending and receiving of TCP/IP packets on the TCP/IP link. This includes buffering of packets, sending/resending and reordering of packets if required. Examples of issues to be considered here are:

- Size of input buffer: The size of the input buffer should be sufficient to contain the maximum payload returned by a metering request. The size of these buffers is usually allocated as part of the configuration parameters for the TCP/IP Communication Module.

[4] The addressing of meters is discussed in section 6.4

- Error handling: If an error is detected at the TCP/IP link this information should be reported to all modules within the driver that requires this information. Examples of such modules are:
  1. If the "Keep-alive" signalling scheme is supported by the driver then the "Keep-alive" module should be informed in order to stop sending "Keep-alive" messages.
  2. The Address Management module should also be informed if a link is down, in order to update the database with addressing table and to reroute the packets.
  3. If the operating system does not perform re-ordering of packets, and requesting the resending of corrupt or lost packets, then this type of error recovery should also be implemented.
- Optimized "Keep-alive" signal management might also be implemented at this module as discussed in the section 6.6 about Optimized "Keep-alive" Signal Management.

5.3 Access Point-Side Driver

The Access Point-Side Driver is situated on the Access Point. The data could be received from the De/Mux or GPRSinterface and sent to the TCP/IP interface, or received from the TCP/IP interface and sent to the De/Mux or GPRS interface. The functional block diagram of the Access Point-Side Driver is given in figure 5.3.

All of the functional units in figure 5.3 are identified with tags [F1, F2,..F6]. The functions of each tagged module are given below:

F1: The function of the **TCP/IP Communication Module** is to manage the sending and receiving of TCP/IP packets on the TCP/IP link. This includes buffering of packets, sending/resending and reordering of packets if required. Examples of issues to be considered here are:

- Size of input buffer: The size of the input buffer should be sufficient to contain the maximum payload returned by a metering request. The size of these buffers is usually allocated as part of the configuration parameters for the TCP/IP Communication Module.

- Error handling: If an error is detected at the TCP/IP link this information should be reported to all modules within the driver that requires this information. An example of such a module is:

  1. If the "Keep-alive" signalling scheme is supported by the driver then the "Keep-alive" module should be informed in order to stop sending "Keep-alive" messages.
  2. If the operating system does not perform re-ordering of packets, and requesting the resending of corrupt or lost packets, then this type of error recovery should also be implemented.

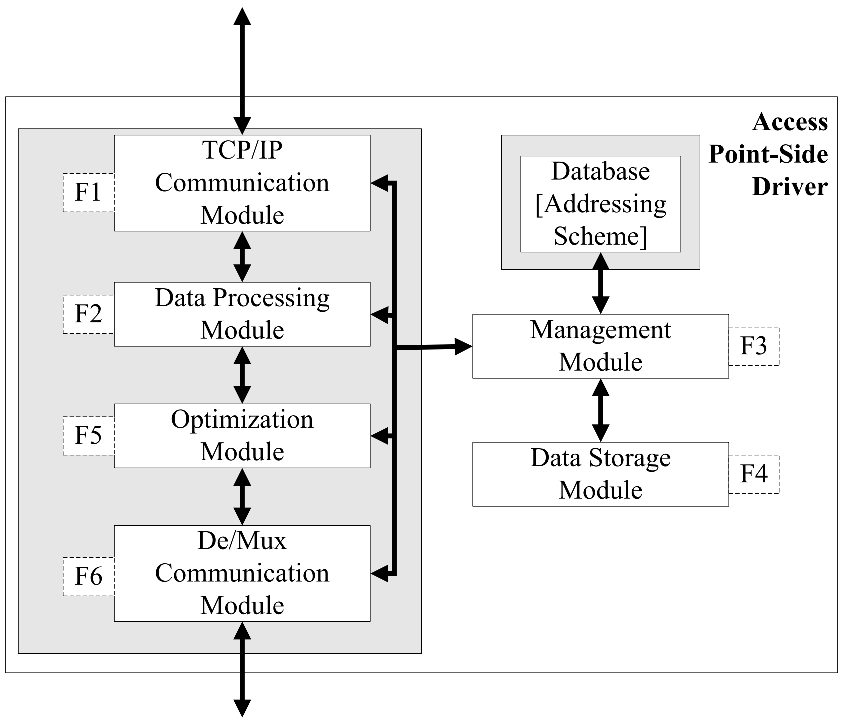


Figure 5.3. The functional block diagram of the Access Point-Side Driver

F2: The function of the **Data Processing Module** is to extract the necessary data out of the APDU, i.e. the equipment address and the RNA, in order to provide it to the management module. The nature of the processing and/or extraction of data out of the APDUs from this module is protocol specific. Thus the data required to be processed by this module is different for each protocol.

F3: The function of the **Management Module** is to control the communication link between the IP network and the PLC network. This includes the sending and receiving of "Keep-alive" packets. This module is used to maintain the state of the current request/response by initiating a timer to determine when the time-out for the current request/response has occurred, if supported.

F4: The function of the **Data Storage Module** is to store, retrieve and maintain small units of data. This module could also be used for maintaining logs of requests processed at the Access Point. This module is optional, depending on available resources at the Access Point.

F5: The function of the **Optimization Module** within the Access Point-Side Driver is to work with the "twin" optimization module situate at the Node-Side Driver. This module optimizes the data sent over the De/Mux or GPRS interface when sent from the Application Server to the Node, or expands the data to its original state if sent from the Node to the Application Server. The data could be compressed before it is sent via the De/Mux interface by this module. If viable this module will also remove any redundant data from the packets in order to minimize the bandwidth used by the data sent over the PLC network. Conversely it will rebuild the data to its original state if received in the minimized or compressed state.

F6: The function of the **De/Mux Communication Module** is to send and receive data via the De/Mux interface. Data received from the TCP/IP interface is processed by the driver and then sent to the De/Mux interface, which delivers the APDU to the Node via the PLC network. Conversely, data received from the De/Mux interface to the PLC network is processed by the driver and then sent to the Application Server via the TCP/IP interface. This module is also responsible for reacting to error messages received from the De/Mux module, by informing all the other modules in the driver that require this information, about the error. Alternatively if GPRS is used to interface with the Node then this module will have to support the GPRS interface.

5.4 NODE-SIDE DRIVER

The Node-Side Driver is situated on the REMPLI Node. The data could be received from the Metering Equipment and sent to the De/Mux interface, or received from the De/Mux interface and sent to the Metering Equipment via the protocol specific hardware driver for the HyNet board. The functional block diagram of the Node-Side Driver is given in figure 5.4.

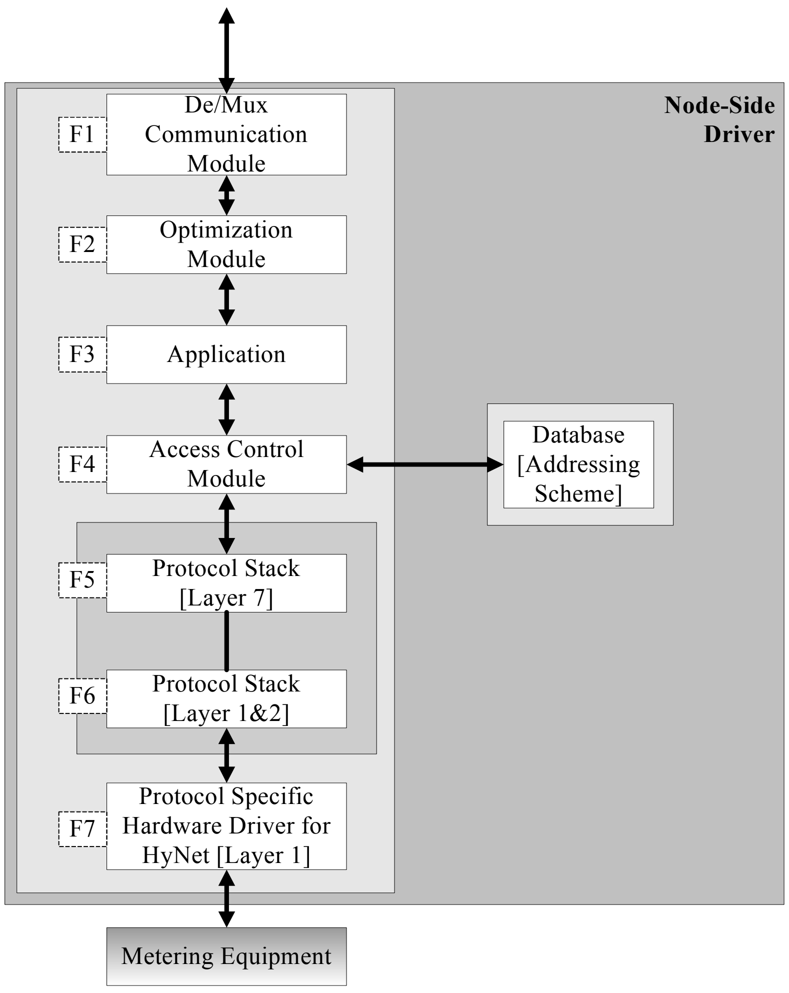

Node-Side Driver
F1
De/Mux Communication Module
F2
Optimization Module
F3
Application
F4
Access Control Module
Database [Addressing Scheme]
F5
Protocol Stack [Layer 7]
F6
Protocol Stack [Layer 1&2]
F7
Protocol Specific Hardware Driver for HyNet [Layer 1]
Metering Equipment

Figure 5.4. The functional block diagram of the Node-Side Driver

All of the functional units in figure 5.4 are identified with tags [F1, F2,..F7]. The functions of each tagged module are given below:

F1: The function of the **De/Mux Communication Module** is to send and receive data via the De/Mux interface. Data received from the De/Mux interface is processed by the driver modules and sent to the Metering Equipment via the HyNet driver.

Conversely data received from the HyNet driver is processed by the driver modules and sent to the Access Point via the De/Mux Communication Module. This module is also responsible for reacting to error messages received from the De/Mux module, by informing all the other modules in the driver that require this information, about the error.

F2: The function of the **Optimization Module** is responsible for minimizing the size of the data sent over the De/Mux interface. The reverse is of course true at the receiving end where this module would be responsible for rebuilding the minimized data to its original form. The data could be compressed before it is sent via the De/Mux interface by this module. If viable this module will also remove any redundant data from the packets in order to minimize the bandwidth used by the data sent over the PLC. Conversely it will rebuild the data to its original state if received in the minimized or compressed state.

F3: The function of the **Application Module** is to provide processing capability for the driver at the Node. This functionality meets the requirement of scalability for future protocols. This processing capability could be used to process raw data to obtain information, for example using meter readings to calculate the flow rate of the heaters. This is required in certain instances for translation purposes, for example some protocols like the S0 metering protocol require this translation module Within REMPLI however protocols like IEC 62056, M-Bus and IEC 60870-5-101 do not require this module and will be tunnelled through the REMPLI network. In the case when conversion is required the function of the Application Module is to convert between different protocols, received over the De/Mux interface into the metering protocol. The reverse is also true when sending the data from the meters back to the Application Server, then this module will be responsible for converting the metering protocol into one of the tunnelled protocols to be transported back to the Application Server.

F4: The function of the **Access Control Module** is control and limiting access only from known Application Servers to known installed meters at the Node side. This includes requests made by the Application Servers and alarm signals sent to

Application Server. The access control is achieved by accessing a database containing all the known addresses of installed meters and Application servers. This module should also update and maintain the access control database. If the data sent over the De/Mux interface is compressed, then it will be difficult to apply this module higher up in the communication chain, for example just after the De/Mux communication module, because the data will be unreadable. This scenario is preferable because less processing of data is done before it is accepted or rejected. Having this module after the protocol converter makes the system susceptible to denial of service attacks however. Figure 5.4 is a general structure and these scenarios should be taken into account when deciding where to implement the access control module. The limitations on where this module could be placed are also dependent on the functionality of the Application Module.

F5: The function of the **Protocol Stack at Layer 7** is to process and then manage data relevant to layer seven of the protocol stack of the metering protocol. This is only needed if some additional processing is required due to the layer 7 information stored within the APDU or if proprietary operations must be performed by the Node-Side Driver on layer 7 data.

F6: The function of the **Protocol Stack at Layer 1 and 2** is used for managing the headers, transmission services and functions provided at these levels of the M-Bus protocol. These include the 4 frame formats supported by M-Bus and the transmission rules applied to the M-Bus protocol. Layer 1 functionality is only required if no driver support is provided.

F7: The function of the **Protocol Specific Hardware Driver for HyNet Module** is to communicate with the HyNet driver in order to communicate directly with the M-Bus Metering Equipment.

The resultant driver structure with a detailed expansion is given in figure 5.5. In this figure the modules situated on the left hand side within the light grey area are modules directly responsible for and involved within the communication process. The APDUs flow through all of these modules. Modules that interface software modules outside the data flow process are placed on the right hand side of the driver modules.

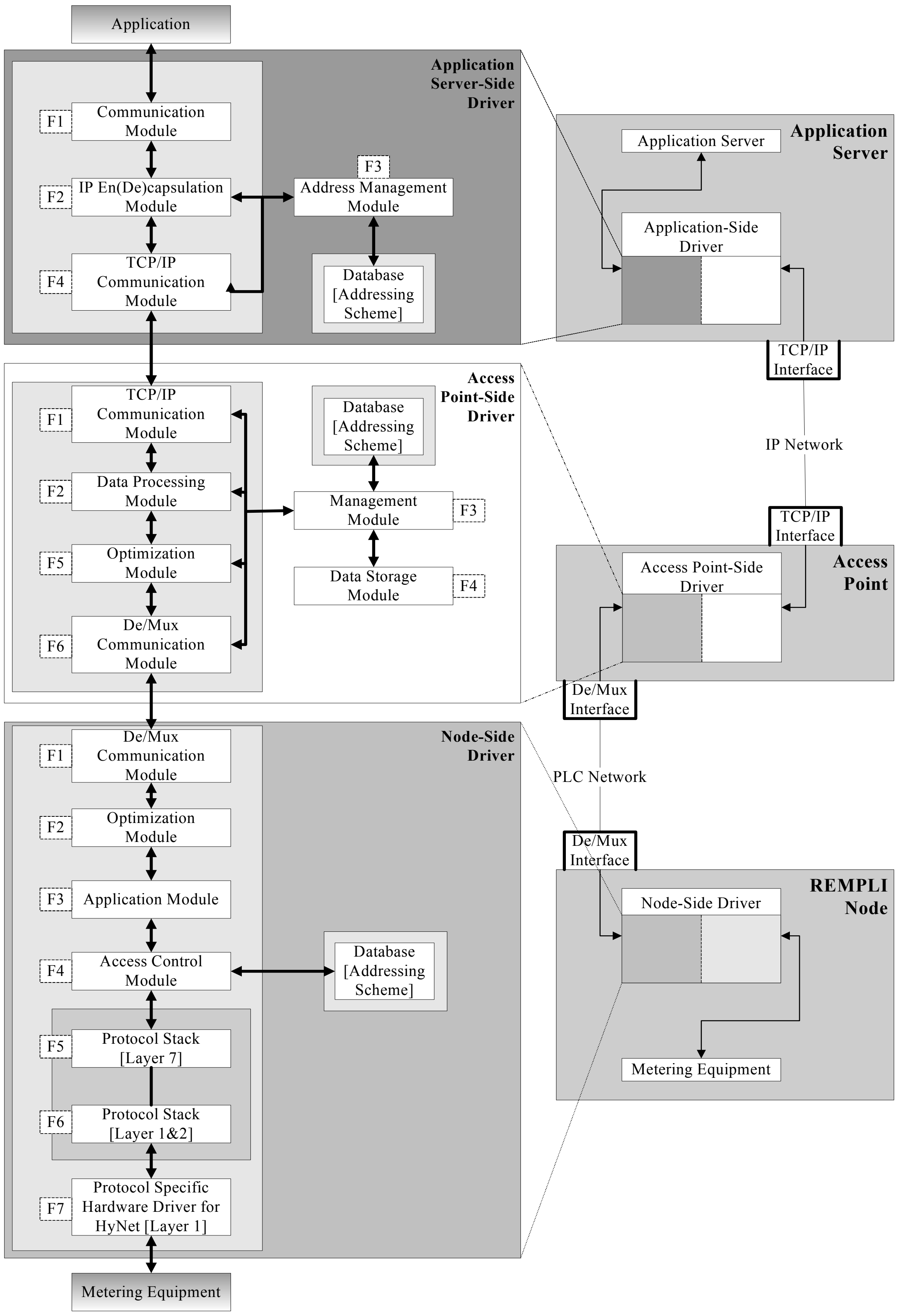


Figure 5.5. The driver structure for the REMPLI Driver, with functional blocks.

Chapter 6

6 M-BUS DRIVER ARCHITECTURE

As a proof of concept the general driver structure given in chapter 5 is applied to the M-Bus protocol. The resultant driver structure is implemented and tested in a real world scenario. As mentioned in chapter 5, several modules in the general driver architecture are protocol specific and the implementation of them depends on the protocol being implemented. After the general REMPLI driver structure has been

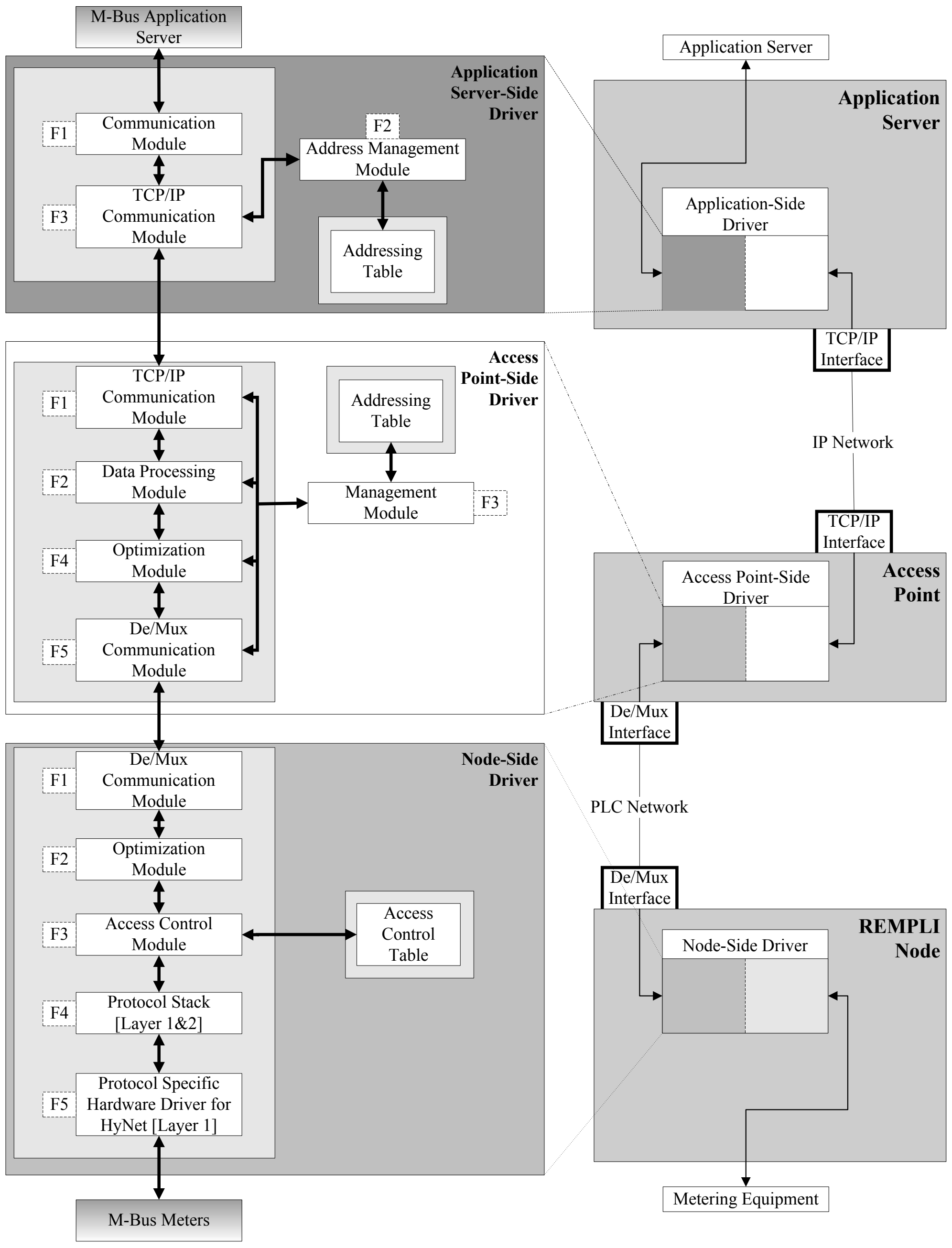


Figure 6.1. The driver structure for the M-Bus protocol, with functional blocks.

adapted and implemented for the M-Bus protocol, the resultant driver structure is given in figure 6.1. The functioning of the M-Bus Protocol Driver is explained at the hand of an example as discussed in the sections below.

6.1 DATA FLOW EXAMPLE OF A SINGLE FRAME M-BUS REQUEST/RESPONSE GENERATED BY THE APPLICATION SERVER

This section describes the flow and processing of data for a simple request/response sequence from the Application Server to and from an M-Bus meter.

In this example the Application Server generates an M-Bus request for class 2 user data from M-Bus meter number 3, connected to REMPLI Node 27. The Application Server generates an APDU which contains an M-Bus frame that is given in figure 6.2.

