Cape Peninsula
University of Technology

# VIRTUALIZATION OF A SENSOR NODE TO ENABLE THE SIMULATION OF IEC 61850-BASED SAMPLED VALUE MESSAGES

**By**

EMMANUEL LUWACA

**Thesis submitted in fulfilment of the requirements for the degree**

**Master of Technology:** Electrical Engineering

**in the Faculty of** Engineering

**at the Cape Peninsula University of Technology**

**Supervisor:** Prof P. Petev
**Co-supervisor:** Mr. C. Kriger

**Bellville Campus**
Date submitted December 2014

## Declaration

I, Emmanuel Luwaca, declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

| | |
|---|---|
| **Signed** | **Date** |

## Abstract

The IEC 61850 standard, "Communication networks and systems in substations" was promulgated to accommodate the need for a common communication platform within substations for devices from different vendors. The IEC 61850 standard proposes a substation automation architecture that is Ethernet-based, with a "station-bus" for protection devices within the substation and a "process bus" where raw data from the voltage and current transformers are published onto the data network using a device known as a Merging Unit.

To date, most of the standardization efforts were focused at the station bus level where event-triggered messages are exchanged between the substation automation devices, commonly referred to as Intelligent Electronic Devices (IEDs). These messages are known as Generic Object Oriented Substation Event messages. Equipment from vendors to accommodate the "process bus" paradigm, however is still limited at present.

The Centre for Substation Automation and Energy Management Systems was established within the Electrical Engineering Department at the Cape Peninsula University of Technology with one of its objectives being the development of equipment either for simulation or real-time purposes in compliance with the IEC 61850 standard. In order to fulfil this long-term objective of the Centre, an in-depth understanding of the IEC 61850 standard is required.

This document details the efforts at acquiring the requisite knowledge base in support of the educational objectives of the Centre and the research project implements a simulation of a merging unit which is compliant with the functional behavior as stipulated by the standard. This limited functional implementation (i.e. non-real-time) of the merging unit, is achieved through the development of a virtualized data acquisition node capable of synthetic generation of waveforms, encoding of the data and publishing the data in a format compliant with the IEC 61850-9-2 sampled value message structure.

This functional behavior of the virtual sensor node which was implemented has been validated against the behavior of a commercial device and the sampled value message structure is validated against the standard. The temporal behavior of the proposed device is commented upon. This research project forms the basis for future real-time implementation of a merging unit.

## Acknowledgements

**I wish to thank:**

Prof P. Petev, Prof R. Tzoneva, Mr. S. Behardien and Mr. C. Kriger for providing great technical and administrative support during this Master of Technology thesis.

The financial assistance of the National Research Foundation towards some of the test equipment used in this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.

## Dedication

Dedicated to my family

# Table of Contents

# LIST OF FIGURES

## LIST OF TABLES

# Glossary of Terms

## Acronyms

| | |
|---|---|
| 1PPS | One Pulse Per Second |
| AC | Alternating Current |
| ADC | Analogue to Digital Converter |
| APDU | Application Protocol Data Unit |
| ASDU | Application Service Data Unit |
| API | Application Program Interface |
| ASN.1 | Abstract Syntax Notation One |
| CDC | Common Data Class |
| CPU | Central Processing Unit |
| CT | Current Transformer |
| CVT | Capacitor voltage transformer |
| DC | Direct Current |
| DNP | Distribution Network Protocol |
| DSP | Digital Signal Processor |
| EMI | Electromagnetic interference |
| FC | Functional Constraint |
| GOOSE | Generic Object Oriented Substation Event |
| GPS | Global Positioning system |
| HMI | Human Machine Interface |
| HV | High Voltage |
| ICD | IED Capability Description |
| IEC | International Electrotechnical Commission |
| IED | Intelligent Electronic Device |
| IEEE | Institute of Electrical and Electronic Engineers |
| IP | Internet Protocol |
| LAN | Local area network |
| LN | Logical Node |
| LV | Low voltage |
| MAC. | Media Access Control |
| MU | Merging Unit |
| MV | Medium Voltage |
| NCIT | Non-Conventional Instrument Transformers |
| OSI | Open Systems Interconnection |
| PC | Personal Computer |
| PD | Physical Device |
| RMS | Root Mean Square |
| RSTP | Rapid spanning tree protocol |
| RTU | Remote terminal unit |
| SA | Substation Automation |

| | |
|---|---|
| SAS | Substation Automation System |
| SCADA | Supervisory Control and Data Acquisition |
| SCD | Substation Configuration Description |
| SCL | Substation Configuration Language |
| SCSM | Specific Communication Services Mappings |
| STP | Spanning Tree Protocol |
| SV | Sampled Values. |
| TCP | Transmission Control Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| UCA | Utility Communications Architecture |
| UDP | User Datagram Protocol |
| VT | Voltage Transformer |
| XML | Extensible Mark-up Language |

| **Terms** | **Definition** |
|---|---|
| Abstract Communication Service Interface (ACSI) | Virtual interface to an IED providing abstract communication services |
| Current Transformer | A current transformer (CT) is a device used for measurement of alternating electric currents. Current Transformer. It is also to reduce the current magnitudes, when current in a circuit is too high to apply directly to measuring instruments. |
| Data object | Parts of a logical node object representing specific information, for example, status or measurement. From an object-oriented point of view, a data object is an instance of a data object class. Data objects are normally used as transaction objects; i.e., they are data structures. |
| Device | Mechanism or piece of equipment designed to serve a purpose or perform a function |
| Intelligent Electronic Device (IED) | Any device incorporating one or more processors with the capability of receiving or sending data/control from or to an external source (for example, electronic multifunction meters, digital relays, controllers) |
| Interchangeability | Ability to replace a device supplied by one |

| | |
|---|---|
| | manufacturer with a device supplied by another manufacturer, without making changes to the other elements in the system |
| Interoperability | Ability of two or more IEDs from the same vendor, or from different vendors, to exchange information and use that information for correct execution of specified functions |
| Logical Node (LN) . | Smallest part of a function that exchanges data. A LN is an object defined by its data and methods |
| Merging Unit | Merging Unit is a device used to publish Sampled Values for currents and or voltages. |
| Voltage Transformer | A voltage transformer (VT) is a device used for measurement of alternating electric currents. Current Transformer. It is also to reduce the voltage magnitudes, when voltage at the measuring point is too high to apply directly to measuring instruments. |
| Non-Conventional Instrument Transformers | Instrument transformers that are not based on iron wound cores like conventional Current Transformers and Voltage Transformers |

# CHAPTER 1

## Introduction

The history of electricity dates back to the early 19th century, when an Italian scientist by the name of Alessandro Volta made a discovery by soaking paper in salt water, placing zinc and copper on opposite sides of the paper, and observing that the chemical reaction produces an electric current. In 1831 Michael Faraday discovered the "induction" or generation of electricity in a wire by moving a wire through a magnetic field (Online: http://inventors.about.com). The induction ring was the first electric transformer and both the electric generator and the electric motor are based on this principle (Online: www.fi.edu/learn/case-files/energy.html). The challenge was then to use these principles to develop an electrical system that would provide people with a practical source of energy. Thomas Edison took on this challenge. He designed and built the first electric power plant that was able to produce electricity, and distribute it to people's homes. The Pearl Street power station was put into service on the 4th September 1882 in New York City (Online: http://www.pearlstreet.inc.com) and is considered the world's first central electricity generating station.

Alternating current electrical power systems began developing in the early 20th century (Nordell, 2008) and are divided into three different entities, Generation, Transmission and Distribution as depicted in Figure 1.1.

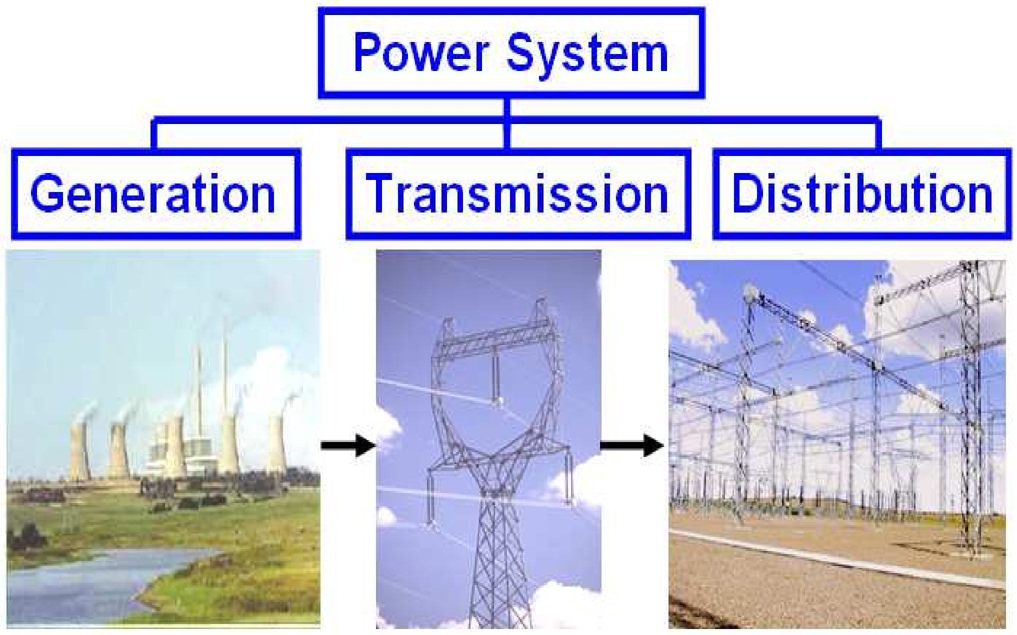


Figure 1.1: Power system

Electricity is generated at a generation plant and transmitted using high voltage power lines. An electrical substation is a subsidiary station of an electricity generation and transmission system where voltages are transformed from high to low (step down) or low to high (step up) using transformers. Substations use protective devices to isolate system failures so that these localised faults do not compromise the power system network integrity.  In substation power protection systems deployed in the early days there was no automation, and humans were often stationed at key points in the electrical system to monitor and respond to problems (Nordell, 2008).

One of the main drivers behind the recent developments in power system protection and control has been to try and minimize system faults, maintain continuity of supply, minimize human intervention and to implement automated system-solutions for the protection and control of power networks. A Substation Automation System (SAS) refers to the creation of a highly reliable substation environment that rapidly responds to real-time events with appropriate action, ensuring an uninterrupted power service.

## 1.1. Introduction to substation automation systems

Substation automation systems were introduced in the early 1990’s (Kirkman, 2007). During this period bay controllers and protection devices with inherent bay control functionality became available. These bay controllers could provide limited substation automation capabilities by connecting substation protection devices to the substation controller, typically a Remote Terminal Unit (RTU) using serial communication.

The early substation automation systems had master/slave architectures and typically utilised legacy communication protocols to transfer substation data to a remote location. These legacy protocols in general were simple protocols that were designed to be byte-efficient. During this period, communications protocols such as Modbus and Distribution Network Protocol (DNP) became industry standards.

These are tag-based protocols, where users accessed data by specifying a tag number or index number. The implementation of these protocols resulted in a simple reliable communication network architecture, however the effort required during system engineering made the use these protocols unnecessarily complex.

Substation automation and the integration of substation automation systems to enterprise level have continued to develop since the early 1990’s. Furthermore, technical advances in microprocessor technology and data networking have meant that Ethernet-based systems are preferred as opposed to serial communication such as RS232 and RS485, as the communication medium of choice. Ethernet bus-based substation automation systems can be divided into two categories, non-standardised and standardised substation automation systems.

### 1.1.1. Substation automation systems using non-standardized intelligent electronic devices

Since the late 1990's protection Intelligent Electronic Devices (IEDs) with inherent bay control functionality and Ethernet communication capabilities have become readily available. Intelligent Electronic Devices (IEDs) is a name given to protection and control devices. The term IED in substation automation system refers to microprocessor based protection, metering and monitoring devices (Kezunovic and Popovic, 2005). Prior to the introduction of IEDs the protection and control devices were known as electromechanical relays – as the name suggests, these earlier devices had no intelligence.

A large number of power utilities have deployed many of these protective devices (IEDs) in the field without using the communication capabilities of these devices due to the fact that these relays are non-standardized. The main drawback of using non-standardized IEDs is that each manufacturer's device is unique. As a result, utilities is either locked into using one manufacturer's product or need to have the resources to maintain devices from different vendors. Figure 1.2 depicts a solution where non-standardized Ethernet based IEDs are deployed.

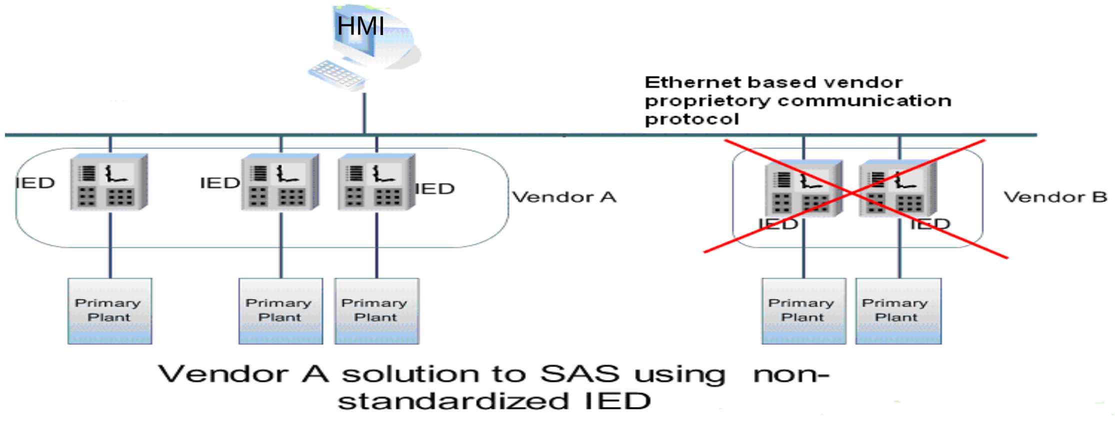

HMI
Ethernet based vendor proprietory communication protocol
IED
IED
IED
Vendor A
IED
IED
Vendor B
Primary Plant
Primary Plant
Primary Plant
Primary Plant
Primary Plant
Vendor A solution to SAS using non-standardized IED

Figure 1.2: Substation architecture using non-standardized IED

The benefits that accompany an Ethernet-based system includes the reduction of wiring time, the provisioning of network addressability for the IEDs, and a significant reduction in cabling cost. However, since proprietary data protocols are used in non-standardized systems, there is no interoperability, and substation integration between vendor A and vendor B IEDs is impossible without the use of very complex and expensive protocol convertors (see Figure 1.2 above). As a result, users of non-standardized IEDs are often bound to a single vendor.

Inadvertently the introduction of non-standardized IEDs resulted in several proprietary communication protocols being introduced into the market. The large number of competing proprietary solutions for communication in substation automation and SCADA systems started becoming too costly for industry. This resulted in a lack of interoperability, high recurring development costs and lack of support (long term stability) for a number of protocols. The need for standardization existed. Figure 1.3 below shows a standardized substation automation system.

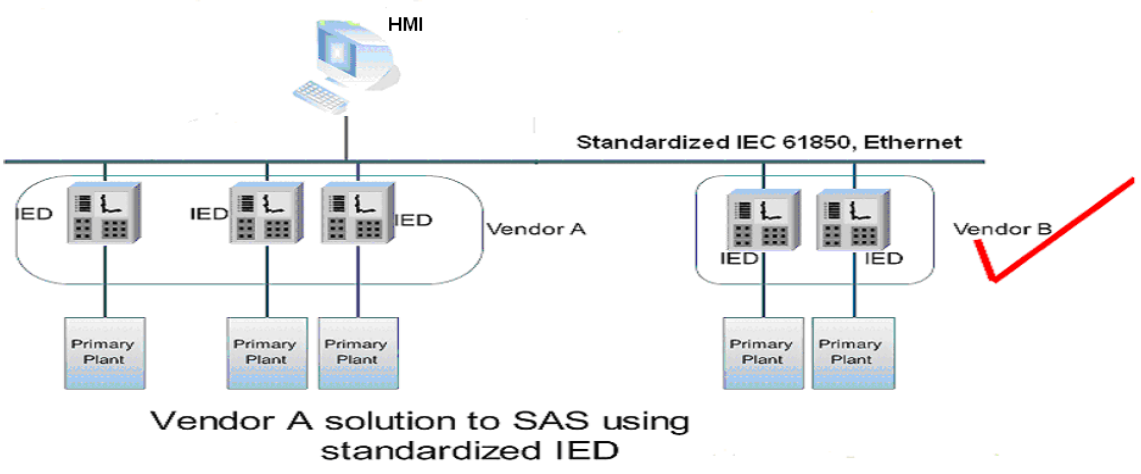


Figure 1.3: Standardized substation automation system

Figure 1.3 shows that the same advantages as in Figure 1.2 can be achieved with a standardized substation automation system, with the added advantage of interoperability and scalability (as is shown between vendor A and vendor B in Figure 1.3 above).

## 1.2. The IEC 61850 standard

The IEC 61850 standard is a current international standard for substation automation systems. The IEC 61850 standard defines a substation automation framework that is future proof, flexible, scalable, maintainable, and provides interoperability between substation automation devices from multiple vendors.

The IEC 61850 standard provides this framework by defining a) the data that has to be communicated between IEC 61850 compliant devices, b) by defining information exchange mechanisms, and c) by defining the format and content of files to enable system engineering for substation automation systems.

The standard recommends a hierarchical Ethernet-based substation automation architecture, with a station level, bay level and a process level, as depicted in Figure 1.4 below.

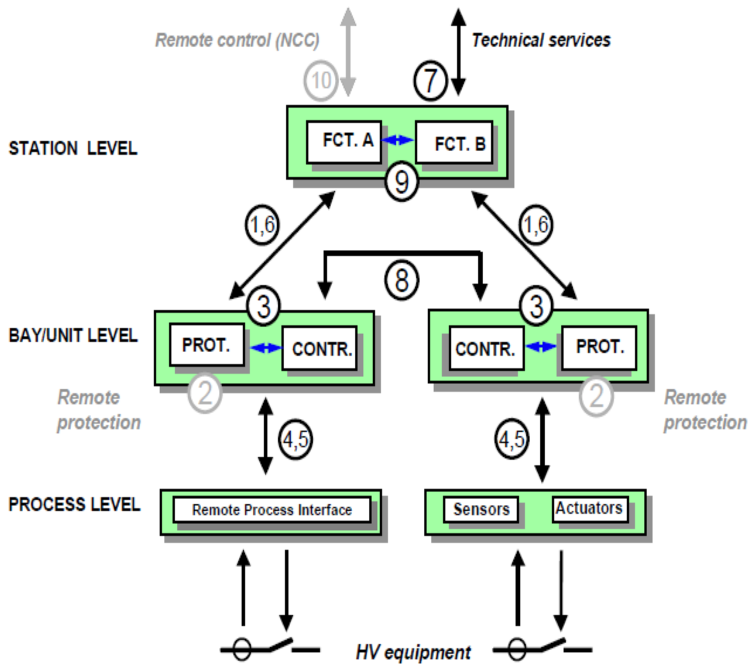


Figure 1.4: IEC 61850 substation architecture (IEC 61850, 2003)

At the process level, a data acquisition node that can be used for measurement distribution from primary plant equipment is defined. This data acquisition node is referred to as the "Merging Unit". The merging unit is discussed in detail in chapter 4 section 4.3.

## 1.3. IEC 61850-9-2 Process Level

The process level is the lowest level of the three tier architecture defined in the IEC 61850 standard (see Figure 1.4 above). The process level is used to distribute primary plant equipment status and analogue measurements to the bay level devices. The measurement distribution method utilised in the IEC 61850 standard is referred to as sampled values. Sampled values are digital measurements published from a Merging Unit connected to the instrument transformer as shown in Fig. 1.5.

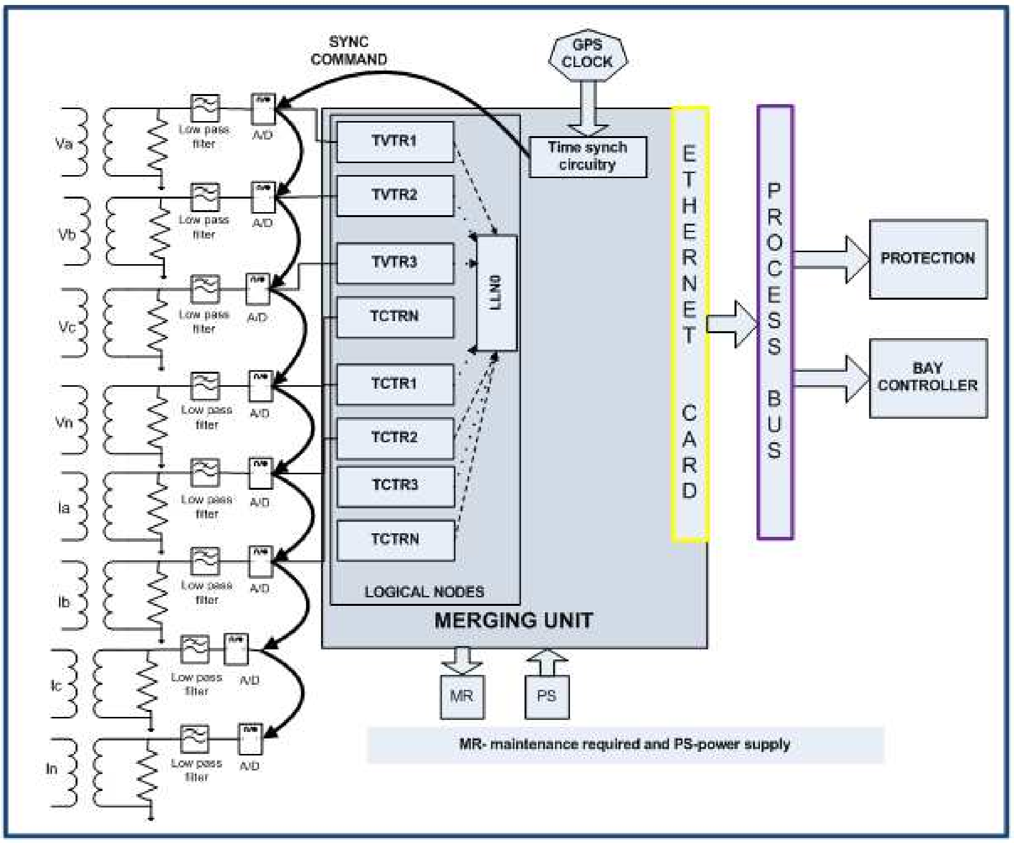


Figure 1.5: Simplified structure of merging unit (IEC 60044-8)

Figure 1.5 above shows the simplified structure of merging unit. A Merging unit receives a time-coherent set of samples of three-phase voltages, three-phase currents, and neutral voltages and currents from the instrument transformer. The data collected from the instrument transformer is digitized and stored within the device. The data is then periodically retrieved from the logical nodes; is encoded into a dataset; packaged according to the IEC 61850-9-2 message format and transmitted to the process-bus as sampled values. This will be explained in more detail in chapter 3 section 3.5.2.3.

The IEC 61850 standard has been widely accepted and is slowly being deployed by power system utilities across the globe. The technology roadmap for most power system utilities is converging towards the general use of the IEC 61850 standard. The implementation of the IEC 61850 process bus model is however still limited. There is a need to train and equip substation automation engineers and students in general with knowledge of the IEC 61850-9-2 process bus technology.

## 1.4. Motivation for research and problem definition

### 1.4.1. Motivation for research

Standards-based equipment compliant with the IEC 61850 standard has become important in modern substation automation systems in order to reduce labour costs, to ease system scalability, and reduce maintenance costs. This is due to benefits derived from a combination of reduced cabling, “Plug and Use” capability, ease of equipment addressability, interoperability of equipment and the availability of software environments for describing system architecture. Equipment from vendors to accommodate the “Process bus” paradigm is however limited at present.

The knowledge about the methods to model developed devices that can publish sampled value messages in a manner compliant with the IEC 61850-9-2 standard currently resides within the vendor environment.

This proposed project is motivated by the need to build the foundation for the creation of a virtual sensor node to contribute to, and assist with the understanding of the process-bus concepts and to develop a pool of expertise with knowledge of the IEC 61850 standard and the engineering tools required for configuration and simulation within an education environment. This proposed virtualised sensor node would represent a limited functional implementation of the merging unit that will represent a virtualised data acquisition node capable of synthetic generation of waveforms through synthesis data generation, encoding of the data and transmitting the data in a format compliant with the IEC 61850 protocol.

The role of this project in an academic environment therefore is to build the knowledge base required to assist with the understanding of the functioning and implementation of the virtual sensor node, and to lay the groundwork for a future real-time implementation.

### 1.4.2. Problem definition

The challenge with IEC 61850-9-2 sampled value message distribution is that it occurs at the Datalink layer. Messages that occur at the Datalink layer are much more difficult to program on PC platforms as opposed to the higher-level protocols in the Open System Interconnect (OSI) model, such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). For this project methods to generate the sampled value frame structure, encoding of the data and transmission at the Datalink layer in a format compliant with the IEC 61850-9-2 sampled value message structure, have to be investigated. The operating systems that allow for the generation, binding and transmission of frames at the Datalink layer also have to be investigated.

## 1.5. Research aim and objectives

### 1.5.1. Aim

The aim of this Postgraduate Project is to examine the standardization developments within the IEC 615850 substation automation environment, to study the substation automation framework presented in the IEC 61850 standard, and measurement distribution methods proposed. The knowledge gained would be used to implement certain aspects of a Merging Unit. This limited functional merging unit represents a virtualized data acquisition node capable of synthetic generation of waveforms, encoding of the data and transmitting the data in a format compliant with the IEC 61850-9-2 sampled value message structure.

### 1.5.2. Research objectives

An initial research objective would be to examine the chronological evolution of substation automation systems and to examine the IEC 61850 framework for substation automation systems with emphasis on the measurement distribution methods proposed.

Another objective would be - through the literature search and literature review to gain the fundamental understanding of IEC 61850 messaging mechanisms, and to be able to accurately provide a breakdown of the message structure or data packets that are used for communication between process level and bay level and to simulate these messages using a software program.

Specific objectives:

i. To identify current and emerging substation automation trends;
ii. To Investigation whether Microsoft Windows and Linux operating systems allow for the generation, binding and transmission of packets at the Datalink layer; and
iii. To thoroughly understand the sampled value messages utilised in the IEC 61850 and to be able to simulate sampled values on a personal computer (PC).

In order to achieve the stated objectives certain sub-objectives have been identified and these are listed below:

1. A literature review for the identification of current and emerging substation automation trends will be conducted.
2. A comparative analysis of the developments of measurement distribution methods utilised in substation automation systems within the literature will be performed.
3. An overview of the IEC 61850 standard with emphasis on sampled values and GOOSE messaging will be conducted.
4. The sampled value message structure breakdown will be performed and the other IEC 61850 related standards will be outlined.
5. The implementation of a low-level network programming Application Programming Interface (API) on both Microsoft Windows and Linux operating systems will be investigated.
6. The software development on Linux Ubuntu to simulate the functional behaviour of sampled value messages will be conducted
7. The development of a test scenario to validate the structure of the sampled value from the developed sensor node will be performed, and a comparative analysis with a commercial unit will be performed.

## 1.6. Project Assumptions and delimitation

### 1.6.1. Assumptions

The following assumption for this research has been established:

i. Modern computers have a fast enough processing speed and have operating systems that allow for Datalink message generation and transmission, enabling accurate emulation of a merging unit.
ii. The use of custom-built networking libraries will not degrade the performance of the generated program.

### 1.6.2. Data Acquisition

i. The data acquisition node will be explored on a PC platform only
ii. The verification of sampled values will be performed using software tools only.
iii. The research project will not be integrated into an IEC 61850 system; no substation configuration language (SCL) files will be produced for the proposed device (MU); the Publisher/subscriber mechanism will not be explored; sampled values to be verified using only software tools such as Wireshark.

### 1.6.3. Simulation

i. The proposed project will not be implemented on a protection network or interfaced to any current and voltage transformers.
ii. Current and voltage values will be generated internally, packaged and transmitted as multicast messages - no sensors will be integrated into the virtual device.
iii. Analogue values published by the sensor node will be monitored using Wireshark

### 1.6.4. Implementation

i. The software programs will be developed using the Python programming language.
ii. Time synchronization and time stamping of the sampled value frames will be implemented using the computer’s internal clock - no integration of a Global Positioning System (GPS) clock source.

### 1.6.5. Significance

**Substation protection simulation and validation:** the process bus technology introduced by the IEC 61850 standard has presented a paradigm shift in protection and control systems within the power system industry. As such test procedures, commissioning tools and commissioning processes utilised in traditional protection and control schemes within substations are being reviewed and altered accordingly. This project is aimed at building an academic framework for a future real-time implementation of a data acquisition node that can be used for IEC 61850-9-2 substation automation simulation and validation. These simulation tools can be used in the academic environment for safe testing of IEC 61850 protection schemes. The future real-time implementation data acquisition node envisioned be used to facilitate testing and would be capable of integrating to both Conventional Power Transformer (CPT) and non-conventional power transformers (NCPT). The Merging unit is fundamental to the integration of the NCPT to SAS.

**Financial:** The Merging unit implementation will result in significant reduction of the amount of wiring between the primary and secondary plant equipment in a substation environment; thus significantly reducing the commissioning costs and reducing the fault-finding and maintenance time. Less down-time equals more revenue for the power utilities. However before this stage is realized, a significant amount of revenue has to be invested in order to ensure adequate training for students and substation automation engineers.

## 1.7. Research methodology

### 1.7.1. Literature review

A literature search and review on the substation automation and measurement distribution framework provided by the IEC 60044-7, IEC 60044-8, IEC 1451 and IEC 61850 standards is conducted in chapter 2 of this document. The literature search and literature review is used to form the theoretical base, in defining software algorithms and for the development of software to simulate IEC 61850 sampled value messages. The literature search and review will also extend to low-level networking software development in both Microsoft Windows and Linux operating.

### 1.7.2. Method for simulation

- A Python program running on a personal computer is proposed for the simulation of the IEC 61850-9-2 sampled value messages.
- Wireshark is proposed for the capture and analysis of the data packets of the generated sampled value messages.

### 1.7.3. Experimental methods

- Practical experiments will be conducted with the Omicron CMC to develop an in-depth understanding of the IEC 61850 sampled value frame structure.
- A Virtual Local Area Network (VLAN)-enabled switch will be used during the verification of the developed sensor node
- The published sampled value messages from the developed sensor measurement will be captured using Wireshark and the structure will be comparatively analysed to the sampled value messages generated by the Omicron CMC.

## 1.8. Organisation of the Thesis

The thesis is divided into six chapters detailing the introduction and background into substation automation systems, problem definition, literature review into substation automation trends, an investigation into the IEC 61850 GOOSE and Sampled value messaging system, and the results of the software development of the simulated sampled value messages.

Chapter 1: presents an overview of the substation automation developments and highlights the problem definition, project aims and objectives, and research methodologies.

Chapter 2: presents the literature search and literature review into measurement distribution developments in the substation automation environment.

Chapter 3: discusses an overview of the IEC 61850 standard with emphasis on sampled value messaging and GOOSE messaging.

Chapter 4: presents the project theory and context detailing an investigation into the methods used to achieve low-level networking using the Python programming language on both Microsoft Windows and Linux Ubuntu operating systems.

Chapter 5: provides the results of the implemented virtual sensor node and describes the case study using an Omicron CMC. This forms the foundation for the software development which is presented later in the chapter. The software development techniques applied in this project to model a virtual device that is used to simulate IEC 61850 sampled value messages is discussed. The published sample value messages generated by the developed sensor node are compared with those published by the Omicron CMC.

Chapter 6:  presents the conclusion to the research project and provides expansion prospects such as real-time implementation for future research projects.

## 1.9. Publications

1. Luwaca. E., Petev P., Kriger C., and Behardien S., 2014. “Virtualization of a sensor node to enable the simulation of IEC 61850-based sampled value messages”. Submitted to the International Journal of Computers, Communications and Control (awaiting confirmation for publication).
2. Kriger C., Retonda J., Luwaca E., and Behardien S., 2011. “Analysis of GOOSE and Sampled Value Message Structure for Educational Purposes”, PAC World Conference, Dublin.

# CHAPTER 2

## Literature review

### 2.1. Introduction

In this chapter a literature search and literature review of standards and papers discussing measurement distribution in substation automation systems is conducted. As part of the literature search and literature review the chronological evolution of substation automation systems is presented. Measurement distribution mechanisms in substation automation systems are discussed.

The chapter is structured into five sections. Section 2.2 discusses the background associated with substation automation systems, followed by a literature search in Section 2.3. A literature review of the papers discovered through the literature search is conducted in Section 2.4. In Section 2.5 a comparative analysis of measurement distribution systems in the substation environment is performed and Section 2.6 concludes on what has been done in the chapter and what will be covered in the next chapter.

### 2.2. Background

A substation is divided into two sections, primary plant and secondary plant. As shown in Fig. 2.1, primary plant consists of voltage transformers (VT's), current transformers (CT's), circuit breakers (CB's) etc. and secondary plant consists of AC/DC panels, protection relays and remote terminal units (RTUs).

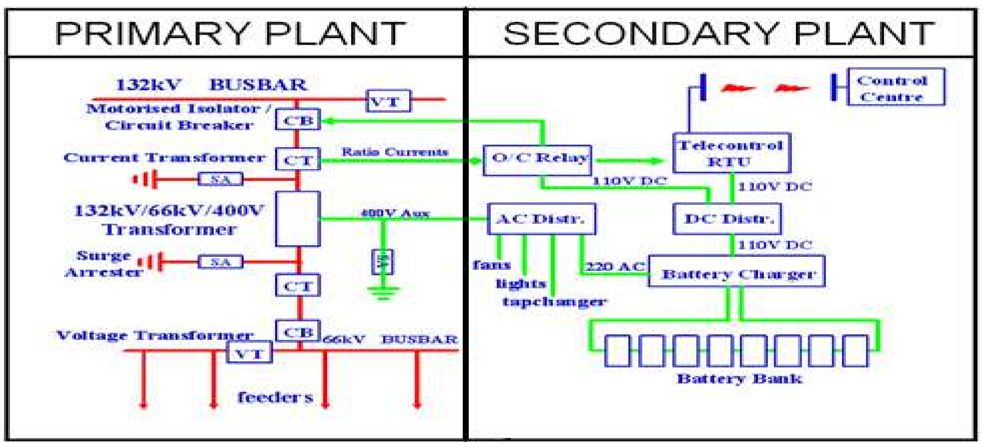


Figure 2.1: Single line diagram of a substation system

The primary plant equipment in Fig. 2.1, receives the incoming supply through the high voltage busbar, steps the supply down and distributes the supply to the feeders through the medium voltage busbar. The secondary plant equipment monitors and protects primary plant equipment.

Before the 1990’s the substation system shown in Fig. 2.1 was designed using protection and control schemes implemented with single-function electromechanical, relay logic, as depicted in the left-most panel of Fig. 2.2, labelled “Hardwired Substation” (Leupp and Rytoft, 2011) .

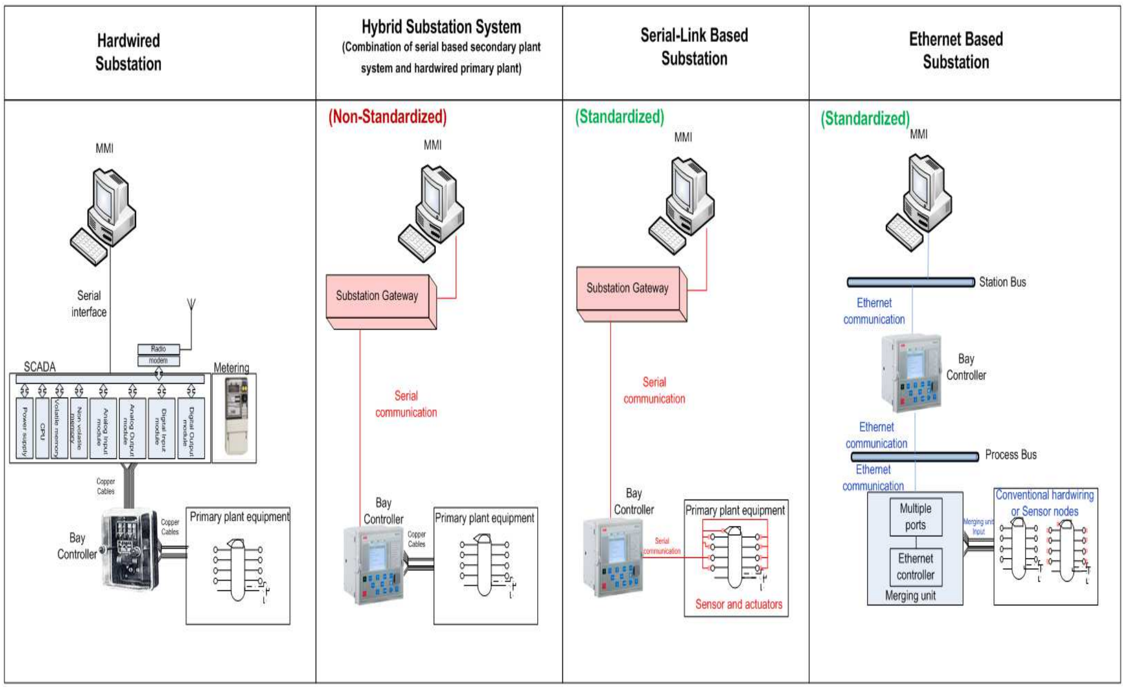


Figure 2.2: Migration from hardwired system towards substation automation

Hardwired substation layouts have been used in industry for decades. One of the reasons for this form of substation layout was the unavailability of intelligent bay controllers and smart sensor nodes before the 1990’s.  This form of substation layout suffered from various disadvantages, such as:

- No addressability:  the different feeders are hardwired, with labels used at either end of the cable being the only form of addressing
- There is no data communication between the various feeders.
- The bulk of the cost for setting up such a system is incurred on cable costs and labour.
- The system is not easily scalable
- No substation automation: the system is hardwired and there is no communication between devices at the same level in the hierarchy.

The equipment associated with monitoring, protection and control functions has in the recent decade's undergone evolution from electromechanical devices to microprocessor-based devices, leading to increased levels of substation automation (Leupp and Rytoft, 2011). During the 1990's intelligent electronic devices with inherent bay control functionality became available (Apostolov, 2006). The main attraction of these bay controllers was their ability to communicate information to the user, using point-to-point protocols. RS232 or RS485 provided a means to communicate to these intelligent electronic devices. In terms of chronological evolution, the column labelled "Hybrid Substation System" in Fig.2.2 above shows a typical example of this layout (Leupp and Rytoft, 2011).

The main drawback with Hybrid Substation layouts as shown in Fig.2.2 is that these systems convey information at slow data rates and cannot be used for protection functions. In most cases the protocols used in this kind of system layout are non-standardized. Proprietary communications protocols and mainly protocols that are limited to a master/slave topology are used for this kind of layout. These protocols are tag-based protocols, where users access data by specifying a tag number or index number. The advantages and disadvantages of Hybrid Substation automation systems are listed below (Flynn, 2007).

**Advantages:**

- Reduced cabling
- Addressability

**Disadvantages:**

- Non-standardized communication protocols
- Primary plant and secondary plant hardwired, no addressing in that regard
- Speed limitation, the maximum speed on an RS232 link is ±14.5kbps
- Costs, RS485/RS422 to RS232 converters are often required. Lots of money still spent on cables between primary and secondary plant.
- Restricted in terms of scalability
- Only processed measurement values are available
- Only physical layer standard: no standardized data communication

Substation automation systems have continued to grow since the 1990s with further developments in microprocessor-based relays and developments in primary plant equipment. Transformers with optical interfaces referred to as Non-Conventional Instrument Transformers (NCIT) were developed in the 1990s. NCITs systems for high voltage applications provide a series of advantages not obtainable with conventional transformers (Christen et al., 1995). These advantages are (Rahmatian et al., 2002):

- Safety - optics provides separation between the measuring point at high voltage (HV) potential and the measuring equipment at ground potential.
- Compactness, reduce the weight of the transformers.
- Better performance than conventional transformers due to the reduced noise induction and Ferro-resonance.
- Better interface to electronic equipment such as personal computers

With the development of NCITs, the need for a standardized interface to NCITs was soon identified. In response to this, the Instrument Transformers Standard IEC 60044 was published in December 1999. Part 7: Electronic voltage transformers (IEC 60044-7) and Part 8: Electronic current transformers (IEC 60044-8) define interfaces to voltage and current instrument transformers respectively.

Part 8 of the IEC 60044 standard goes on to further specify among other things, (Brunner, 2011):

- The digital output for both electronic current and voltage transformers
- Physical, Data link and Presentation layers of the IEC 60044-8 protocol
- Defines a node for smart sensors to interface to instrument transformers.

The smart sensor interface node introduced through IEC 60044 is referred to as a merging unit (Brunner, 2011). A merging unit provides the interface from the electronic current and voltage transformers to the secondary equipment. A typical method of communicating measurements using a merging unit, as defined in IEC 60044, is shown in Fig. 2.3 (IEC 60044-8, 2002).

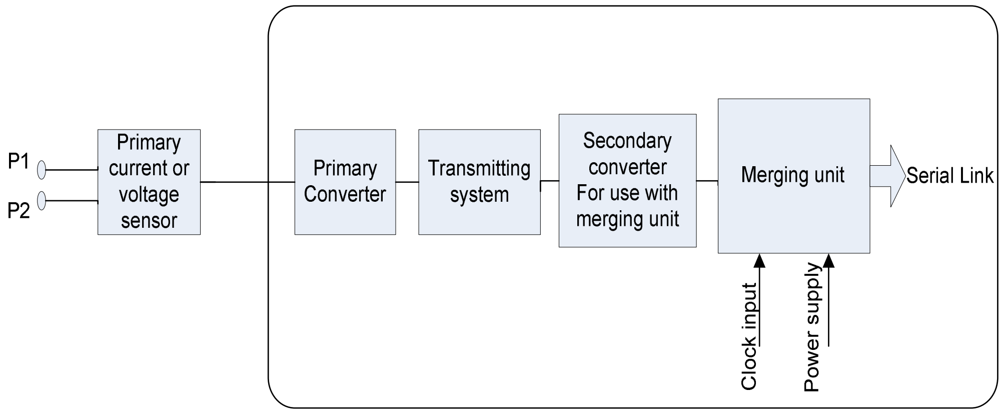


Figure 2.3: A simplified block diagram of measurements distribution using IEC 60044-8 standard (IEC 60044-8, 2002)

Current or voltage sensors are connected to the NCIT. The primary measurements from the sensors are then converted into secondary measurements using a transducer of sorts. The secondary measurements are then used as inputs to the merging unit, which then publishes the raw measurement values over a serial link. The introduction and use of the IEC 60044 standard, lead primary plant manufacturers to start to incorporate some sort of sensor into their product to maintain the competitive edge. This allowed other standards such as the Standard for a Smart Transducer Interface for Sensors and Actuators, i.e. IEEE 1451, first published in 1997, to get into the power system environment. IEEE 1451 attempted to alleviate the problems of analogue measurement distribution in power systems. A Typical system using the IEEE 1451 standard to publish measurements onto a serial link is depicted in Figure 2.4. (Zheng et al., 2006)

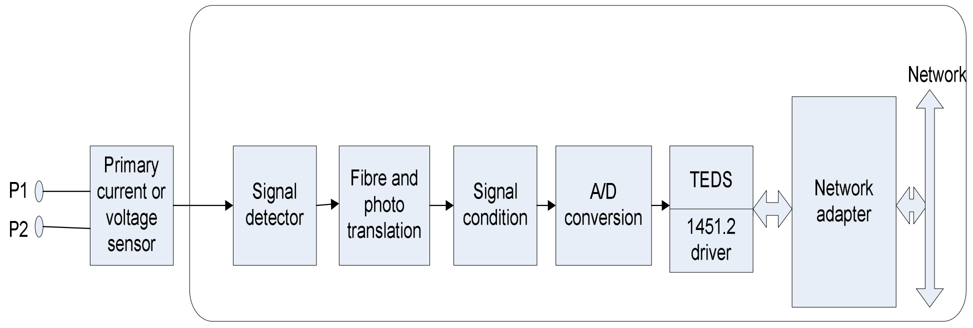


Figure 2.4: Measurements communication system using IEC IEEE 1451 standard (Zheng et al., 2006)

At first glance Fig 2.3 and Fig 2.4 look very similar, with the only major difference being the inclusion of Transducer Electronic Datasheets (TEDs) in the published message in IEEE 1451. Basically the TEDs allow a data system to choose the correct template for the interpretation of the stored data. By including calibration information in the TEDs, IEEE 1451 allows intelligent devices to adjust for variations in transducer sensitivity (Mark and Hufnagel, 2004). This was a major breakthrough in substation measurement systems as potential measurement errors could now be compensated for in the IEDs. The network interface specified in the original IEEE 1451 standard, however was still a serial link. Some of the advantages and disadvantages of serial-linked substation systems as shown in Fig. 2.2 in the panel labelled “Serial-link Based Substation” are stated below.

**Advantages: serial-linked substation system:**

- Reduced cabling
- Addressability
- Raw measurements and not processed measurements are published from a sensor node.

- Calibration information is communicated using IEEE 1451 TEDS
- Provides an interface to NCITs.

**Disadvantages: serial-linked substation system:**

- Speed limitation: the maximum speed on an RS232 link is ±14.5kbps
- Costs: RS485/RS422 to RS232 converters are often required.
- Restricted in terms of scalability
- Only physical layer standard specified
- Data communication is not standardized
- Only Unicast (Point to point) measurement communication
- No interface to conventional transformers defined.

Due to the disadvantages of serial based substation systems, the goal to define a communication infrastructure that would allow seamless integration of substation automation components continued into the middle and late 1990's. The focus had now shifted from serial-based substations to Ethernet-based substation systems - culminating in the publication of the Communication Networks and Systems in Substations standard in 2003, i.e. the IEC 61850 standard. The objective of the standard is to specify requirements and to provide a framework to achieve interoperability between the IEDs supplied from different vendors (IEC 61850-1, 2003). To achieve interoperability, the IEC 61850 specifies logical interfaces between the various components of a substation automation system (Apostolov, 2005) as shown in Fig.2.5.

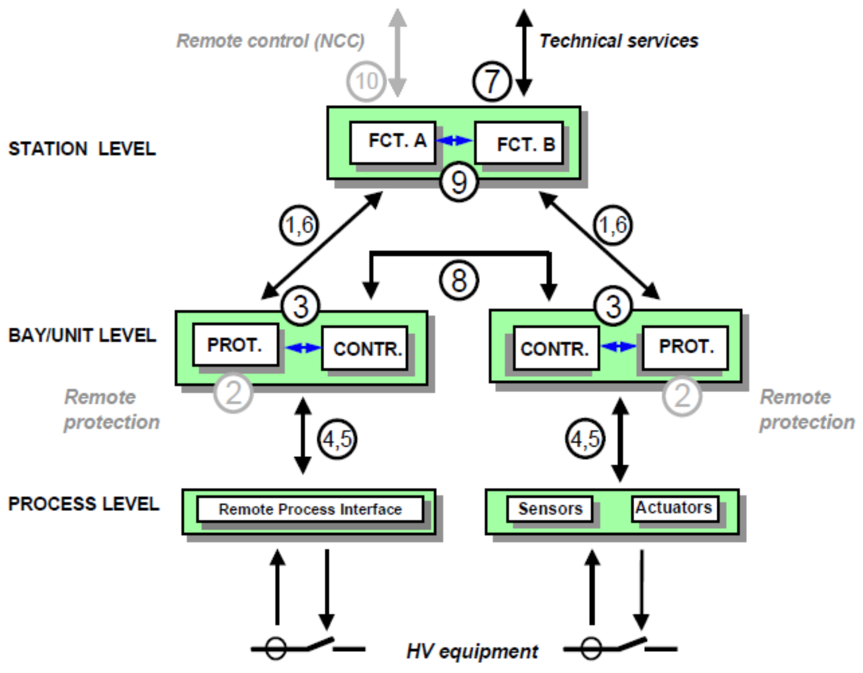


Figure 2.5: Interface model of a substation automation system specified within part 1 of the IEC 61850 standard (IEC 61850, 2003)

The description of the interfaces is outlined below:

- IF1: protection-data exchange between bay and station level.
- IF2: protection-data exchange between bay level and remote protection
- IF3: data exchange within bay level.
- IF4: CT and VT instantaneous data exchange (especially samples) between process and bay level.
- IF5: control-data exchange between process and bay level.
- IF6: control-data exchange between bay and station level.
- IF7: data exchange between substation (level) and a remote engineer’s workplace.
- IF8: direct data exchange between the bays especially for fast functions such as interlocking.
- IF9: data exchange within station level.
- IF10: control-data exchange between substation (devices) and a remote control centre

Fig. 2.5 above shows a hierarchical architecture proposed by the IEC 61850 standard, and the interfaces between the various components of a substation automation system. In IEC 61850 the functions of a substation automation system can be distributed across three levels. These levels are:

- The station level,
- The bay level, and
- The process level.

The substation automation components at the same level or at different levels of the hierarchy are then linked together by the various interfaces. It should be noted that the interfaces 2 and 10 (greyed out) are mentioned and shown in Fig.2.5 but do not form part of version 1 of the IEC 61850 standard.

The station level often referred to as “station-bus” connects all of the IEDs to one another and to the human machine interface (HMI) using interfaces 1,3,6,7 and 8. The IEDs communicate to the Station level devices using the messaging services provided by the Manufacture Message Specification (MMS) protocol in a client/server architecture. Inter-IED communication on interface 3 is achieved by using the publisher/subscriber communication services. Interfaces 4 and 5 connect Process level or “process bus “equipment to the Bay level and vice versa. The process level conveys raw power system information from switchyard source devices to the Bay Level through publisher/subscriber communication services.

Subsequent to the publication of the IEC 61850 standard, equipment compliant with the standard has become increasingly important in substation automation systems. This is due to benefits derived from a combination of reduced cabling, “Plug and Use” capability, ease of equipment addressability, interoperability of equipment and the availability of software environments for describing system architecture.

Intelligent Electronic Devices (IED’s) compliant with the standard have been developed, are widely available, and are slowly being deployed in industry but these IED’s are largely for station bus use only and are hardwired to primary plant. Equipment from vendors to accommodate the “Process bus” paradigm is however almost non-existent at present. Fig. 2.6a shows the current implementation and Fig. 2.6b shows the future implementation of the IEC 61850 standard.

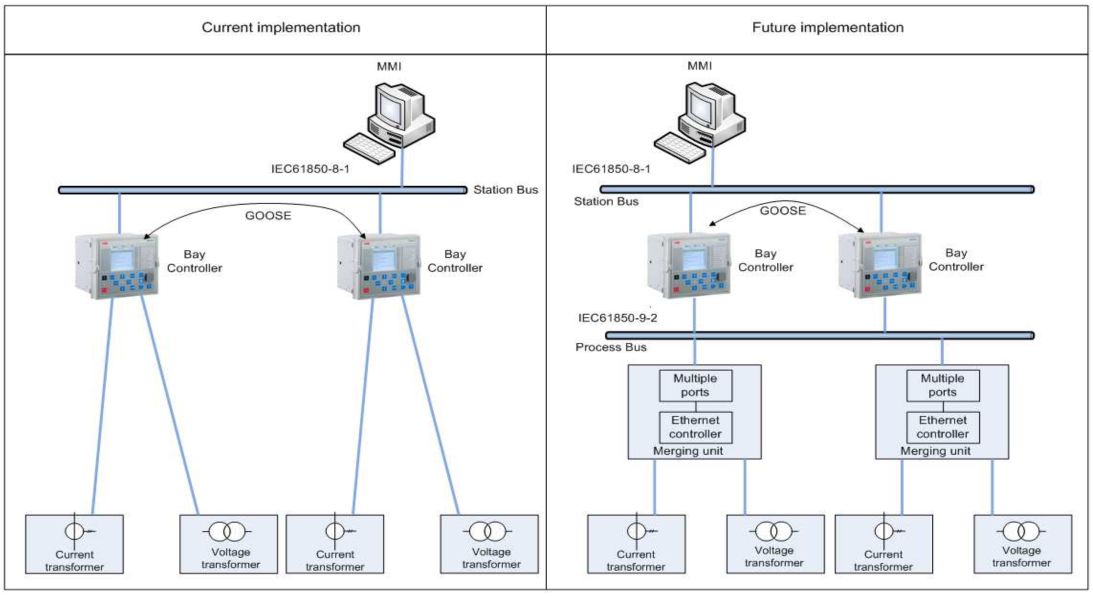


Figure 2.6a: Current implementation  Figure 2.6b: Future Implementation

In Fig.2.6a the current transformers (CTs) and voltage transformers (VTs) are hardwired to the bay controller. The bay controller has to process the measurements and generate Generic Object Orientated Substation Event (GOOSE) messages based on some pre-set values.