10_{16}	$5B_{16}$	03_{16}	$6E_{16}$	16_{16}

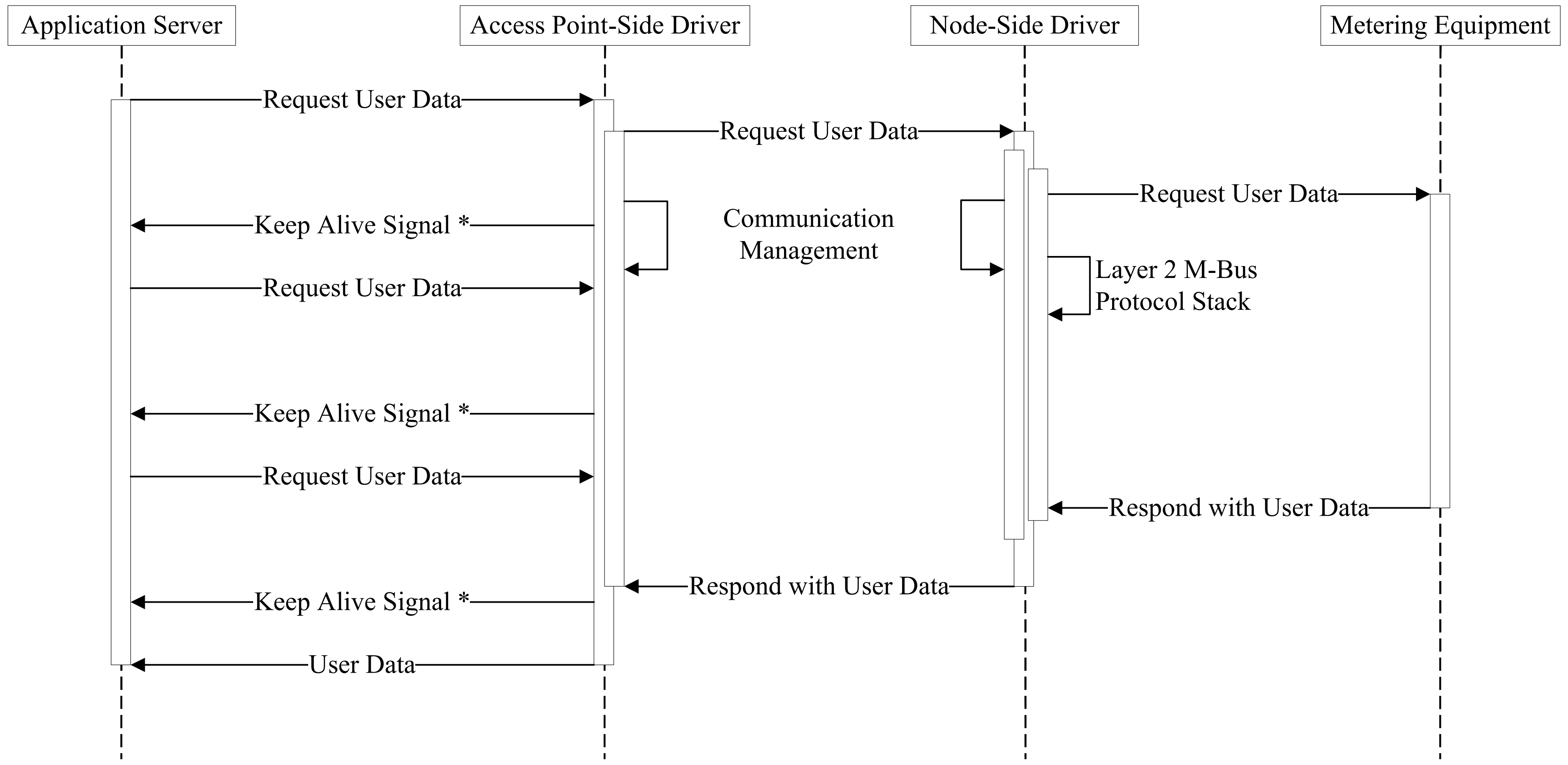
Figure 6.2. The M-Bus frame generated by the Application Server, all values are given in hexadecimal format.

The request is sent to the Access Point over an IP network where it is processed by the Access Point-Side Driver and then forwarded to the REMPLI Node over a PLC network. At the REMPLI Node the request is processed by the Node-Side Driver and forwarded to the Metering Equipment. After the Metering Equipment has processed the request, it sends the response back to the Node-Side Driver which processes the data and then forwards it to the Access Point over the PLC network. The response generated by the Metering Equipment is dependent on the configuration, which is setup by the application software. The meter in this example is setup to respond with the data shown below in figure 6.3. All values are given as hexadecimal values.

68	1F	1F	68	08	03	72	78	56	34	12	24	40
01	07	55	00	00	00	03	13	15	31	00	DA	02
3B	13	01	8B	60	04	37	18	02	18	16		

Figure 6.3. The M-Bus frame generated by the Application Server, all values are given in hexadecimal format.

After receiving the data from the meter, the Node-Side Driver forwards the M-Bus frame to the Access Point via the PLC/GPRS network interface. At the Access Point the meter's response is processed by the Access Point-Side Driver. The Access Point-Side driver then forwards the metering response to the Application Server-Side Driver over the IP network, which sends the meter's response to the Application Server. The Application Server processes and stores the response it received. For this example it is implied that the IP address used by the Application Server is predefined. Figure 6.4 gives the sequence diagram for the communication process described above.



* Required for Successful Communication

Figure 6.4. Sequence diagram for a single frame request/response generated by an Application Server.

6.1.1 Data Flow From Application Server To TCP/IP Communication Module

The communication procedure at the Application Server side commences with the Application Server initiating a request for class 2 data from meter 3 situated at Node 27 via the M-Bus Application Server-Side Driver. When the request is made by the Application Server it sends the generated request to the corresponding REMPLI Protocol Driver address. For this example the Application Server request is forwarded to the M-Bus Protocol Driver. No errors occur during the entire transmission period for this scenario. Even though this example shows an M-Bus request/response procedure, the M-Bus driver structure is derived from the general REMPLI Protocol Driver structure, which is used by all REMPLI communication protocols and thus representative of the general REMPLI Protocol Driver. For this example the request is sent to the M-Bus Driver address and is as follows:

1. The **Application Server** generates an M-Bus request to readout class 2 user data from meter nr. 3, which is connected to a Node with a specific address (RNA = 27). This request is in the form of an APDU (Application Protocol Data Unit). It is assumed that this APDU contains the entire M-Bus frame, which includes all layer 2 and layer 7 data needed for the request. For this example it is also assumed that the response from the metering device will fit into a single M-Bus frame. It could also be assumed that the Application Server generated request will also contain some other data apart from the M-Bus frame in the APDU, from which the destination RNA (REMPLI Node Address) can be derived. The Application Server forwards the request for class 2 user data to the M-Bus Application Server-Side Driver by sending the request to the M-Bus Driver address.

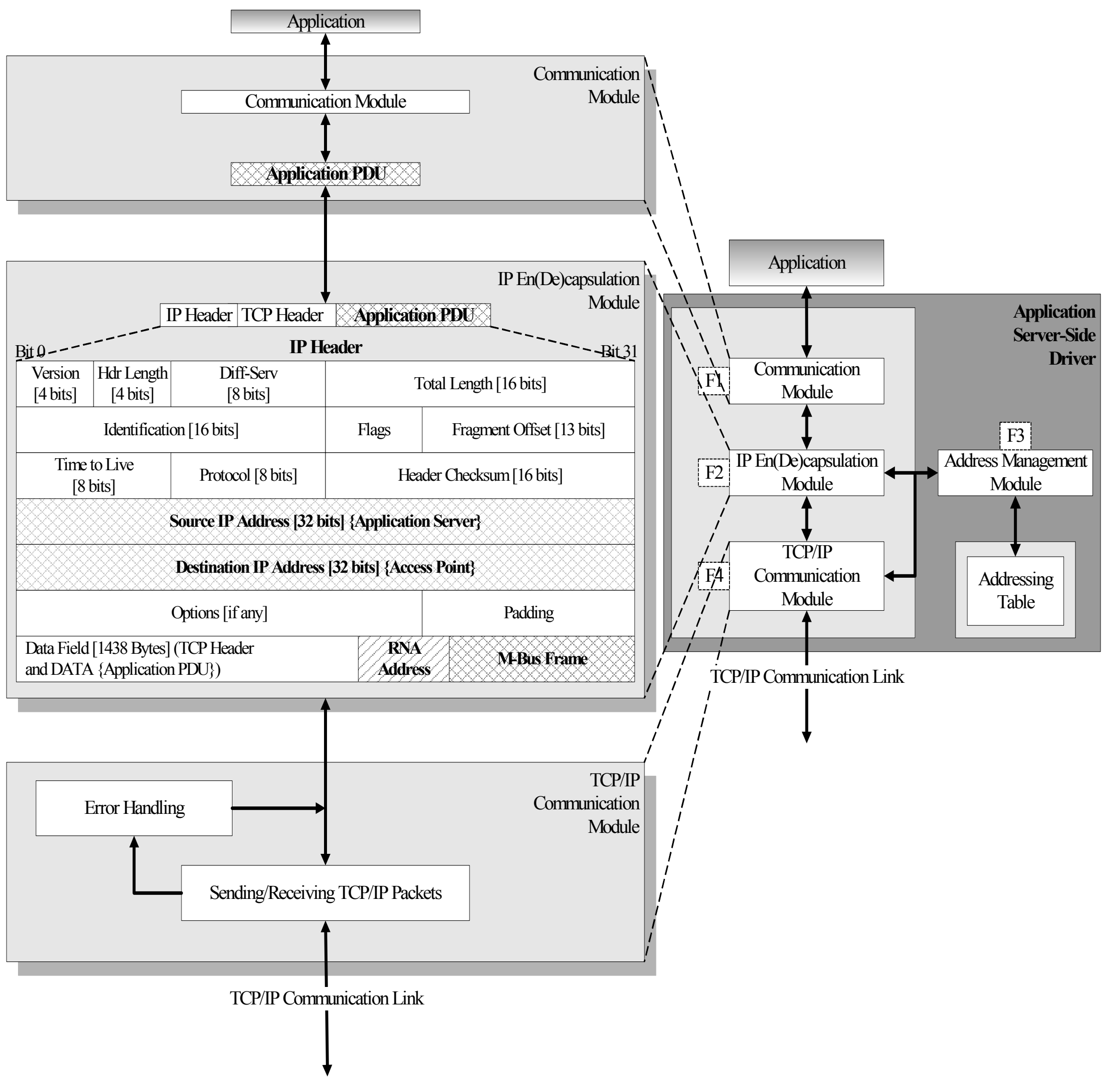

Application
Communication Module
Communication Module
Application PDU
IP En(De)capsulation Module
IP Header
TCP Header
Application PDU
Bit 0
IP Header
Bit 31
Version [4 bits]
Hdr Length [4 bits]
Diff-Serv [8 bits]
Total Length [16 bits]
Identification [16 bits]
Flags
Fragment Offset [13 bits]
Time to Live [8 bits]
Protocol [8 bits]
Header Checksum [16 bits]
Source IP Address [32 bits] {Application Server}
Destination IP Address [32 bits] {Access Point}
Options [if any]
Padding
Data Field [1438 Bytes] (TCP Header and DATA {Application PDU})
RNA Address
M-Bus Frame
Error Handling
TCP/IP Communication Module
Sending/Receiving TCP/IP Packets
TCP/IP Communication Link
Application
Application Server-Side Driver
F1
Communication Module
F2
IP En(De)capsulation Module
F3
Address Management Module
F4
TCP/IP Communication Module
Addressing Table
TCP/IP Communication Link

Figure 6.5. Functional block diagram for the M-Bus driver at the Application Server.

2. The Application Server generated request is received by the Application Server-Side Driver via the **Communication Module**, which is responsible for communicating with the Application Server. After the Communication Module has successfully received the APDU, it forwards it to the TCP/IP Communication Module.

3. The **TCP/IP Communication Module** takes the entire APDU and extracts the RNA and M-Bus frame before encapsulating it into a TCP/IP packet as illustrated in figure 6.5. It uses the destination Access Point IP address provided by the Application Server to add the destination IP address into the IP headers. In this example the destination address is predetermined and inserted into the IP header.

If the Application server does not include the destination Access Point IP address then the Access Point IP address will have to be derived from the destination address provided by the Application Server. To provide this functionality the TCP/IP Communication Module interfaces with the Address Management Module, this will provide the required Access Point destination IP address[5]. The TCP/IP packet is then sent over the IP network to the predetermined address. The IP packet then traverses across the IP network until it reaches the destination Access Point-Side Driver.

6.1.2 Data Flow from the TCP/IP Communication Module at the Access Point to the De/Mux Communication Module at the Access Point

The data flow process is further explained by the following section which describes the functionality of the **Access Point-Side Driver**. A detailed functional block diagram is given in figure 6.8.

1. The TCP/IP packet is received by the **TCP/IP Communication Module** at the Access Point-Side Driver. Once the IP packet is successfully received it can be passed on to the Data Processing Module.

2. The **Data Processing Module** is used to remove the TCP/IP headers from the TCP/IP packet, if not provided by the operating system, or TCP/IP communication libraries. The Data Processing Module now extracts the necessary data out of the payload data, i.e. the M-Bus frame and the RNA, and sends it to the Management Module. This data is required to maintain the status of the current request/response sequence at the Access Point. After processing the APDU the Data Processing Module forwards the M-Bus frame to the Optimization Module.

3. The **Management Module** is used to maintain the status of the current request/response sequence. Because a thread is used to maintain the communication link, the only status information required by this module is a timer, which is used to determine when a time-out has occurred. The timer is

[5] Address translation is discussed in section 6.4.

started and terminates the communication process if a time-out occurs to make the communication link available for the next Application Server request. This module could also be used to maintain logs of requests and responses processed at the Access Point for auditing purposes, this is however not implemented in the M-Bus driver. In REMPLI so called "Keep-alive" signals are used to maintain the communications link between the Application Server and the Metering Equipment. This module is also responsible for managing these "Keep-alive" messages at the Access Point. These "Keep-alive" signals are however protocol specific and the use of keep-alive signals depend on whether the protocol implemented supports such functionality. The M-Bus protocol does not support this functionality and therefore does not implement a keep-alive signalling scheme.

4. The **Optimization Module** is used to minimize the size of the data sent over the PLC network. Therefore after receiving the M-Bus frame it removes some of the level 2 headers in order to minimize the size of the frame. The role of the Optimization Module for the M-Bus protocol is discussed in section 6.7 of this document. The Optimization Module will perform similar functions to other protocols implemented within REMPLI. The class 2 user data request is a small data packet only containing 5 Bytes, of which 3 headers could be removed to reduce it to 2 bytes. The resultant M-Bus frame is given in figure 6.7. It is not worth the effort to compress the rest of the data if there is no reduction in size to gain from the data compression, a gain of at least one byte needs to be achieved to justify using compression. This function is available for longer datagrams which might contain metering logs, with long sequences of values. In this example data compression is not justified and therefore not implemented.

$5B_{16}$	03_{16}

Figure 6.7. The reduced M-Bus frame generated by the Optimization Module, all values are given in hexadecimal format.

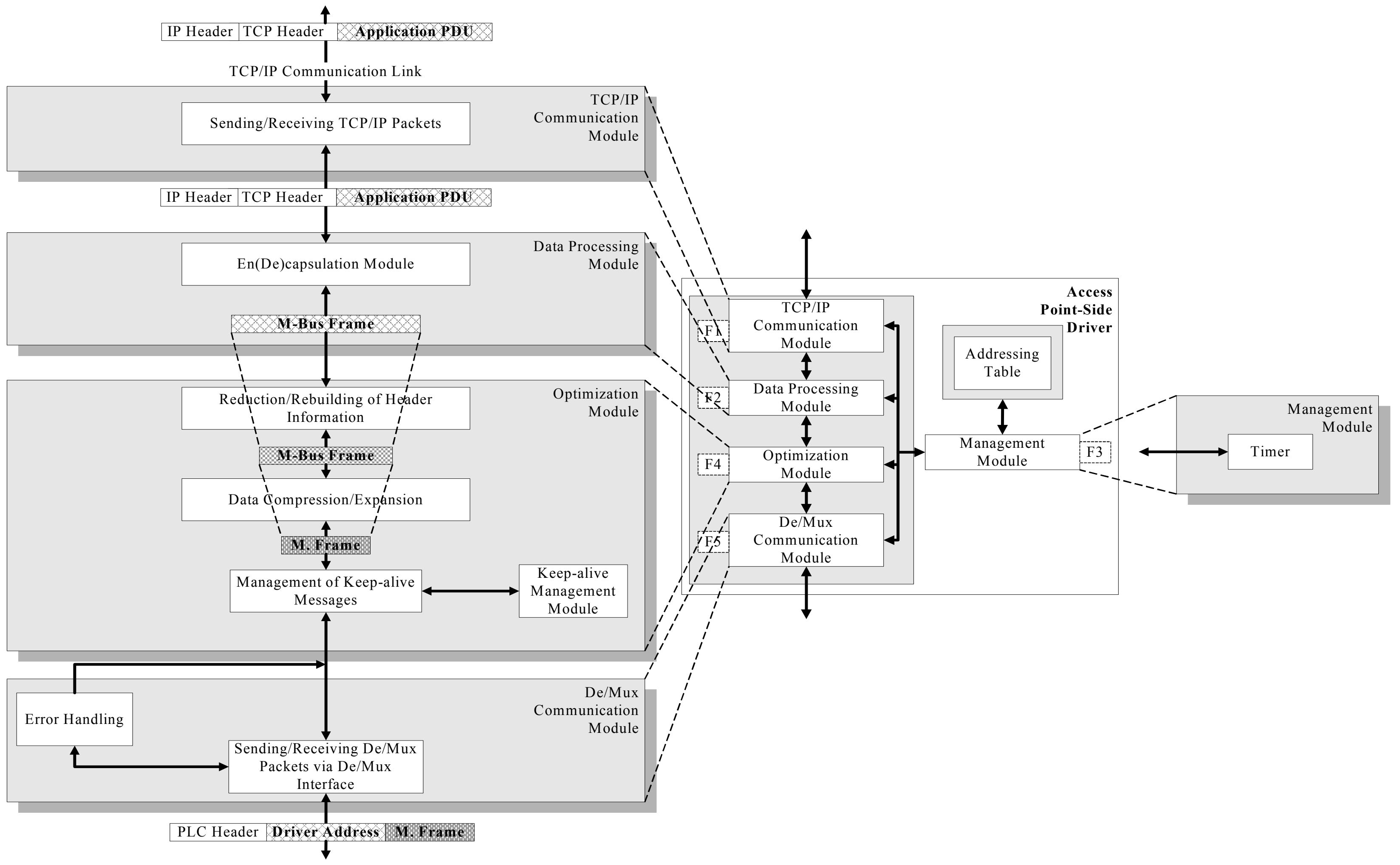

IP Header
TCP Header
Application PDU
TCP/IP Communication Link
Sending/Receiving TCP/IP Packets
TCP/IP Communication Module
IP Header
TCP Header
Application PDU
En(De)capsulation Module
Data Processing Module
M-Bus Frame
Reduction/Rebuilding of Header Information
Optimization Module
M-Bus Frame
Data Compression/Expansion
M. Frame
Management of Keep-alive Messages
Keep-alive Management Module
Error Handling
De/Mux Communication Module
Sending/Receiving De/Mux Packets via De/Mux Interface
PLC Header
Driver Address
M. Frame
Access Point-Side Driver
F1
TCP/IP Communication Module
F2
Data Processing Module
F4
Optimization Module
F5
De/Mux Communication Module
Addressing Table
Management Module
F3
Management Module
Timer

Figure 6.8. Functional block diagram for the M-Bus driver at the Access Point.

5. The compressed or minimized M-Bus frame is now passed on to the **Management of "Keep-alive" Packets Module**, which sends a signal to the "Keep-alive" Management Module to inform it of the status of the packet to be updated. The Management of Keep-alive Packets Module optimizes the bandwidth utilization of the PLC network by combining keep-alive data with APDU data sent via protocol drivers that have free, available bandwidth to carry the extra data. This concept and functionality of optimizing the PLC bandwidth is discussed in greater detail in section 6.7.

6. After the Access Point Management module has been informed the minimized APDU is sent over the PLC network via the De/Mux interface, at the **De/Mux Communication Module**.

6.1.3 Data Flow from the De/Mux Communication Module at the Node-Side Driver to the Metering Equipment

The data flow process is further explained in this section which describes the functionality of the REMPLI Node-Side Driver. A detailed functional block diagram is given in figure 6.9.

1. The minimized M-Bus frame is received at the Node-Side driver via the De/Mux interface at the **De/Mux Communication Module**. If the M-Bus frame is compressed before it is sent over the PLC network it is not possible to determine the destination address until the packet has been restored to its uncompressed state. Thus no access control can take place until the M-Bus frame has been uncompressed.

2. The compressed or minimized M-Bus frame is now passed on to the **Management of "Keep-alive" Packets Module**, which sends a signal to the "Keep-alive" Management Module to inform it of the status of the packet. The

"Keep-alive" Management Module will have to start a timer[6] once the packet is received.

3. The minimized M-Bus frame is forwarded to the **Data Compression/Expansion Module** where the payload will be expanded to its original state if it was compressed at the Access Point side. For this example data compression was not implemented and no action is taken at this module.

4. The **Reduction/Rebuilding of Header Information** module now rebuilds the 3 M-Bus headers that were dropped at the Access point, to restore the M-Bus frame to its original state, as conceived at the Application Server. The M-Bus frame is then forwarded to the Access Control Module for processing.

5. The **Access Control Module** compares the origin of the M-Bus frame and the equipment address with a list of known Application Servers and equipment addresses and either accepts or rejects the packet. If the M-Bus frame is rejected it is dropped and the "Keep-alive" Management Module is informed, if the M-Bus frame is accepted it is forwarded to the Protocol Stack.

6. The **Layer 2 Protocol Stack** manages the transmission of data between the REMPLI Node-Side Driver and the Metering Equipment[7].

7. The M-Bus frame is sent to the Metering Equipment via the **HyNet Driver**. The HyNet Driver translates the software data packet into protocol specific hardware data and transmits it directly to the metering equipment connected to the bus.

[6] Discussed in section 6.6 of this document.

[7] Discussed in section 1.4 of this document.