IEC 61850 however, proposes a system not only based on GOOSE, but on raw sampled measurement values published from a merging unit. Part 9 of the IEC 61860 standard specifies an interface between the merging unit, process and bay level equipment using the digital output of the merging unit for distribution to protection or metering equipment. Fig. 2.7 shows an example of measurements being published from a merging unit (IEC 61850-9-1, 2004).

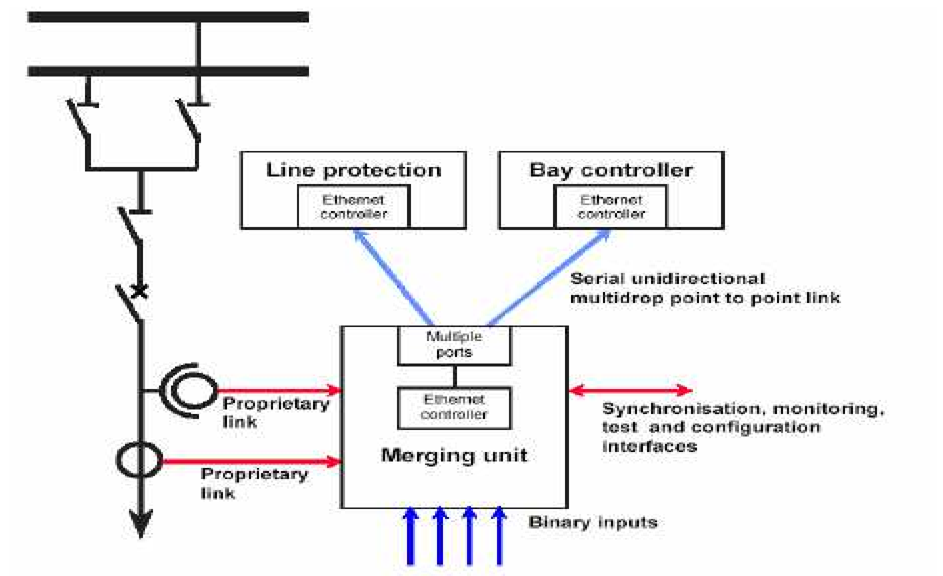


Figure 2.7: Example of measurements being published from a merging unit (IEC 61850-9-1)

In Fig.2.7, the merging units receive analogue measurements from the plant, digitize the analogue values, encode, and publish the raw data in a sampled value format onto the process-bus. Listed below are some of the advantages that can be derived with the implementation of the process bus. These are:

- Raw, unprocessed measurement values
- Support for both Unicast and Multicast published measurements
- Substation system not only based on substation events, but on actual raw measurements
- Standardized interface to non-conventional instrument transformers
- Improving the system performance in a conventional transformer system. The A/D conversion is done close to the source and much of the burden of processing can be removed from the bay controllers
- Reduced wiring

Based on the advantages and disadvantages itemised for the various standards, IEC 61850 provides greater advantages than IEC 60044 and IEEE 1451 for measurement distribution in substation automation systems. Compared to IEC 60044 and IEEE 1451, IEC 61850 is not restricted only to the interface between NCITs and bay level devices. IEC 61850-9 also provides the means to interface conventional transformers to bay level devices. The original IEC 61850-9-2 standard has, however, proved not to be detailed enough to allow implementation and to achieve interoperability of merging units. The IEC 61850-9-2 standard leaves the implementation of certain parameters as configurable parameters. As an example, the implementation of data sets and the sampling rate are left as configurable parameters in the IEC 61850-9-2 standard. To reduce problems due to differing implementations by manufacturers, the UCA User Group published the “Implementation Guideline for Digital Interface to Instrument Transformers” often referred to as IEC 61850-9-2LE. IEC 61850-9-2LE has recommended values for sampling rate for protection and metering applications, and proposes a fixed dataset consisting of three phase current and voltage measurements and neutral current and voltage measurements.

The aim of this project is to gain a fundamental understanding of the IEC 61850 standard, and through this foundation to implement certain aspects of a sensor node as shown in Fig.2.8.

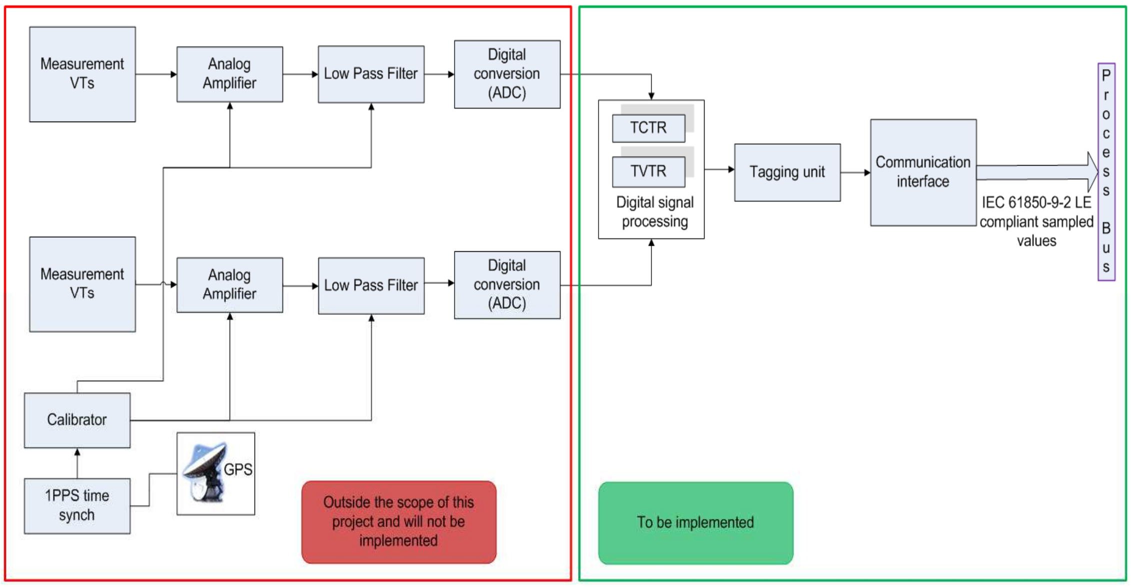


Figure 2.8: Basic architecture of a Merging Unit (Tholomier and Chatrefou, 2008)

This limited functional implementation of the sensor node would represent a virtualized sensor node capable of synthetic generation of waveform data, encoding of the data, and transmitting the data in a format compliant with the IEC 61850 protocol.

For the development of this sensor node, a thorough knowledge and understanding of IEC 61850 client/server, peer-to-peer communication, device model, process-bus and sampled value concepts are fundamental. The merging unit device model sampled value message structure and sampled value control blocks are some of the key requirements to the development of the sensor node.

## 2.3. Literature search

In order to gain this knowledge a literature search on the trends and methodologies of measurement distribution systems within the substation environment had to be conducted and it has been established that the following are some of key papers discussing measurement distribution in power systems under the following standards:

IEC 60044

- (Sawa et al., 1990)
- (Kezunovic et al., 1990)
- (Anderson and Brand, 2000)

IEEE 1541

- (Lee and Schneeman, 2000)
- (Wiczer, 2001)
- (Lee, 2003)
- (Elmenreich et al., 2005)

IEC 61850

- (Apostolov, 2006)
- (Apostolov, 2010)
- (Pan et al., 2012)

From the papers listed above and references from these papers, key-phrases that could be utilized in the literature search were identified as:

- "Power system", "substation automation", "current transformer", "voltage transformer", "non-conventional current transformer", "non-conventional voltage transformer", "IEEE 60044-7", "IEC 60044-8", "IEEE 1451 smart sensors", "IEC 61850", "IEC 61850 process bus", "IEEE 1588", "Merging unit".

The chronological spread of papers discovered using these key-phrases during the literature search is shown in Fig. 2.9, and a literature review of these papers is conducted in section 2.4. The main sources for these papers were the Institute of Electrical and Electronics Engineers (IEEE), Science Direct, Academia, and CiteSeerx websites.

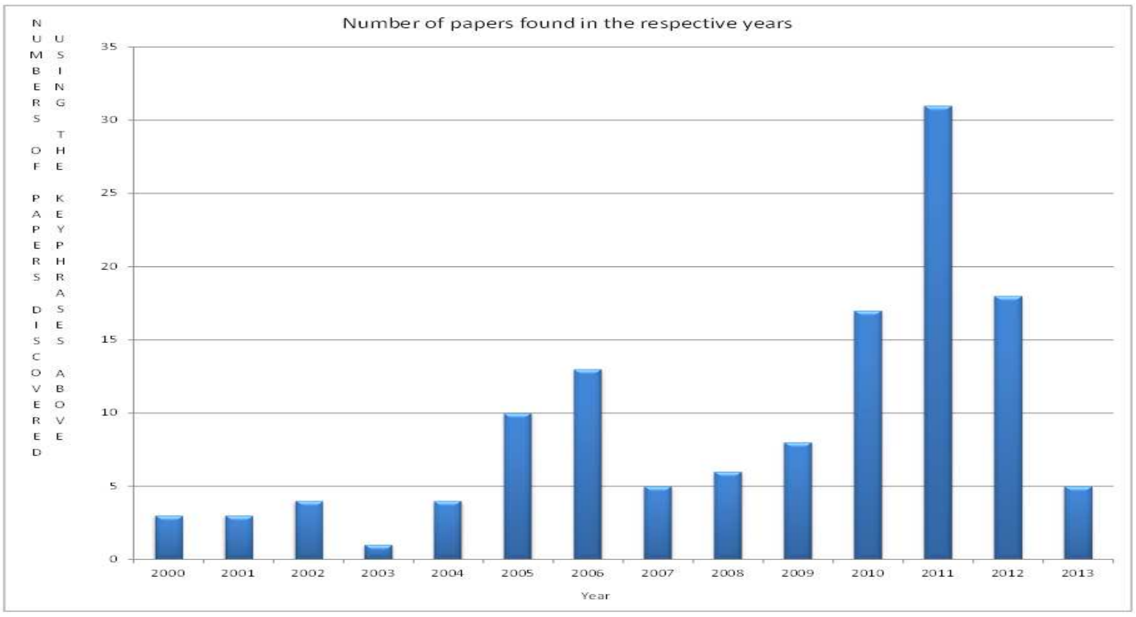


Figure 2.9: Papers per year using key-phrase searches

The literature search reveals that most of the research focus between the years 2000 and 2004 was on NCITs, and interfacing NCITs to the merging unit device defined in the IEC 60044-7/8 standard. Between the years 2004 and 2006, the research focus area had shifted slightly onto the integration of the IEEE 1451 standard to power systems. During the same period, the IEC 61850 standard was introduced and has since become the international standard for substation automation and control. Papers between the years 2005 and 2010 focused mainly on substation automation systems using the IEC 61850 standard. In recent years between 2011 and 2013 most of the research has been focused on time synchronization techniques for IEC 61850 substation automation systems. In Section 2.4 a sample of the papers found through the literature search are catalogued, commented upon and the criteria for comparative analysis decided upon.

## 2.4. Literature review

| IEC 60044-7 and IEC 60044-8 NCIT standards | | | | | |
|---|---|---|---|---|---|
| Paper | Aim of paper | Development Platform | Advantages and disadvantages outlined | Summary | Results |
| (Sawa et al., 1990) | Discusses the design and assembly procedure of optical current transformer (CT) and voltage dividing-type voltage (PD) transformer. | N/A | Advantages: Compact, high performance and reliable optical CT and PD transformer | Designing and manufacturing of the optical CT and PD have been described together with test results. New techniques for current and voltage measurement using optical sensors have been discussed | Test results for the developed optical CT and PD showed that their basic properties conform with Japanese standard for electric-power instrument transformers |
| (Christensen et al., 1995), | Describes a voltage transformer based on the Pockels effect | N/A | Advantages: Separation between the measuring point and the high voltage potential.<br><br>Compact transformer.<br><br>Better interface to electronic equipment such as personal computers (PCs) | An optical voltage transformer based on the Pockels effect is described. | Theoretical description of an optical voltage transformer design. |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| (Rahmatian et al., 2002) | Discusses the design and testing of optical voltage transducers for metering and protection applications in high-voltage systems | Hardware using an optical voltage transformer and sensors | Advantages: Optical voltage transducer offer advantages in terms of size, weight, and bandwidth compared to conventional transducers | A novel high voltage system using optical voltage transducers suitable for both metering and protection is described. | Optical voltage transducer that measure voltage by using three optical electric field sensors, passed the accuracy and insulation testing according to IEC |
| Kezunovic et al., 2006) | Looks to evaluate and quantify the influence of optical current transducers characteristics on protective relay performance | PC platform using PSPICE and Alternative Transients Program (ATP) simulation | Advantages: From the evaluation results it is expected that optical transducers will positively influence the performance of protection and control IED. | Using of simulation combined with statistical analysis, it was shown in this paper that the influence of NCIT on powers systems can be quantified and the results show that the use optical transducers is expected to positively influence the performance of protection and control IED. | Educational |
| (Pan et al., 2012) | The objective is to develop a testing prototype to measure the ratio error and phase error of the transformer under test | PC platform using LabVIEW software | Advantages: Accurate measurement of the ratio error and phase error on transformers | A device based on the digital signal processing technology, that can test both analogue and digital outputs of NCIT is proposed, for the implementation of NCIT | A device suitable for the testing transformers with precision classes up to 0.2 at the frequency of 50 Hz or 60 Hz, is introduced |

| | | | | | |
|---|---|---|---|---|---|
| (Bertolotto et al., 2007) | Discusses the evolution transformer analogue measurement systems by deploying data acquisition nodes using smart transducers to interface to optical voltage | N/A | Advantages: significant improvement of transformers reliability and maintenance techniques | The mix of well-known and reliable technologies, like HV capacitor dividers and Rogowski coils, together with high performance electronic devices and digital signal processing allow for the design of a measurement transformer compliant with IEC 60044-8 | development of an electronic measurement transformer for high voltage system |
| **IEEE 1451 sensor node** | | | | | |
| Paper | Aim of paper | Development Platform | Advantages and disadvantages outlined | Summary | Results |
| (Elmenreich et al., 2005) | Describes the main concepts of smart transducer interfaces | Embedded platform using microcontroller and sensor technology | Advantages: Reduced system complexity using smart transducer interfaces | Using two case studies, proof of concept for the smart transducer interface standard was achieved | Reduced system complexity using smart transducer interfaces |
| (Lee and Schneeman, 2000) | Describes the framework for standardized distributed measurement and control system. | System demonstration performed using simulation network program | Advantages: Distributed intelligence. Raw measurement values are broadcasted from the process level to the various network devices, allowing | Using existing object oriented programming systems and networking standards, a multi-level measurement and control system has | A framework for distributed application using open network communication and standardized transducer interfaces is developed |

| | | | | | |
|---|---|---|---|---|---|
| | | (SIMNET) | intelligent devices in the network nodes to make decisions based on raw, unprocessed measurement data | been developed. | |
| (Wiczer, 2001) | Analyses the strengths and weaknesses of smart sensor technology and strategies for successful implementation | N/A | Disadvantages: Implementing smart sensors can be very costly. A feasibility study has to be conducted for each project. | Two scenarios one of a real-world example of smart sensor technologies that suffered from significant economic and technical feasibility challenges and another example of a smart interface applied to an unenhanced sensor to create a functioning smart sensor unit are discussed in this paper | This paper provides a good base for designers and system developers to assess the feasibility of employing smart sensor technology in their system. |
| (Mark, 2004) | Looks into the IEEE 1451 modelling templates, as a set of properties or values that describe key characteristics of the transducer | N/A | Advantages:<br>P1451.2 TEDs provide a self-description mechanism for smart transducers | A roadmap of transducer standards from inception till it was accepted as a full-use standard by the IEEE Standards Association and the modelling approach is discussed in this paper | This paper is mainly Educational, focusing on the modelling of smart transducer nodes and smart nodes network interfaces |
| (Lee, 2004) | Describes the development | Software using UML | Advantages:<br>A fully executable C++ | A UML model that includes the data model | UML model for the IEEE 1451.1 |

| | | | | | |
|---|---|---|---|---|---|
| | of an UML model for the IEEE 1451.1 specification | tool in Visual C++ | code can be generated using UML tools from the model developed. | and object model of the IEEE 1451.1 standard has been developed. The model developed captures the attributes, operations, and behavioural information of the IEEE 1451.1 objects. | Standard has been developed |
| (Kim et al., 2006) | Presents definitions of the basic TEDs of electronic tongue system and propose a new template TEDs potentiometric devices | Hardware platform using an electronic tongue as the sensor | Advantages:<br>With IEEE 1451 implementation sensors and actuators become smaller, intelligent and provide communication capabilities | The capabilities of a sensor are beyond that of traditional sensors. These capabilities are widening the use of sensors to intelligent in industries. | Successful organisation of basic TEDs version letters and version numbers to indicate the type of sensor |
| (Pal and Rakshit, 2004) | Presents schemes for the development of network capable smart transducers | System proposed is an embedded system using AT89S8251 microcontroll er | Advantages:<br>cost effective and flexible in the sense that the user need not keep separate transducer for each sensor or actuator | A smart transducer with network interface that can accommodate Profibus, Fieldbus and Modbus as well as provided the program loading can be developed in an embedded system such as the AT89S8252 | Theoretical paper |
| (Zheng et al., 2006) | Look into the development of an optical sensor to measure HV | A hardware circuit using optical sensor | Advantages:<br>A safe, accurate and efficient method measurement of | An optical fiber measuring technique for transformer winding temperature is | An IEEE 1451.2 compliant temperature sensor of power transformer winding temperature was developed |

| | | | | | |
|---|---|---|---|---|---|
| | transformer winding temperature based on IEEE1452.2 | analogue to digital converters and 89S52R MCU | transformer winding temperature. | developed using smart IEE1451 transducer, with each transducer using a its own port. | |
| | | | **IEC 61850** | | |
| Paper | Aim of paper | Development Platform | Advantages and disadvantages outlined | Summary | Results |
| (Apostolov, 2006) | To describes the different communications related implementation of the new IEC 61850 standard. | N/A | Advantages of IEC 61850:<br>Introduction of Ethernet applications based on Client/Server and high-speed Peer-to-Peer communications in a substation environment | Different applications imposing different requirements that can be met by proper selection of the communications type in IEC 61850. The communications in the substation automation system can be between devices at the same level of the functional hierarchy or between bay and process levels of the system. | Educational. To raise awareness of the benefits of the IEC 61850 |
| (Anderson and Brand, 2000) | discusses the motivation behind the development of the IEC 61850 standard and the benefits | N/A | Advantages:<br>IEC 61850 is standardized and future proof. | IEC 61850 provides a future proof architecture that can be easily integrated to enterprise systems | Educational. A raised awareness of the IEC 61850 architecture, IEC 61850 system configuration and design considerations when dealing with substation automation |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  |  |  |  | system |
| (Baigent et al., 2004) | To provide an overview of an overview of the IEC 61850 protocol and IEC 61850 device model | N/A | IEC 61850 can be easily interfaced with existing legacy protocols OPC and Relational RDBMS | The paper teaches about the standardization data names, creation of services and the implementation of IEC 61850 | Educational |
| (Janssen and Apostolov, 2008) | Describes the different components of IEC 61850 substation systems and the distribution of functions between the devices | N/A | Advantages: Introduction of distributed functions in substation systems and reduced wiring | IEC 61850 allows the development of new approaches to substation design and maintenance processes. | The paper is mainly educational focusing on new design approach based on distributed functions |
| **IEC 61850 process bus** |  |  |  |  |  |
| Paper | Aim of paper | Development Platform | Advantages and disadvantages outlined | Summary | Results |
| (Apostolov, 2010) | Discusses the implementation agreement that ensures the interoperability between merging units and protection devices | N/A | Advantages: Cost reduction and interoperability | The traditional large number of copper cables connecting primary and secondary plant is replaced by just a few fiber optic cables. | Educational |
| (Sonawane et al., 2007) | Looks at the latency and deterministic | Real time digital | Advantages: The RTDS platform is | The goal of IEC 61850 is to provide a | An IEC 61850 messaging capable system was |

| | | | | | |
|---|---|---|---|---|---|
| | performance appropriate for an IEC 61850 real-time simulation. | simulator (RTDS) | capable of providing binary input and output as well as sampled value. | comprehensive set of functions to replace proprietary protocols. This allows the development of simulation test product using only the one protocol. | successfully developed for the RTDS Simulator. |
| (Skendzic et al., 2007), | Explores the Merging Unit Concept and analyses the reliability of protection systems based on sampled values (SVs). | N/A | Network Time Protocol (NTP) is recommended in IEC 61850 as the primary time however NTP cannot achieve the µs level time synchronization required for SV. However the standard allows time to be distributed by independent means (IREGB) | The Process Bus technology described in IEC 61850-9-2 offers a variety of new possibilities in designing Ethernet-based protection and control systems. | An important property of the SV-based process bus that is evident is from the investigation conducted is high bandwidth requirement and high processing speed required from the process bus and merging unit respectively |
| (Konka et al., 2012) | Emulation of IEC 61850-9-2 standard. Outlining the testing procedure for merging units | PC platform using NS3 and Wireshark | Advantages: Cheap testing tool for emulation of sampled value messages. | In this paper, an accurate and realistic model of SV traffic generator was described. Packet traces generated using the model was shown to be realistic as they included the full byte structure of Sampled Values APDU and ASDU. | A SV traffic generator has been developed.<br>The generated sampled values have been verified using Wireshark |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| (Yang et al., 2012) | Design and simulation of the communication network based IEC 61850 | PC platform using NS2 and Wireshark | Cost effective tool for IEC 61850 network simulation | A topology that satisfies the timing requirement of IEC 61850 proposed | Design and simulate the station bus and process bus was achieved. |
| **Substation automation time synchronization** |  |  |  |  |  |
| Paper | Aim of paper | Development Platform | Advantages and disadvantages outlined | Summary | Results |
| (Ingram et al., 2012), | Performance evaluation of commercially available PTP devices to meet the functional requirements of a SAS. PTP Time synch methods required and the effect of SV traffic on PTP performance. | RTDS, PTP clocks, Endace DAG7.5G4 Ethernet capture card, Tektronix DPO2014 digital oscilloscope | Advantages:<br>PTP devices intended for power system use are interoperable, as each grandmaster and slave clock combination successfully synchronized | Tests conducted with commercially available PTPv2 clocks prove PTPv2<br>to be a viable method of providing time synch for a sampled value process bus using IEC 61850-9-2 | The jitter between the grandmaster and slave clock is deterministically measured in the nano second range. This proves PTP time distribution methods are suitable for use in synchronizing sampled values |
| (Ingram et al., 2011) | To examine PTPv2 grandmaster holdover and recovery from loss of GPS synch. | Hardware implementation using: PTP clocks, Endace DAG7.5G4 Ethernet capture card, Tektronix DPO2014 digital oscilloscope | Compensation of Propagation delays when using PTPv2 for time synch of sampled values, providing benefits over various other 1PPS systems. | Propagation delays are compensated for by PTPv2, providing benefits over various other 1PPS systems. | From the tests conducted PTPv2 was proved as a viable source of 1PPS timing signals. |

The papers catalogued in section 2.4 are divided into five headings. The headings represent the various initiatives to address the problem of measurement distribution in substation systems.  General comments on the papers reviewed are discussed in the paragraphs to follow.

The concepts of Instrument transformers discussed in the papers above sought to improve on the conventional transformer concepts to achieve higher performance, compactness and better reliability by having more monitoring systems. Instrument transformers have standardized sampling rates and interfaces according to IEC 60044-7 and IEC 60044-8 standards. The IEC 60044-8 Instrument transformer standard also introduced methods to interface to the instrument transformers using smart sensors.

The standards discussed above reflect an attempt to meet this challenge of distributing measurements in substation systems in the 1990s. While the IEEE 1451 standard was very successful in most industries such as gas, oil etc., it still lacked a lot of the elements that were needed in power systems. In 1996, Technical Committee 57 of the IEC began work on IEC 61850, with the objective of specifying the requirements and to provide a framework to achieve interoperability between the intelligent electronic devices supplied from different vendors

IEC 61850 proposes a digital bus-based architecture comprising of station bus and process bus. The Process layer of the substation is related to gathering information, such as voltage and current, from transducers connected to primary power plant equipment. The quantities are then digitized at the source by the merging unit, and the resulting sampled values are transmitted onto the process bus in a format compliant with the IEC 61850-9-2 standard.

A comparative analysis of the various epochs of measurement distribution systems in the substation environment is performed in section 2.5. Some of the advancements made in each era are discussed.

## 2.5. Comparative analysis of the developments in the existing literature

Before the 1990's, the primary research focus on transformers was on methods to improve conventional transformer performance by improving grading, stress loads and mechanical aspects of conventional transformers. One of the major focus areas was on improving the transient performance of bushings and the reduction of high voltage breakdowns between the bushing oil end-shielding and the turret (Minker, 2005).

In the early 90's studies incorporating optical technology in transformers began. Suva et al., 1990 laid most of the groundwork in his paper that discussed the design and development of optical instrument transformers based on Faraday and Pockels effects. Following the publication of this paper, more papers on instrument transformer design and instrument transformer transducer design that supported the Sawa et al., 1990 proposal, and also offered an alternative method for instrument voltage transformer design, were published. The Christen et al., 1995 paper discusses an alternative for instrument voltage transducers and proposes VT's without a capacitive voltage divider, instead using optical voltage transformers. Results from these studies and various other studies were conclusive as to the benefits that could be derived from the implementation on optical instrument transformers. The international bodies responded to the positives outlined in the various studies on instrument transformers by publishing the instrument transformer standard IEEE C57.13 in 1993, and a standard to interface instrument transformers to secondary plant. The IEC 60044 standard was published in December 1999.