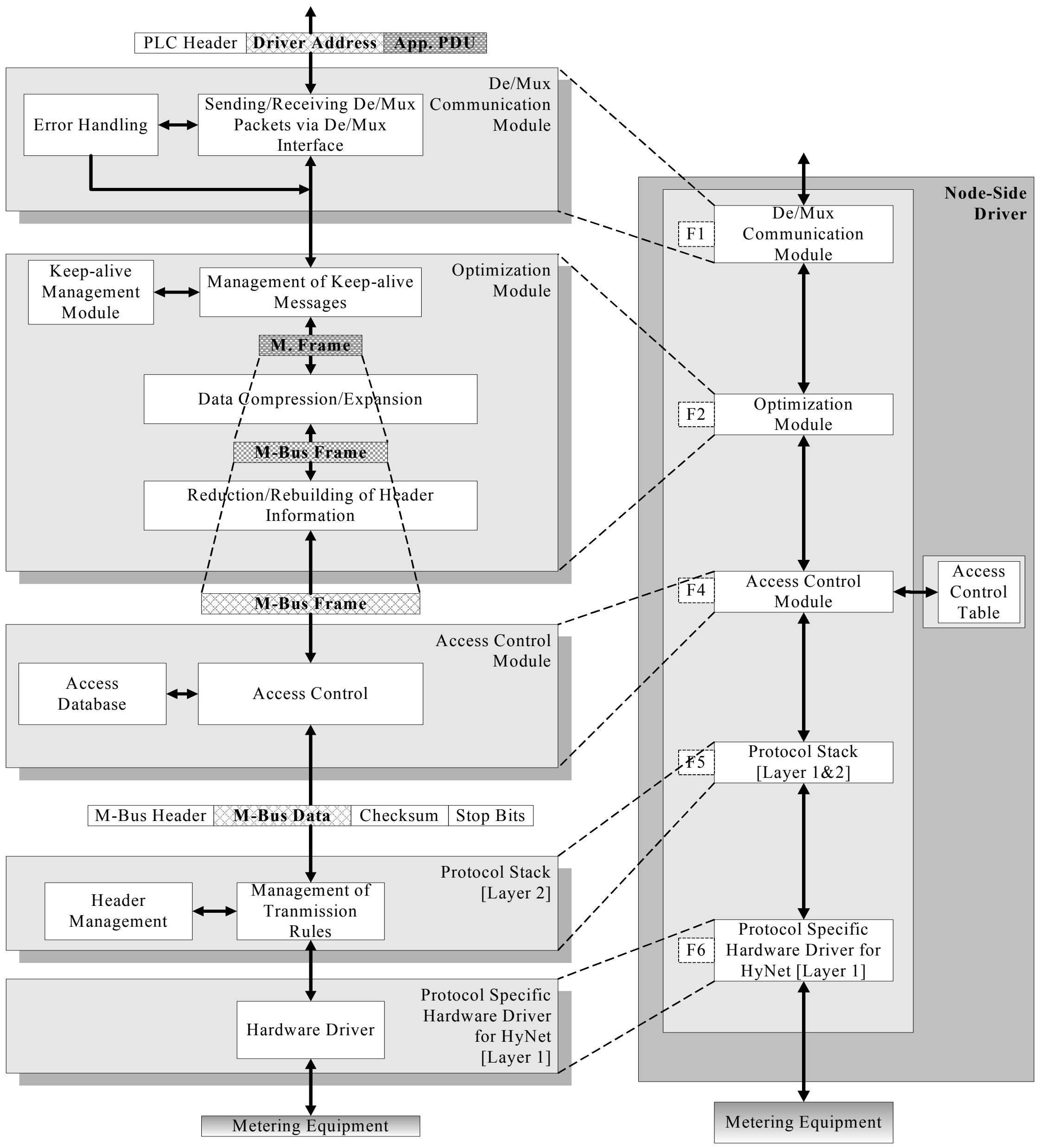

PLC Header
Driver Address
App. PDU
De/Mux Communication Module
Error Handling
Sending/Receiving De/Mux Packets via De/Mux Interface
Keep-alive Management Module
Management of Keep-alive Messages
Optimization Module
M. Frame
Data Compression/Expansion
M-Bus Frame
Reduction/Rebuilding of Header Information
M-Bus Frame
Access Control Module
Access Database
Access Control
M-Bus Header
M-Bus Data
Checksum
Stop Bits
Protocol Stack [Layer 2]
Header Management
Management of Tranmission Rules
Protocol Specific Hardware Driver for HyNet [Layer 1]
Hardware Driver
Metering Equipment
Node-Side Driver
F1
De/Mux Communication Module
F2
Optimization Module
F4
Access Control Module
Access Control Table
F5
Protocol Stack [Layer 1&2]
F6
Protocol Specific Hardware Driver for HyNet [Layer 1]
Metering Equipment

Figure 6.9. Functional block diagram for the M-Bus driver at the REMPLI Node.

6.1.4 Data Flow from the Metering Equipment to the De/Mux Communication Module at the Node-Side Driver.

The following section discusses the flow of the response containing the user 2 data which is generated by the Metering Equipment and forwarded to the Application Server.

1. Once the Metering Equipment has replied to the request with a response M-Bus frame it is sent to the **Protocol Stack [Layer 2]** at the Node via the HyNet driver. The layer 2 protocol stack at the M-Bus Node-Side Driver module ensures that the frame that was received is correct by checking the length of the, the frame checksum and the parity bit. Once the information in the frame is validated it is forwarded to the Access Control module. In this example the resultant M-Bus frame is as given in figure 6.3.

2. The **Access Control Module** checks that meter 3 on REMPLI node 27 is allowed to be forwarded to the Application Server and then either accepts or rejects the frame. If the frame is allowed to be forwarded to the Application Server then it is sent to the Reduction/Rebuilding of Header Information Module.

3. The **Reduction/Rebuilding of Header Information Module** then removes any headers that can be rebuilt at the Access Point side as explained in section 6.7.1 and forwards the reduced M-Bus frame to the Data Compression/Expansion Module**.** The reduced M-Bus frame is given in figure 6.10.

				08	03	72	78	56	34	12	24	40
01	07	55	00	00	00	03	13	15	31	00	DA	02
3B	13	01	8B	60	04	37	18	02				

Figure 6.10. The reduced M-Bus frame generated by the Metering Equipment, all values are given in hexadecimal format.

4. The **Data Compression/Expansion Module** does not play any role in this example as there is no data compression/Expansion applied.

5. The **Management of "Keep-alive" Signals Module** then informs the Node Side "Keep-alive" Management Module of the intent to forward the message to the Access Point and the timer that was started at the beginning of the communication at this module is stopped. After the timer has stopped the M-Bus frame is sent to the De/Mux Communication Module.

6. The minimized M-Bus frame, as given in figure 6.10 is sent over the PLC network via the De/Mux Interface, by the **De/Mux Communication Module**.

6.1.5 Data Flow from the De/Mux Communication Module at the Access Point to the TCP/IP Communication Module at the Access Point.

The following section describes the flow of the APDU from the Access Point-Side Driver to the IP network.

7. The **De/Mux Communication Module** receives the minimized M-Bus frame from the De/Mux interface and forwards it to the Management of "Keep-alive" Messages Module.

8. The **Management of "Keep-alive" Messages Module** sends the relevant data to the "Keep-alive Management module, which updates the "Keep-alive" status at the Access Point by terminating the timer which was started when the request for user 2 data was received from the Application Server. The M-Bus frame is sent to the Data Compression/Expansion Module.

9. If the M-Bus frame was compressed the **Data Compression/Expansion module** is responsible for expanding the data to its original uncompressed state. This module was not utilized during this example and thus no expansion of data is required.

10. The uncompressed M-Bus frame is now forwarded to the **Reduction/Rebuilding of Header Information**, where all the dropped headers are restored to their original state, as shown in figure 6.3. This includes the 2 start bytes, 2 length bytes, the checksum byte and the stop byte.

11. The **Data Processing Module** now retrieves the necessary data out of the Management Module at the Access Point-Side driver, i.e. The Application Server IP-address, if not supported by the underlying TCP/IP communication module and forwards it to the TCP/IP Communication Module. The Data Processing Module then encapsulates the M-Bus frame into a TCP/IP packet and sends it to the TCP/IP Communication Module.

12. The **TCP/IP Communication Module** forwards the TCP/IP encapsulated M-Bus frame over the TCP/IP link to the Application Server.

6.1.6 Data Flow from the TCP/IP Communication Module at the Application Server-Side Driver to the Application Server

The following section describes the data flow of the response M-Bus frame at the Application Server's side, by the Application-Side Driver.

13. The **TCP/IP Communication Module** at the Application-Side Driver receives the TCP/IP packets, decapsulates the data and then forwards the response M-Bus frame to the Communication Module.

14. The **Communication Module** sends this resultant M-Bus frame in the form of an APDU to the Application Server, which takes the appropriate action.

6.2 DATA FLOW EXAMPLE OF A MULTI-FRAME M-BUS REQUEST/RESPONSE GENERATED BY THE APPLICATION SERVER

This section describes the flow of an M-Bus Application Server generated request for user 2 data from meter number 5, which is connected to REMPLI Node 15. The

Metering equipment is configured to respond with user data that is 531 bytes long, which results in the meter returning 3 M-Bus frames: two with 252 bytes of user 2 data and the last M-Bus frame containing 27 bytes of user 2 data. The sequence diagram for the communication process given in this example is given in figure 6.11.

During a multi-frame request by the Application Server the communication process is essentially the same as the process described in section 6.1 therefore this section gives a brief overview of the data flow from the Application Server to the Metering Equipment and the resultant response. The functional blocks and functions performed by each module remain the same as given in section 6.1.

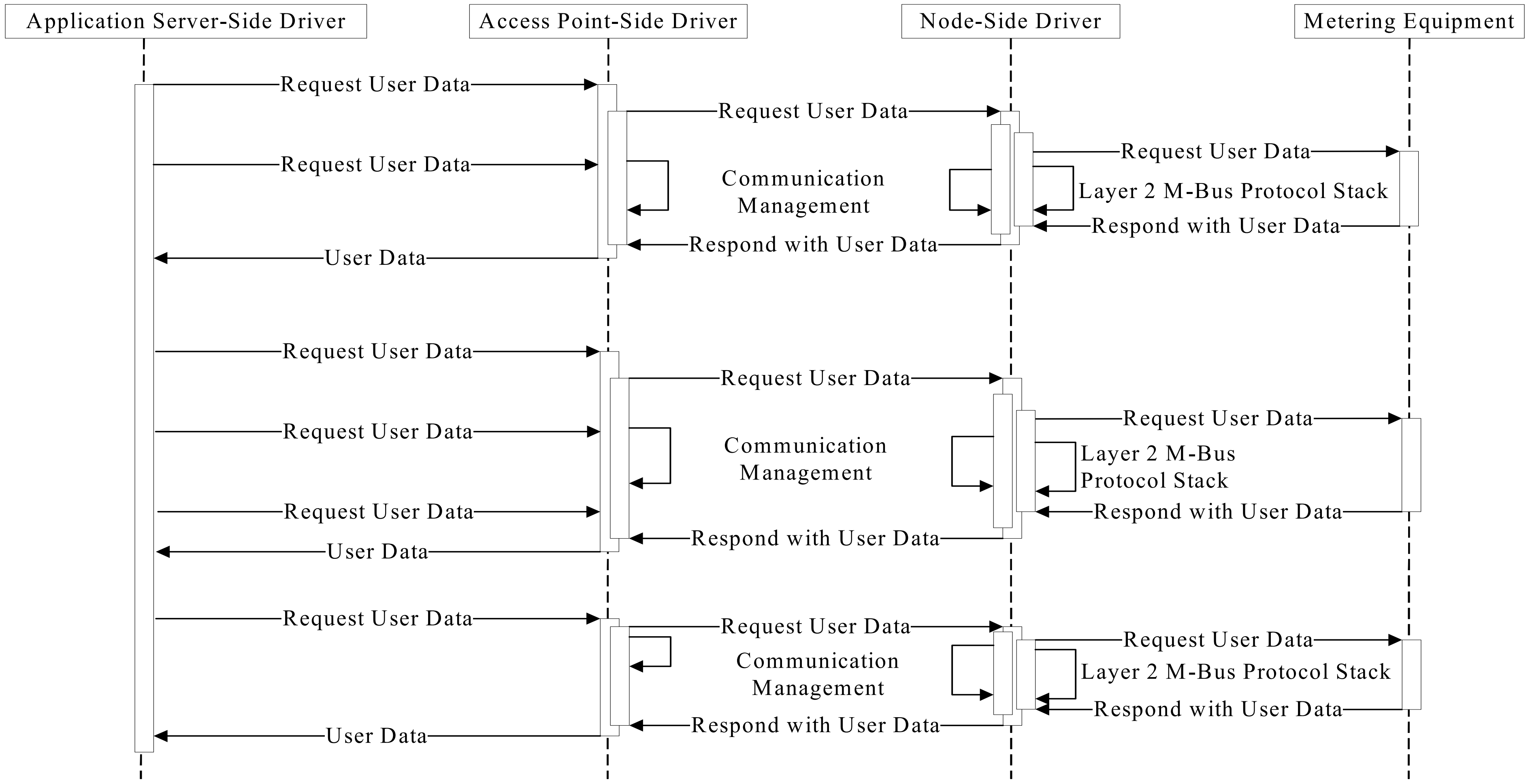


Figure 6.11. Sequence diagram of a multi-frame M-Bus request/response generated by an Application Server.

6.2.1 Multi-frame Data Flow from Application Server to the Metering Equipment and Response, figure 6.11.

This section gives brief description of the flow of data, which occurs when the M-Bus Application Server request user 2 data and the M-Bus Metering Equipment responds with a multi frame response. The functional communication is the same as given in 6.1 but the process repeats itself until the entire response has been received by the Application Server.

The Application Server generates the request for user 2 data from M-Bus meter nr 5, which is connected to REMPLI Node 15. The Application Server generated M-Bus request frame is given in figure 6.12 as shown below.

10_{16}	$5B_{16}$	05_{16}	$6E_{16}$	16_{16}

Figure 6.12. The M-Bus frame generated by the Application Server, all values are given in hexadecimal format.

The Application Server-Side Driver receives the APDU from the Application Server in which user 2 data is requested. The Application Server Side-Driver encapsulates the APDU into a TCP/IP packet and forwards it to the predetermined Access Point. The Access Point-Side Driver receives the TCP/IP packet via a TCP/IP interface. The Access Point-Side Driver performs the necessary processing and informs the required modules of the new request, as given in 6.1.2. The Access Point-Side Driver then forwards the reduced M-Bus frame $5B05_{16}$ to the Node-Side Driver via the PLC interface. At the Node-Side Driver the minimized M-Bus frame is restored to its original form and the layer 2 protocol stack is then responsible for managing the communication between the M-Bus metering equipment and the Node-Side Driver by requesting the user 2 data from the Metering Equipment as given in the M-Bus frame. The communication between Metering Equipment and the Node-Side Driver takes place via the HyNet driver. While this request takes place between the Node-Side Driver and the Metering Equipment, the time elapsed during this data flow process has exceeded the time allowed for the M-Bus response to be sent to the Application Server, thus in accordance with the M-Bus protocol the Application sends a second request for user 2 data from the same M-Bus meter located at REMPLI Node 15. Once this request is received at the Access Point the Management Module determines that the request is a duplicate of the current request and does not forward it over the limited bandwidth PLC network, thus saving PLC bandwidth. Once the M-Bus response is received by the Node-Side Driver it is processed, reduced and forwarded to the Access Point-Side Driver via the PLC network. The M-Bus response contains 252 bytes of user data and a Control (C) field, Address (A) field and a Control Information (CI) field. The Access Point-Side Driver receives the M-Bus response and informs the Management of Communication Module of the event and performs the required tasks and forwards the data to the Application Server-Side Driver via the TCP/IP network. The Application Server checks if the M-Bus response is part of a multi-frame response by checking the CI field of the M-Bus response frame. Knowing that the frame is part of a multi-frame response allows the Application Server-Side Driver to forward the response M-Bus frame to the Application Server and wait for the second response from the Application Server for the second frame of the multi-frame response.

The Application Server generates a new request for the remaining metering response by toggling the FCB-bit in the Control field. The Application Server-Side Driver then forwards this request to the Access Point-Side Driver with the flow of data the same as explained in the section above. The only difference is that during this instance the time lapse during the traversal of the PLC network is such that the Application Server repeats the request for the user data 3 times before the response is received. Once the response is received by the Application Server-Side Driver it checks again to see if another response is expected, in this case the response is not finished yet and another response is required, thus the Application Server-Side Driver waits for the Application Server to send the next request for the user data from the M-Bus meter.

The next request is generated by the Application Server and passed on to the M-Bus meter via the Access Point-Side Driver and the Node-Side Driver. The delay caused in traversing the network is minimal in this instance and the final M-bus response is received by the Application Server before another request for this response has to be generated and sent to the meter. The fact that this is the final response allows the Application Server-Side Driver to terminate further communication with the corresponding instance of the Access Point-Side driver, while the Node-Side driver terminates itself through a timer which controls this communication, thus minimizing the traffic flow on the PLC network.

Due to variability in the PLC network the traversal time of a data packet can not be predetermined and as given in this example has an impact on the number of requests for a response telegram from the Application Server. It is however critical for successful transmission of M-Bus data across the REMPLI system that the underlying network strictly adhere to the QoS requirements set by the M-Bus protocol. During this example the baud rate of the Application Server was set at 300 bits/sec., in order to maximize the time window in which the M-Bus response could be generated and returned to the Application Server. The baud rate on the physical connection between the REMPLI Node and the M-bus meter was set at 9600 baud. This results in a faster request/response time at the REMPLI Node. The resultant setup of the communication system optimally utilizes the allotted time for a response by minimizing the time of request and response between the REMPLI Node and the physical M-Bus meter and maximizing the time frame for the request/response between the Application Server and the REMPLI Node.

6.3 DATA FLOW EXAMPLE OF M-BUS REQUEST/RESPONSE GENERATED BY THE APPLICATION SERVER WITH TRANSMISSION FAULTS

The communications process remains the same as given in the examples above, but if a communication fault occurs, or the underlying network is dropped, the communication process is described in the sequence diagram of the communication process for the M-Bus Protocol Driver is given in figure 6.13. When a transmission fault occurs, the Application Server is responsible for requesting the required measurement value again. This can happen immediately after the fault has occurred or at a later stage, depending on the configuration of the Application Server.

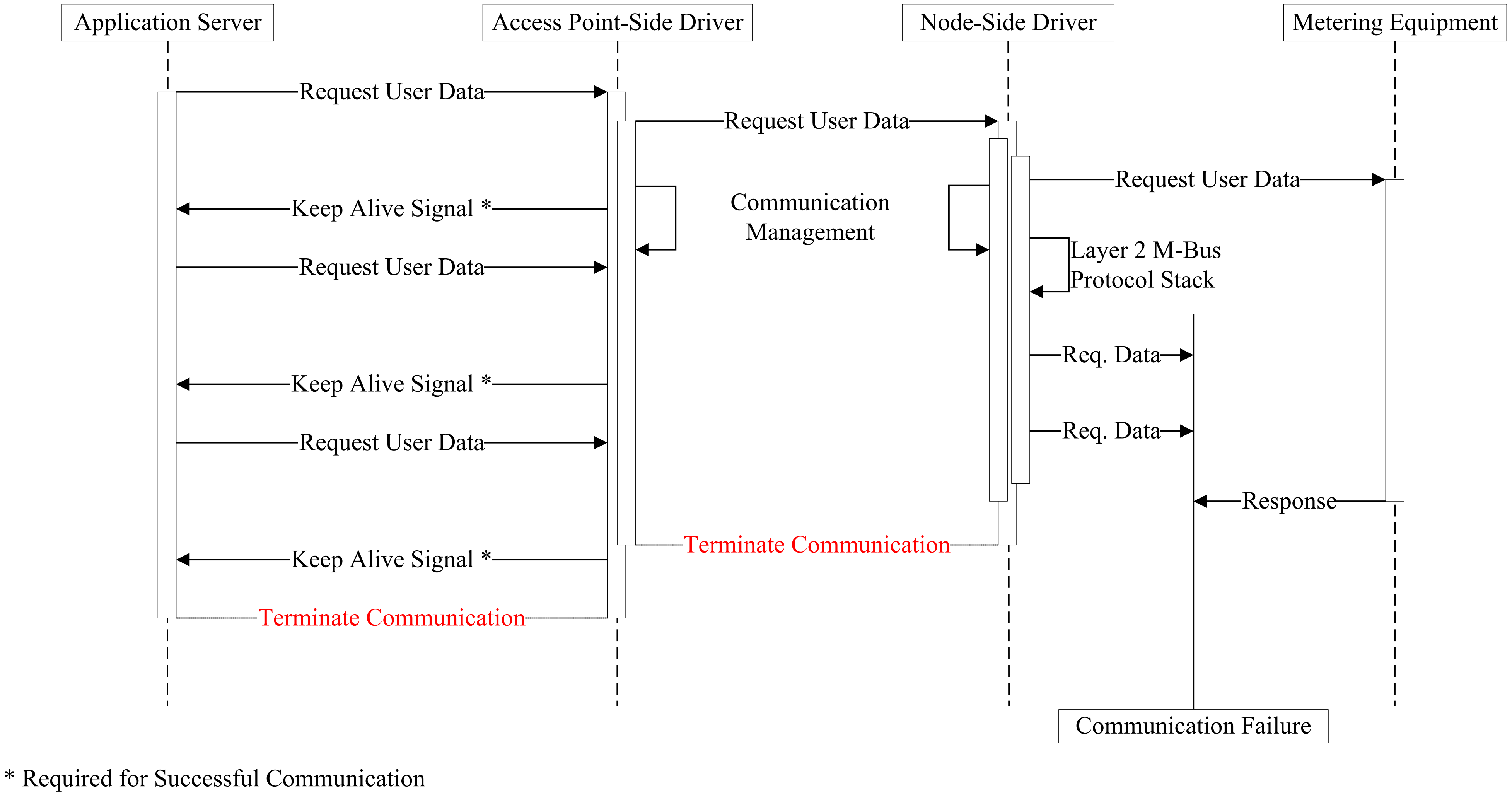


Figure 6.13. Sequence diagram in case of faults during the communication process.

6.4 ADDRESS TRANSLATION

The format of the Application Server destination address is implementation specific and the derivation algorithm will depend on the format of the Application Server destination address. At the moment the example given in section 3, assumes a predetermined IP address is used, an alternative to using one predefined destination TCP/IP address is to create a list which is stored at the Application Server and contains all the TCP/IP address of all the Access Points connected to the REMPLI network. There are currently two implementation options:

- Deriving the Access Point IP address and RNA with a general algorithm
- Using a table with stored values to obtain the Access Point IP address and RNA.

6.4.1 General Algorithm

The first implementation option is to implement a general algorithm to derive an Access Point IP address and an RNA from the Application Server destination address. This method requires little storage space and is a general way of deriving Access Point destination addresses. But it is questionable if a general algorithm could be implemented to derive Access Point destination addresses from different Application Server protocols, and destination addresses. Because different application server supply their destination addresses in different formats a general algorithm may not exist that caters for all the different formats supported by different Application Servers. This method is therefore limited in its ability to expand to additional protocols and or Application Servers.

6.4.2 Address Management Table

The second implementation method is to use a table with stored destination Access Point values for each Application Server destination address. These tables would have to be generated manually and maintained. The tables require storage space. However they are more flexible than a general algorithm, and incurs no limitation on the addressing of Nodes and Metering Equipment, as a general address derivation algorithm might do.

6.5 ACCESS CONTROL

The Access Control Module could be implemented at the Node-Side Driver, or the Access point. The purpose of this module is to restrict the incoming and outgoing packets, by only allowing predefined destination and origin addresses to send and receive APDUs via the Access Point and REMPLI Node.

6.5.1 Access control implemented at the Access Point

If access control is implemented at the Access Point it limits traffic over the PLC, but there is a large amount of processing overhead placed on the Access Point. Another problem with implementing access control at the Access Point is that all the Access Points will need to have the same access control lists, therefore these lists have to be distributed and duplicated on each and every Access Point on the Network. The instant there is a change in one of the lists located at an Access Point, this list has to be distributed to all other Access Points. This scenario is not preferred because of the extra processing and relaying capabilities that would be required at the Access Point. This possibility also opens up the possibility of a malicious Access Point that does not allow for the storage of an access list at the Access Point rendering the access list useless. In order to obtain the origin address of the APDU used to accept or reject a request in the access control list it can be obtained from the source and destination address from the TCP/IP packet. This is not a processor intensive operation and no additional data, containing the origin address needs to be sent via the payload of a data packet. The single most important factor is however that Access Points are owned by different entities within the REMPLI network, and they might have objections to the access control being centralised over all Access Points.

6.5.2 Access control implemented at the Node side

If the access control list is implemented at the REMPLI Node it is difficult to determine the origin of the data and therefore this data needs to be added by the driver to the data sent over the PLC. If the data is compressed before it is sent over the PLC link some decompression processing will have to be done before the access control could take place. A possible solution for obtaining the origin of the APDU is to retrieve the origin from the signature of the APDU if security is implemented. However this method is reliant on the security module (signature) being implemented and if a sender chooses not to implement it more data will have to be added at the Access Point to the PLC packet in order to identify the origin of the packet.

Because the Access Points are already performing a lot of processing, it could be possible to overload the Access Points with too much processing requirements and it is preferable to distribute some processing operations to the REMPLI Node. If the Access Points is not owned by the same company, which it could be assumed, then different vendors might not want to implement this extra processing overhead on their Access Points. Each owner would determine the access control applied to its Access Point.

If the Access Control is implemented at the REMPLI Node, then the origin of the APDU will have to be retrieved from the signature of the APDU. This signature should be generated at the Application Server, in order to make the Access Point layer transparent as an origin. By implementing it this way, APDUs does not have to be forwarded through a specific Access Point, but rather originate from a specific Application Server. This will allow an Access Point which cannot directly connect to a certain Node to forward it to another Access Point and then forward it from there to the destination REMPLI Node.

It is therefore preferable to place the access control functionality at the REMPLI Node, this distributes processing between the Access Point and Node and it is a central unit to which all requests relevant to this Node are sent.

6.6 OPTIMIZED “KEEP-ALIVE” SIGNALLING CONCEPT

Due to the nature of the transmission services and the use of a connectionless transport protocol like IP and a very bandwidth limited network like the PLC network between Application Server and metering devices, communicating the request of an Application Server to a metering device could take up a significant amount of time. In order to allow requests generated by Application Servers to traverse down through the REMPLI network and back up again to the Application Server without timing out, a “Keep-alive” signalling scheme is required for each protocol implemented within the REMPLI project.

The purpose of these “Keep-alive” signals is to keep the Application Server “interested” in the request until the response can be sent back to the Application Server. These “Keep-alive" signals needs to be sent between the Access Point and the Application Server and the REMPLI Node and the Access Point.

A general "Keep-alive" management function is needed to control the sending and receiving of these "Keep-alive" signals. It is required that this "Keep-alive" signalling scheme be optimized as much as possible in order to limit the possible overhead it could cause over the PLC network. The purpose of this general "Keep-alive" management module is also to perform these optimization functions. The status of each meter and protocol will be maintained within the general "Keep-alive" management module. The general "Keep-alive" management module must maintain a timer to determine when a time-out is going to occur. The general "Keep-alive" module is then responsible for sending the "Keep-alive" signals to the appropriate destination (Access Point, Application Server) before the timeout for the protocol occurs. This module could be used to optimize this "Keep-alive" signalling scheme by adding "Keep-alive" packets to protocol specific APDUs that do not fill up an entire PLC packet. This is possible because PLC packets are of fixed length and if the APDU used by a protocol does not fill it up, there are empty bytes available within the PLC packet that could be used to transport "Keep-alive" data.

Some protocols do not support "Keep-alive" signalling capabilities and these protocols will not be able to use the "Keep-alive" signalling functions provided by this general "Keep-alive" management module.

6.7 FUNCTIONALITY OF THE OPTIMIZATION MODULE

This section describes the functions that are performed by the Optimization Module at the Access Point-Side Driver and the Node-Side Driver. This includes the reduction of headers from APDUs and the compression of data before it is sent over the PLC network via the De/Mux interface.

6.7.1 Header Reduction

This section will focus on the header reduction within the M-Bus protocol, but is however not limited to the M-Bus protocol, as similar operations can be performed on all the Metering PDUs. In the M-Bus protocol four different telegram formats are defined. Three of these telegram formats can be recognised by special start characters, as shown in figure 6.14.

Single Character

E5h

Short Frame

Start 10h	C Field	A Field	CS	Stop 16h

Control Frame

Start 68h	L Field 3	L Field 3	Start 68h	C Field	A Field	CI Field	CS	Stop 16h

Long Frame

Start 68h	L Field	L Field	Start 68h	C Field	A Field	CI Field	User Data	CS	Stop 16h

Figure 6.14. Telegram formats defined within the M-Bus protocol.

All of these telegram formats can be uniquely identified by the length of the data payloads they have. This allows the Optimization Module to drop all the headers for all the different telegram formats while still being able to identify the type of telegram that is sent or received. The optimization of headers is applied as follows for each telegram format. The header reduction can be applied due to the guarantee of data correctness by the underlying networks.

- Single Character: No further optimization is necessary. This telegram consists of a single byte without any headers. This telegram format is identified if the length of the data is 1 byte.
- Short Frame: The optimization comprises the dropping of the Start byte, Check Sum byte and the Stop byte. Only the Control field and the Address fields are required to stay in tact. This type of telegram is identified by the length of its payload which is 2 bytes.
- Control Frame: The optimization comprises the dropping of both the Start bytes, both the Length fields, the Check Sum byte and the Stop byte. Only the Control field, the Address field and the Control Information fields are required to remain in tact. This type of telegram is identified by the length of its payload which is 3 bytes.
- Long Frame: The optimization comprises the dropping of both the Start bytes, both the Length fields, the Check Sum byte and the Stop byte. Only the Control field, the

Address field and the Control Information fields are required to remain in tact. This type of telegram is identified by the length of its payload which is greater than 3 bytes.

6.7.2 Data Compression

The optimization process can further be improved by compressing the data before it is sent over the De/Mux interface. Compression should only be performed if there is some sort of gain by applying a data compression algorithm, i.e. if the APDU and the security data and the "Keep-alive" byte fit into one PLC packet then there is nothing to gain by compressing this data and unnecessary processing is done. If the data, required to transmit over the PLC network requires multiple PLC packets and the number of PLC packets can be reduced by even only one byte, then compression should be implemented. Some sort of Heuristic decision making process should be implemented to determine whether there is some gain out of applying compression to the PLC packet's payload.

Chapter 7

7 IMPLEMENTATION SOLUTIONS

This chapter discusses the different implementation options for each of the different modules contained within the REMPLI driver. Due to the distributed nature of the REMPLI driver, each of the different modules: Application Server-Side Driver, Access Point-Side Driver and Node Side Driver are discussed within a separate section in order to avoid ambiguity.

7.1 SOFTWARE ARCHITECTURE

This section describes the software architecture implemented for the entire system by employing the 4+1 view paradigm methodology suggested by Kruchten, [32]. The design view is given in chapters 4 and 5. The sequence diagrams shown by figures 6.4, 6.11 and 6.13 describe the process view of the M-Bus protocol driver. This section will expand the views mentioned above by discussing the implementation and deployment views.

7.1.1 Implementation View

The class diagram representing the M-Bus Driver architecture is given in figure 7.1. The class diagram gives the main classes contained within the M-Bus Driver. An M-Bus Driver contains a client, used for connecting, and a server, used for listening. Either one of the server or client will be utilized or both depending on where the driver is deployed and the role the driver fulfils i.e. does the current driver fulfil a dual client/server role and do both communicate over a specific communications link. Thus if the M-Bus Protocol Driver is deployed at the Access Point or the Node it will require both a TCP client and TCP server class for communicating over a network utilizing the TCP/IP protocol. The Application Server-Side Driver contains a server that listens to the requests made by the M-Bus application server and a TCP client which connects and forwards the requests to the Access Point-Side Driver's TCP server over a TCP/IP connection.

At the Access Point the Application Server-Side Driver's TCP client connects to the Access Point TCP server and the Access Point-Side Driver uses the client class to connect to the Node-Side Driver's server. The Node-Side Driver in turn uses the client and the M-Bus Protocol class to connect to the metering equipment. The M-Bus Frame class is a class which contains all the parameters associated with an M-Bus frame. The M-Bus Frame class also contains all the validation functionality required by the layer 2 M-Bus protocol for an M-Bus frame, such as the checksum validation, parity validation and length validation. The Node-Side Driver communicates with the M-Bus metering equipment using the M-Bus Protocol class.

It is important to note that the client and server do not have to be limited to the TCP protocol. The client/server should use the underlying network's communication protocol. If PLC is used and a proprietary interface and protocol is used then this protocol must be implemented by the client/server. Alternatively a protocol like UDP (User Datagram Protocol) could be used as a connection orientated method of communicating over the network interface.

The M-Bus Protocol class is responsible for maintaining all layer 2 protocol communication procedures. These procedures include:

- Send/Confirm
- Request/Response
- Transmission procedures in case of errors.

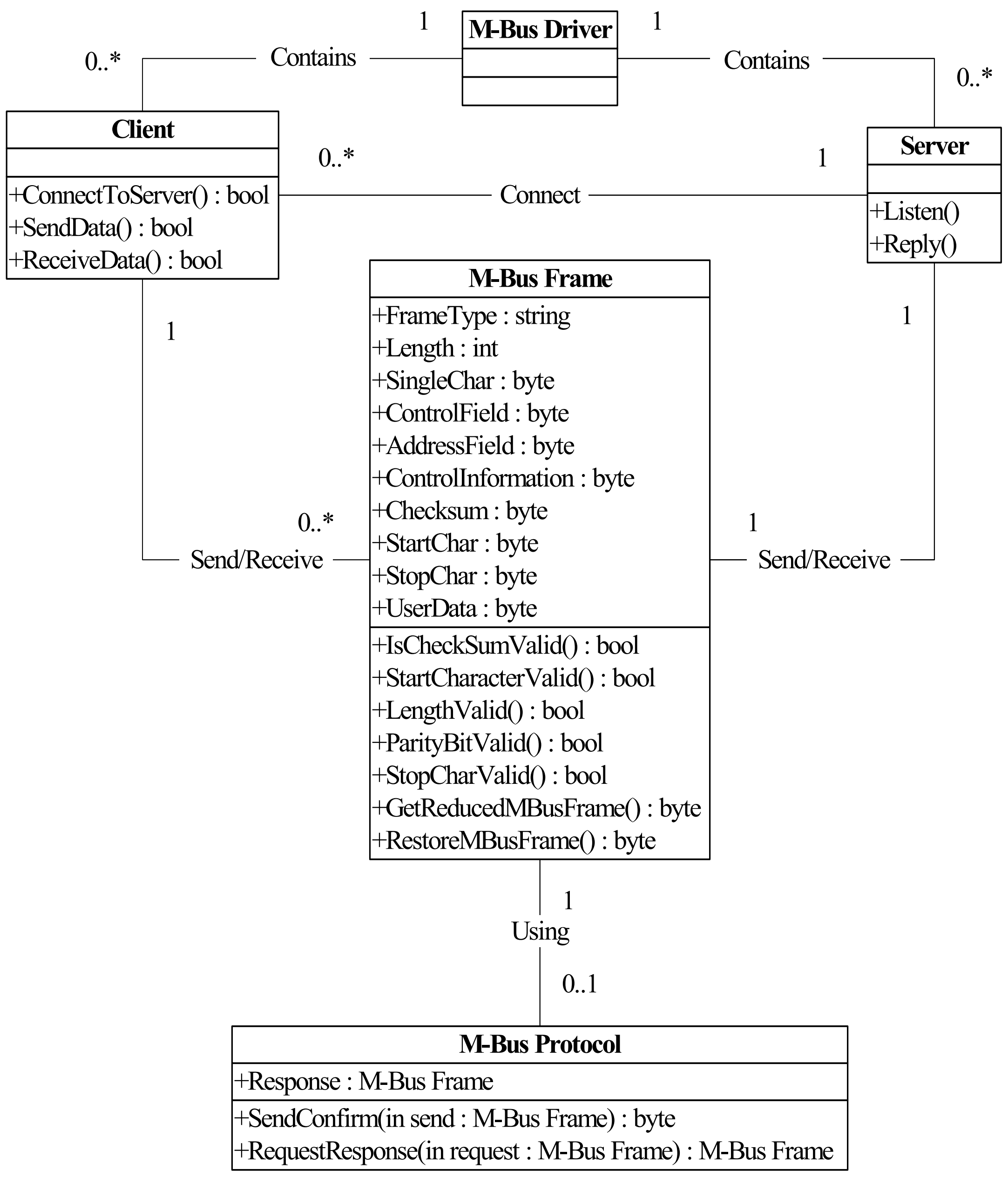


Figure 7.1. Simplified class diagram of the M-Bus driver architecture.

The M-Bus Protocol class is also responsible for handling communication errors and determining if the transaction completed successfully, see section 1.4.16. The main role of the M-Bus drivers at the Access Point-Side Driver is for the server and client to perform the tunnelling functionality, i.e. forwarding the received M-Bus protocol data units to and from the Application Server and Nodes by traversing the underlying network.

7.1.2 Deployment View

The M-Bus Driver software will be deployed on multiple hardware devices. These devices are remotely distributed and this section gives the deployment view for all the software modules implemented by the M-Bus protocol driver. The software modules are deployed as given in figure 7.2.

The Application Server-Side Driver is deployed at the Application Server. This is usually a PC which contains the software M-Bus application server. The Application Server is responsible for managing the entire M-Bus metering network. The Access Point which is connected to the Application Server through a TCP/IP interface contains the Access Point-Side Driver and the Node contains the Node-Side Driver.

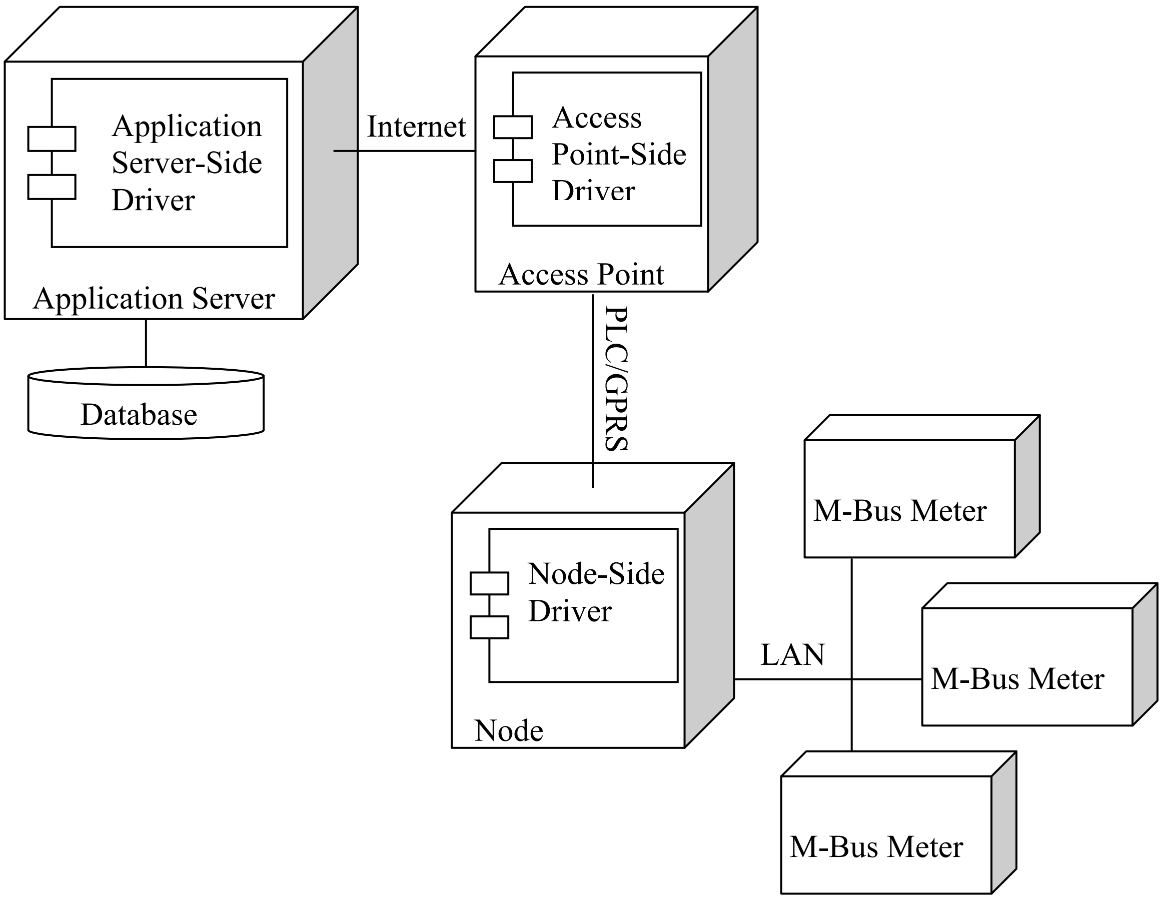


Figure 7.2. Deployment view of the M-Bus driver architecture.

7.1.3 Implementation Platform

The M-Bus driver was implemented on the .NET 2.0 Framework. The C# programming language was used.

The .Net Framework consists of two main components: the .NET Framework class library and the common language runtime. The runtime environment is responsible for managing code at execution time, this include the management of threaded processes and the allocation of memory. At the same time the runtime environment enforces various forms of code accuracy management, which in turns ensures security and robustness is maintained. The class library contains a comprehensive collection of reusable object-oriented types that support the development of software. Classes utilized in the M-Bus protocol driver include:

- The RS232 class for communicating over a serial line with the metering equipment.
- The TcpListener class which served as a TCP Server and the TcpClient class which served as a TCP client. The TcpListener also supports access control.
- The .Net framework also supports sockets, which can be used for implementing the TCP server and TCP clients.

The concurrent sending and receiving of requests and responses take place in a threaded process managed by the runtime environment. The classes provided above are also supported in other environment such as the Linux environment, but the management of these processes will have to be managed by the driver itself and not a runtime environment used by the .NET Framework.

The .Net Framework was thus chosen as a development platform for its comprehensive set of class libraries, the management of the runtime environment and the ease of use when implementing software. There is a slight disadvantage in terms of performance when using the .Net Framework. Due to the management of code by the runtime environment an implementation in the .Net Framework is slightly slower in terms of execution time. Another inhibiting factor about the C# language is that the first time an object is invoked within the runtime environment a lot of time is taken to assign the necessary resources, however after the initial allocation of resources the program execution times are close to pure native C program implementations. The difference in execution time between C# and C is negligible for the M-Bus protocol driver implementation. Therefore using the C# language will have a negligible impact on the implementation of the M-bus protocol driver.

The C# language was chosen because it gives the user easy access to native Windows services. These include networked objects and communication services, all of which are used extensively by the M-Bus protocol driver.

7.2 APPLICATION SERVER-SIDE DRIVER

This section contains the implementation solutions for all of the modules that are situated at the Application Server side. All of these modules are contained within the Application Server-Side Driver as shown in figure 6.1.

7.2.1 Communication Module

The data from the Application Server comprises two parts:

- M-Bus frame
- Addressing information

This module splits the data received from the Application Server into the two different data units as mentioned above. The Addressing information received from the Application Server is used to translate to corresponding Access Point IP-address. The M-Bus frame is then sent to the Access Point, as is, via a TCP/IP network interface. The following preconfigured Application Server data 155.239.172.69 | ****M-Bus Frame**** was used. The Communication Module extracted the destination IP address and M-Bus frame received from the Application Server. Thus the resultant Access Point IP-address 155.239.172.69 was used by the TCP/IP Communication Module to connect and send the M-Bus frame to the Access Point.

7.2.2 TCP/IP Communication Module

The TCP/IP Communication Module was implemented using the .Net TcpClient class. The TcpClient class was used to setup a client TCP connection which connected to the TCP/IP server situated at the destination Access Point. The TCP client was setup to communicate on port 399 using the destination IP address 155.239.172.69. Alternatively a DNS name could have been used as the destination of the TCP/IP server, however DNS lookups are quite time consuming, decreasing the available time for the M-Bus request/response transaction. The TCP/IP Communication Module also retrieves the destination REMPLI Node Address from a lookup table by combining the address of the communications port with data contained within the original application server request as the key. Thus the destination RNA and the M-Bus request is forwarded to the Access Point-Side Driver. The TCP/IP communication takes place in a thread, thus allowing for multiple concurrent TCP/IP connections to be made to multiple Access Points and allowing the Application Server-Side Driver to continue with other processes while the TCP/IP communication is managed within the thread by the .NET runtime environment.

7.2.3 Address Management Module

The Address Management Module uses the Addressing information contained in the data to obtain the IP address of the destination Access Point. This implementation had a fixed Access Point address and retrieved the RNA from a hash table which uses the communications port address and the data contained in the Application Server data as the key. The hash table with all its RNA mappings was serialized (saved to the hard-drive in binary format) and the RNA retrieved by deserializing the hash table and looking up the destination RNA.