The implementation of the IEC 60044 standard for metering and protection applications is discussed in Rahmatianet al., 2002. Bertolotto et al., 2007 extends on what is discussed in the Rahmatianet al., 2002 paper and discusses the introduction of data acquisition nodes using smart transducers to interface to optical voltage and current measuring systems based on a digital signal processing system compliant with the IEC 60044-7 and IEC 60044-8 standards for protection and metering, and the benefits that can be derived from these systems.
Another standard that attempted to meet the challenge of sampled values in the 1990's was the IEEE 1451. The Lee and Schneeman, 2000 paper presented the framework for measurement distribution using a bus-based architecture.

The Sal and Rakshit, 2004, and Zeng et al., 2006 papers proved that IEEE 1451 smart transducers could be used in power systems. Sal and Rakshit, 2004 used smart transducers for pressure and temperature monitoring of transformers and Zeng et al., 2006 discusses the measurement of transformer winding temperature using smart transducers.

In 2003, the IEC 61850 standard for substation communication networks was published. Apostolov, 2006, Anderson and Brand, 2000, and Baigent et al, 2004 introduced the concepts and motivation behind IEC 61850 and the device model approach in IEC 61850. The IEC 61850 incorporated the standards and framework that had previously existed. IEC 60044-8 standard was incorporated into the IEC 61850-9-1 standard. IEC 61850 standard uses an architecture very similar to the framework proposed by Lee and Schneeman, 2000.

IEC 61850-9-2 defines the mapping of the abstract sampled value services to an Ethernet communication network. As discussed in Apostolov, 2010, IEC 61850-9-2 presents a serious challenge when it comes to the actual implementation and interoperability of merging units. Because of the incompleteness of the IEC 61850-9-2 standard, an implementation guideline “IEC 61850-9-2LE” was published by the UCA International User Group, to facilitate the implementation and interoperability of devices utilizing sampled values (SVs).  The Apostolov, 2010 paper discusses the implementation guideline and implementation agreement for sampled values. Skendzic et al, 2007 discusses the merging unit concepts, sampled value message structure as outlined in IEC 61850-9-2.

Even with the implementation of IEC 61850-9-2LE, the implementation of the process bus paradigm is still almost non-existent in industry. As a result several research investigations have looked into simulation of communication services defined in the IEC 61850 standard. The Konka et al., 2012 and Yang et al., 2012 papers investigate the simulation of sampled values using Network Simulator version 3 (NS3) and Network Simulator version 2 (NS2) respectively. However both the systems developed are non-real time. While this proposed research project is also non-real time, the intent is to provide the foundation and framework for a real-time implementation.

## 2.6. Conclusion

The idea of sampled values is a very old concept in substation automation systems. The performance of the communication technologies in the past however was just not sufficient for real-time measurement distribution. As technology developed and with the introduction of non-conventional instrument transformers, bus-based real-time measurement systems started becoming a reality. The need for a standardized protocol interface was soon identified and from this IEC 60044-7 and IEC 60044-8 emerged. IEC 60044-7 and IEC 60044-8 standards reflects an attempt to meet this challenge of sampled values in the 1990s.

Another standard that attempted to meet the challenge of sampled values in the 1990’s was the IEEE 1451. The IEEE 1451 standard was however never really accepted in the power system industry as its origins were not in the power industry. Secondly the IEEE standard is a national standard not an international standard, so it never took off in certain parts of the world.

A need for an international standard still existed. When the working groups for IEC 61850 were discussing the SV concept, IEC 60044-8 was about to be finalized and released. The IEC 61850-9-1 became a link between the standards. IEC 61850-9-1 was later superseded by IEC 61850-9-2. IEC 61850-9-2 defines the mapping of the abstract sampled value services to Ethernet. This document however allowed developers a lot of freedom as to the actual implementation, for example the sample rates or the actual content of dataset is not defined. This presented a serious challenge when it came to the actual implementation of merging units by different vendors which could be interoperable. An implementation guideline “IEC 61850-9-2LE” was published by the UCA International User Group, to facilitate the implementation and interoperability of devices utilizing SVs. IEC 61850-9-2LE specifies a fixed dataset containing four currents and four voltage measurements. IEC 61850-9-2LE also specifies fixed sampling rates of 80 samples per cycle for protection applications and 256 samples per cycle for metering.

Even with the implementation of IEC 61850-9-2LE, the implementation of the process bus paradigm is still almost non-existent in industry. Devices that publish sampled values in a format compliant with IEC 61850-9-2LE guideline are still almost non-existent in industry. This project investigates the development of a sensor node on a PC platform.

Chapter three of this document discusses the details of the IEC 61850 standard and various investigations conducted to understand the sampled value device model and its message structure.

# CHAPTER 3

# An overview of the IEC 61850 standard with emphasis on GOOSE messaging and Sampled Values

## 3.1. Introduction

In order to implement a virtual sensor node based on the IEC 61850, an in-depth investigation into the IEC 61850 standard has to be conducted. The aim of this chapter is to examine the standardisation developments within the IEC 61850 substation automation environment, to study the substation automation framework presented in the IEC 61850 standard, and to outline the measurement distribution methods proposed in the IEC 61850 standard. This requires an in-depth understanding of the tools, methods and services specified in the IEC 61850 standard. In this chapter the various components that have been found to be fundamental to the understanding of substation automation systems and the IEC 61850 standard are deliberated upon.

From the literature review in the previous chapter, three main concepts/drivers for the development of the IEC 61850 standard have been identified as:

a) The provision of a virtualized substation automation framework for all substation automation devices.
b) The provision of a future-proof substation automation framework that is flexible, scalable, maintainable, and provides interoperability between substation automation devices.
c) The specification of the tools and processes for easing the engineering of substation automation systems.

This chapter examines the aforementioned IEC 61850 concepts in detail. Section 3.2 discusses the chronological development of the IEC 61850 standard. Section 3.3 provides an introductory overview of the IEC 61850 standard. Section 3.4 discusses the IEC 61850 substation automation virtualization framework (as mentioned in (a) above), focusing on the modelling philosophy and the abstract object services. Through the virtualization framework provided, the IEC 61850 standard supports Object-Oriented (OO) data modelling of all substation automation processes that are implemented and controlled. Section 3.5 discusses the mechanisms (as mentioned in (b) above) which aids in allowing flexibility, scalability, and interoperability – such as the IEC 61850 naming convention, and the process mapping, abstract services and mapping onto real communication protocols.

Section 3.6 discusses the IEC 61850 system engineering tools (as mentioned in (c) above). Section 3.7 examines network topologies as one of the critical items to be considered when designing IEC 61850, Ethernet-based substation automation systems. The network integrity and hence the topology deployed takes additional importance when dealing with the process level in IEC 61850 - this is due to the additional network traffic loading imposed by the time synchronization systems required by the Merging Units. Section 3.8 concludes on what has been discussed in this chapter and what will be discussed in the next chapter.

## 3.2. Overview of the chronological development of the IEC 61850 standard

The IEC 61850 standard is a result of various initiatives aimed at providing seamless integration of substation automation systems to higher level systems (enterprise systems), and to provide inter-device communication within the substation. Beginning in the early 1990s, the European Parliaments Research Initiative (EPRI) and the Institute of Electrical and Electronic Engineers (IEEE) spear-headed an effort to define a Utility Communications Architecture (UCA) for substation automation systems. The focus was on Inter-Control Center communications and Substation-to-Control-Center communications. These standardization initiatives culminated in the publication of the Inter-Control Center Communications Protocol (ICCP) specification which was later adopted by the International Electrotechnical Commission (IEC) as 61850 TASE.2

In 1994, EPRI/IEEE initiated the next phase of the UCA standardization efforts. This time more emphasis was placed on defining Station Level communication of a substation automation system. Two years later the Technical Committee 57 of the IEC began work on IEC 61850 with the similar agreement of defining a Station level. In 1997, the two groups agreed to work together. The results of the harmonization efforts are the IEC 61850 standard specification IEC 61850 is a superset of the UCA 2.0 standard. IEC 61850 contains almost all of the original UCA specification and offers additional features as shown in Fig. 3.1.

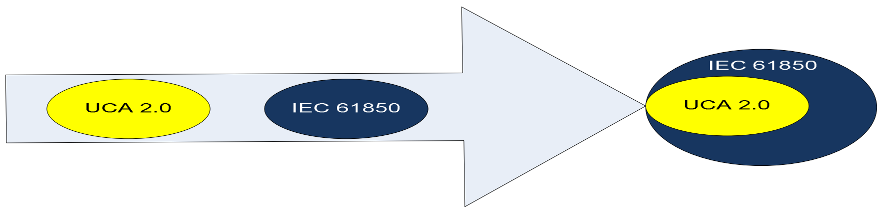
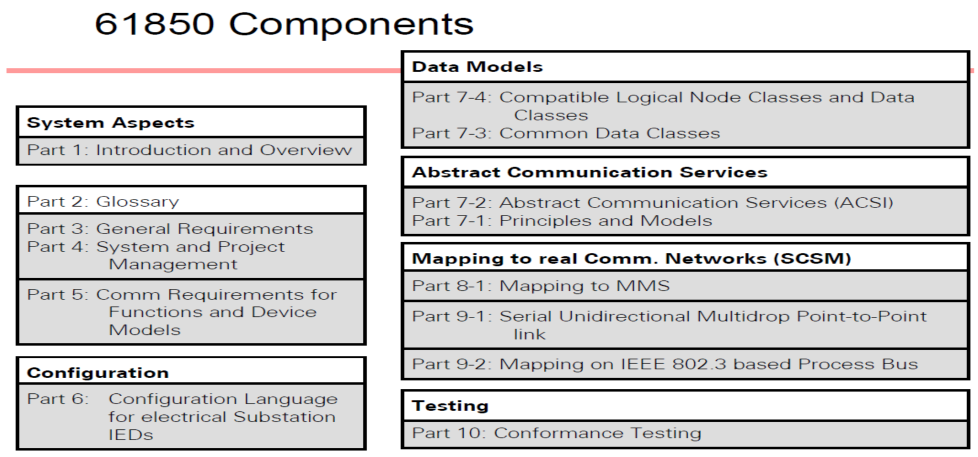


Figure 3.1: The migration of the UCA 2.0 standard to the IEC 61850 standard (Pofoud, 2002)

## 3.3. Introduction to the IEC 61850 standard

The IEC 61850 standard is divided into ten parts as is shown in Fig. 3.2 below. Part 1 of the IEC 61850 standard is the introduction and overview of the standard. Part 2 introduces the terms used within the standard. Parts 3, 4, and 5 of the standard identify the general and specific functional requirements for communication within the substation environment (Udren et al., 2000). These requirements are then used in the identification of the services, data models and protocol requirements for a particular application (Mackiewicz, 2006).



61850 Components
System Aspects
Part 1: Introduction and Overview
Part 2: Glossary
Part 3: General Requirements
Part 4: System and Project Management
Part 5: Comm Requirements for Functions and Device Models
Configuration
Part 6: Configuration Language for electrical Substation IEDs
Data Models
Part 7-4: Compatible Logical Node Classes and Data Classes
Part 7-3: Common Data Classes
Abstract Communication Services
Part 7-2: Abstract Communication Services (ACSI)
Part 7-1: Principles and Models
Mapping to real Comm. Networks (SCSM)
Part 8-1: Mapping to MMS
Part 9-1: Serial Unidirectional Multidrop Point-to-Point link
Part 9-2: Mapping on IEEE 802.3 based Process Bus
Testing
Part 10: Conformance Testing

Figure 3.2: IEC 61850 standard parts (IEC 61850, 2003)

Through these ten parts, the IEC 61850 standard specifies the substation automation requirements and provides a future-proof substation automation framework that allows for interoperability, scalability, flexibility, and maintainability of substation automation systems.  This future-proof substation automation framework utilizes, as some of its mechanisms, virtualization (virtualized SAS device model), interfaces, services and messaging systems. These will be discussed in detail in sections 3.4 through 3.7 below.

## 3.4. Virtualization

### Overview

The concept of virtualization began in the 1960's and originates from the Information Technology (IT) environment. In the IT environment, virtualization refers to the act of creating a virtual version of devices and its sub-components. Virtualization tools such as Hypervisor and VMware provide the framework or methodology for dividing a device's resource into multiple execution environments, by applying various technologies.

Virtualization is often used in the IT space as a means to fully maximize the potential use of a specific hardware investment.  Virtualization technologies/tools provide a means to virtualize the hardware platform, Operating System (OS), storage device(s), network resources, etc. An example of virtualized IT servers is illustrated in Fig. 3.3 below.

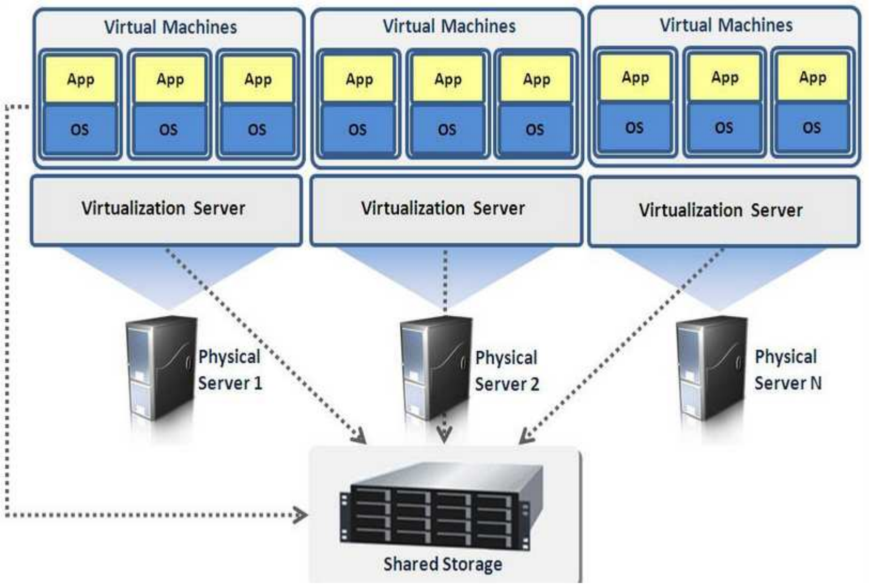


Figure 3.3: Example of virtualized IT servers

In Fig. 3.3, multiple physical servers are virtualized within one shared storage device. Several applications are running within the virtualized servers shown in Fig. 3.3, essentially sharing the shared storage server's memory, processing power, network resources, etc. Some of the advantages of virtualization are:

- Improved System Reliability and Security. Virtualization is used to prevent system crashes and improve the overall system venerability due to memory corruption caused by software drivers and provides a means of isolating the Input/output (I/O) resources.
- That it allows for powerful debugging and performance monitoring.
- That the virtualization framework allows for emulation or simulation of a complete device.
- Scalability.
- The provision of interoperability with minimal co-dependencies.

The concept of virtualization has in recent years been introduced into the Operation Technology (OT) environment. The IEC 61850 has leveraged the concept of virtualization from the IT environment, to provide a representation of the behaviour of real substation devices (Brunner et al., 2007) through the specification of standardized models representing the actual equipment (Mackiewicz, 2006). This is expanded upon in more detail in sections 3.4.1 and 3.4.2 below.

### 3.4.1 Virtualization – in the standards-based power system industry

The main functions of Substation Automation Systems (SASs) are to protect, control and monitor plant equipment used in the substation and the substation feeders. These functions are used as the building blocks for the object orientated physical and logical device information models defined in the IEC 61850 standard. The information models and the device modelling methods form part of the core of the IEC 61850 standard. The models defined in the IEC 61850 standard enable virtualizing of real devices as depicted in Fig. 3.4.

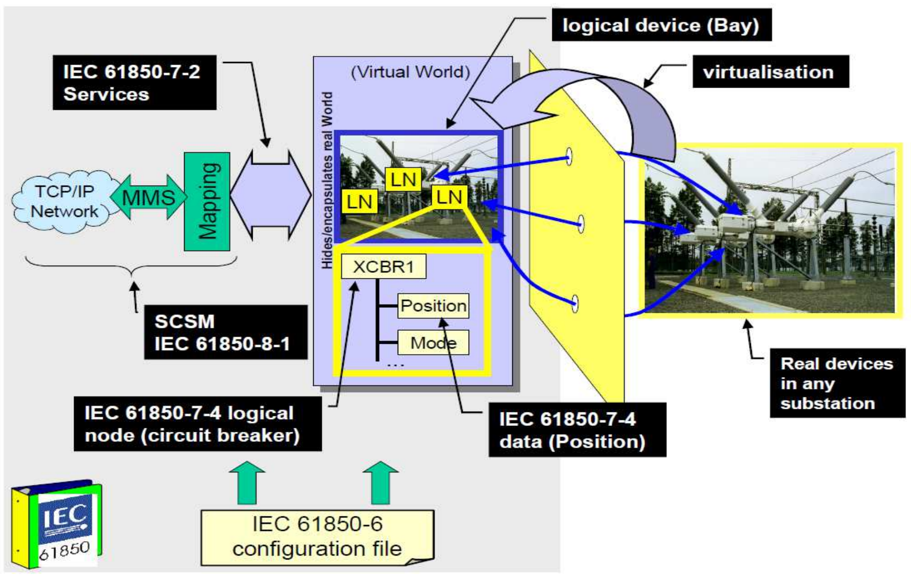

logical device (Bay)
IEC 61850-7-2 Services
(Virtual World)
virtualisation
Hides/encapsulates real World
TCP/IP Network
MMS
Mapping
LN
LN
LN
XCBR1
Position
Mode
SCSM IEC 61850-8-1
Real devices in any substation
IEC 61850-7-4 logical node (circuit breaker)
IEC 61850-7-4 data (Position)
IEC 61850
IEC 61850-6 configuration file

Figure 3.4: IEC 61850 modelling approach (IEC 61850-7-1, 2004)

The IEC 61850 standard utilizes abstract models to define information and information exchange in a manner that is self-determining and independent of protocol implementation. The standard uses the concept of virtualization to provide a view of some of the real device aspects that are fundamental for information exchange with other devices (IEC 61850-7-1).

The IEC 61850 standard further defines an Abstract Communication Service Interface (ACSI) and a Specific Communication Service Mapping (SCSM) that should be supported by IEC 61850 compliant devices to provide interoperability. As described in IEC 61850-5, the approach of the standard is to decompose the application functions into the smallest entities, which are used to exchange information. The data modelling and object oriented mechanisms utilised within the IEC 61850 standard are outlined further in section 3.4.2 below.

### 3.4.2 IEC 61850 – Data modelling

The IEC 61850 standard divides the SAS functions into thirteen different logical groupings of data as is shown in Fig. 3.5. The IEC 61850 standard attempts to model all data that could originate in a substation automation system to one of these groups. The groups are subdivided into 92 different Logical Nodes and each of these logical nodes is composed of 355 data classes that represent some application-specific meaning. A complete list of logical nodes is defined in the IEC 61850 standard and is presented in Appendix 3 of the standard (IEC 61850-7, 2004).

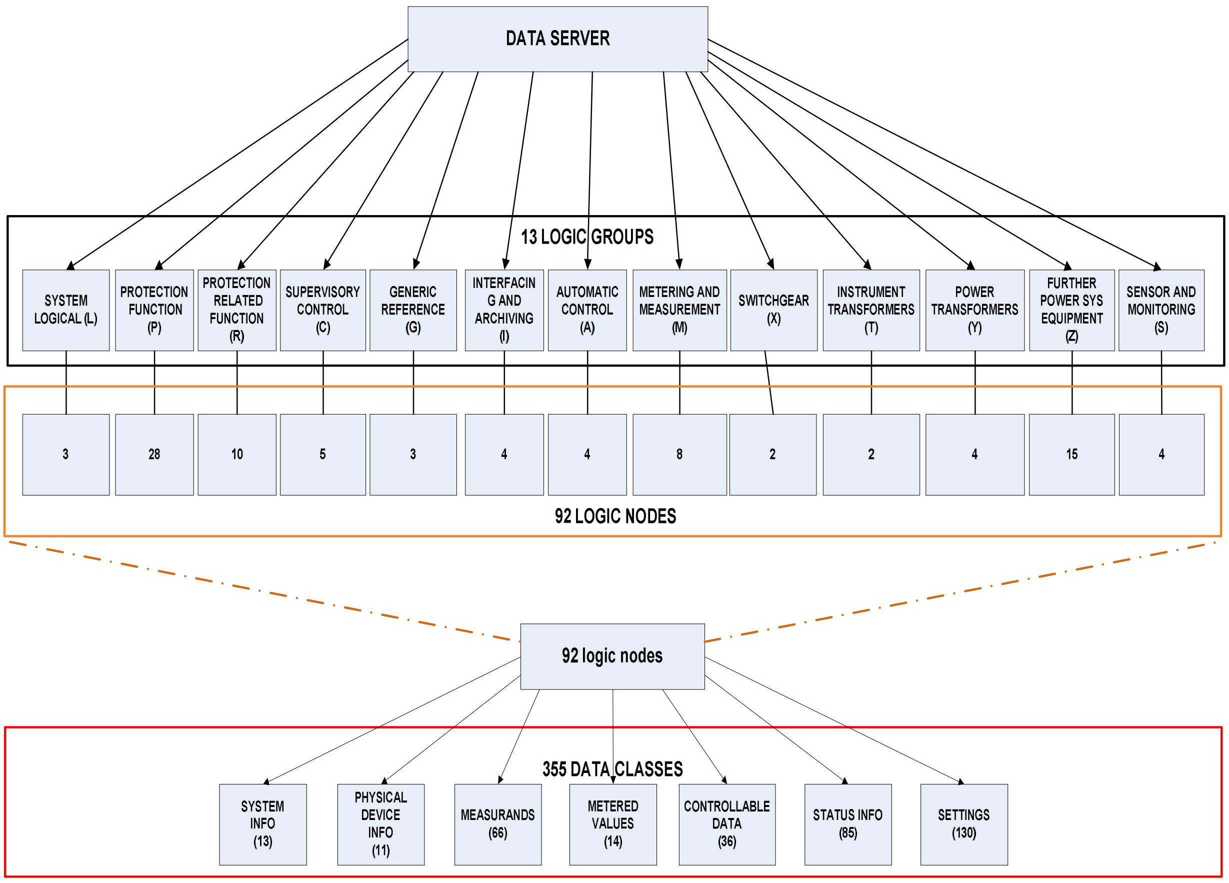


Figure 3.5: Logic grouping of substation data according to IEC 61850 standard.

The data model shown in Fig. 3.5 is hierarchical, with the logical nodes as the key object in the IEC 61850 data modelling structure. The logical nodes represent particular functions within a device (Mackiewicz, 2006) and contain data linked to that specific function.

A Logical Device (LD) forms a container for three or more Logical Nodes (LN). A logical device is always encapsulated within a physical device. In the IEC 61850 standard, the device model begins with the physical device (PD). A physical device consists of a Logical Device(s) as down in Fig. 3.6 below. In Fig. 3.6, the outermost shell represents the "Real devices" (Physical devices). Inside the physical device is a virtual model of the actual physical device.

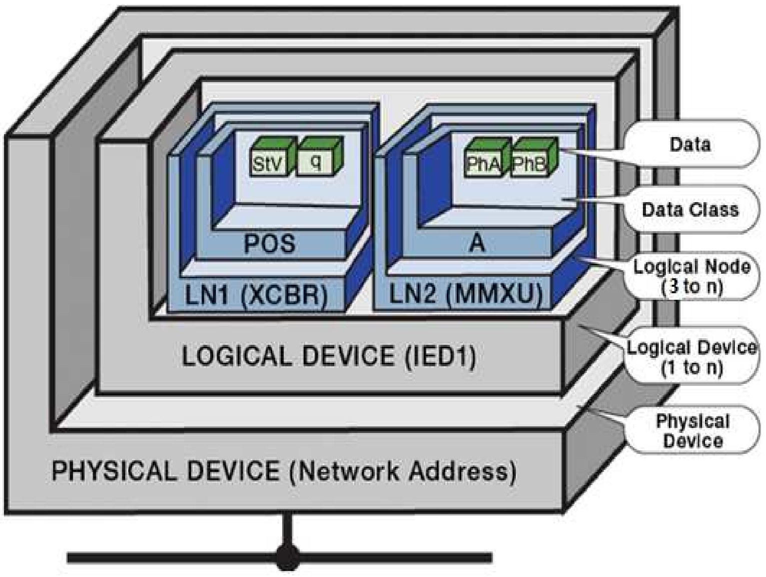


Figure 3.6: An IEC 61850 device representation (Gers, 2004)

A logical device is a compound of logical nodes that are related to a specific function inside a substation. At least three logical nodes must be within a logical device. As an example, a logical device is usually comprised of two LNs related to common functions within the logical device (such as LLN0 and LPHD), and at least one LN performing some protection and/or control function.

The architectural model that the IEC 61850 standard adopts is that of "abstracting" the definition of the data items and the services. As a result, many of the data objects are made up of elements such as Status, Control, Measurement, etc., thus the concept of "Common Data Classes" or "CDC" was developed which defined common building blocks for creating the larger data objects. Fig. 3.7 shows a Unified Modeling Language (UML) diagram of a conceptual IEC 61850 device.

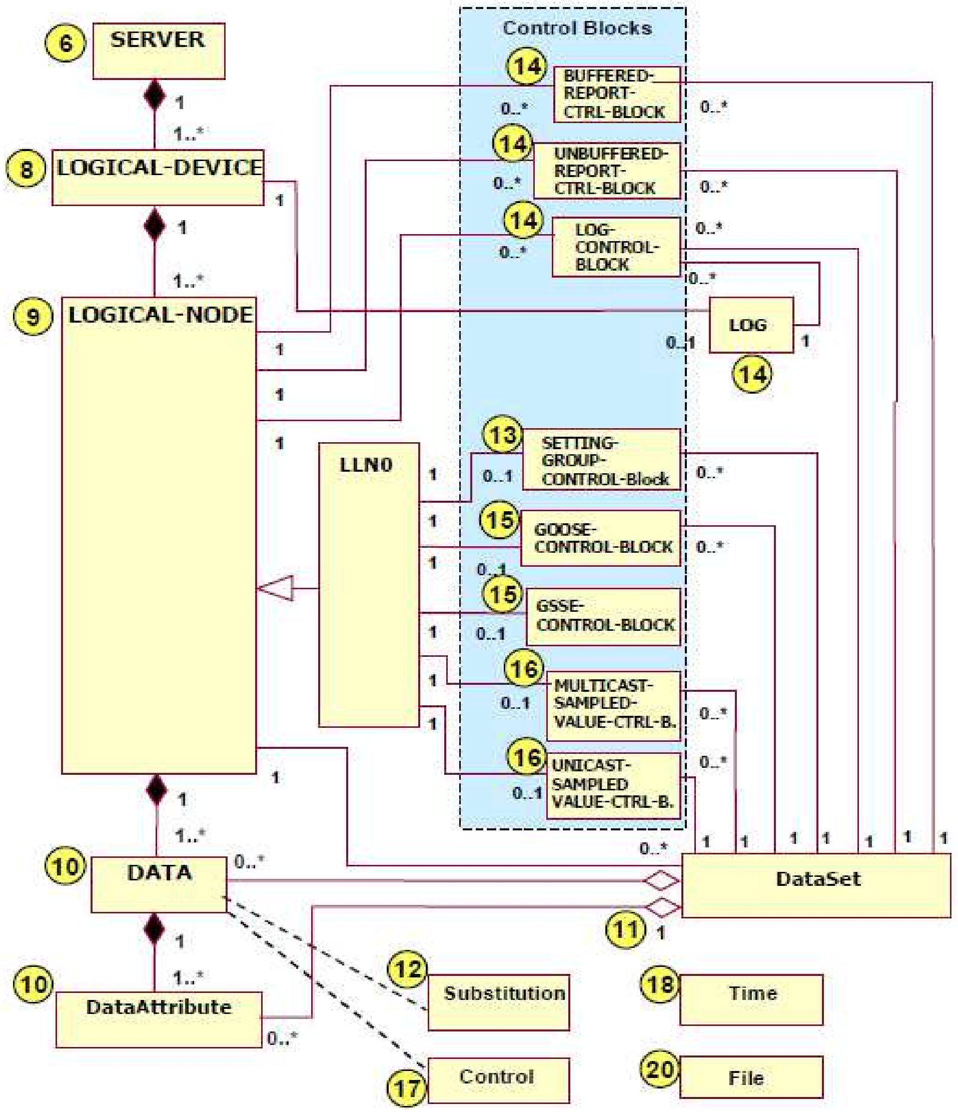


Figure 3.7: Conceptual IEC 61850 device and services model (IEC 61850, 2003)

In Figure 3.7: above, Block 6 which is the physical data server is the top-most component of the hierarchical model. It serves as an encapsulating entity for logical devices (block 8) and logical nodes (block 9). The Logical Device (LD) in turn is the logical correspondence of a physical device. The logical device is a group of logical nodes performing similar functions. Each logical device contains three or more Logical Nodes, each of which contains elements of data. Each of the data elements conforms to the IEC 61850-7-3 specifications of Common Data Class (CDC). The CDC's describe the type and structure of the data within a logical node. An example of the use of the logical node modelling technique using the measurement logical node MMXU is shown in Fig. 3.8.

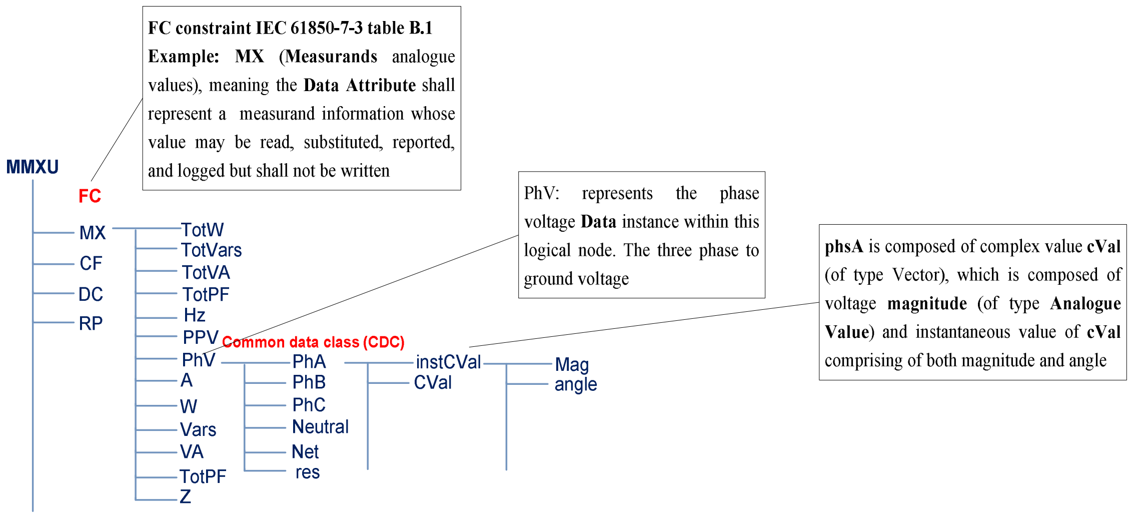

FC constraint IEC 61850-7-3 table B.1
Example: MX (Measurands analogue values), meaning the Data Attribute shall represent a measurand information whose value may be read, substituted, reported, and logged but shall not be written
MMXU
FC
MX
CF
DC
RP
TotW
TotVars
TotVA
TotPF
Hz
PPV
PhV
A
W
Vars
VA
TotPF
Z
Common data class (CDC)
PhV: represents the phase voltage Data instance within this logical node. The three phase to ground voltage
PhA
PhB
PhC
Neutral
Net
res
instCVal
CVal
Mag
angle
phsA is composed of complex value cVal (of type Vector), which is composed of voltage magnitude (of type Analogue Value) and instantaneous value of cVal comprising of both magnitude and angle

Figure 3.8: Example showing the decomposition of the MMXU logical node

With reference to this example, the CDC belongs to Function Constraints (FC) that groups CDC's into categories, as depicted in Fig. 3.8. The Functional Constraints (FC) is a property of the "data attribute" that characterizes the specific use of the "attribute". The CDC defines the structure for common types that are used to describe "data objects".

The major architectural model of IEC 61850 is that of "abstracting" so that the data objects and services can be mapped to any protocol that can meet the data and service requirements (Mackiewicz, 2006). By decoupling the data objects and services from the underlying protocol, the IEC 61850 standard provides a future-proof architecture that enables all the substation automation servers to behave in an identical manner from the communication network point of view (Piirainen, 2010).

## 3.5. IEC 61850 – Substation Automation Future Proofing

The IEC 61850 proposes a "future-proof" substation architecture that is capable of providing interoperability between substation automation devices (Mohagheghi et al., 2007).

Future proofing in the IEC 61850 standard is facilitated by:

i) Decoupling of the data objects and services from the underlying protocol. This provides a physical separation between the slow growing Substation-Specific Applications (SSA) from the fast advancing communication network infrastructure. This separation is provided by the Abstract Communication Services Interface (ACSI).

ii) Using a hierarchical naming convention instead of indexed addressing. The IEC 61850 standard naming convention provides a more descriptive context for the data (Mohagheghi et al. 2007).

iii) Defining a system configuration language that can be easily extendable and is open for future data types and services.

These are discussed in more detail below.

### 3.5.1. IEC 61850 – Naming Convention

The IEC 61850 standard specifies a method for transforming the virtualized device models into named variable objects with unique and unambiguous references. Fig. 3.9 below shows an example of the IEC 61850 object naming convention.

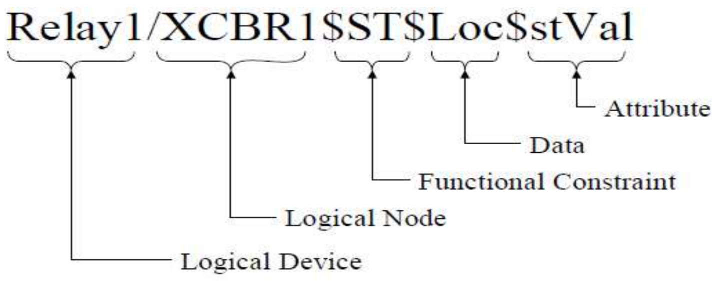


Figure 3.9: Anatomy of an IEC 61850 Object Name (Mackiewicz, 2006)

The IEC 61850 standard provides a framework for a hierarchical naming convention. The naming convention references the named grouping of data and associated services that are logically related to a substation automation function. In Fig. 3.9 *Relay 1* refers to the logical device. The Logical device has a circuit-breaker object that is linked to it, and is referenced by *XCBR* in Fig. 3.9. The circuit-breaker object contains Data that belongs to a certain Functional Constraint and has a certain Attribute.

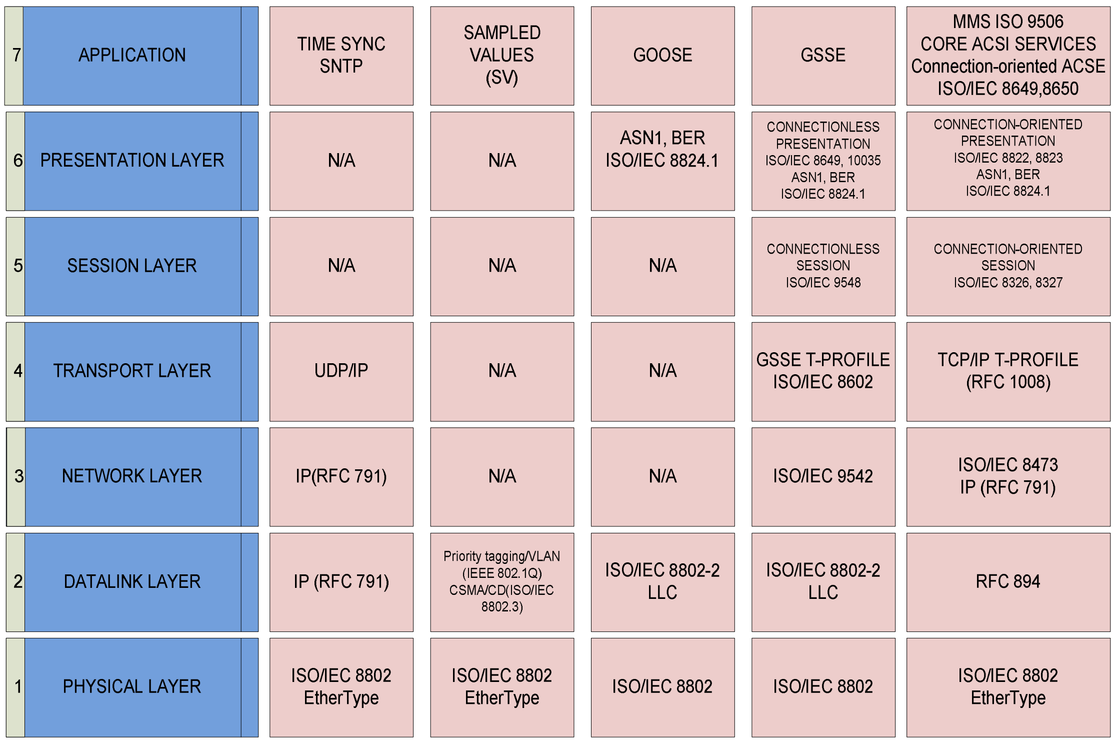
By providing the possibility to create a naming hierarchy for functional names, the IEC 61850 standard allows the generic binding of all information in the IEDs to the application (Brunner et al., 2012). In this way future-proofing for IEC 61850 substation automation systems is provided, as generic logical nodes can be added. This allows for extension of the IEC 61850 standard naming convention to other application domains, the naming convention has already been used to accommodate the hydro-electric industry, which did not form part of the original IEC 61850 SAS naming convention. Additionally the IEC 61850 naming convention allows for the addition of a suffix to Logical Node names, enabling identification of the Logical Node’s purpose.

### 3.5.1.1 Compatibility of the IEC 61850 naming convention with other International standards

IEC 61850 aligns and integrates well into smart grid philosophy. IEC 61850 is structured after the Common Information Model (CIM) standards (IEC 61968 and IEC 61970). CIM is the general information model of utility-specific data and underlies the information structures of most, if not all, IEC utility standards. IEC 61968 defines standards for information exchanges between electrical distribution systems. The IEC 61970 defines standards for the application program interfaces for Energy Management Systems (EMS)

## 3.5.2. Mapping to an actual protocol

The IEC 61850 standard has adopted a layered Open System Interconnect (OSI) model for all substation automation needs. The adopted model has leveraged the use of existing standard protocols in all layers up to layer 7 in the OSI reference model. Fig. 3.10 shows the OSI model adopted by the IEC 61850 standard and the various standards which were leveraged in the process.



| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | APPLICATION | TIME SYNC SNTP | SAMPLED VALUES (SV) | GOOSE | GSSE | MMS ISO 9506 CORE ACSI SERVICES Connection-oriented ACSE ISO/IEC 8649,8650 |
| 6 | PRESENTATION LAYER | N/A | N/A | ASN1, BER ISO/IEC 8824.1 | CONNECTIONLESS PRESENTATION ISO/IEC 8649, 10035 ASN1, BER ISO/IEC 8824.1 | CONNECTION-ORIENTED PRESENTATION ISO/IEC 8822, 8823 ASN1, BER ISO/IEC 8824.1 |
| 5 | SESSION LAYER | N/A | N/A | N/A | CONNECTIONLESS SESSION ISO/IEC 9548 | CONNECTION-ORIENTED SESSION ISO/IEC 8326, 8327 |
| 4 | TRANSPORT LAYER | UDP/IP | N/A | N/A | GSSE T-PROFILE ISO/IEC 8602 | TCP/IP T-PROFILE (RFC 1008) |
| 3 | NETWORK LAYER | IP(RFC 791) | N/A | N/A | ISO/IEC 9542 | ISO/IEC 8473 IP (RFC 791) |
| 2 | DATALINK LAYER | IP (RFC 791) | Priority tagging/VLAN (IEEE 802.1Q) CSMA/CD(ISO/IEC 8802.3) | ISO/IEC 8802-2 LLC | ISO/IEC 8802-2 LLC | RFC 894 |
| 1 | PHYSICAL LAYER | ISO/IEC 8802 EtherType | ISO/IEC 8802 EtherType | ISO/IEC 8802 | ISO/IEC 8802 | ISO/IEC 8802 EtherType |

Figure 3.10: IEC 61850 OSI model

The benefit of the model adopted by the IEC 61850 standard is that there is no new protocol development required. The standard leverages and integrates internationally recognized and standardized protocols. The layered protocol stack approach adopted by the IEC 61850 standard provides a small set of specific services to the layer below and the layer above, thus providing independence between the layers. As a result, the functions of a specific layer can be modified without changing the overall structure of the model. This is, provided that the services provided by the layer which is modified, maintains its relationship to the layer above and below. The protocols defined at each layer establish a peer-to-peer relationship with the corresponding layer of the receiving device.

### 3.5.2.1 Interfaces and Services

Fig. 3.11 shows the hierarchical architecture model of the IEC 61850 standard and the interfaces between the various components of an IEC 61850 substation automation system. The interface functions have been elaborated on previously in chapter 2. The hierarchical architecture depicts three levels on which substation automation devices can be interfaced, namely:

- The station level,
- The bay level and
- The process level

The substation automation components at the same level or at different levels of the hierarchy are then linked together by the various interfaces, using the communication services/protocols defined in the standard. The station level, often referred to as the "station bus" connects all of the Intelligent Electronic Devices (IEDs) communicating with one another and to the Human Machine Interface (HMI) using interfaces 1,3,6,7 and 8.

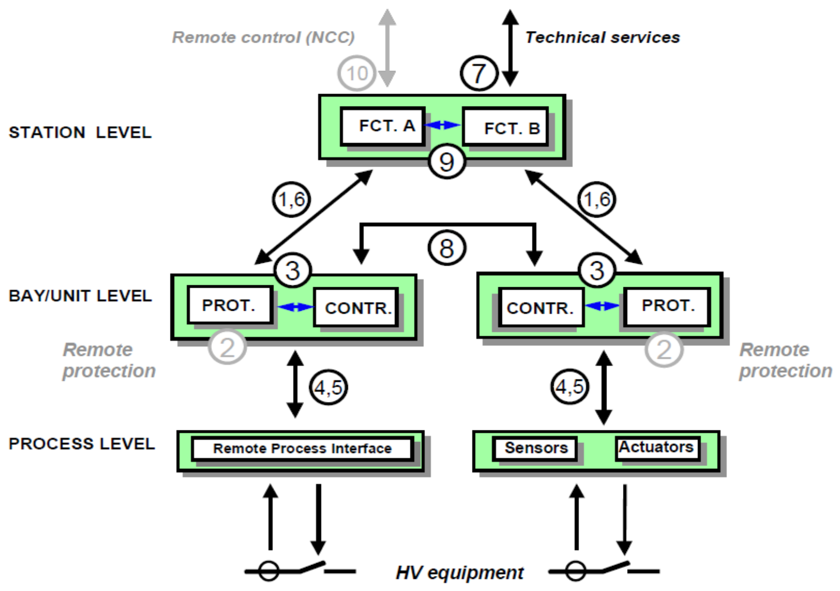


Figure 3.11: IEC 61850 substation automation system interfaces (IEC 61850-1, 2004)

The IEC 61850 standard defines communication interfaces as shown in Fig. 3.11, to support:

a) Publisher/Subscriber services, and
b) Client/Server services

These services are used to facilitate communication between IEC 61850 devices using Manufacturing Messaging Specification (MMS) for client/server applications, Generic Object Oriented Substation Events (GOOSE) messaging for fast peer-to-peer communication between the publishing and subscribing device(s), and Sampled Value (SV) messaging for measurement distribution.

IEDs communicate upstream to the HMI using the messaging services provided by the Manufacture Message Specification (MMS) protocol through interface (7) in Fig. 3.11 above. Inter-IED communication is achieved through interface (8) using a publisher/subscriber messaging system referred to as GOOSE. Interfaces 4 and 5 connect Process level or "process bus" equipment to the Bay level and vice versa. The process level devices distribute power system information from switchyard source-devices to the Bay Level, utilising a publisher/subscriber mechanism.

The IEC 61850-8-1 standard defines the mapping of the abstract objects and services of process level, bay level and station level devices to the application layer. In addition to mapping to the application layer, IEC 61850-8-1 defines profiles for the data-link, networking, transport, session and the presentation layers of the communication stack as shown in Fig. 3.12.

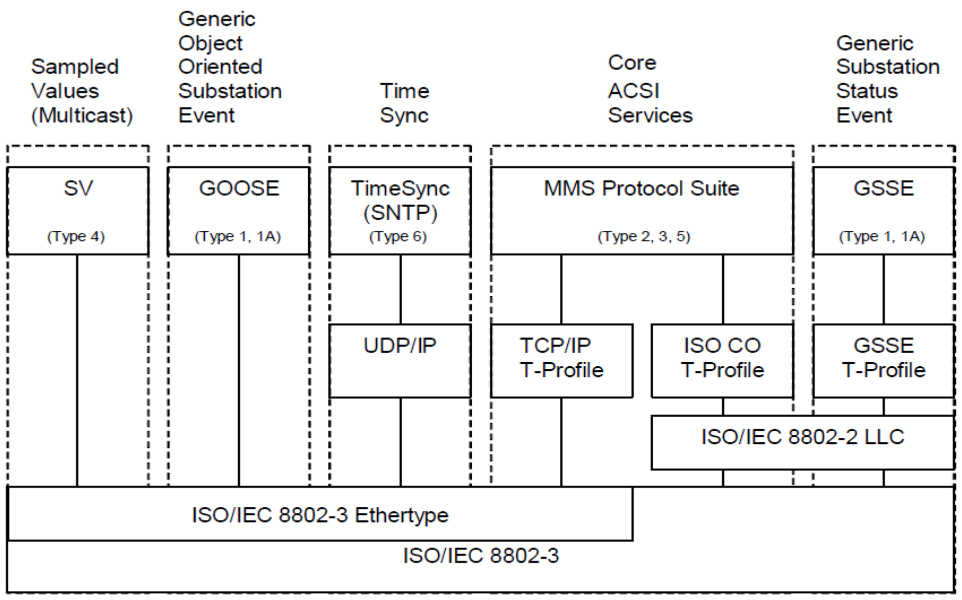


Figure 3.12: Overview of communication functionality and profiles (IEC 61850-8-1, 2004)

Fig. 3.12 shows the Specific Communication Services Mapping (SCSM) defined in the IEC 61850 standard. GOOSE and Sampled Values applications map directly into the Datalink layer (Layer 2) thereby eliminating additional processing of any middle layers, whereas MMS uses all seven layers of the Open System Interconnect (OSI) stack. All data maps onto an Ethernet link using either the data type “Ethertype” in the case of Sampled Values (SV), GOOSE and Time Synchronization (TimeSync), or using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) method defined in “IEEE 802.3” for MMS and Generic Sub-Station Event (GSSE) messages. Table 3.1 below compares the IEC 61850 layer 2 messaging systems, GOOSE and Sampled Values.

Table 3-1: Comparison between GOOSE and Sampled Values

| Comparison between GOOSE and Sampled Values | | |
|---|---|---|
| | GOOSE | Sampled Values |
| Reporting | Event-based | Non-event Based (Periodic) |
| Detection Loss | Yes | Yes (through VAR. Counter) |
| Re-transmission | Yes | No (Sequential) |
| Relation | 1:n ( Layer 2 Multicast) | 1:n ( Layer 2 Multicast) |
| Timelessness | Yes (few milliseconds) | Yes (few milliseconds) |
| Message Priority | High | High |
| Priority Tagging | Yes | Yes |
| Data Type | Maps on Layer 2 (Datalink) using Ethertype | Maps on Layer 2 (Datalink) using Ethertype |
| Message content encoding | ASN.1 BER | ASN.1 BER |
| OSI layers utilised | 1,2 and 7 application layer | 1,2 and 7 application layer |
| Max. Message Size | < 1500 octets | < 1500 octets |
| Document | IEC 61850-8-1 | IEC 61850-9-2 |

Goose and Sampled value messaging systems are outlined in more detail in Sections 3.5.2.2 and 3.5.2.3 respectively.

### 3.5.2.2 GOOSE Messaging

GOOSE messaging refers to the time-critical messaging, which has been defined for fast peer-to-peer communication between substation automation Intelligent Electronic Devices (IEDs). GOOSE messages are generally used to transfer status information between two or more devices, namely the publisher and the various subscribing devices. They are transmitted as a multicast over Local Area Network (LAN). An example of the Publisher/Subscriber mechanism used in the GOOSE message exchange is illustrated in Fig. 3.13.

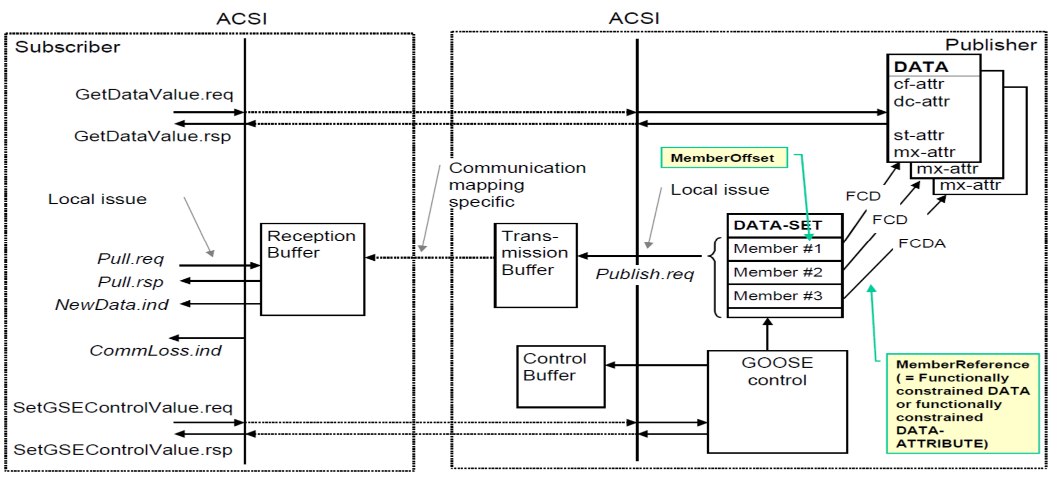
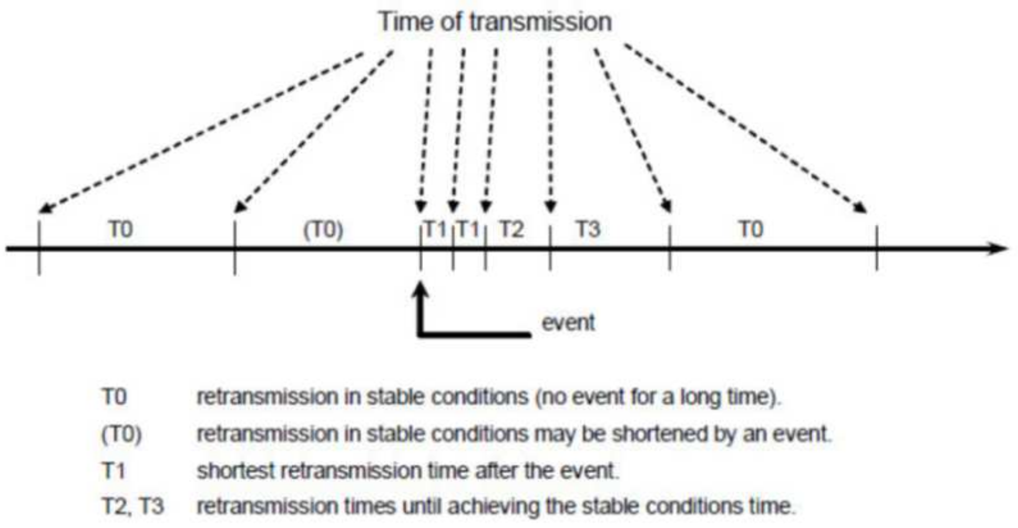


Figure 3.13: An example of the GOOSE Publisher-Subscriber mechanism (IEC 61850-7-2, 2004)

Fig. 3.13 gives an overview of the classes and services of the GOOSE model. The publishing device (Publisher) comprises of logical correspondents of the physical device. Each of these logical correspondents contains one or more Logical Nodes, which contain some elements of data. The data is contained within a dataset that groups various data attributes which belong to a specific functional constraint. A change in the data contained within the dataset will result in the transmission buffer of the publisher being updated with the local service “publish” and the values are transmitted as a GOOSE message.