The Address Management Module is responsible for storing and maintaining a database with available Access Point and RNA addresses. The Hashtable class was chosen because the Hastable class is contained within the .NET class library set, which manages a table with key indexers and corresponding values. It is also used to save and retrieve the saved configuration by using .NET serialization.

7.3 ACCESS POINT-SIDE DRIVER

This section discusses the software components defined at the Access Point-Side Driver as given in figure 6.8.

7.3.1 TCP/IP Communication Module

The TCP/IP Communication Module receives the TCP/IP encapsulated data and decapsulates it. The resultant payload data is then forwarded to the Data Processing Module. This module is implemented as a TCP/IP server using the TcpListener class contained in the .NET Framework.

7.3.2 Data Processing Module

The Data Processing Module takes the received data from the Application Server-Side Driver and extracts the RNA. The tunnelling approach chosen for the M-Bus protocol driver does not make use of data loggers, however for experimental reasons this module was also used to store the requests and responses with a time-stamp for auditing and experimental verification.

7.3.3 Management Module

The Management Module is responsible for managing the current communication thread. For the M-Bus protocol it has only one function: Start a timer that determines when the current application request/response times out. To implement this function of determining if the current request/response has timed out, a timer is needed that provides a small enough resolution for the protocol implemented. In the case of the M-Bus protocol, a timeout is determined by using 330 bit periods + 50 ms to determine if a timeout has occurred, as discussed in section 3.2.1.3. In order to determine the exact timeout window, the speed of communication used by the M-Bus meters is required. For the test case this data rate is predetermined at 300 baud. Knowing the baud rate, the exact timeout window could be calculated for the request/response to be completed. This timeout window can be expressed as given in formula 7.1

$$T_{\mathrm{Req/Res}} = 3 \times [\frac{330}{Baudrate} + 0.05] \quad \text{(s)} \qquad (7.1)$$

where $Baud\ rate$ is given in bits per second and $T_{Req/Res}$ is measured in seconds. If the predetermined baud rate of 300 baud is substituted in formula 7.1 a timeout interval of 3.45 seconds is given. This is the absolute maximum time that an M-Bus protocol request/response transaction can take if the M-Bus protocol is used. Therefore the

Management Module has to start a timer the moment a request is received from the Application Server and terminate the communication link after the timeout period has expired. If the timeout window expires before a response is given to the Access Point, then the Management Module is responsible for terminating the current Application generated request/response thread. This involves the termination of the TCP client connected to the TcpListener and the termination of the client connected to the REMPLI Node server. The Access Point is only responsible for managing and maintaining the communications link to and from the Application Server and the REMPLI Node. The TcpClient class provides a ReceiveTimeout property that is set to 3.45 seconds which determines the time a TcpClient will wait for receiving data, before it terminates.

7.3.4 Data Storage Module

For experimental reasons this module was used to store M-Bus requests and corresponding responses with a timestamp. But in the final implementation this module is dependent on available resources and user needs.

7.3.5 Optimization Module

The user 2 data that is received is reduced at this module as discussed in section 6.7.1 header reduction. The reduction of layer 2 headers was implemented within the M-Bus Frame class. The M-Bus Frame class contains a method GetReducedMBusFrame() which retrieves the user 2 data to traverse the PLC/GPRS network.

7.3.6 De/Mux Communication Module

For the experimental setup TCP over GPRS was used. Thus the reduced M-Bus frame was sent to the Node by connecting with a TCP client to the TCP server contained at the Node.

If the REMPLI PLC communication interface is used the reduced data received from the Optimization Module is sent over the PLC-network by using the De/Mux Communication Module which interfaces with the REMPLI Communication Interface (RCI) which manages the power line communication. Thus this module only forwards the received data to the RCI and if an error message should be received from the RCI it inform the Management Module.

7.4 NODE-SIDE DRIVER

This section discusses the software components defined at the Access Point-Side Driver as given in figure 6.9.

7.4.1 De/Mux Communication Module

This module contains a TcpServerListener (TCP server) which listened for clients connecting to it. The reduced M-Bus frame was received by this module after it was sent by the Access Point-Side Driver's De/Mux Communication Module.

If the REMPLI PLC communication interface is used, the reduced M-Bus frame is received from the De/Mux Communication Module by interfacing with the REMPLI Communication Interface (RCI) which manages the power line communication.

7.4.2 Optimization Module

The M-Bus class contains a method called RestoreMBusFrame, which restores the reduced M-Bus frame to its original state. This process does not change any of the layer 2 data, it only restores overhead such as the length, start and stop fields in the M-Bus frame.

7.4.3 Access Control Module

The Access Control Module was implemented to use the IP address of the Access Point TCP client connected to the Node-Side Driver's TcpListener. The Access Point's IP address is checked against a local access control list to see if it is allowed to connect to the Node. The .NET Framework provides an AccessControlList class for managing the access control list functionalities.

7.4.4 Protocol Stack (Layer 2)

The layer 2 protocol stack is implemented in the M-Bus Protocol class contained within the M-Bus protocol driver. The M-Bus Protocol class starts the communication process with the M-Bus metering equipment and maintains the communication process.

7.4.5 Protocol Specific Hardware Driver for HyNet (Layer 1)

The meter was connected to a PC via a serial port, thus in the experimental setup this module was implemented by utilizing the SerialPort class supplied by the .NET

Framework class library. The SerialPort class was responsible for communicating via the RS232 protocol over the serial line with the metering equipment.

In the REMPLI network the protocol specific HyNet driver, specifically developed for the REMPLI system will be used by the Node-Side Driver to communicate with the metering equipment. In the test implementation the SerialPort class was used to interface between the Node-Side Driver and the metering equipment using the procedures contained in the M-Bus Protocol class.

For experimental reasons this module was also used to store the requests and responses received/sent at the Node and associated with a time-stamp for auditing and experimental verification.

Chapter 8

8 RESULTS AND CONCLUSION

This chapter presents the results obtained during experiments conducted as well as conclusions reached after the results were obtained.

8.1 EXPERIMENTAL SETUP

As a test case scenario, the M-Bus driver architecture discussed in chapter 6 was implemented as discussed in chapter 7 and deployed in a test system. The experimental setup consisted of an Application Server, which was responsible for collecting and storing measurements taken by remote meters. The Access Point was connected to the Application Server via a fixed line network on which TCP over IP was used to maintain a connection. The fixed line connection provided a high-bandwidth, stable network interface. The Access Point was connected to the Node PC via a GPRS connection. The GPRS connections demonstrated similar characteristics to the REMPLI PLC network in the REMPLI network which include the following:

- Low bandwidth.
- Frequent fluctuations in signal strength. The metering equipment was situated in a remote location and the signal strength was inconsistent.
- The GPRS connection is dropped on a regular basis and they have to be re-established, causing long delays and variance in round trip time for requested data.
- Unstable power supply also had a big influence on the underlying network, because of regular intervals at which the power failed at the remote station where the remote Node and metering equipment was situated.

A typical usage scenario in the REMPLI system would require that one reading is taken per meter per day. In the experimental case the Application Server requested readings every 30 minutes. The experimental setup took the readings of three parameters:

- Temperature
- Left Longitudinal Force
- Right Longitudinal Force

The readings mentioned above were obtained from a railway track. The temperature measured is the air temperature in close proximity to the track, the left longitudinal force is the force exerted on a force strip by the expanding left segment of railway track. Similarly the right longitudinal force is the force exerted by the expansion of the right segment of the railway track. Thus as the temperature rises the railway segments expand and exert a force in a longitudinal direction (relative to the rail segment), as shown in figure 8.1.

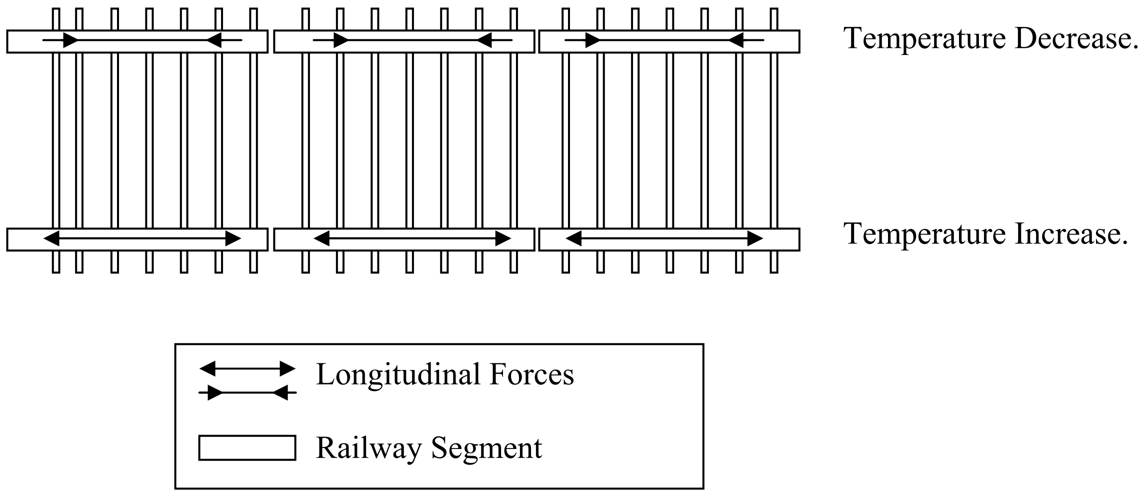


Figure 8.1. This figure demonstrates the longitudinal forces exerted on a railway track as the temperature increases and decreases.

The typical readings taken in the REMPLI system would include electricity, gas usage and temperature etc. The values obtained in this experimental setup are not of the same units as the typical measured units of a typical REMPLI scenario, but serve as sufficient representative data to be collected by an M-Bus metering system, with similar data sizes and characteristics. At the Node PC the meter was connected to the serial port utilizing RS232 communication. The Node-Side driver was setup to store the requested readings on the Node. If no request was received from the Application Server, the Node-Side driver would request a reading and store it on the local machine. The results stored on the Node were compared to the readings received at the Application Server in order to determine the success rate of the readings requested by the Application Server from the metering equipment. The Application Server collected meter readings over a period of several days.

8.2 MEASUREMENT RESULTS

The resultant readings retrieved are listed below in table 8.1. The values listed in table 8.1 contain were measured from 7 December 2005 to 8 December 2005 between 18:21 and 10:54 respectively.

Table 8.1. The measurement values received by the Application Server after requesting the values from the remote meter over a 17 hour period.

Left Longitudinal Force	Right Longitudinal Force	Temperature	Time Requested
33.119	39.6747	37.0283	2005/12/07 18:21
34.3371	40.7937	37.0146	2005/12/07 18:22
37.4112	43.6807	36.9481	2005/12/07 18:24
96.8217	107.067	34.8035	2005/12/07 18:54
162.396	176.739	31.9262	2005/12/07 19:24
212.326	230.199	29.5468	2005/12/07 19:54
248.549	267.654	27.7326	2005/12/07 20:24
280.147	299.989	26.1059	2005/12/07 20:54
305.254	325.88	24.773	2005/12/07 21:24
325.423	346.552	23.6541	2005/12/07 21:54
342.7	364.244	22.7063	2005/12/07 22:24
304.071	315.988	24.5036	2005/12/07 22:54
332.662	346.861	23.1799	2005/12/07 23:24
358.083	374.404	21.8901	2005/12/07 23:54
357.809	365.198	21.908	2005/12/08 00:24
376.919	388.738	20.9126	2005/12/08 00:54
393.437	407.454	20.021	2005/12/08 01:24
374.171	390.455	20.7275	2005/12/08 01:54
374.171	390.455	20.7275	2005/12/08 02:24
374.171	390.455	20.7275	2005/12/08 02:54
374.171	390.455	20.7275	2005/12/08 03:24
374.171	390.455	20.7275	2005/12/08 03:54
374.171	390.455	20.7275	2005/12/08 04:24
374.171	390.455	20.7275	2005/12/08 04:54
374.171	390.455	20.7275	2005/12/08 05:24
374.171	390.455	20.7275	2005/12/08 05:54
374.171	390.455	20.7275	2005/12/08 06:24
374.171	390.455	20.7275	2005/12/08 06:54
374.171	390.455	20.7275	2005/12/08 07:24
374.171	390.455	20.7275	2005/12/08 07:54
374.171	390.455	20.7275	2005/12/08 08:24
374.171	390.455	20.7275	2005/12/08 08:54
374.171	390.455	20.7275	2005/12/08 09:24
374.171	390.455	20.7275	2005/12/08 09:54
374.171	390.455	20.7275	2005/12/08 10:24
374.171	390.455	20.7275	2005/12/08 10:54

In order to allow the M-Bus Protocol Driver to run at the optimal execution speed, the first 3 readings are requested within minutes of each other in order to initialize the communication network and allocate all the required resources within the .NET Framework.

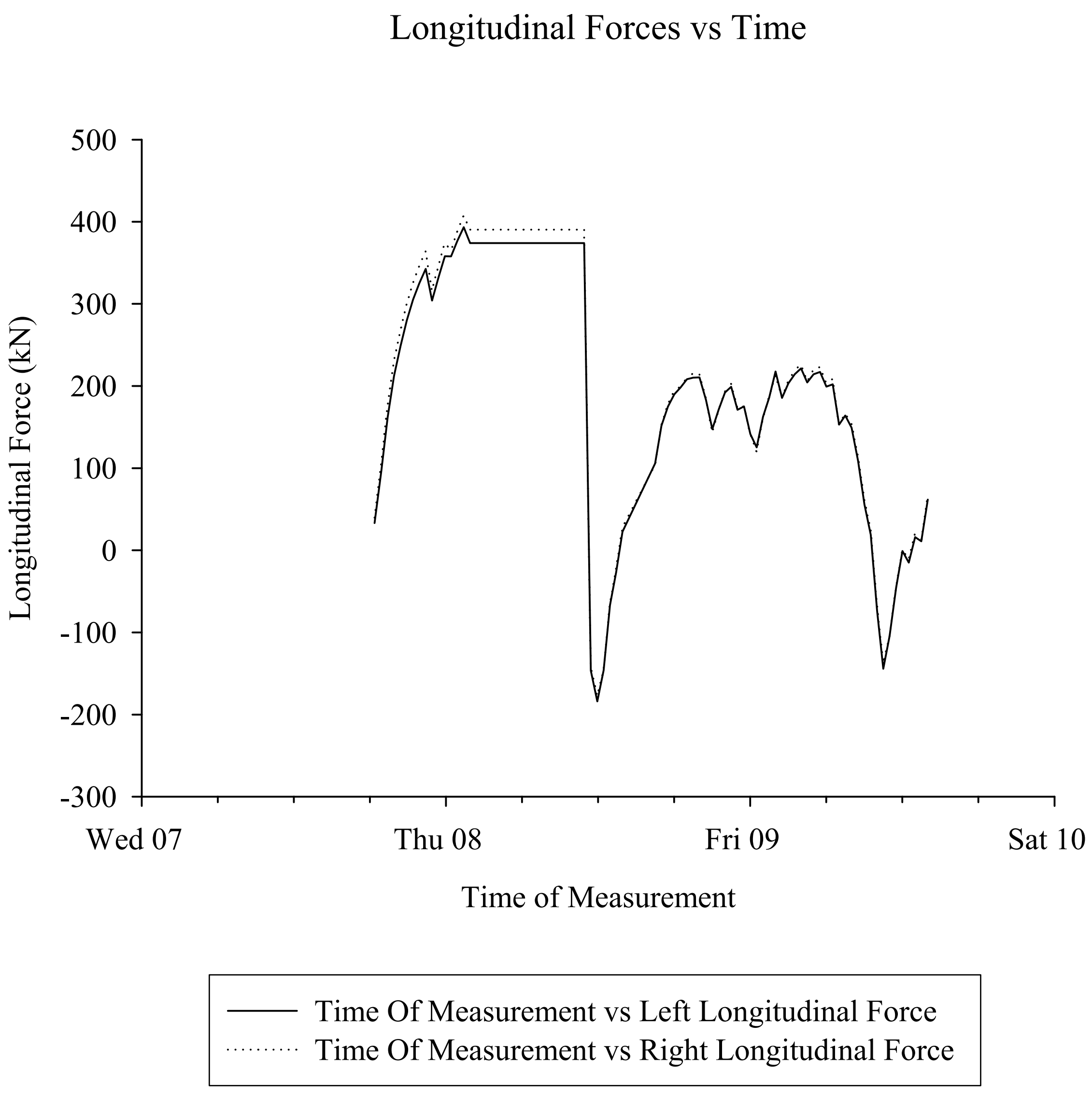


Figure 8.2. Graph displaying the combined measured values of the left and right longitudinal forces, measured from 07/12/05 – 09/12/05.

Values requested in such a short space of time also serve as control values to verify that the meter setup is correct and that valid values are received. It can clearly be seen that there is a small change in the temperature and both the longitudinal forces measured during the first two readings, which are taken one minute apart. This is as expected, because the temperature and thus the longitudinal forces should vary only slightly after a minute has passed. The metering equipment is thus functioning correctly and all the driver modules initiated and functioning as intended. After this initial setup control steps have successfully been completed the Application Server can continue with requesting measurements every 30 minutes.

Table 8.2. The next 13 measurement values received by the Application Server after requesting the values from the remote meter over a 9 hour period.

Left Longitudinal Force	Right Longitudinal Force	Temperature	Time Requested
-146.459	-142.008	30.9078	2005/12/08 11:25
-184.124	-177.225	33.1174	2005/12/08 11:56
-146.171	-146.905	32.0739	2005/12/08 12:26
-66.7688	-65.8767	28.4759	2005/12/08 12:56
-26.7663	-21.8699	26.3369	2005/12/08 13:26
22.4886	27.9764	23.9095	2005/12/08 13:56
105.823	104.082	19.9678	2005/12/08 16:30
151.78	152.688	17.7461	2005/12/08 17:00
175.001	177.837	16.4192	2005/12/08 17:30
189.893	194.123	15.5896	2005/12/08 18:00
198.516	199.552	15.1367	2005/12/08 18:30
208.339	209.53	14.6054	2005/12/08 19:00
210.243	215.446	14.394	2005/12/08 19:30

Frequent fluctuations in the underlying network caused errors during the communication process of the request/response transaction. The error handling for such sudden drops in the communication line was not sufficiently implemented in the experimental M-Bus Protocol Driver. Hence the software was updated and upgraded throughout the network. The intervals between meter reading requests drift in the time requested field given in table 8.2. This is due to the upgrading of the M-Bus Protocol Driver taking place during the measurement period shown in this table. There is also a 150 minute period without readings, this is due to the reconfiguration and setup of the system after the M-Bus Driver was upgraded. The M-Bus Protocol Driver had to be upgraded at the Application Server, Access Point and the Node.

The graph displaying the resultant data is shown in figure 8.2. The fluctuation in force can clearly be seen over the measurement period. The metering equipment appear to have malfunctioned during the period 2005/12/08 01:54 to 2005/12/08 10:54. All the values returned during this period are exactly the same. The temperature will vary as the time progresses and thus so should the measured longitudinal forces. The backup values stored at the Node confirm that the values received at the Application Server are the same as the values measured by the metering equipment, thus the values returned by the metering equipment were faulty. This could be due to many factors. A likely cause is condensation on the meter probes, causing a possible short.

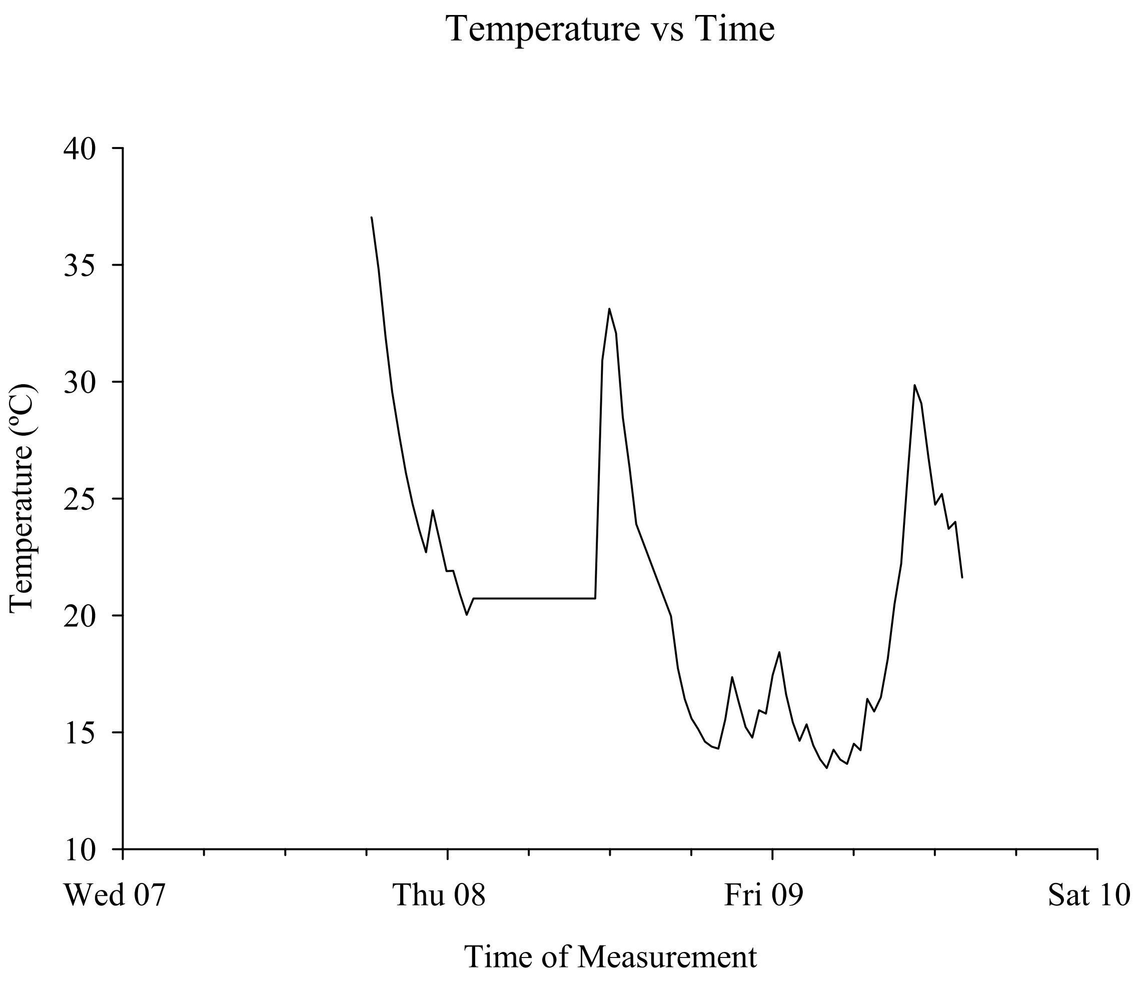


Figure 8.3. Graph displaying the measured temperature values for the period 07/12/05 – 09/12/05.