The GOOSE messages contain information that allows the receiving device to know that a status has changed and the time of the last status change. The time of the last status change allows a receiving device to set local timers relating to a given event. GOOSE is an event-driven, fast, connection-less communication service. GOOSE security is achieved by the repetition of messages a number of times as shown in Fig. 3.14.



| | |
|---|---|
| T0 | retransmission in stable conditions (no event for a long time). |
| (T0) | retransmission in stable conditions may be shortened by an event. |
| T1 | shortest retransmission time after the event. |
| T2, T3 | retransmission times until achieving the stable conditions time. |

Figure 3.14: GOOSE message transmission time (IEC 61850-8-1, 2004)

The Quality of Service (QoS) specified within the IEC 61850 requires the repetition of GOOSE messages a number of times to achieve higher reliability. A device newly connected to the network shall send current status values as its initial GOOSE message. Moreover, all devices sending GOOSE messages shall continue to send messages between each other with a long cycle time, even if no status value change has occurred as shown in Fig. 3.14 (T0). This ensures that devices that have been connected recently will know the current status values of their peer devices.

GOOSE re-transmission may be shortened by an event. A status value change in the data contained within the dataset will result in an event, and a GOOSE message being published repeatedly a number of times with a short cycle time Fig. 3.14 (T1). The duration of the cycle will gradually increase, until the pre-event status has been reached, as shown in Fig 3.14 (T2, T3 and T0).

From a practical perspective, this means that a specific GOOSE control block (GOCB) has to be configured for each GOOSE message. The GOOSE control block includes the information which a dataset needs for transmission and information that can be used for the GOOSE validation on the subscribing device.

The GOOSE control block specifies information such as the GOOSE control block name, GOOSE control block reference, GOOSE enable and the various services that enable the transmission of a GOOSE message, as shown in Figure 3.15: GOOSE Control Block Class.

| GoCB class | | | | |
|---|---|---|---|---|
| **Attribute name** | **Attribute type** | **FC** | **TrgOp** | **Value/value range/explanation** |
| **GoCBName** | ObjectName | GO | - | Instance name of an instance of GoCB |
| **GoCBRef** | ObjectReference | GO | - | Path-name of an instance of GoCB |
| **GoEna** | BOOLEAN | GO | dchg | Enabled (TRUE) \| disabled (FALSE) |
| **AppID** | VISIBLE STRING65 | GO | | Attribute that allows a user to assign a system unique identification for the application that is issuing the GOOSE. DEFAULT GoCBRef |
| **DatSet** | ObjectReference | GO | dchg | |
| **ConfRev** | INT32U | GO | dchg | |
| **NdsCom** | BOOLEAN | GO | dchg | |
| **Services**<br>SendGOOSEMessage<br>GetGoReference<br>GetGOOSEElementNumber<br>GetGoCBValues<br>SetGoCBValues | | | | |

Figure 3.15: GOOSE Control Block Class (IEC 61850-7-2, 2004)

GOOSE messaging has been discussed above, and sampled value messaging at the process level is discussed in more detail in the section below.

### 3.5.2.3 Process Level

The Process level often referred to as the process-bus, is defined in the IEC 61850-9-2 standard for gathering information, such as Voltage and Current, from the transducers connected to the primary power system equipment as shown in Fig. 3.11. The quantities are then digitized at the source by a device referred to as a “Merging Unit” and the resulting sampled values are generated and published onto the data network.

The Merging Unit at the process level receives a time-coherent set of samples of three phase voltages, three phase current and neutral values of both voltage and current. The values are then converted to digital values as shown in Fig 3.16.

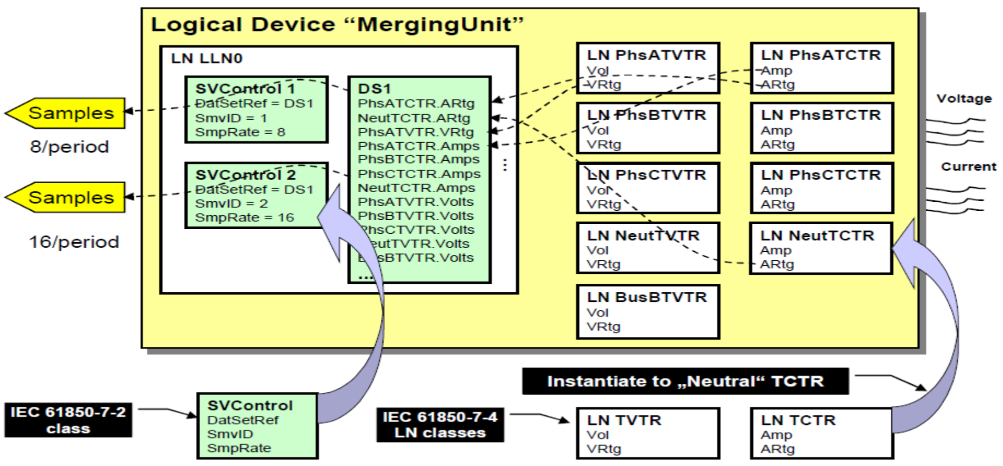
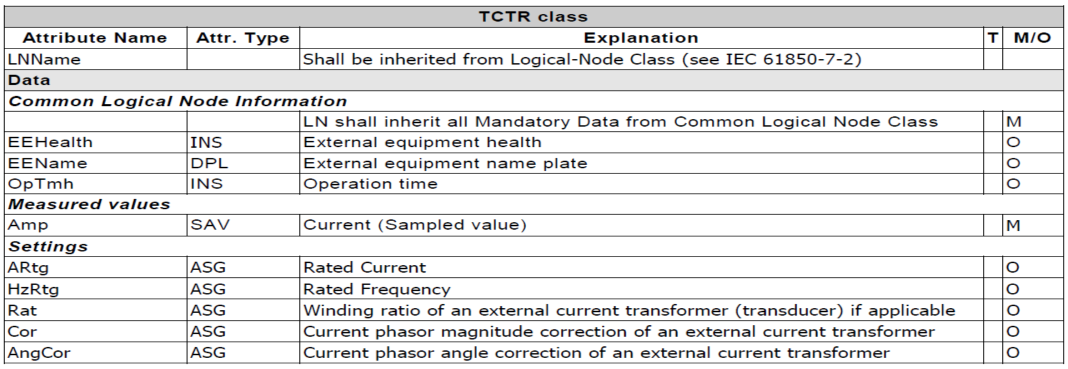


Figure 3.16: Merging unit and Sampled Value exchange (IEC 61850-7-1 2004)

The merging unit encodes the digitized values, packages the information into a "configurable" dataset that can be communicated through a multi-cast/Uni-cast transmission onto the process level (IEC 61850, 2003).

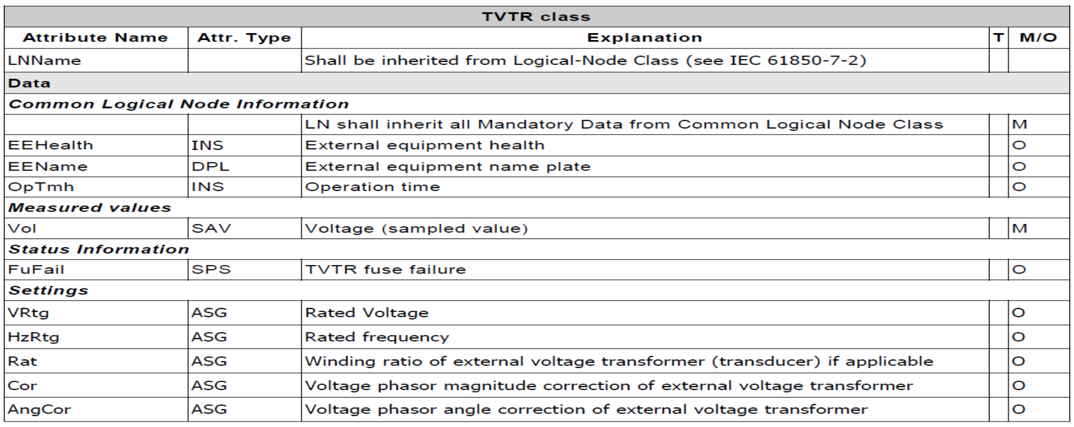
With reference to Fig. 3.16 above, the logical nodes involved in the acquisition of measured data are TVTR and TCTR for voltage and current respectively. Four instances of the TCTR and TVTR logical nodes are shown in Fig. 3.16 because measurements representing primary equipment is typically done via a single phase representation, therefore the need for four instances of the logical nodes TCTR or TVTR representing the three phases and neutral phase are required (Brunner et al., 2005).

Each of the logical nodes shown in Fig. 3.16 in essence can be thought of as an object with both attributes and operations. These objects are an instance of a class, which describes the properties and behaviour of the objects. So for every object type, there needs to be a defined class model. The IEC 61850-7 standard specifies the general definition of such a class model, the logical node class model. The detail of these classes defining the logical nodes is shown in Fig. 3.17 and Fig. 3.18 below.



| TCTR class | | | | |
|---|---|---|---|---|
| **Attribute Name** | **Attr. Type** | **Explanation** | **T** | **M/O** |
| LNName | | Shall be inherited from Logical-Node Class (see IEC 61850-7-2) | | |
| **Data** | | | | |
| ***Common Logical Node Information*** | | | | |
| | | LN shall inherit all Mandatory Data from Common Logical Node Class | | M |
| EEHealth | INS | External equipment health | | O |
| EEName | DPL | External equipment name plate | | O |
| OpTmh | INS | Operation time | | O |
| ***Measured values*** | | | | |
| Amp | SAV | Current (Sampled value) | | M |
| ***Settings*** | | | | |
| ARtg | ASG | Rated Current | | O |
| HzRtg | ASG | Rated Frequency | | O |
| Rat | ASG | Winding ratio of an external current transformer (transducer) if applicable | | O |
| Cor | ASG | Current phasor magnitude correction of an external current transformer | | O |
| AngCor | ASG | Current phasor angle correction of an external current transformer | | O |

Figure 3.17: TCTR logical node class (IEC 61850-7-4, 2004)

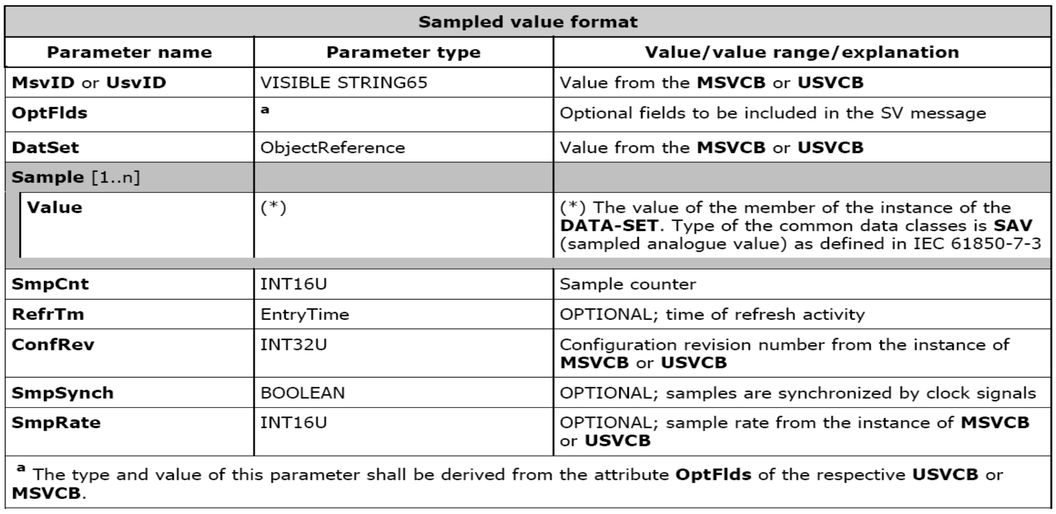


| TVTR class | | | | |
|---|---|---|---|---|
| **Attribute Name** | **Attr. Type** | **Explanation** | **T** | **M/O** |
| LNName | | Shall be inherited from Logical-Node Class (see IEC 61850-7-2) | | |
| **Data** | | | | |
| ***Common Logical Node Information*** | | | | |
| | | LN shall inherit all Mandatory Data from Common Logical Node Class | | M |
| EEHealth | INS | External equipment health | | O |
| EEName | DPL | External equipment name plate | | O |
| OpTmh | INS | Operation time | | O |
| ***Measured values*** | | | | |
| Vol | SAV | Voltage (sampled value) | | M |
| ***Status Information*** | | | | |
| FuFail | SPS | TVTR fuse failure | | O |
| ***Settings*** | | | | |
| VRtg | ASG | Rated Voltage | | O |
| HzRtg | ASG | Rated frequency | | O |
| Rat | ASG | Winding ratio of external voltage transformer (transducer) if applicable | | O |
| Cor | ASG | Voltage phasor magnitude correction of external voltage transformer | | O |
| AngCor | ASG | Voltage phasor angle correction of external voltage transformer | | O |

Figure 3.18: TVTR logical node class (IEC 61850-7-4, 2004)

Sampled Values from these logical nodes are then transmitted as primary current values. This means that the transformer ratio and the correction factors are of no interest for the transmitted samples, but can be added as optional parameters within the logical node for maintenance purposes. In addition, status information and optional parameters/settings can be added to the logical nodes (TCTR and TVTR).

The abstract sampled value format used for the sampled value message defined in part 7 of the IEC 61850 standard is shown in Fig. 3.19.

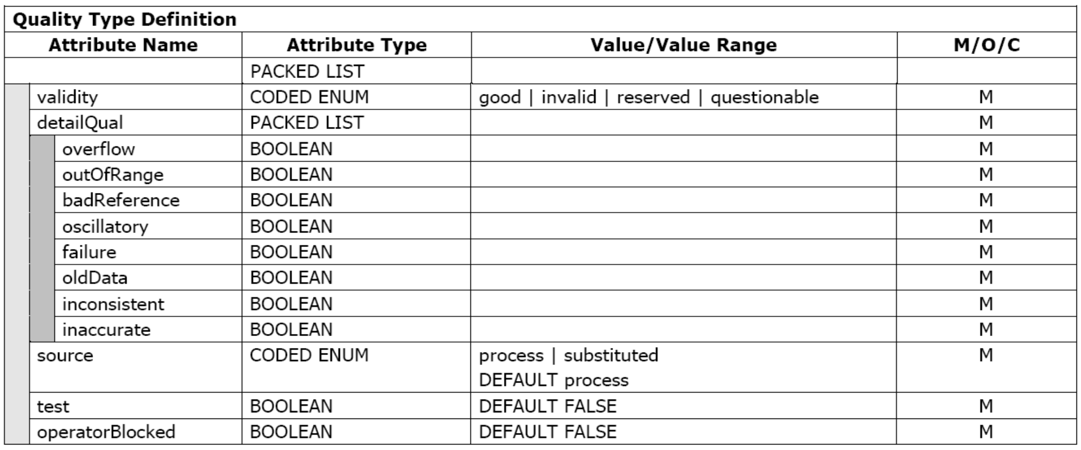


| **Sampled value format** | | |
|---|---|---|
| **Parameter name** | **Parameter type** | **Value/value range/explanation** |
| **MsvID** or **UsvID** | VISIBLE STRING65 | Value from the **MSVCB** or **USVCB** |
| **OptFlds** | [a] | Optional fields to be included in the SV message |
| **DatSet** | ObjectReference | Value from the **MSVCB** or **USVCB** |
| **Sample** [1..n] | | |
| **Value** | (*) | (*) The value of the member of the instance of the **DATA-SET**. Type of the common data classes is **SAV** (sampled analogue value) as defined in IEC 61850-7-3 |
| **SmpCnt** | INT16U | Sample counter |
| **RefrTm** | EntryTime | OPTIONAL; time of refresh activity |
| **ConfRev** | INT32U | Configuration revision number from the instance of **MSVCB** or **USVCB** |
| **SmpSynch** | BOOLEAN | OPTIONAL; samples are synchronized by clock signals |
| **SmpRate** | INT16U | OPTIONAL; sample rate from the instance of **MSVCB** or **USVCB** |

[a] The type and value of this parameter shall be derived from the attribute **OptFlds** of the respective **USVCB** or **MSVCB**.

Figure 3.19: Sampled value format (IEC 61850-7-2, 2004)

The model for transmitting sampled values is covered in IEC 61850-7-2. The model applies to the exchange of values of a data set. The data contained within the data set is of the common data class SAV (IEC 61850-7-3, 2003) and shown in Fig. 3.20.



| **Quality Type Definition** | | | |
|---|---|---|---|
| **Attribute Name** | **Attribute Type** | **Value/Value Range** | **M/O/C** |
| | PACKED LIST | | |
| validity | CODED ENUM | good \| invalid \| reserved \| questionable | M |
| detailQual | PACKED LIST | | M |
| overflow | BOOLEAN | | M |
| outOfRange | BOOLEAN | | M |
| badReference | BOOLEAN | | M |
| oscillatory | BOOLEAN | | M |
| failure | BOOLEAN | | M |
| oldData | BOOLEAN | | M |
| inconsistent | BOOLEAN | | M |
| inaccurate | BOOLEAN | | M |
| source | CODED ENUM | process \| substituted<br>DEFAULT process | M |
| test | BOOLEAN | DEFAULT FALSE | M |
| operatorBlocked | BOOLEAN | DEFAULT FALSE | M |

Figure 3.20: Sampled value model (IEC 61850-7-3, 2004)

#### 3.5.2.3.1 Sampled Value Messaging

Sampled Value (SV) messages are related to measurement distribution as stated in the IEC 61850 standard. They are transferred between the bay and process levels.

The transmission of sampled values is time constraints. The IEC 61850 model provides transmission of sampled values in an organized and time controlled manner so that the combined jitter of sampling and transmission is minimized to a degree that an explicit allocation of the samples, times, and sequence is provided. Figure 3.21: Sampled message transmission timevalues.

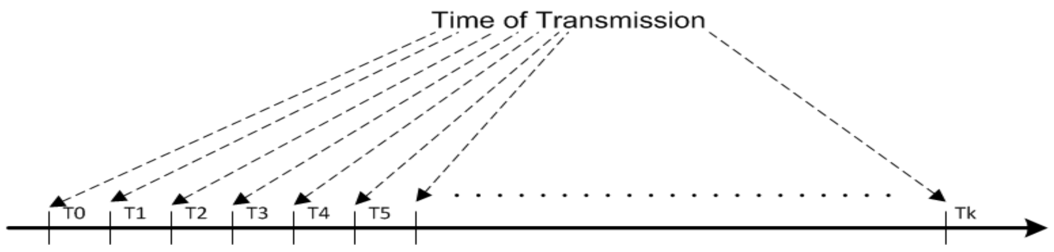


Figure 3.21: Sampled message transmission time

SV messages are time critical messages that are communicated periodically between two or more devices in a chronological manner as shown in Fig. 3.21. These messages can be sent as multicast to several receivers. Fig. 3.22 below shows the Sampled Values Control classes and the services that should be supported by a device publishing or a device subscribing to sampled values.

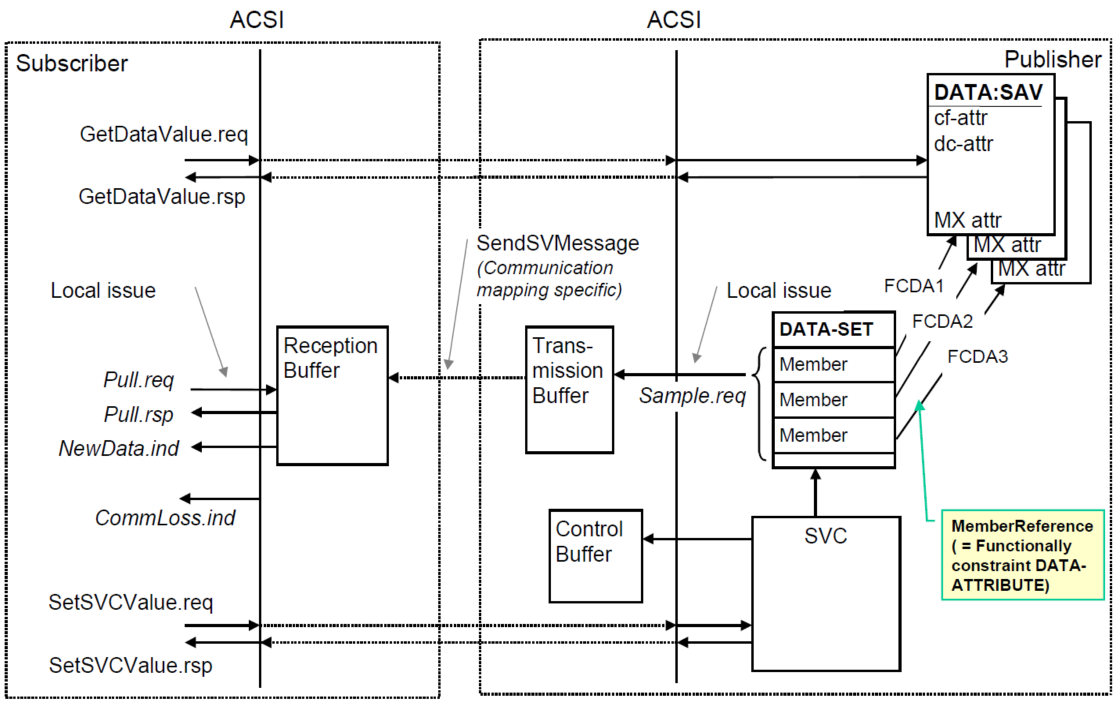
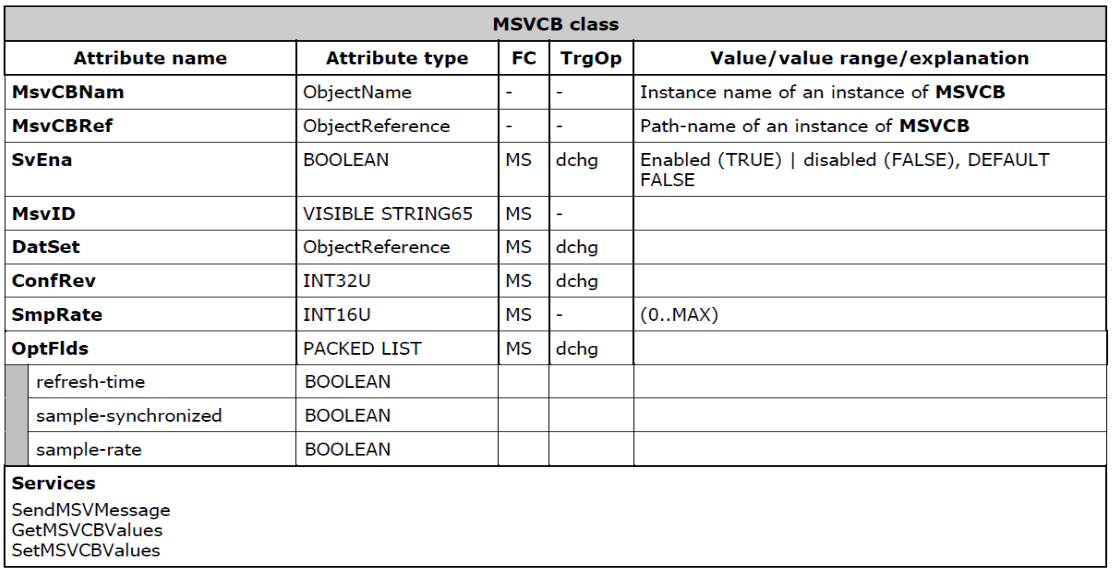


Figure 3.22: Sampled Value Control classes and services (IEC 61850-7-2, 2004)

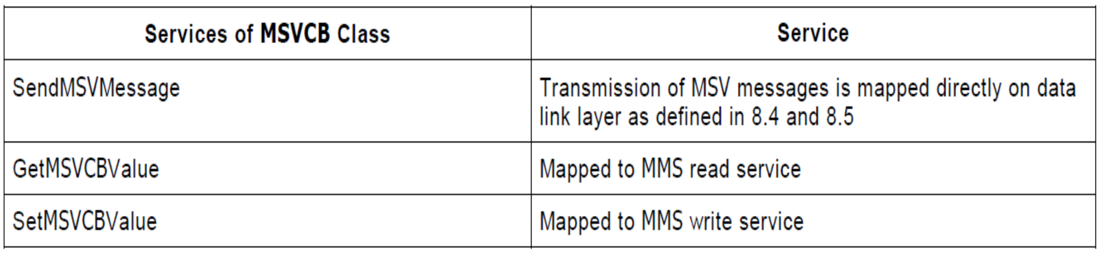
The Sample Value information exchange is based on a publisher/subscriber mechanism. The publisher writes the values in a local buffer at the sending side and the subscriber reads the values from a local buffer at the receiving side. The sampled value count is added as the time stamp, so that the subscriber can verify the timeliness of the values.

The Sampled Value Control (SVC) block shown in Fig. 3.23 below and the Sampled Value services as detailed in Fig. 3.24 are used by the publisher and the subscriber to control the communication process.



| **MSVCB class** | | | | |
|---|---|---|---|---|
| **Attribute name** | **Attribute type** | **FC** | **TrgOp** | **Value/value range/explanation** |
| **MsvCBNam** | ObjectName | - | - | Instance name of an instance of **MSVCB** |
| **MsvCBRef** | ObjectReference | - | - | Path-name of an instance of **MSVCB** |
| **SvEna** | BOOLEAN | MS | dchg | Enabled (TRUE) \| disabled (FALSE), DEFAULT FALSE |
| **MsvID** | VISIBLE STRING65 | MS | - | |
| **DatSet** | ObjectReference | MS | dchg | |
| **ConfRev** | INT32U | MS | dchg | |
| **SmpRate** | INT16U | MS | - | (0..MAX) |
| **OptFlds** | PACKED LIST | MS | dchg | |
| refresh-time | BOOLEAN | | | |
| sample-synchronized | BOOLEAN | | | |
| sample-rate | BOOLEAN | | | |
| **Services**<br>SendMSVMessage<br>GetMSVCBValues<br>SetMSVCBValues | | | | |

Figure 3.23: Multicast SV Control Block Model (IEC 61850-7-2, 2004)



| Services of **MSVCB** Class | Service |
|---|---|
| SendMSVMessage | Transmission of MSV messages is mapped directly on data link layer as defined in 8.4 and 8.5 |
| GetMSVCBValue | Mapped to MMS read service |
| SetMSVCBValue | Mapped to MMS write service |

Figure 3.24: Multicast SV services (IEC 61850-7-2, 2004)

#### 3.5.2.3.2 Sampled value frame format

To provide a better understanding of sampled value messages an examination into the sampled value message structure is conducted in this section. The IEC 8802-3 sampled value frame structure shown in Fig. 3.25 below and the various fields that comprise this SV structure are discussed.