If the convention used to measure the longitudinal forces is inverted, such that an increase in temperature causes an increase in force, the relation between the temperature and the forces measured can be compared. The graph displaying the measured forces by using the inverse convention is given in figure 8.4. If the measured temperature given in figure 8.3 is compared to the measured longitudinal forces given in figure 8.4 the correlation between the temperature and forces are clearly shown. There is a perfect correlation between the increase/decrease in temperature and the increase/decrease of the longitudinal forces exerted by the railway tracks.

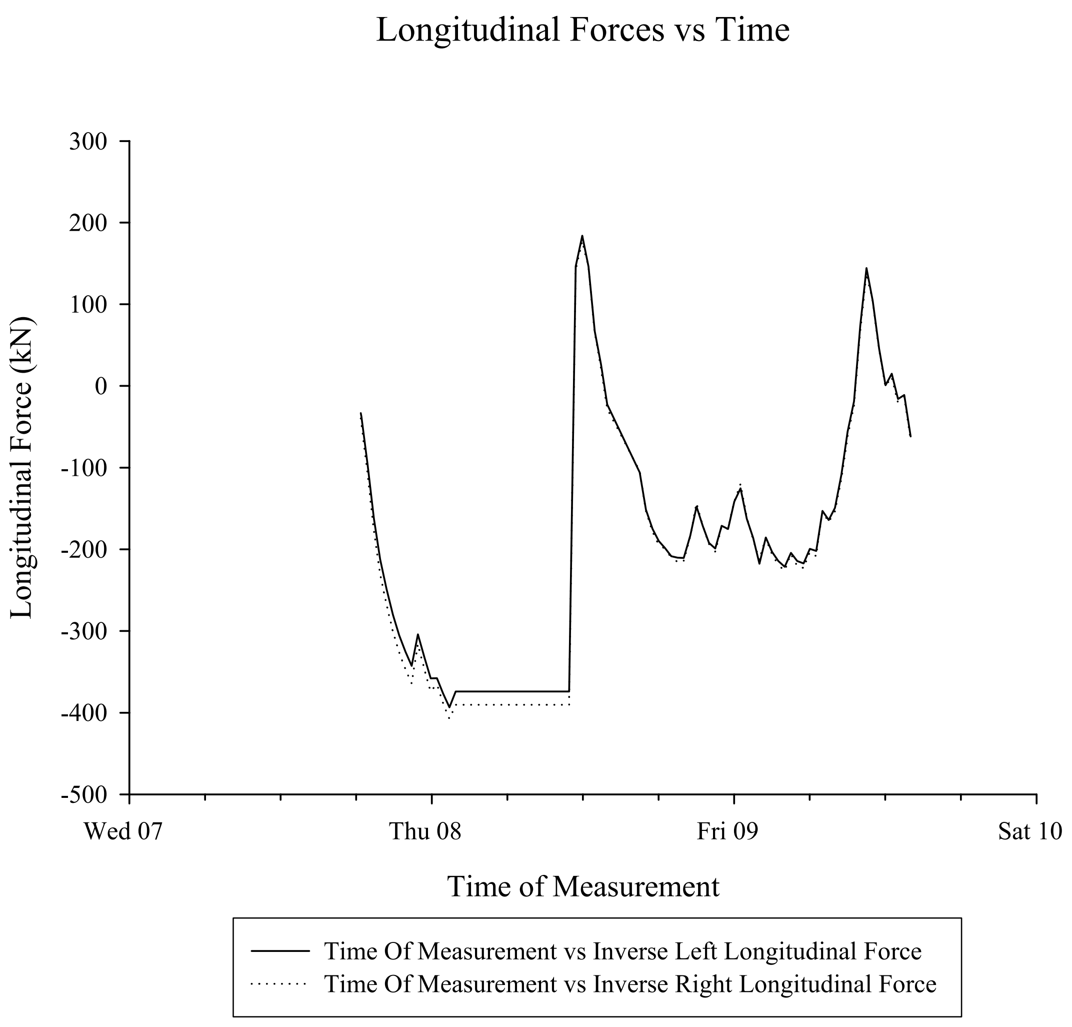


Figure 8.4. Graph displaying the combined measured values of the left and right longitudinal forces, measured from 07/12/05 – 09/12/05.

The remaining measurement values retrieved are given in table 8.3.

Table 8.3. The next set of measurement values received by the Application Server after requesting the values from the remote meter over a 19 hour period.

Left Longitudinal Force	Right Longitudinal Force	Temperature	Time Requested
210.678	214.267	14.3012	2005/12/08 20:00
183.202	184.712	15.5378	2005/12/08 20:30
147.568	143.489	17.3592	2005/12/08 21:00
171.324	170.613	16.26	2005/12/08 21:30
192.186	193.18	15.2232	2005/12/08 22:00
199.018	202.76	14.7693	2005/12/08 22:30
171.04	172.584	15.9393	2005/12/08 23:00
175.281	175.194	15.7943	2005/12/08 23:30
141.54	141.219	17.438	2005/12/09 00:00
125.332	120.087	18.4258	2005/12/09 00:30
162.627	160.827	16.6181	2005/12/09 01:00
186.074	187.332	15.4219	2005/12/09 01:30
217.614	213.235	14.6338	2005/12/09 02:00
185.494	186.116	15.3408	2005/12/09 02:30
203.273	206.387	14.4287	2005/12/09 03:00
214.348	218.878	13.8511	2005/12/09 03:30
221.458	226.611	13.4696	2005/12/09 04:00
204.341	205.666	14.2519	2005/12/09 04:30
214.262	220.122	13.8281	2005/12/09 05:00
217.152	222.979	13.6433	2005/12/09 05:30
199.27	203.243	14.505	2005/12/09 06:00
201.991	208.659	14.2332	2005/12/09 06:30
153.002	153.069	16.4332	2005/12/09 07:00
164.072	166.92	15.8858	2005/12/09 07:30
149.251	153.81	16.5002	2005/12/09 08:00
108.782	114.049	18.1453	2005/12/09 08:30
56.0104	61.4791	20.4826	2005/12/09 09:00
18.9685	25.3996	22.2145	2005/12/09 09:30
-73.5914	-66.3771	26.1543	2005/12/09 10:00
-144.296	-137.448	29.8491	2005/12/09 10:30
-104.049	-104.266	29.0679	2005/12/09 11:00
-45.6182	-44.4528	26.7639	2005/12/09 11:30
-8.60E-01	8.46E-01	24.7338	2005/12/09 12:00
-14.8363	-11.4137	25.1948	2005/12/09 12:30
15.9441	20.5975	23.7031	2005/12/09 13:00
11.0812	13.3416	24.0015	2005/12/09 13:30
61.9283	66.5577	21.6229	2005/12/09 14:00

Due to power outages the meter reading process was interrupted. The power outages continued for the duration of the experiment. The final set of readings was obtained by utilizing an uninterrupted power supply. The values obtained during this final session are contained in table 8.4 after restarting the metering process.

Table 8.4. The final set of measurement values received by the Application Server after requesting the values from the remote meter over a 13 hour period.

Left Longitudinal Force	Right Longitudinal Force	Temperature	Time Requested
-246.888	-236.745	36.6868	2005/12/13 17:40
-232.541	-222.782	36.2536	2005/12/13 17:40
-427.203	-427.323	47.3743	2005/12/14 14:22
-428.966	-429.171	47.4387	2005/12/14 14:22
-408.262	-407.125	46.3611	2005/12/14 14:53
-415.458	-414.207	46.3437	2005/12/14 15:23
-388.564	-397.65	45.1444	2005/12/14 15:53
-379.54	-379.9	44.0128	2005/12/14 16:23
-372.475	-382.582	43.61	2005/12/14 16:53
-324.627	-324.329	41.3645	2005/12/14 17:23
-275.499	-270.713	39.1512	2005/12/14 17:53
-209.148	-203.881	36.3723	2005/12/14 18:23
-147.26	-143.843	33.7905	2005/12/14 18:53
-88.685	-84.5479	31.0567	2005/12/14 19:23
-76.6351	-74.4994	30.3398	2005/12/14 19:53
-38.9762	-35.397	28.4618	2005/12/14 20:23
-11.7798	-7.20312	27.0122	2005/12/14 20:53
7.43651	12.715	26.0068	2005/12/14 21:23
21.6381	27.2523	25.2088	2005/12/14 21:53
29.5255	35.4913	24.7285	2005/12/14 22:23
30.7563	36.524	24.6007	2005/12/14 22:53
38.5829	44.9084	24.1249	2005/12/14 23:23
41.4331	47.8881	23.9215	2005/12/14 23:53
48.2366	54.9218	23.6018	2005/12/15 00:23
31.0888	31.2183	24.477	2005/12/15 00:53
95.188	95.9729	21.3279	2005/12/15 01:23
105.022	107.825	20.6317	2005/12/15 01:53

During the measurement period the underlying network was stable and met the required QoS requirements imposed by the M-Bus protocol. Every first-time request sent by the Application Server was successfully responded to by the metering equipment. The first measurement value was received during the initial setup stages to test if the communication process is successful. The first two readings retrieved are requested only two minutes apart as part of the initial setup process. This initial setup process also allows the .Net Framework to allocate the required resources (memory). The time consuming process of allocating resources only occurs the first time the software is executed. After the initial invocation the system will execute at optimal speed.

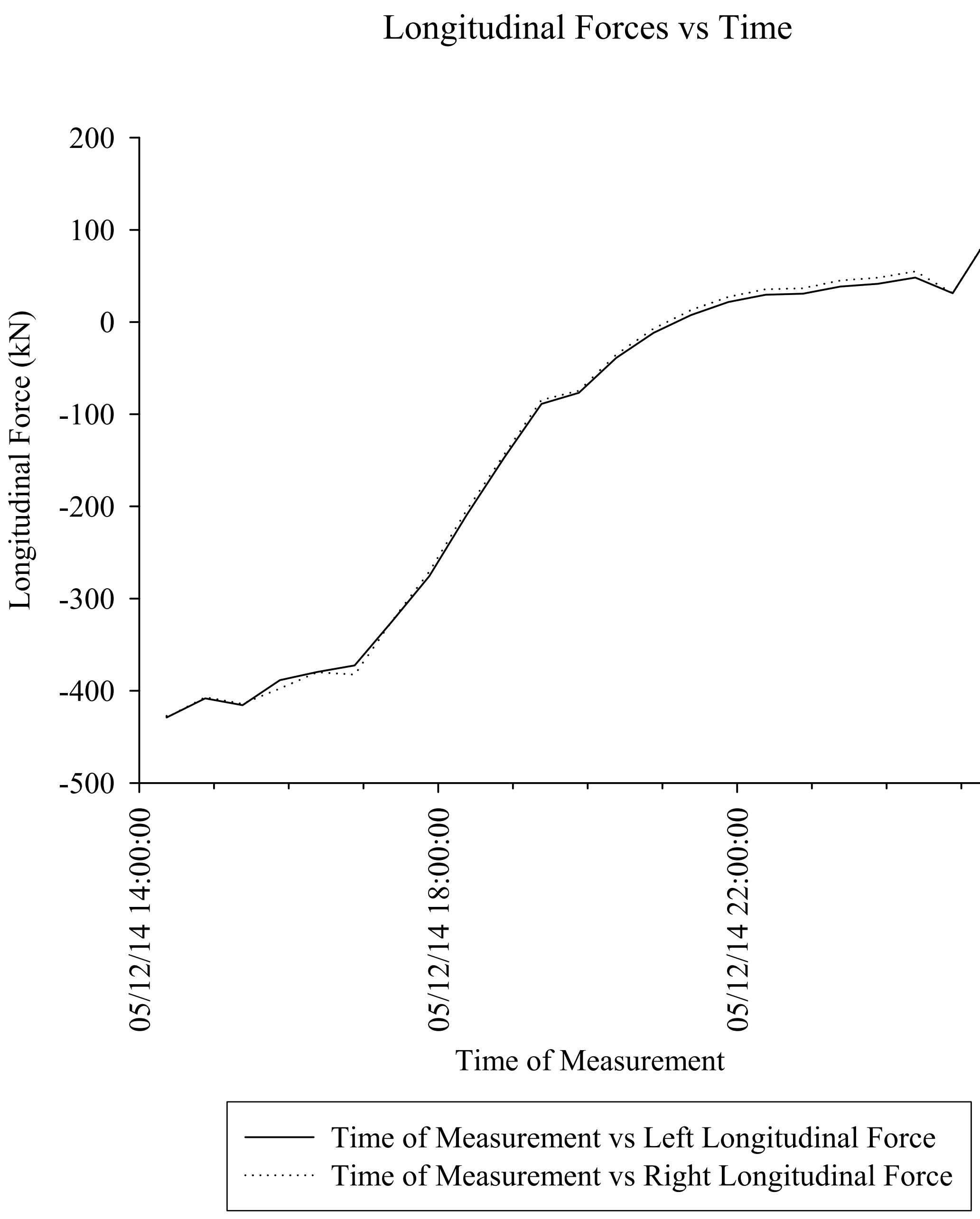


Figure 8.5. Graph displaying the combined measured values of the left and right longitudinal forces.

The negative valued forces measured in each track is due to the convention used, the forces exerted in each track segment changes as a result of the temperature changes.

Figure 8.5 shows a chart depicting the resultant longitudinal forces measured for the left and right rail segments. The measured temperatures are given in figure 8.6.

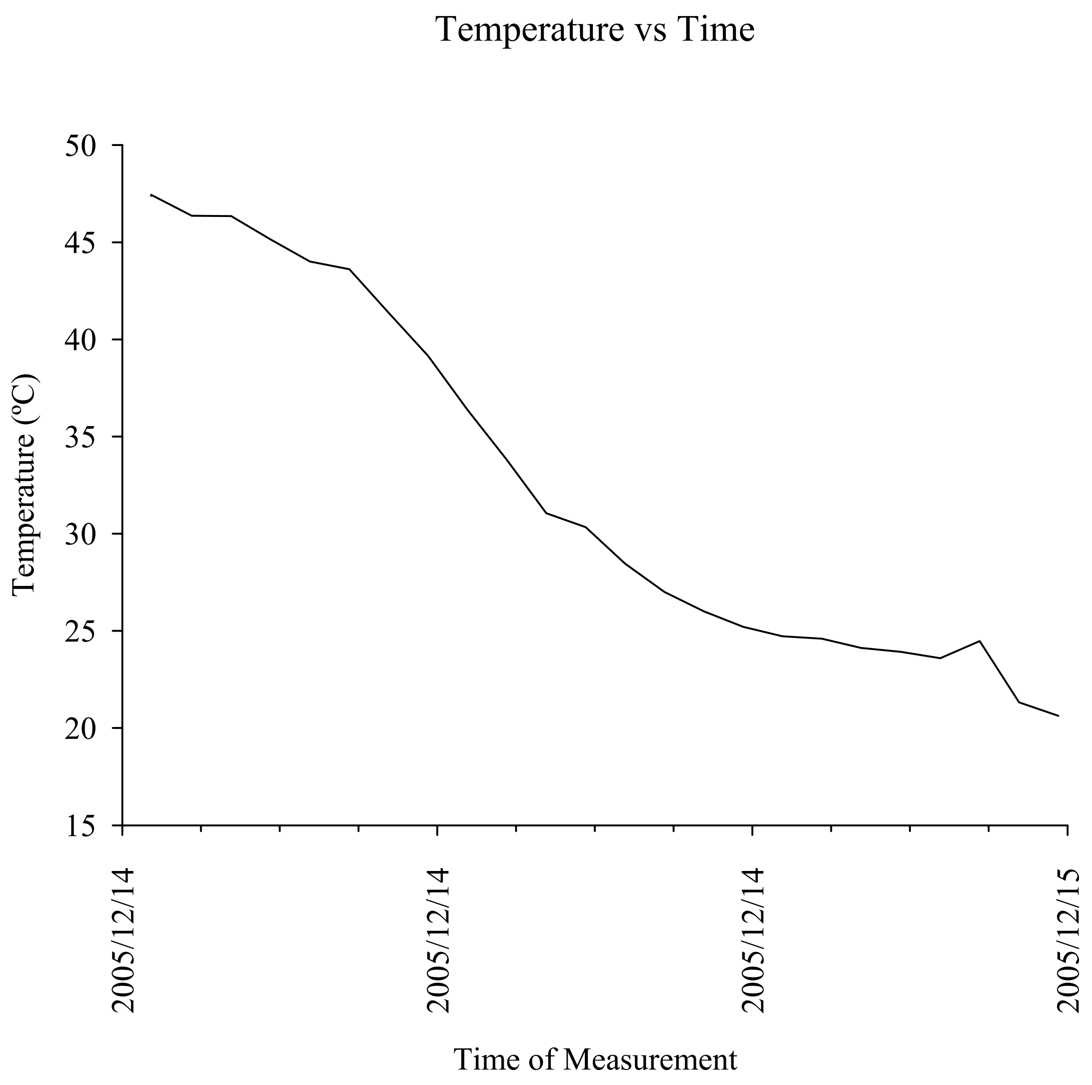


Figure 8.6. Graph displaying the temperature measured over a 24 hour period.

8.3 M-BUS REQUEST/RESPONSE TRANSACTION SUCCESS RATE

Due to instability in the underlying network, the M-Bus request/response transaction is not always successful. A failure happens when the Application Server has requested the measurement value 3 times with all the necessary timeouts as described in section 1.4.19 and still do not obtain the requested value. If an M-Bus request/response transaction failure occurs the Application Server will have to re-request the measurement values at a later time. Due to the fact that the Application Server requests measurements in a serial manner: one M-Bus meter after the other, a request/response failure has a negative impact on the entire communication process as it leaves fewer open slots for the Application

Server to send requests to other M-Bus meters. It is however important to state that the success rate of request/response transactions are entirely dependent on the quality of service provided by the underlying network. An unstable low bandwidth network will result in more request/response failures, which in tern will put a greater burden on the entire system to obtain all the measured values. As the quality of service from the underlying network increases, so will the success rate of M-Bus meter request/response transactions. The number of requests made by the Application Server to obtain each value during the experimental setup is given in figure 8.7.

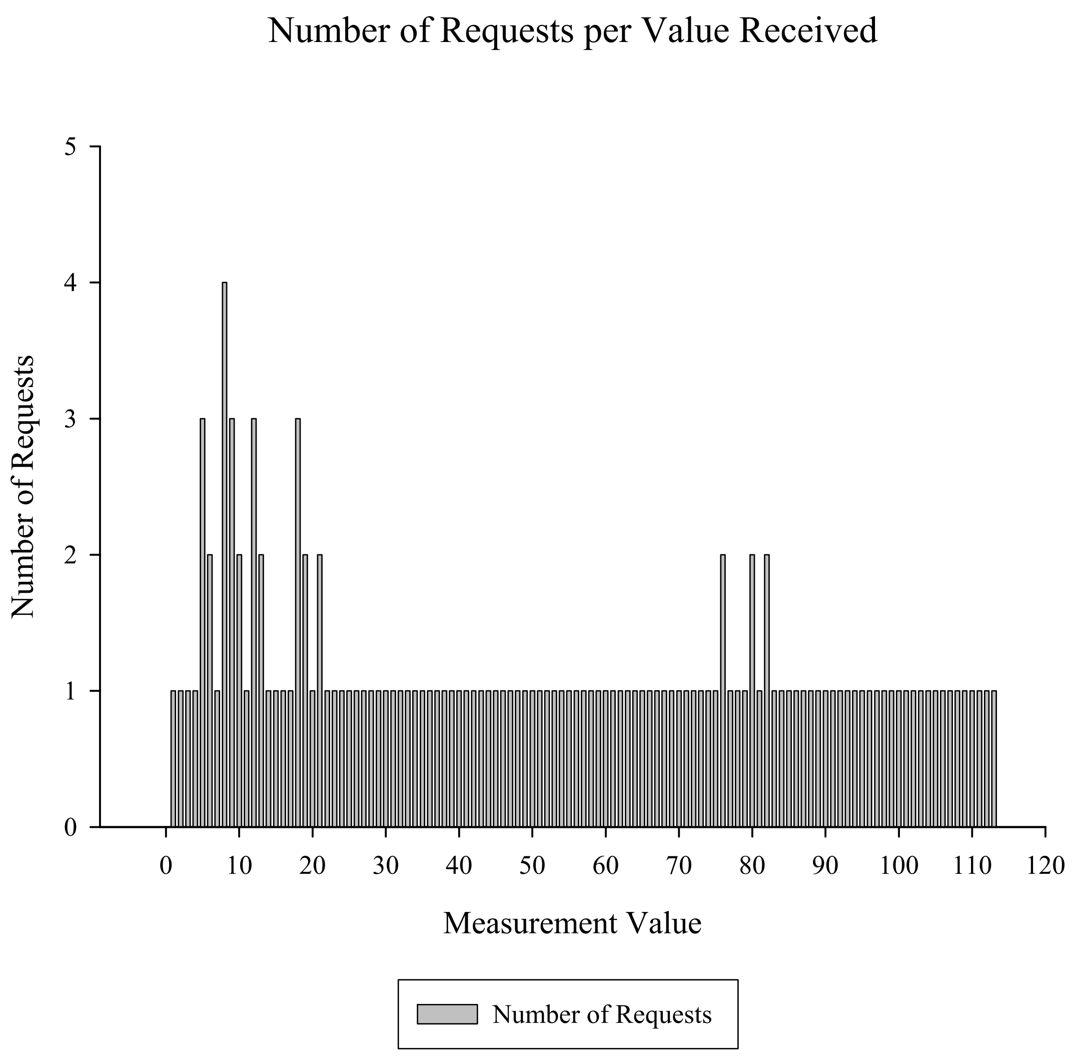


Figure 8.7. Graph displaying the number of Application Server requests required to obtain each measurement value.