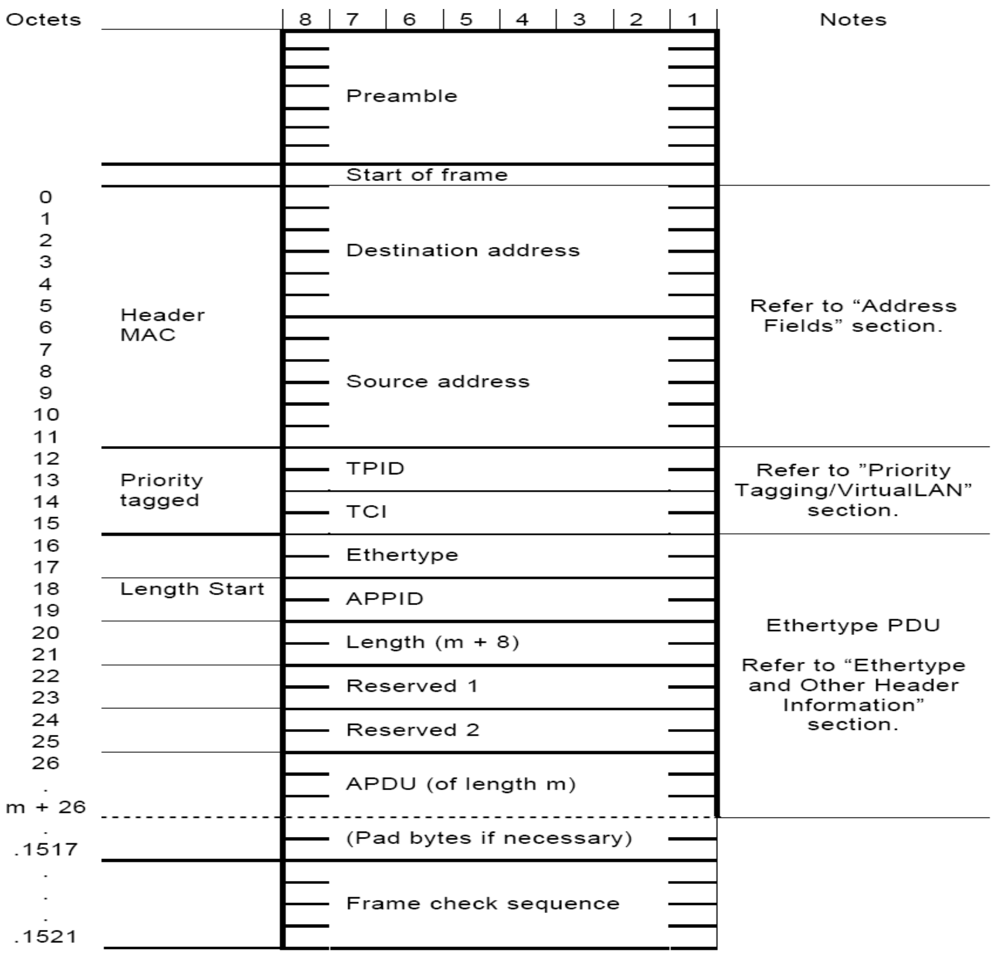


Figure 3.25: ISO/IEC 8802-3 frame format (IEC 61850-9-2, 2004)

The ISO/IEC 8802-3 frame format shown in Fig. 3.25 consists of a Preamble, which consists of seven octet streams of alternating 1's and 0's that allows the Ethernet card to synchronize. The preamble is followed by the Start Frame Delimiter which is a sequence of alternating bits (10101011) (Apostolov, 2006), indicating the start of a frame. This is followed by the Media Access Control (MAC) source and destination addresses, the Virtual Local Area Network (VLAN) Identifier, the Application Protocol Identifier and the Application Protocol Data Unit (APDU). The various fields that collectively form the ISO/IEC 8802-3 frame are described in detail in the following sections.

#### 3.5.2.3.2.1 Media Access Control

Media Access Control (MAC) addresses are unique address assigned to network interface cards for communications on the physical network. The MAC address is typically assigned by the manufacturer of a Network Interface Card (NIC) and is shown in Fig. 3.26.

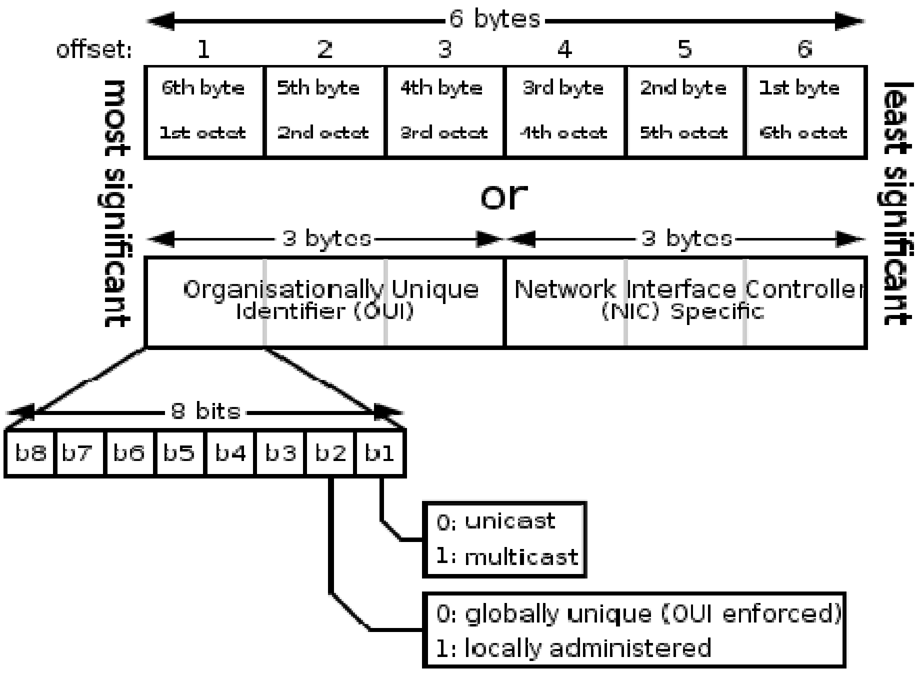
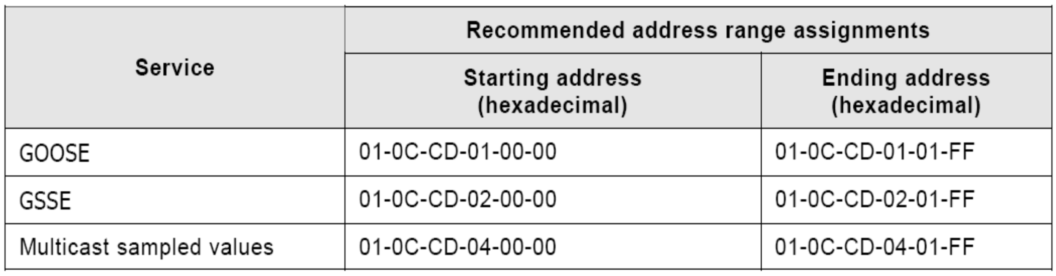


Figure 3.26: ISO/IEC 8802-3 source MAC address format

The assigned multicast/unicast address assignment should however be within the standard range for sampled value messages (Annex C of IEC 61820-9-2, 2003). The sampled value address range specified in the standard is shown in Fig. 3.27.



| Service | Recommended address range assignments | |
|---|---|---|
| | Starting address (hexadecimal) | Ending address (hexadecimal) |
| GOOSE | 01-0C-CD-01-00-00 | 01-0C-CD-01-01-FF |
| GSSE | 01-0C-CD-02-00-00 | 01-0C-CD-02-01-FF |
| Multicast sampled values | 01-0C-CD-04-00-00 | 01-0C-CD-04-01-FF |

Figure 3.27: Multicast address within IEC 61850 (IEC 61850-9-2, 2004)

The first three bytes are assigned by the Institute of Electrical and Electronic Engineers (IEEE) as 01-0C-CD. The fourth byte defines the publishing mechanism Multicast or unicast sampled values. The last two octets are individual assigned addresses.

#### 3.5.2.3.2.2 Virtual local Area Network (VLAN)

The 802.1Q VLAN tag is a 32-bit field added between the destination/source address and the EtherType fields of the original frame or the IEEE 802.3 frame - allowing priority tagging and protocol identification as shown in Fig. 3.28.

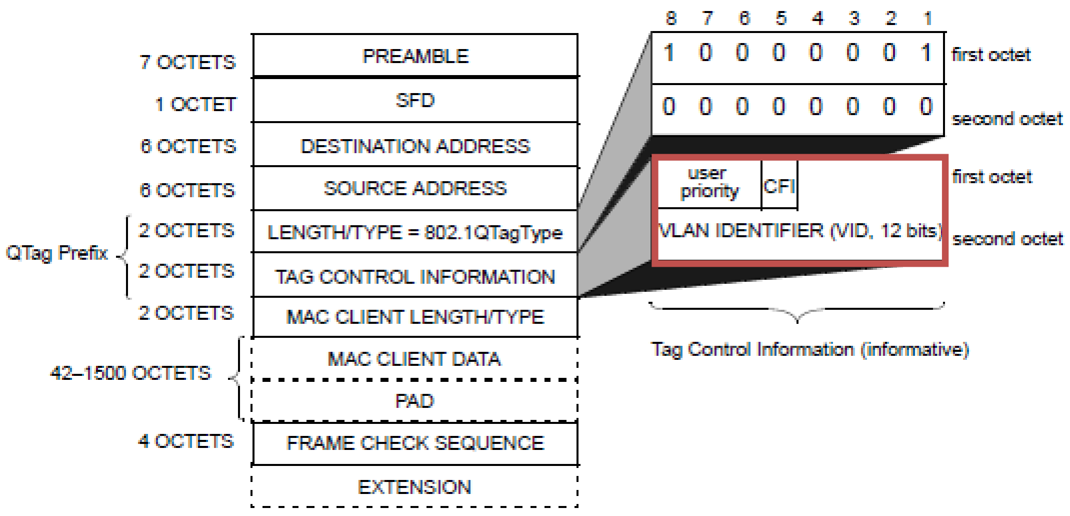


Figure 3.28: Tagged IEEE 802.3 MAC frame format (IEEE802.1Q, 2005)

Two bytes within the VLAN field are used for the Tag Protocol Identifier (TPID), the other two bytes for tag control information (TCI). The TCI field is further divided into User priority, Canonical Format Identifier (CFI), and VLAN identifier (ID) as shown in Fig. 3.29.

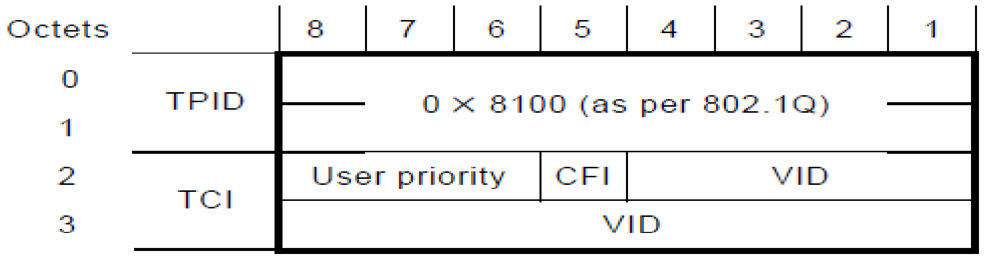


Figure 3.29: Sampled Value VLAN specification in the IEC 61850-9-2

The Tag Protocol Identifier (TPID) is a 16 bit field to identify the frame as an IEEE 802.1Q tagged frame. When set to 0x8100 it identifies the frame as an IEEE 802.1Q frame (IEEE802.1Q, 2005).

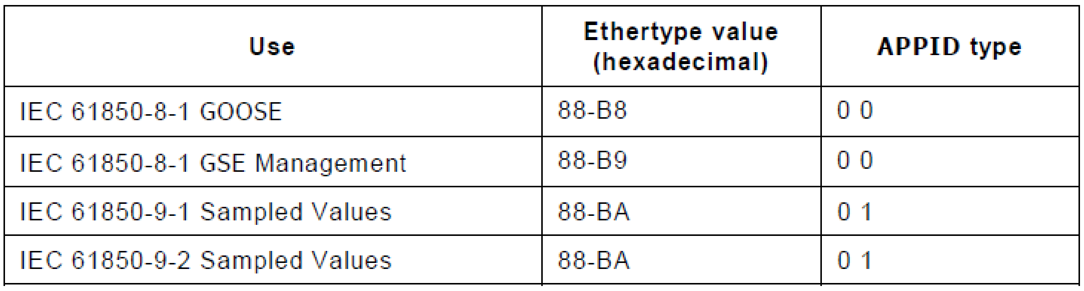
Tag Control Identifier (TCI) represents the user assigned priority. TCI is generally used to separate time critical and high priority bus traffic for protection-relevant applications from low priority busload. User priority is a 3-bit field indicating the frame priority level (IEEE802.1Q, 2005), with 7 representing the highest priority and 1 representing the lowest priority (The default value for SV = 4).

Canonical Format Indicator (CFI) is a 1-bit field. CFI bit specifies the method utilised for the ordering of bits within a frame. CFI is used for compatibility between Ethernet and Token Ring technologies. If the value of this field is 1, the MAC address is in non-canonical format. If the value is 0, the MAC address is in canonical format, meaning the tag does not contain an Embedded Routing Information Field (IEEE 802.1Q, 2005)

The VLAN Identifier (VID) is a 12-bit field specifying the VLAN to which the frame belongs. The hexadecimal values of 0x000 and 0xFFF are reserved, all other values may be used as VLAN identifiers, allowing up to 4,094 VLANs.

#### 3.5.2.3.2.3 Ethertype

Ethertype is a two-octet field in an Ethernet frame. It is used to indicate which protocol is encapsulated in the Payload of an Ethernet Frame. Ethertype numbering generally starts from 0x0800. Ethertypes based on ISO/IEC 8802-3 MAC address sub-layer is registered with the IEEE.  The IEC 61850 registered Ethertypes are listed in Fig. 3.30.



| Use | Ethertype value (hexadecimal) | APPID type |
|---|---|---|
| IEC 61850-8-1 GOOSE | 88-B8 | 0 0 |
| IEC 61850-8-1 GSE Management | 88-B9 | 0 0 |
| IEC 61850-9-1 Sampled Values | 88-BA | 0 1 |
| IEC 61850-9-2 Sampled Values | 88-BA | 0 1 |

Figure 3.30: IEC 61850 Ether-types based on ISP/IEC 8802-3 (IEC 61850-9-2)

#### 3.5.2.3.2.4 Application identifier (APPID)

APPID is a two-octet field in an Ethernet frame that identifies the network information propagating on a particular network, this is set to 0x4000 for SV messages.

#### 3.5.2.3.2.5 SV Protocol data unit

An example of the SV Protocol Data Unit (PDU) can be found in the IEC 61850-9-2 standard and is shown in Fig. 3.31. The SV PDU consists of one or more Application Service Data Units (ASDUs), a Sampled Value Identifier (svID), Sample Count (smpCnt) and a dataset containing the three-phase and neutral value current and voltage measurements as shown in Fig. 3.31.

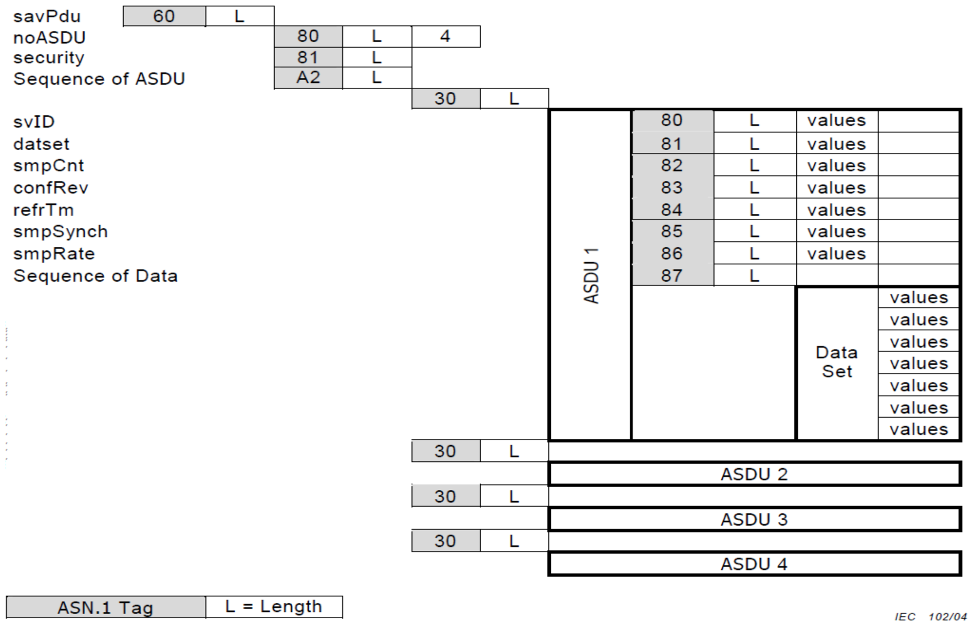
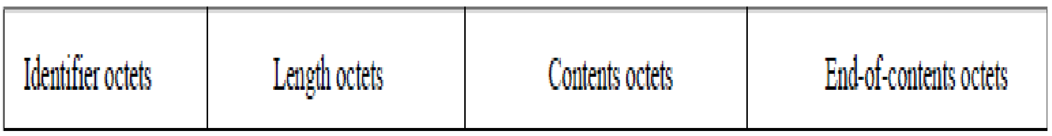


Figure 3.31: Example of a Sampled Value Protocol Data Unit (IEC 61850-9-2, 2004)

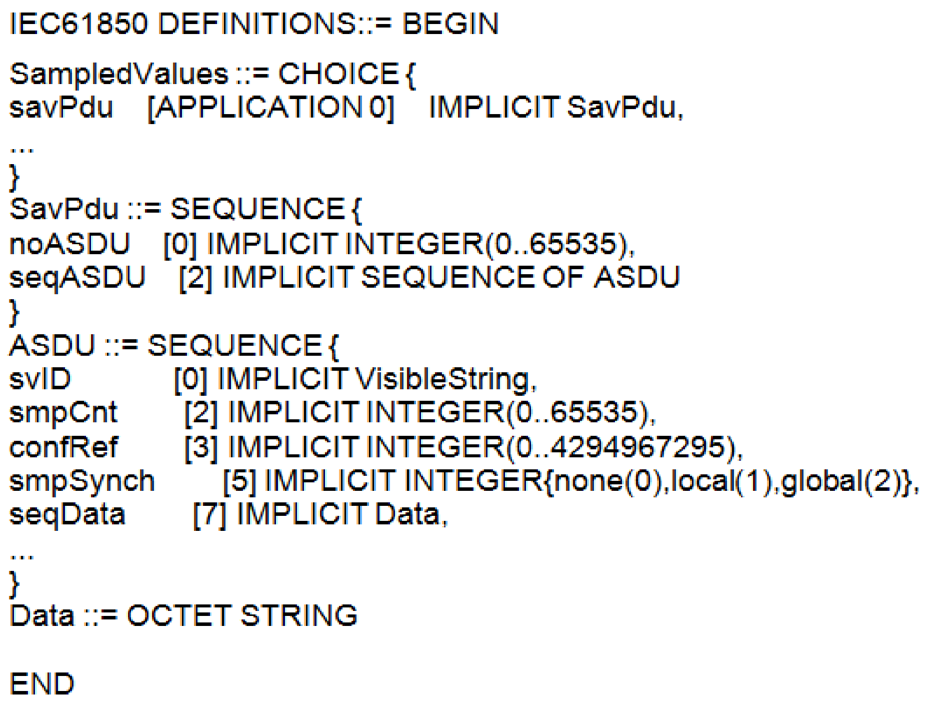
In Fig. 3.31 the Abstract Syntax Notation One (ASN.1) is generally used as padding for the various fields within the Protocol Data Unit (PDU). ASN.1 is a set of rules for representing OSI objects as strings of 1's and 0's. This allows one to define a variety of data types, such as integers, sets, bit strings, and sequences. The main encoding principles for SV messages as discussed in 61850-9-2, is an adapted version of ANS.1 tags for MMS. An example of the tag structure is shown in Fig. 3.32.



| Identifier octets | Length octets | Contents octets | End-of-contents octets |
|---|---|---|---|

Figure 3.32: Format of ASN.1 tags (Falk, H. and Burns, 1996)

The TAG byte has several sub-fields designated. Bits 7 and 6 define the tag type, bit 5 defines whether the tag is primitive or constructed, and Bits 0-4 represents the tag value. The LENGTH has a simple and extended form. If the length of the data is less than 128 bytes, the LENGTH is equal to the number of bytes. However, if the length of the data is greater than or equal to 128 bytes, the LENGTH is encoded as several bytes.  The DATA portion of the production contains the actual information to be exchanged. The content is described by the TAG, discussed above, and can consist of primitive types and constructed types. An example of the SV PDU is shown in Fig. 3.33 below.



```
IEC61850 DEFINITIONS::= BEGIN

SampledValues ::= CHOICE {
savPdu    [APPLICATION 0]    IMPLICIT SavPdu,
...
}
SavPdu ::= SEQUENCE {
noASDU    [0] IMPLICIT INTEGER(0..65535),
seqASDU    [2] IMPLICIT SEQUENCE OF ASDU
}
ASDU ::= SEQUENCE {
svID            [0] IMPLICIT VisibleString,
smpCnt          [2] IMPLICIT INTEGER(0..65535),
confRef         [3] IMPLICIT INTEGER(0..4294967295),
smpSynch            [5] IMPLICIT INTEGER{none(0),local(1),global(2)},
seqData         [7] IMPLICIT Data,
...
}
Data ::= OCTET STRING

END
```

Figure 3.33: Example of Sampled Value's ANS.1 tags

The example shows the order in which each element must appear within the SV PDU according to IEC 61850-9-2 and the implementation guideline IEC 61850-9-2 LE.

This section highlighted the future proofing mechanisms incorporated into the IEC 61850 standard. The specification of the tools and processes for easing the engineering of substation automation systems are presented in section 3.6 below.

## 3.6. IEC 61850 System Engineering

In traditional substation systems, IEDs are typically configured using the vendor's native IED Configurator. IED Configurators are manufacturer-specific tools that can be used to import or export files to a specific device, such as IED settings and generally the device's configuration file. IED configuration tools are supplied by the IED vendors.

The IEC 61850 standard differentiates between IED configuration tools and system engineering/configuration tools. The role of the IED configuration tools has not changed and is primarily related to everything that is specific for the IED. System engineering tools are defined in the IEC 61850 standard as tools that "shall be used to configure everything that is needed to define the interaction between the different IEDs to fulfil the system functionality" (IEC 61850-6, 2004).

The main drivers with IEC 61850 system engineering are (Brunner et al., 2011):

- To reduce engineering time of substation automation systems
- To provide power utilities independence from vendors by introducing the possibility of vendor-independent engineering tools.

The IEC 61850-6 achieves its objectives by stating that vendor IEDs compliant with the IEC 61850 series shall, “be accompanied either by a Substation Configuration description Language (SCL) file describing its capabilities, or by a tool, which can generate this file from the IED”. This file is referred to as the IED Capability Description (ICD) file. The ICD file describes the IED’s capabilities i.e. the functions and data models supported within the IED. ICD files are used for data exchange from the IED configuration tools to the system configuration (Apostolov, 2010).

The system configuration is assembled based on a System Specification Description (SSD) and the IED Capability Description files (Brunner et al., 2011). The system configuration tool is used to configure the entire substation automation system. The result is the Substation Configuration Description (SCD) file that is then used for the IED configuration tools to create the configuration download.

IEC 61850-6, 2004 specifies a configuration file format for IEC 61850 substation automation systems. The configuration file format describes communication parameters, IED setting, primary plant structures and the relations between these functions. The specified configuration file format is often referred to as Substation Configuration description Language (SCL).

The SCL is used in the IEC 61850 standard to model substation automation systems in Extensible Markup Language (XML) version 1.0, according to IEC 61850-5 and IEC 61850-7.

The model used in the SCL is based on the Unified Modelling Language
(UML) and was intended to define (IEC 61850-6, 2004):

[1] SAS functional specification
[2] IED capability description
[3] SA system description

The motivation for the development of SCL is to aid/ease the process of SAS engineering, SAS communication engineering and to provide a means of describing

SA engineered systems in a standardised manner. The SCL object model defined in the IEC 61850-6, 2004 is shown in Fig. 3.34 below.