The M-Bus protocol is limited in that it does not provide for a keep-alive signalling mechanism, thus a request/response transaction is limited by the available timeout window provided by the layer 2 M-Bus protocol to obtain the data from the meter. It is important to assess the trade off between the quality of service provided by the underlying network and the number of meters connected to the network, before implementing the M-Bus protocol over a remote network infrastructure. If the quality of service is not good enough the negative impact of the re-requests will reach a point where it is impossible to obtain all the values required by the Application Server from all the meters connected to the network. Figure 8.6 show that 113 measurements were received of which 8 values required 2 requests from the Application Server and 4 values required 3 requests from the Application Server and 1 value required 4 requests from the Application Server. Most of the request/response failures occurred during the period when the 4th and 22nd values where requested. This indicates that the underlying network was very unstable during this period. On average the Application Server required 1.168 requests per measurement received. The underlying network used during the experimental gathering of measured values thus resulted in a success rate of 88.5% of requests being received during the first Application Server request, 7.07% of requests being received during the second Application Server request, 3.54% of requests being received during the third Application Server request and 0.88% of requests being received during the forth Application Server request.

8.3.1 Network Quality of Service versus Number of Requests per M-Bus Meter Trade-Off

When implementing the remote metering communication network as proposed in this document a trade-off exists between the network QoS and the number of remote meters that can be connected to a single Application Server. The QoS of the underlying network thus have an impact on the scalability of the remote metering infrastructure, and also plays an important role in the reliability of measurements received. Therefore it is important to be able to determine the trade-off impact that the network QoS has on the remote metering infrastructure. The relationship between the number of meters connected to a single Application Server and the QoS provided by the underlying network is given in equation 8.1

$$N_m = \frac{T_{Avail.}}{RTT_{avg} \times Nr_{req/m}} \tag{8.1}$$

where N_m is the total number of meters that can be connected to an Application Sever, T_{Avail} is the available time in seconds to obtain all the required measurements, RTT_{avg} is the average Round Trip Time (RTT) in seconds and $Nr_{req/m}$ is the average number of request made per meter during the available time period. Thus if a typical usage scenario for the Application Server is to poll each meter once a day (24 hours) for a single value and the average number of requests per meter is 1.16 as obtained in the experiment given in section 8.3 with an average RTT of 1.5 and the these values are substituted into equation 8.1, a total of 49655 M-Bus meters is obtained. If each Node contains 250 M-Bus meters this would result in 199 Nodes connected to a single Application Sever. It is clear out of equation 8.1, that an increase in the RTT or the average number of requests per meter would result in a decrease of the total number of M-Bus meters that can be connected to the Application Server. If the underlying network in the remote communication infrastructure is inaccessible for extended periods of time, the required measurements might be lost, as the Application Server may no longer be able to request values from all the meters connected to it, thus it is also important to take this into account when determining the number of M-Bus meters that can be accessed by one Application Server.

8.4 M-BUS PROTOCOL DRIVER EXECUTION TIME

The execution time taken by the M-Bus Protocol Driver has an impact on the tunnelling architecture as it creates overhead on the communication network and uses up some of the limited time allowed for a request/response transaction by executing software procedures, which cannot be eliminated as part of the communication mechanism, [33]. Measuring the impact of the executing procedures contained within the M-Bus Protocol is thus important in assessing the impact of the execution time on the available timeout window allowed for the request/response transaction.

8.4.1 Experimental Setup

The experimental M-Bus Protocol Driver execution times were performed on an AMD Sempron 2800+ PC, running at 2.00GHz with 512MB RAM running Windows XP with Service Pack 2. Execution times will vary based on the implementation platform and the quality of the compiler, but these results serve as an indicator of the impact that the M-Bus Protocol Driver execution time will have relative to the M-Bus request/response timeout window and should give an accurate enough indication to this affect.

To determine the total execution time required by the M-Bus Protocol Driver, the individual execution times of the Application Server-Side Driver, Access Point-Side Driver and the Node-Side Driver were measured and added together. Each individual component was executed 25 times in obtaining execution times with a high level of confidence. Each component's average execution time was added to determine the total execution time.

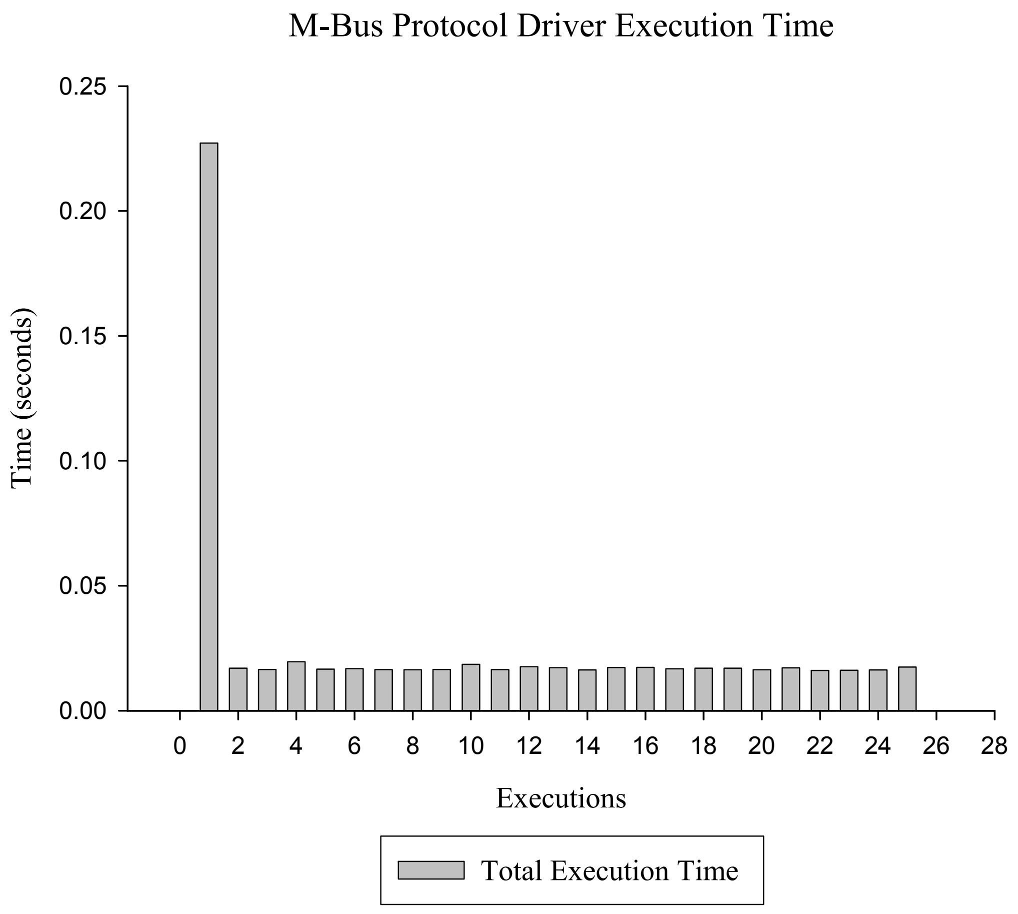


Figure 8.4. Graph displaying the total execution time measured over 25 experiments.

The graph shown in figure 8.4 shows the total M-Bus Protocol Driver execution times measured. The measured values are given in table 8.2. It is interesting to notice the very first measurement is 0.227 seconds, which is nearly 14 times greater than any of the other measured values. This can be expected as the .NET framework's runtime environment is responsible for allocating memory to all the objects contained in the M-Bus Protocol Driver. The allocation of resources takes place the very first time the software is executed and the objects are created. The first reading will be ignored in determining the average execution time and the variance as this is special case which only occurs the first time the program is executed. The average times calculated for the Application Server-Side Driver, Access Point-Side Driver and Node-Side Driver respectively are: 0.0055, 0.0080, 0.0033 seconds. The total time calculated by adding these values is thus 0.016933 seconds. The standard deviation is less than 0.04% which indicates a very high level of confidence. The measured total execution time of 0.016933 seconds is only 0.5% of the total M-Bus request/response timeout window of 3.45 seconds. The impact that the execution time of the M-Bus Protocol Driver will thus have on the timeout window is very small and will on average reduce the total timeout window to 3.433 seconds, which is an acceptably low impact.

Table 8.2. Execution times for each software component contained in the M-Bus Protocol Driver. All values are measured in seconds.

Application Server Exec. Time	Access Point Exec. Time	Node-Side Exec Time
0.019204117	0.112420992	0.095576952
0.005531709	0.008152433	0.003302375
0.005482261	0.007926426	0.00308
0.005411023	0.008192661	0.005984559
0.005606858	0.008000179	0.003038934
0.005591772	0.007986769	0.003238959
0.005489245	0.007864687	0.003018261
0.005503772	0.007905195	0.002948419
0.005587302	0.007976153	0.002923277
0.006501385	0.008934376	0.003063238
0.005528077	0.007917207	0.002927746
0.00556188	0.008520915	0.003527823
0.005479188	0.007869995	0.003802439
0.005479188	0.007865245	0.00291741
0.005544001	0.008286249	0.003397639
0.005532826	0.007919163	0.00383708
0.005467455	0.008111087	0.003174705
0.005479467	0.008170312	0.003331429
0.005475556	0.008225906	0.003292877
0.005488686	0.007886198	0.002945067
0.005409905	0.008225906	0.003488432
0.005403759	0.007788699	0.002931937
0.005439518	0.007832001	0.002935848
0.005439518	0.007829207	0.003007924
0.00546969	0.00845303	0.003534248

8.5 CONCLUSION

The results obtained show that the tunnelling approach used to implement the remote infrastructure for acquiring measurement values from M-Bus meters is a viable and scalable system. The results show that the proposed remote communications infrastructure suggested, allows for diversity by supporting any one of a multitude of communication mediums, including GPRS and PLC. Due to the lack of a keep-alive mechanism within the M-Bus protocol the success of request/response transactions are dependent on the RTT of the underlying network. The QoS provided by the underlying network thus has a big influence in the success of the implemented system. Therefore it is essential that the underlying network conform to the QoS requirement of the implemented remote metering system.

The overhead added to the communication process due to the execution times required by the distributed M-Bus driver is marginal and for practical purposes can be ignored. Thus the distributed driver system is a viable way to implement the addressing requirements and the protocol diversity requirements of the REMPLI system and the protocol tunnelling approach. Further more the triplet of drivers provide a transparent manner of communicating between application servers and metering equipment.

The results also show that this protocol tunnelling approach taken for the acquisition of M-Bus meter values is clearly not suited to mission critical systems or real-time based system. The M-Bus protocol is more suited to situations where the need arises to obtain relatively small amounts of data from a limited number of meters over the remote network. The risks involved in using the M-Bus protocol for obtaining measurement values are as follows:

- Failure of underlying network causes failure of entire measurement system.
- No guarantee of obtaining the values from the remote meters.
- Success is heavily reliant on the QoS provided by the underlying network.

These risks have to be taken into consideration when setting up an M-Bus based protocol tunnelling remote metering system. Systems ideally suited to this type of implementation include:

- Readings by utility companies, which monitor the average usage of the resources/services that they provide.
- Reading requirements where a small amount of data is required from all or several meters once a day/week/month, where the resultant measurements are not mission critical.

The results show that the distributed driver architecture implemented in this document allows application servers to transparently retrieve values from meters which are remotely located over TCP/IP and GPRS based networks.

8.6 FURTHER WORK

Further research to be done include the following:

- In a full scale fieldbus implementation multiple concurrent metering protocols will be operating at the same time. The influence of increasing metering protocols and the influence it will have on the available bandwidth of the underlying network needs to be investigated to determine the overall QoS requirements for a full scale fieldbus system.
- The influence of the QoS and the variance thereof provided by the underlying network must be investigated and related to the number of nodes and meters connected to the fieldbus system, in order to ensure that the required levels of reliability and data integrity are maintained for the fieldbus system
- The need arises to determine the optimal number of remote meters per metering protocol and the optimal number of nodes that the fieldbus system can support. This research suggest an equation to roughly determine the maximum number of meters that can be used based on the number of readings required per meter per time unit, but more work needs to be done to take the number of concurrent protocols and other factors into account to more precisely determine the limitations of the fieldbus system.
- An entire address translation algorithm or scheme can be developed according to specific user needs. The two main approaches are mentioned in this research, but it can be further optimized and compared to each other.

- The optimization scheme suggested in this research can also be further developed according to the number of protocols running concurrently and to optimally utilize the underlying network and communication system in order to improve the performance of the entire communication structure provided by the distributed driver architecture.
- Currently the M-Bus protocol is implemented with the connection-orientated nature of the M-Bus meters in mind. Perhaps the M-Bus user group could consider a keep-alive scheme for the M-Bus protocol as supported in some of the other metering protocols.

References

[1] A. Bratukhin, G. Pratl, M. Lobashov, P. Maas, "Data transparency in PLC-based SCADA and metering protocols", *2005 International Symposium on Power Line Communications and Its Applications*, pp. 76 – 79, 6-8 April 2005.

[2] M. Lobashov, G. Pratl, T. Sauter, "Applicability of Internet protocols for fieldbus access", *4th IEEE International Workshop on Factory Communication Systems*, pp. 205 – 213, 2002.

[3] F. Miehlisch, "The M-Bus: A Documentation Rev. 4.8", 1998, http://www.m-bus.com/mbusdoc/default.html. Last accessed on 13 May 2006.

[4] M. Felser, T. Sauter, "The fieldbus war: history or short break between battles?", *4th IEEE International Workshop on Factory Communication Systems*, pp. 73-80, 2002.

[5] L. Ferreira, E. Tovar, "Timing analysis of a multiple logical ring wired/wireless PROFIBUS network", *2004 IEEE International Workshop on Factory Communication Systems, 2004. Proceedings*, pp. 81 – 90, 22-24 Sept. 2004.

[6] A. Willig, "An architecture for wireless extension of PROFIBUS", *The 29th Annual Conference of the IEEE Industrial Electronics Society, 2003. IECON '03*, pp. 2369 - 2375 Vol.3, 2-6 Nov. 2003.

[7] A. Willig, A. Kopke, "The adaptive-intervals MAC protocol for a wireless PROFIBUS", *Proceedings of the 2002 IEEE International Symposium on Industrial Electronics*, pp. 61 – 66, 8-11 Jul. 2002.

[8] W. Stallings, *Data and Computer Communications 6th Edition*, Prentice-Hall, 2000.

[9] J. P. Thomesse, "Fieldbuses and interoperability", *Control Engineering Practice*, vol. 7, 1999, pp. 81-94.

[10] Jun Xu, Yan-Jun Fang, "Profibus automation technology and its application in DP slave development", *International Conference on Information Acquisition*, pp. 155-159, 21-25 Jun. 2004.

[11] L. Lo Bello, O. Mirabella, "A multi-ring scheduling strategy for Profibus networks", *The 27th Annual Conference of the IEEE Industrial Electronics Society, 2001. IECON '01*, pp. 2144 – 2148, vol. 3, 29 Nov.-2 Dec. 2001.

[12] REMPLI overview, http://www.rempli.org/, last accessed 13 May 2006.

[13] E. Nikoloutsos, A. Prayati, A.P. Kalogeras, V. Kapsalis, S. Koubias, G. Papadopoulos, "Integrating IP traffic into fieldbus networks", *Proceedings of the*

2002 IEEE International Symposium on Industrial Electronics, pp. 67-72, 8-11 Jul. 2002.

[14] M. Knizak, M. Kunes, M. Manninger, T. Sauter, "Applying Internet management standards to fieldbus systems", *IEEE International Workshop on Factory Communication Systems*, pp. 309 – 315, 1-3 Oct. 1997.

[15] M Kunes, T. Sauter, "Fieldbus-internet connectivity: the SNMP approach", *IEEE Transactions on Industrial Electronics*, volume 48, pp. 1248 – 1256, Dec. 2001.

[16] P. Neumann, C. Diedrich, R. Simon, "Necessary extensions of fieldbus systems for distributed processing", *1995 IEEE International Workshop on Factory Communication Systems, 1995. WFCS '95, Proceedings*, pp. 247 – 254, 4-6 Oct. 1995.

[17] S. Deter, "Fieldbus device description using tag-based trees", *6th Africon Conference in Africa, IEEE AFRICON*, volume 1, pp. 263 – 268, 2–4 Oct. 2002.

[18] A.F.J. von Gordon, G.P. Hancke, "Protocol conversion for real-time energy management systems", *2004 IEEE International Workshop on Factory Communication Systems, 2004 Proceedings*, pp. 319 – 322, 22-24 Sept. 2004.

[19] H.C. Ferreira, H.M. Grove, O. Hooijen, A.J. Han Vinck, "Power line communications: an overview", *4th AFRICON, 1996*, volume 2, pp. 558 – 563, 24–27 Sept. 1996.

[20] D. Cooper, "Low-data-rate narrow-band power-line communications on the European Domestic Mains: symbol timing estimation", *IEEE Transactions on Power Delivery*, pp. 664 – 667 vol. 20 issue 2 part 1, April 2005.

[21] M. Crussiere, J.-Y. Baudais, J.-F. Helard, "Robust and high-bit rate communications over PLC channels: a bit-loading multi-carrier spread-spectrum solution", *2005 International Symposium on Power Line Communications and Its Applications*, pp. 37 – 41, 6–8 April 2005.

[22] Chan Tat-Wai, Lim Tian-Yew, "Application of A-Band LV power line standards for remote metering at populous residential buildings", *Asia Pacific. IEEE/PES Transmission and Distribution Conference and Exhibition 2002*, volume 2, pp. 969-974, 6-10 Oct. 2002.

[23] M. Wada, O. Nakamura, K. Akiyama, "Development of remote meter reading system for distribution automation", *Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply*, pp. 212 - 216, 17-19 Nov. 1992.

[24] J. Newbury, W. Miller, "Multiprotocol routing for automatic remote meter reading using power line carrier systems", *IEEE Transactions on Power Delivery*, Volume 16, Issue 1, pp. 1 – 5, Jan. 2001.

[25] P. Foord, J. Tsoucalas, "Remote meter reading, load control and distribution system automation utilising SWD technology", *Sixth International Conference on Metering Apparatus and Tariffs for Electricity Supply*, pp. 163 – 167, 3-5 Apr. 1990.

[26] A.D. Craig, P.M. Moore, C.D. Long, "Experience with a high speed MV/LV power line communications system for remote metering, load control and network automation", *Eighth International Conference on Metering and Tariffs for Energy Supply*, pp. 165 – 169, 3-5 July 1996.

[27] I. Erakovic, D. Rakic, V. Lapcevic, S. Marjanovic, M. Djonovic, "A system for remote meter reading and load management", *Ninth International Conference on Metering and Tariffs for Energy Supply*, pp. 196 – 199, 25-28 May 1999.

[28] S.Y. Kwankam, N.N. Ningo, J. Bilikha, F. Nematchoua, J.M. Ngundam, "Application of a new digital power line communication technology to remote reading of standard electric utility meters", *Proceedings of the 35th Midwest Symposium on Circuits and Systems*, Volume 2, pp. 1534 – 1537, 9-12 Aug. 1992.

[29] J. Newbury, W. Miller, "Potential communication services using power line carriers and broadband integrated services digital network", *IEEE Transactions on Power Delivery*, Volume 14, Issue 4, pp. 1197 – 1201, Oct. 1999.

[30] G. Bumiller, T. Sauter, G. Pratl, A. Treytl, "Secure and reliable wide-area power-line communication for soft-real-time applications within REMPLI", *2005 International Symposium on Power Line Communications and Its Applications*, pp. 57 – 60, 6–8 April 2005.

[31] F. Pacheco, M. Lobashov, M. Pinho, G. Pratl, "A power line communication stack for metering, SCADA and large-scale domotic applications", *2005 International Symposium on Power Line Communications and Its Applications*, pp. 61 – 65, 6-8 April 2005.

[32] P.B. Kruchten, "The 4+1 View Model of architecture", *IEEE Software*, Volume 12, Issue 6, pp. 42 – 50, Nov. 1995.

[33] A. Mendes, L. Ferreira, E. Tovar, "Fieldbus networks: real-time from the perspective of the application tasks", *IEEE International Workshop on Factory Communication Systems*, pp. 275 – 282, 6-8 Sept. 2000.

ADDENDUM A

M-BUS PROTOCOL STACK

The following section gives a full overview of the Data link layer M-Bus protocol stack as described in the M-Bus standard (EN 13757-2), [18] and demonstrates the limited capability of the protocol, because it is originally designed to operate on a serial bus LAN network, as outlined in [3].

Data link layer

This protocol uses asynchronous serial bit transmission, in which the synchronisation is implemented with start and stop bits for each character. This mode of transmission is used because the M-bus protocol makes provision for a party line network configuration also know as a bus topology. There must be no pauses within a telegram, not even after a stop bit. The start bit must be a Space (logical 0) and the stop bit must be Mark (logical 1). Between the start and stop bits, eight data bits and the even parity bit are transmitted, ensuring that at least every eleventh bit is a Mark. The data bits are transmitted in ascending order, i.e. the bit with the lowest value LSB (least significant bit) is the first bit to be found on the transmission line. Transmission takes place in half-duplex at a data rate of at least 300 Baud. Figure A.1 shows how the transmission of a byte in calling and replying direction is represented. In figure A.2 a comprehensive view is given of the general structure of both a character and a frame within the M-Bus protocol. This figure also shows the start, stop and parity bits, with the parity bit set to “1” if there is an odd number of “1”s within the 8 data bits.

Transmission Rules

The transmission rules are summarised in table A.1. The frame is rejected if one of these checks fails, otherwise it can be released to the user.

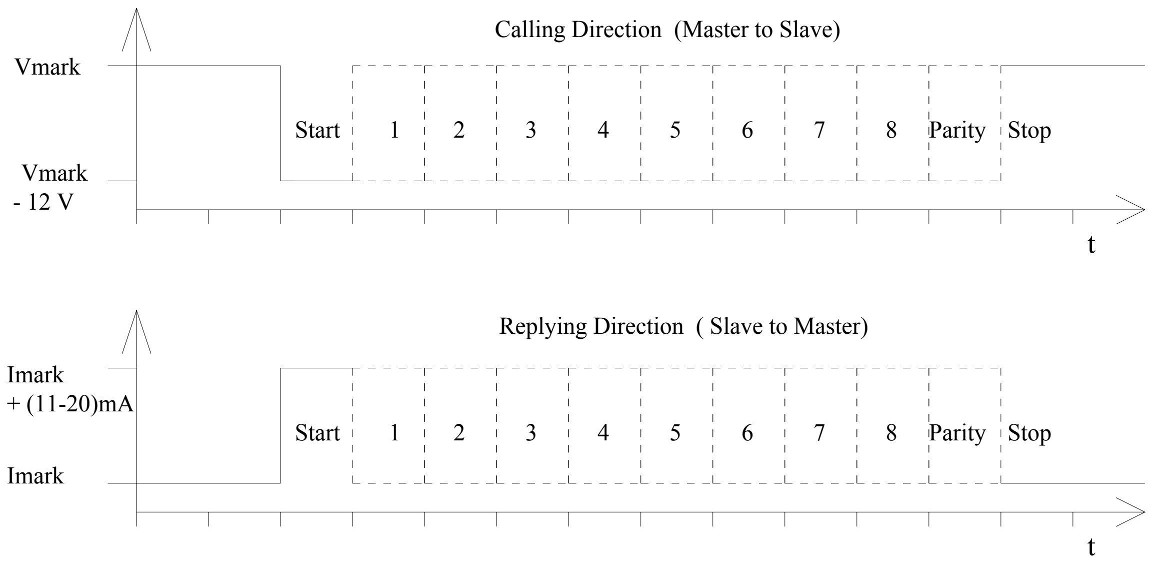


Figure A.1 Transmission of a character in calling and replying direction.

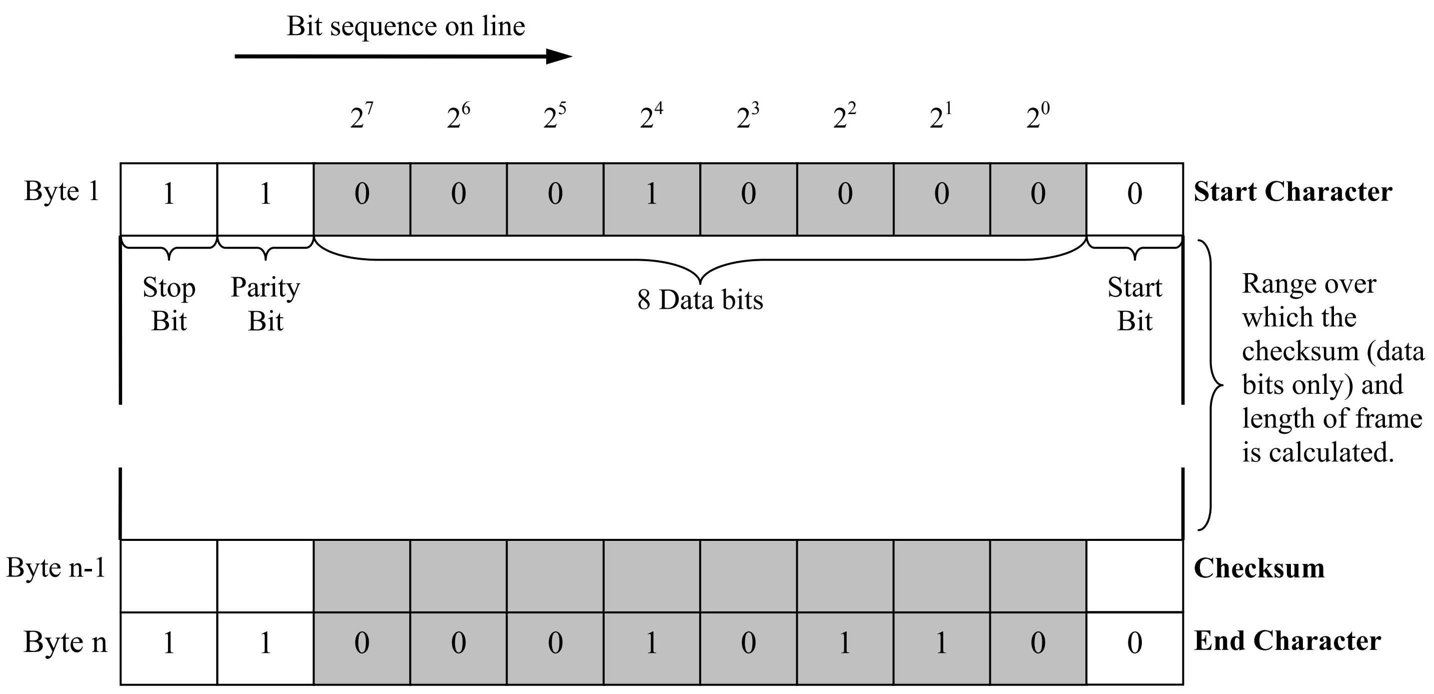


Figure A.2 Structure of an M-Bus frame.

Table A.1 Table with transmission rules for the M-Bus protocol.

Transmission Rules	
1	Line idle is binary 1.
2	Each character has one start bit (binary = 0) 8 information bits, one even parity bit and one stop bit (binary = 1).
3	No line idle intervals are admitted between characters of a frame.
4	Upon detecting an error according to rule 6, a minimum interval of 33 bits (3 characters) is required between frames.
5	The sequence of user data characters is terminated by an 8 bits checksum (CS). The checksum is the arithmetic sum over all user data octets.
6	The receiver checks:
6.1	the start bit, the stop bit and the even parity bit per character.
6.2	the start character, the length (2 bytes in frames with variable lengths), the frame check sum and the end character per frame and, upon detecting an error, the line idle interval specified by rule 4

Frame/Telegram formats supported within the M-Bus Protocol

The frame/telegram format is shown in figure A.3 given below. A subsequent description is given for each frame format.

Single Character	**Short Frame**	**Control Frame**	**Long Frame**
E5h	Start 10h	Start 68h	Start 68h
	C Field	L Field = 3	L Field
	A Field	L Field = 3	L Field
	Check Sum	Start 68h	Start 68h
	Stop 16h	C Field	C Field
		A Field	A Field
		CI Field	CI Field
		Check Sum	User Data (0-252 Byte)
		Stop 16h	Check Sum
			Stop 16h

L Length field range between 0-255 and specify the number of subsequent user data octets including the control, address and control information fields
C Control field
A Address field (link)
CI Control Information field

Figure A.3 Telegram formats used in the M-Bus protocol.

Single Character

This format consists of a single character, namely the E5h (decimal 229), and serves to acknowledge receipt of transmissions.

Short Frame

This format with a fixed length begins with the start character 10h, and besides the C and A fields includes the check sum (this is made up from the two last mentioned characters), and the stop character 16h

Long Frame

With the long frame, after the start character 68h, the length field (L field) is first transmitted twice, followed by the start character once again. After this, the Control field (C field), address field (A field) and control information field (CI field) follows. The L field gives the quantity of the user data inputs plus 3 (for C,A,CI). After the user data is inserted into the frame, the checksum is transmitted, which is built up over the same area as the length field, and in conclusion the stop character 16h is transmitted.

Control Frame

The control frame conforms to the long frame without user data, with an L field with the value of 3. The check sum is calculated at this point from the fields C, A and CI.

M-Bus Fields

The fields indicated within the telegrams as shown in figure 1.8 will be discussed in the following sections. All the fields have a length of 1 Byte, corresponding to 8 bits.

C Field (Control Field, Function Field)

Besides labelling the functions and the actions caused by them, the function field specifies the direction of data flow, and is responsible for various additional tasks in both the calling and replying directions. Figure A.4 shows the coding of the individual bits of the C field:

Bit Number	7	6	5	4	3	2	1	0
Calling Direction	0	1	FCB	FCV	F3	F2	F1	F0
Reply Direction	0	0	ACD	DFC	F3	F2	F1	F0

Figure A.4 Coding of the control field.

The highest value (most significant) bit is reserved for future functions, and at present is allocated the value 0; bit number 6 is used to specify the direction of data flow. The frame count bit FCB indicates successful transmission procedures (i.e. those that have been replied to or acknowledged), in order to avoid transmission loss or multiplication. If the expected reply is missing or reception is faulty, the master sends the same telegram again with an identical FCB, and the slave replies with the same telegram as previously. The master indicates with a 1 in the FCV bit (frame count bit valid), that the FCB is used. When the FCV contains a "0", the slave should ignore the FCB. A summary of each bit and its function is given in table A.2.

In the replying direction, both these bits can undertake other tasks. The DFC (data flow control) serves to control the flow of data, in that the slave with a DFC=1 indicates that it can accept no further data. With an ACD bit (access demand) with a value of 1, the slave shows that it wants to transmit Class 1 data. The master should then send it a command to request Class 1 data. Such Class 1 data is of higher priority, which (in contrast to Class 2 data) should be transmitted as soon as possible. The support of Class 1 data and the bits DFC and ADC is not required by the standard.

The bits 0 to 3 of the control field code the true function or action of the message. Table A.3 shows the function codes used in the calling and the replying directions. The functions shown in this table will be explained in more detail in the next section. All additional function codes defined in IEC 870-5-2 can also be used.

Table A.2 Functions of all the bits contained in the M-Bus control field.

Bit 7	Reserved
Bit 6	Primary message 0 = message from secondary (responding) station. 1 = message from primary (initiating) station.
FCB	Frame count bit: 0 - 1 = alternating bit for successive SEND/CONFIRM or REQUEST/RESPOND services per station. The frame count bit is used to delete losses and duplications of information transfers. The primary station alternate the FCB bit for each new SEND/CONFIRM or REQUEST/RESPOND transmission service directed to the same secondary station. Thus the primary station keeps a copy of the frame count bit per secondary station. If an expected reply is timed out (missing) or corrupted, then the same SEND/CONFIRM and REQUEST/RESPOND service is repeated with the same frame count bit. In case of reset commands the FCB bit is always zero, and upon receipt of these commands the secondary station will always be set to expect the next frame primary to secondary with FCV = valid (FCV = 1) to have the opposite setting of FCB, i.e. FCB equal to one
FCV	Frame count bit valid. 0 = alternating function and FCB bit is invalid 1 = alternating function of FCB bit is valid SEND/NO REPLY services, broadcast messages and other transmission services that ignore the deletion of duplication or loss of information output do not alternate the FCB bit and indicates this by a cleared FCV bit
DFC	Data flow control 0 = further messages are acceptable 1 = further messages may cause data overflow Secondary (responding) stations indicate to the message initiating (primary) station that an immediate secession of further message may cause a buffer overflow.
ACD	Access demand. There are two classes of message data provided, namely class 1 and 2. 0 = no access demand for class 1 data transmission 1 = access demand for class 1 data transmission Class 1 data transmission is typically used for events or for messages with high priority. Class 2 data transmission is typically used for cyclic transmission or for low priority messages.
Bit 3-0	Functions described in table 3

Table A.3 Control Codes of the M-Bus Protocol (F : FCB-Bit, A : ACD-Bit, D : DFC-Bit)

Name	**C Field Binary**	**C Field Hex.**	**Telegram**	**Description**
SND_NKE	0100 0000	40	Short Frame	Initialization of Slave
SND_UD	01F1 0011 F = 1 / 0	53/73	Long/Control Frame	Send User Data to Slave
REQ_UD2	01F1 1011 F = 1 / 0	5B/7B	Short Frame	Request for Class 2 Data
REQ_UD1	01F1 1010 F = 1 / 0	5A/7A	Short Frame	Request for Class1 Data (Alarm Protocol)
RSP_UD	00AD 1000 A,D = 1 / 0	08/18/28/38	Long/Control Frame	Data Transfer from Slave to Master after Request

A Field (Address Field)

The address field serves to address the recipient in the calling direction, and to identify the sender of information in the receiving direction. The size of this field is one Byte, and can therefore take values from 0 to 255. The addresses 1 to 250 can be allocated to the individual slaves, up to a maximum of 250. Unconfigured slaves are given the address 0 at manufacture, and as a rule are allocated one of these addresses when connected to the M-Bus. The addresses 254 (FEh) and 255 (FFh) are used to transmit information to all participants (Broadcast). With address 255 none of the slaves reply, and with address 254 all slaves reply with their own addresses. The latter case naturally results in collisions when two or more slaves are connected, and should only be used for test purposes. The address 253 (FDh) indicates that the addressing has been performed in the Network Layer (see chapter 7) instead of Data link layer. The remaining addresses 251 and 252 have been kept for future applications.

CI Field (control information field)

The control information field is already a part of the Application Layer. It is included in the telegram format, in order to distinguish between the formats of the long and the control frames. The control information allows the implementation of a variety of actions in the master or the slaves.

Check Sum

The Check Sum serves to recognize transmission and synchronization faults, and is configured from specific parts of the telegram. These parts are mentioned when presenting the individual telegram formats. The Check Sum is calculated from the arithmetical sum of the data mentioned above, without taking carry digits into account. In other words the checksum of the telegram 16 5B FA 16 is equal to: 16hex + 5Bhex= 6Bh+FAh=65h+16h=7Bh.

Communication procedures

The Data link layer implements two types of transmission services:

- Send/Confirm: SND/CON
- Request/Respond: REQ/RSP

It is important to note that after the reception of a valid telegram the slave device has to wait between 11 bit periods and (330 bit periods +50ms) before answering also refer to EN1434-3.

Send/Confirm Procedures

SND_NKE → Single control character

This procedure serves to start up after the interruption or beginning of communication. The value of the frame count bit FCB is adjusted in master and slave, i.e. the first master telegram with FCV=1 after SND_NKE contains a FCB=1. The slave responds to a correctly received SND_NKE with an acknowledgment consisting of a single character (E5h).

SND_UD → Single control character

With this procedure the master transfers user data to the slave. The slave can either confirm the correct receipt of data with a single character acknowledge (E5h), or by omitting a confirmation signal that it did not receive the telegram correctly.

Request/Respond Procedures

REQ_UD2 → RSP_UD

The master requests data from the slave according to Class 2. The slave can either transfer its data with RSP_UD, or give no response indicating that the REQ_UD2 telegram has not been received correctly or that the address contained in the REQ_UD2 telegram does not match.

Minimum Communication

According to the European standard EN1434-3, as a minimum for communication the procedures REQ_UD2 / RSP_UD and SND_NKE / $E5 are needed. All other functions are optional.

Transmission Procedures in case of faults

A fault in a received telegram can be detected by the receiver (master or slave), by checking the following points:

- Start /Parity /Stop bits per character.
- Start /Check Sum /Stop characters per telegram format.
- The second Start character, the parity of the two field lengths, and the number of additional characters received (= L Field + 6) with a long or control frame.

When a fault has been detected as a result of the above checks, the transmission will not be accepted, and the reply or acknowledgement will not be sent. After a time limit of (330 bit periods + 50 ms) the master interprets the lack of a reply as a fault and repeats the same telegram up to two times. If a valid telegram has not been received at that time a so called "idle time" of at least 33 bit periods is introduced. When slaves send faulty or corrupt replies, three attempts are also made, and if there is a fault during the last attempt then the 33 bit periods "idle time" is introduced. The master may try a SND_NKE. If this fails also it will continue with the next slave address.

Multi-telegram answers (RSP_UD) from slave to master

If a total answer sequence from a slave will not fit into a single RSP_UD (respond user data) telegram from the slave to the master, the master signals by a toggled FCB-Bit together with a set FCV-Bit in the next REQ_UD (Request user data) telegram that its link layer has properly received the last RSP_UD-telegram from the slave. The slave answers to a REQ_UD-request with toggled FCB-Bit with a set FCV-bit from the master with a RSP_UD containing the next link layer telegram section of a multi-telegram answer, otherwise it will repeat the last telegram. Note that a slave with a single RSP_UD-telegram may simply ignore the FCB in the REQ_UD2-telegram and send always the same (single) telegram. Note also that a slave with exactly two (sequential) RSP_UD-answer telegrams may simply use the FCB of the REQ_UD2 to decide which of both telegrams should be transmitted. Thus a slave with one or two (sequential) RSP_UD answer-telegrams does not require an internal "Last-REQ_UD2-FCB"-image bit. A slave with three or more (sequential) RSP_UD answer telegrams requires such an internal memory bit. Note that such an internal memory bit for the RSP_UD-direction must be independent of a possible additional internal memory bit for the SND_UD direction (see master to slave section).

Frozen answer telegrams from slave to master

In same instances a slave will freeze the data of its last RSP_UD answer telegram into an additional temporary storage and will repeat these previously frozen RSP_UD answers, if the FCB has not been toggled. After the reception of a toggled FCB-Bit with a set FCV-Bit or after the reception of a REQ_UD2 with the FCV-Bit cleared, the slave will generate a next answer telegram reflecting the current state of all its data instead of repeating the data values frozen at the first REQ_UD2 attempt with toggled FCB. In meter applications this frozen-telegram approach will result in possibly very old meter data if the last REQ_UD2 with toggled FCB-bit occurred a long time ago. Thus for meter readout this frozen telegram technique is not recommended.

Multi-telegram data (SND_UD) from master to slave

If the master sends a large (sequential) data block to a slave (e.g. RAM/EEPROM-initialize, code upload) which must be divided into multiple telegrams a similar situation like in the slave to master direction might occur. If the slave receives a telegram correctly and answers with a positive acknowledge (usually by a $E5 single byte answer) but the master does not receive this positive answer correctly, the master will repeat the last telegram with the identical FCB-Bit as in the first attempt. From this the slave can recognize that this next telegram does not contain the next data block but repeats the last data block which has been received correctly. So the slave may either ignore this telegram repetition completely or may accept it thus overwriting the last telegrams data with the second identical data. In both cases an internal telegram sequence counter is not incremented. Note that a slave who will accept only single telegram master to slave communication may simply ignore the FCB in the SND_UD. Note also that a master which can accept exactly two (sequential) SND_UD-telegrams may simply use the FCB of the SND_UD to decide which of both telegrams has been sent. Thus a slave which can accept one or two (sequential) SND_UD answer-telegrams does not require an internal "Last-SND_UD-FCB"-image bit. A slave which can accept three or more (sequential) SND_UD telegrams requires such an internal memory bit. Note that such an internal memory bit for the SND_UD-direction must be independent of a possible additional internal memory bit for the RSP_UD direction.

Incremental actions in slave initiated by master

If single telegram SND_UD will initiate some incremental action in a slave (like toggling a relais or counting something) in contrast to sending some "absolute" data or parameters the FCB-mechanism allows as in the multi-telegram SND_UD situation a distinction between a repetition of the last telegram due to missed acknowledge reception and the next action.

In this case the action is only taken if the FCB of the current SND_UD-telegram is different from the FCB in the previous SND_UD-telegram.

Implementation aspects for primary addressing Master

The master must always contain a "Next REQ_UD2-FCB-image bit" and also a "Next SND_UD-FCB image bit" for each primary slave address used by its application layer. After sending a SND_NKE-request to a slave address both these "Next FCB-image bit" associated with this address contained in the request must be set. Thus for each primary address the first REQ_UD2 or SND_UD telegram after a SND_NKE contains a set FCB-Bit. Note that after a memory loss (usually due to a power failure) of these "Next FCB-image bits" the master is required to send a SND_NKE to all affected addresses. All subsequent RSP_UD2-telegrams must contain the "Next REQ_UD2- FCB-image bit" of the appropriate primary address as a FCB. This "Next REQ_UD2 FCB-image bit" is toggled after an error free link layer RSP_UD telegram has been received. All subsequent SND_UD-telegrams must contain the "Next SND_UD- FCB-image bit of the appropriate primary address as a FCB. If a SND_UD has been successfully transmitted to a slave (reception of a valid acknowledge byte $E5 or a valid RSP_ACK telegram) this "Next SND_UD-FCB-image bit" associated with this address is toggled.

Slave

If a slave wants to utilize the FCB-Bit mechanism for the REQ_UD2-type (slave to master) transfers for more than two sequential telegrams it must provide a "Last REQ_UD2-FCB"-memory bit. If a valid REQ_UD2 telegram with a set FCV-Bit is received its FCB-Bit is compared to this "Last REQ_UD2-FCB-Bit". If they differ or the FCV-bit is clear, the next actual telegram data are used for the RSP_UD answer otherwise the last (stored) telegram is repeated.

If a slave wants to utilize the FCB-Bit mechanism for the SND_UD-type (master to slave) transfers for more than two sequential telegrams it must provide a "Last SND_UD-FCB"-memory bit. If a valid SND_UD telegram with a set FCV-Bit is received, its FCB-Bit is compared to this "Last SND_UD-FCB-memory Bit". If they differ or the FCV-bit is clear, the next actual telegram data are used for the RSP_UD answer otherwise the last (stored) telegram is repeated.

Note that after a valid reception of a SND_NKE to the primary address of the device or to the test address 254 ($FE) or the broadcast address 255 ($255) these internal "Last FCB" memory bits must be cleared.

Implementation for multiple address slaves

A slave might be configured to respond to more than one primary address. This could be useful for slaves which internally consist of more than one independent functional block. If this slave wants to utilize FCB-functionalities they must implement the appropriate number of internal memory bits (0, 1 or 2) for each of these addresses.

Implementation for the primary (broadcast) address 255

All transfers to the primary broadcast address 255 ($FF) are not answered and should hence be implemented by the master with the FCV-Bit cleared. Note that a SND_NKE to primary address 255 will clear the internal "Last received FCB"-Bits of all slaves with primary addresses 0-250 and with FCB-Bit implementation simultaneously.

Implementation for the primary (test) address 254 ($FE)

A slave should answer to all requests to the primary address 254 ($FE=test address) irrespective of its own primary address. The answer must contain its own primary address and not the address 254 ($FE). This test address is used by readout- or test equipment in point-to-point mode. Although this is a second primary address for each slave separate "Last received FCB"- Bit(s) are not required for this special case, since any test equipment or master is required to issue a SND_NKE after each reconnect or power fail thus clearing the "Last received FCB"-Bit(s) and thus preparing for a virgin transaction irrespective of the previous communication history.

Implementation for secondary addressing

The use of the FCB-Bit in secondary addressing is not relevant to the work proposed in this document. The M-Bus Protocol Driver will only operate on the layer 2 protocol stack. Secondary addressing is applied as part of the Application layer protocol stack and is discussed in the application layer section of the M-Bus protocol standard.