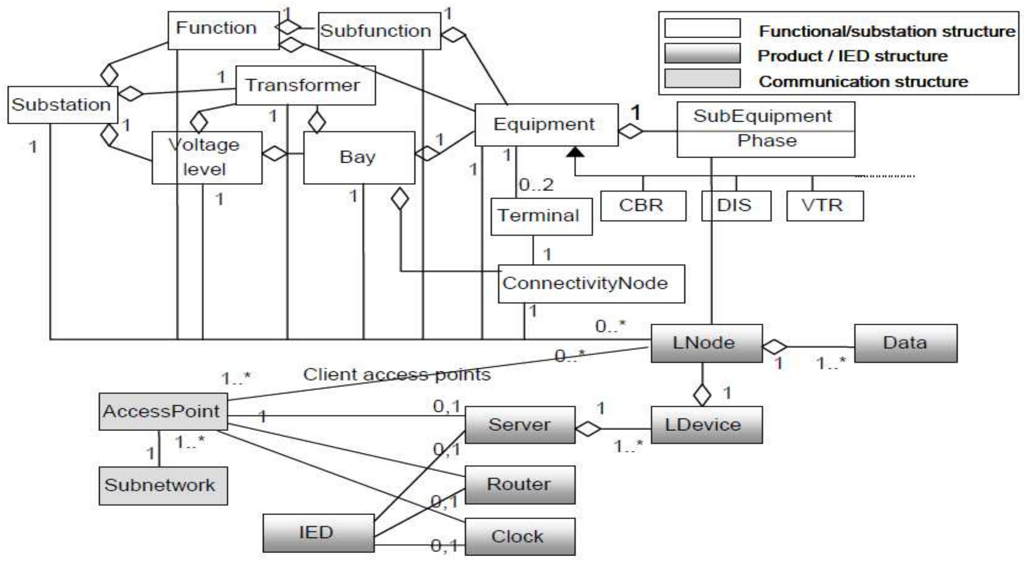


Figure 3.34: SCL Object model (IEC 61850-6, 2004)

The UML SCL object model that is restricted to the object types that are used within a SCL instance file is shown in Fig. 3.34. The object model has three basic parts, namely Substation, Product and Communication that can be described by the four different SCL file types (IEC SCL summary, 2006). The four SCL file types are:

- System Specification Description (SSD) – power system functions
- Substation Configuration Description (SCD) – complete substation
- IED Capability Description (ICD) – the capability of a specific IED
- Configured IED Description(CID) – the configuration of a specific IED

The four SCL files have five sections per file as shown in Table 3.2.

Table 3-2: SCL File Types and Sections (IEC SCL summary, 2006)

| **Section** | **File Type** | | | |
|---|---|---|---|---|
| | **SSD** | **ICD** | **SCD** | **CID** |
| | **Substation One-Line and Functions** | **IED Template** | **Complete Substation** | **Particular IED Configuration** |
| **Header** | Yes | Yes | Yes | Yes |
| **Substation** | Yes | Optional | Yes | Optional |
| **Communications** | Optional | One Instance | Yes | One Instance |
| **IED** | Optional | Yes, values optional | Multiple | Yes, including values |
| **Data Type Templates** | As needed | As needed | Multiple | As needed |

Through the four configuration files and the file types catalogued in Table 3.2, the SCL specifies a hierarchy of configuration files allowing various system engineering levels. IEC 61850 SCL also allows users the ability to compile engineering files (online and offline), as well as provide a means for IED configuration to be passed between Engineering Application systems. Fig. 3.35 shows an example of an IEC 61850 configuration system.

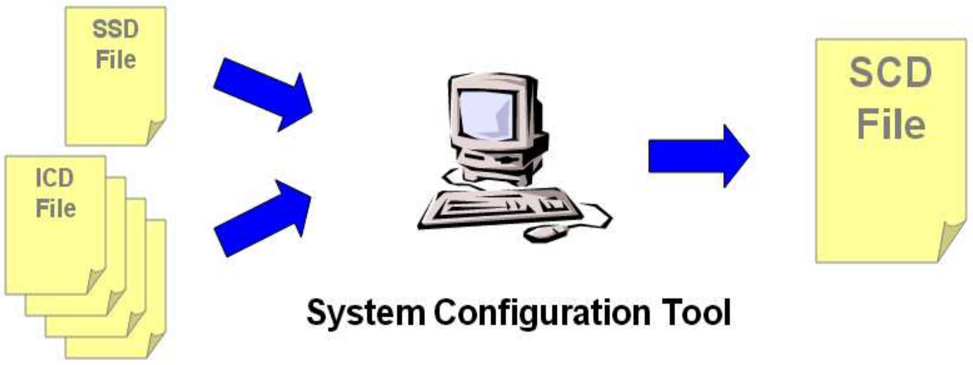


Figure 3.35: IEC 61850 System configuration (IEC SCL summary, 2006)

This section discussed the role of standardized file formats and system engineering tools as a means to optimize and enhance the design, scalability and maintainability of IEC 61850-based substation automation systems.

The next section, section 3.7 examines network topologies as one of the critical items to be considered when designing IEC 61850, Ethernet-based substation automation systems. The network integrity and hence the topology deployed takes additional importance when dealing with the process level in IEC 61850 - this is due to the additional network traffic loading imposed by the time synchronization systems required by the Merging Unit.

## 3.7. Substation automation network topology

The networking requirements for substation automation systems to support the protocols in Fig. 3.10 are also specified in the IEC 61850 standard. To meet the networking requirements as specified in IEC 61850-3, 2004 and to ensure overall system integrity, the network topology has to be carefully designed to minimize system down-time.

The IEC 61850 standard does not define a network topology. The network topology has to be decided upon by the engineering team to meet the performance and reliability criteria specified by the IEC 61850 standard - taking into account the operating conditions and high levels of Electromagnetic Interference (EMI) that characterize the substation environment (Chen, 2008).

A star network topology is often shown when illustrating the implementation of IEC 61850-based systems. However, various other topologies such as ring and dual networks can be deployed in substation automation systems to provide redundancy for critical systems. The topology deployment takes additional importance when dealing with measurement distribution in substation automation systems, due to the additional network loading. Fig. 3.36 shows basic communication networks.

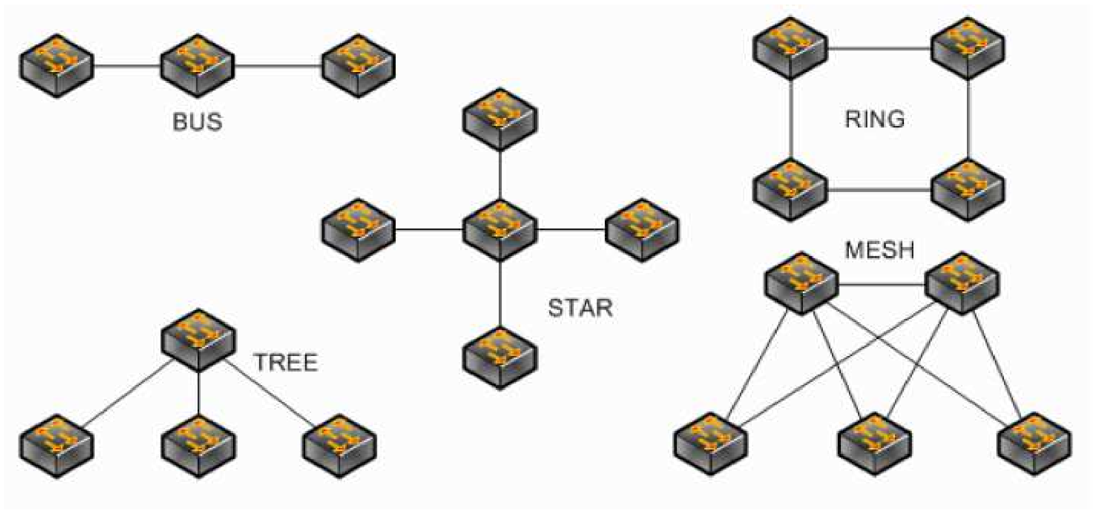
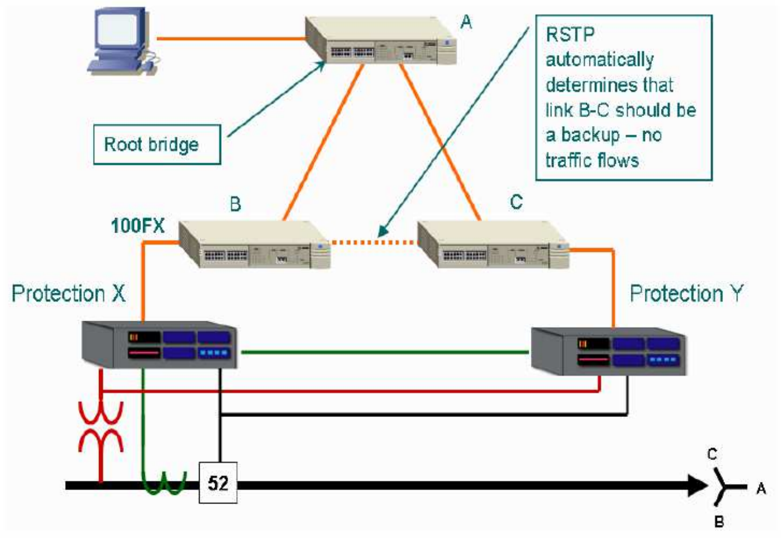


Figure 3.36: Basic network architectures (Midence, 2009)

It is only the interfaces (see Fig. 3.11) that are specified in the IEC 61850 standard, but it should be noted that any of the network topologies shown in Fig. 3.36 can be used for the implementation of IEC 61850 substation automation systems. The trade-off between building a simple star network and redundant networks is often the additional cost and complexity associated with building redundant network.

Ring and dual ring network topologies need managed data network switches and management protocols such as Spanning Tree (STP), Rapid Spanning Tree (RSTP) or Multi-Spanning Tree (MSTP). The Ring network topology shown in Fig. 3.37 is in most cases the preferred choice for substation automation systems due to its flexibility, lower overheads, and ease of installation (Nordell, 2008).



A
RSTP automatically determines that link B-C should be a backup – no traffic flows
Root bridge
B
C
100FX
Protection X
Protection Y
52
C
A
B

Figure 3.37: Illustration of ring network topology using RSTP (Midence, 2009)

A ring topology can be implemented using spanning tree protocol (STP) as per the IEEE 802.1D specification or rapid spanning tree protocol (RSTP) as per IEEE 802.1W specification.

The main difference between STP and RSTP is that in RSTP the STP ports, where the listening, blocking or discarding of packets all learn the MAC addresses of the of the adjacent switch, making recovery time on RTSP slightly faster than on STP - where the switch will only start to respond when it receives a message from the root bridge. Generally the network recovery time when using STP or RSTP is between 1 and 60 seconds (Chen, 2008).

A delay of 60 seconds can cause serious problems for complex substation automation systems and the root bridge is typically a single point of failure in ring networks. Fig. 3.38 shows dual redundant networks with a redundant link to the central control room and backup for the root bridge. The IED devices shown in Fig. 3.38 could be protection devices, merging units or any substation automation device.

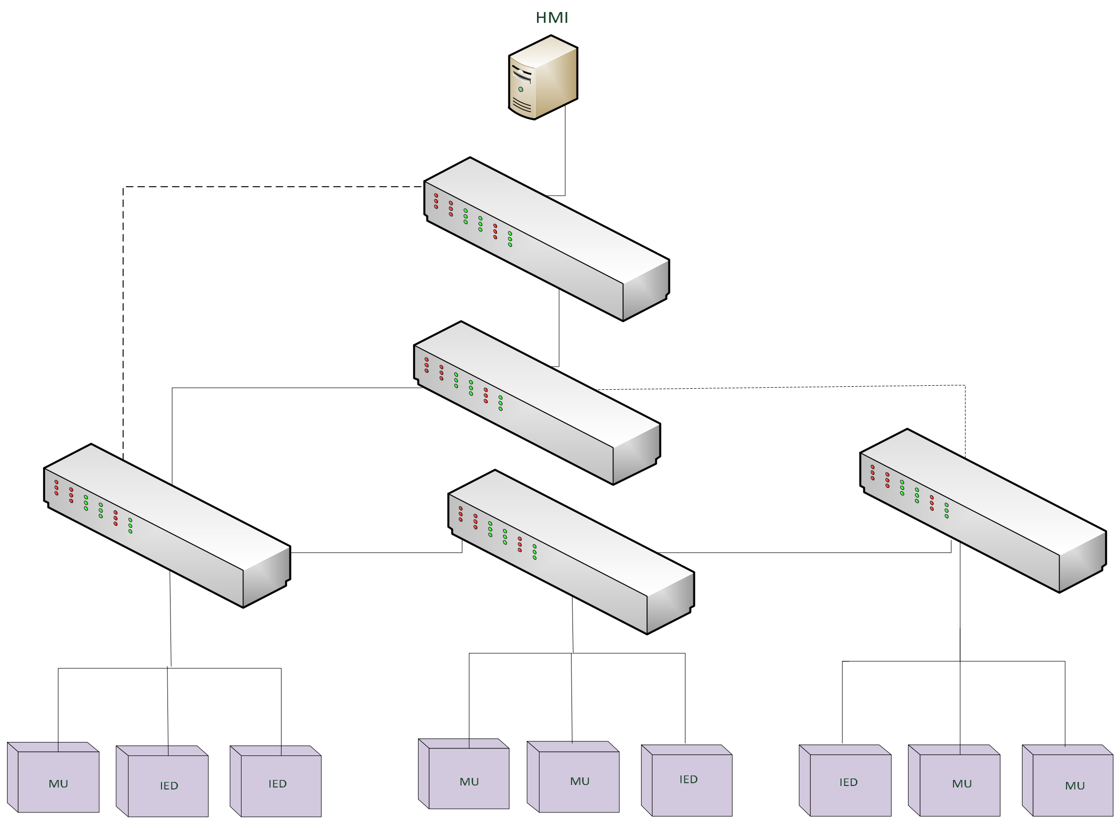


Figure 3.38: Dual homing in a substation automation system (Station bus and Process bus sharing the same LAN).

In Fig. 3.38 a failure of any connection or switch in the ring will not cause a disruption in network communication, the redundant ring immediately restores network communication, and the system runs without interruption. This topology is often referred to as one of high availability due to the added redundancy in the topology. A closer look at the topology shows that the topology is comprised of multiple ring subsets.

There are various other topologies that can be explored, the trade-off is often the fact that very large rings can be difficult and expensive to manage or install, especially over great distances. In addition, network recovery times increase with the size of the ring and it is often more effective to establish a number of smaller rings based on location and system architecture. These are all key factors that have to be considered when designing the network architecture of substation automation systems.

## 3.8. Conclusion

In this chapter an overview of the IEC 61850 standard is covered. The various aspects that have been investigated to facilitate a better understanding of IEC 61850 substation automation systems have been discussed.

The mechanisms and processes in the IEC 61850 standard have all been detailed in support of the main concepts driving the standard, namely:

a) The provision of a virtualized substation automation framework for all substation automation devices.
b) The provision of a future-proof substation automation framework that is flexible, scalable, maintainable, and provides interoperability between substation automation devices.
c) The specification of the tools and processes for easing the engineering of substation automation systems.

It is felt that the knowledge-base required has been developed through this chapter and is sufficient to support the development of the proposed research project, i.e. the development of a virtual sensor node based on the IEC 61850 standard.

The chapter to follow details the project outline and context.

# CHAPTER 4

# Project outline and context

## 4.1. Introduction

The technology introduced by the IEC 61850 standard has presented a paradigm shift in protection and control systems within the electricity delivery industry. IEC 61850 compliant devices are widely available and are slowly being deployed in industry. These devices however are mainly for station level applications at present. The knowledge base in the academic environment and the devices to accommodate the process bus paradigm are still limited (even in the industrial environment). The limited development of devices to accommodate the process-bus paradigm prompted the UCA user group to publish an implementation guideline document "IEC 61850-9-2LE" in 2004 **(**Schaub and Kenwrick, 2009).

The IEC61850-9-2 standard describes the framework for the general design of merging units (Weiss et al., 2011). This document, however allows developers significant leeway which has led to a variety of implementations, with consequent problems with regards to interoperability between different vendor's devices. An implementation guideline "IEC 61850-9-2LE" was published by the UCA International user Group, to facilitate the uniform implementation and interoperability of devices utilizing sampled values. The IEC 61850-9-2LE implementation guideline document specifies a fixed dataset containing four current and four voltage measurements (IEC 61850-9-2LE page 7). The IEC 61850-9-2LE document also specifies fixed sampling rates of 80 samples per cycle for protection applications and 256 samples per cycle for metering applications (IEC 61850-9-2LE page 9). The concepts discussed in the implementation guideline document are further elaborated later on in this chapter.

Since the publication of the implementation guideline document, there has been several pilot projects implemented that support the proposed IEC 61850-9-2 interface convention (Schaub and Kenwrick, 2009). However, there are still only a few substation automation devices available on the market which provides IEC 61850-9-2LE interface connectivity.

The few devices that are available on the market are vendor based. There is limited knowledge in the public domain about the devices that can be used to simulate the process-bus concept of sampled value messaging. This chapter explores the capabilities of personal computer-based platforms as a means to simulate sampled value messages.

The chapter is structured into seven sections. This section (4.1) is the introduction. Section 4.2 provides the project context and outlines the hierarchical structure defined in the IEC 61850 standard. In section 4.3 the merging unit as the device demarcated for distribution of measurements between the process and the bay levels in IEC 61850 systems is discussed, and the detailed structure of merging unit devices is outlined, with comments on its functional and temporal behaviour as per the standard. In section 4.4 the research project is described and the delimitation and scope stated. Section 4.5 addresses the issues of development tools and development platforms which were used. Section 4.6 addresses the challenges that needed to be overcome with respect to data networking at the datalink layer level. Section 4.7 concludes on what has been discussed in this chapter and states what will be discussed in the next chapter.

## 4.2. Project context

The IEC 61850 standard specifies the use of a hierarchal architecture for data modelling and for interfacing between substation devices, as depicted in Fig. 4.1 below.

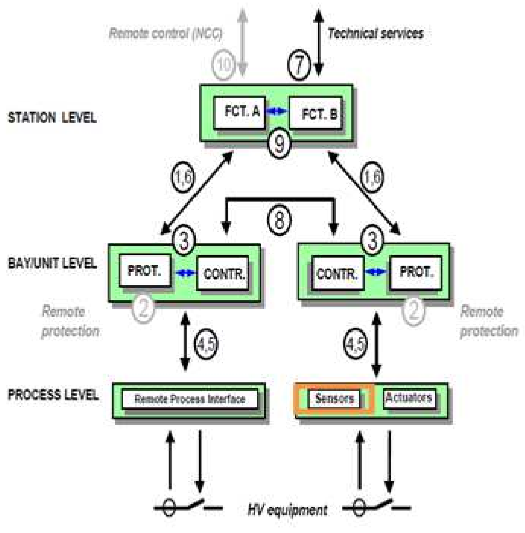


Figure 4.1a: Hierarchical model as specified in the IEC 61850 standard (IEC 61850, 2003)

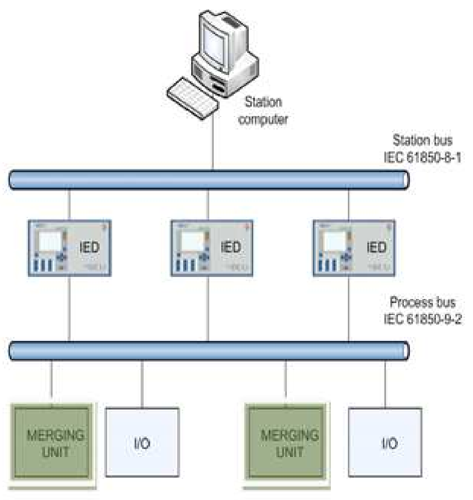


Figure 4.1b: An implementation architecture for IEC 61850 communications (Brunner et al., 2007)

The interfaces defined in the IEC 61850 standard when implemented, manifests as the station and process busses respectively - as shown in Fig. 4.1b. Interfaces four (IF4) and five (IF5) in Fig. 4.1a are defined in the IEC 61850-5 standard for interfacing the process level to the bay level. IF4: Current Transformer (CT) and Voltage Transformer (VT) instantaneous data exchange (including sampled values) between process and bay level; and IF5: control-data exchange between process and bay level.

Interface five (IF5) in Fig. 4.1a is used for GOOSE messaging carrying control data exchange messages between bay level and process level devices (Sidhu et al., 2008). GOOSE messaging refers to the time-critical messaging, which has been defined for fast peer-to-peer communication between substation automation Intelligent Electronic Devices (IEDs). GOOSE messages are generally used to transfer status information between two or more devices (Sidhu et al., 2008). GOOSE messaging as discussed earlier in section 3.5.2.2 is event-based. The current implementation of the IEC 61850 standard in industry is however not through this interface, i.e. IF5.

Figs. 4.2a and b depicts the current deployment of the IEC 61850 standard in industry (within South Africa and internationally). Notable in Fig. 4.2 is that the primary plant equipment is hardwired to the bay level devices. GOOSE message exchange only occurs at the bay and station levels, through interfaces 1, 3, 6, 8 and 9, as shown in Fig. 4.1a (All interfaces are discussed in chapter 2). The current deployment of the IEC 61850 standard **does not** provide connectivity to primary plant equipment through interfaces 4 and 5. The disadvantage of this deployment architecture is that there are **no raw measurements** distributed from the process level to the bay level.

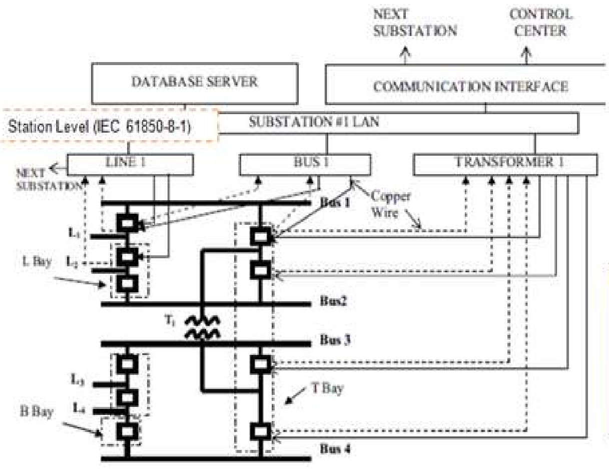


Figure 4.2a: IEC 61850 current deployment architecture (Kezunovic, 2004)

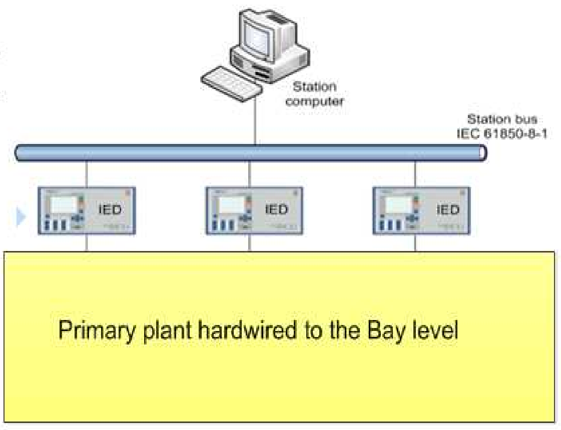

Station computer
Station bus IEC 61850-8-1
IED
IED
IED
Primary plant hardwired to the Bay level

Figure 4.2b: IEC 61850 current deployment communication architecture

[Note: The “Bus” in Fig. 4.2a refers to voltage busbar and not a data network]

[**Note**: The Skaapvlei Wind Farm in the Vredendal (South Africa) is an example of a current implementation within South Africa. The author of the document was part of the team (Buffkins, R., Tshisevhe, H. and Luwaca. E), that designed and commissioned a bus-protection scheme using GOOSE messaging, and the deployed architecture is as depicted in Fig. 4.2b. ]

The Future IEC 61850 deployment architecture is depicted in Fig. 4.3 and IF4 and IF5 is provisioned for. The introduction of the future architecture will introduce a paradigm shift in the industry, as both GOOSE and raw measurements will be available from the process level through interfaces 4 and 5 respectively. **Note**: In Fig. 4.3 the traffic at the process level is segregated with the use of a Virtual Local Area Network (VLAN). The blue lines represent the sampled values VLAN and the brown lines represent the GOOSE VLAN.

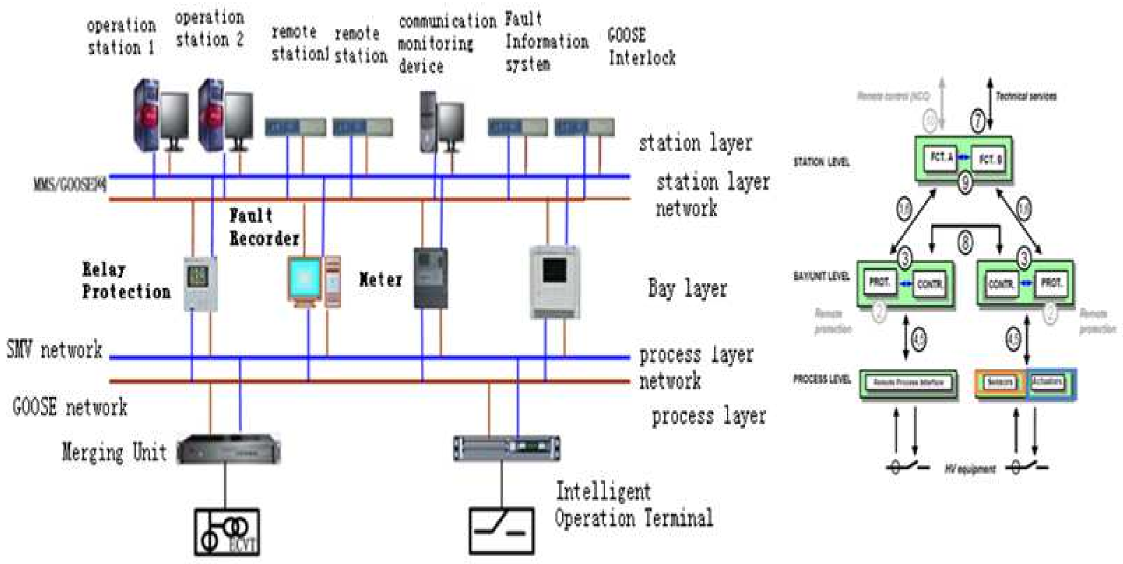


Figure 4.3: IEC 61850 future deployment architecture

Interface five (IF5) is defined in the IEC 61850-5 standard for sampled value measurement distribution from the process level to bay level devices. Sampled value messaging as discussed in chapter three, section 3.5.2.3.1 is related to periodic measurements that are distributed between the process level and the bay level. The current implementation of the IEC 61850 standard in industry does not include the process bus. In Fig. 4.2a, above the primary plant equipment is hardwired to the protection devices. The process bus concept is not applied.

The previous paragraphs attempted to state what the current status quo is – that GOOSE messaging is predominant and that the process bus concept has not even begun to come to fruition. The paragraphs to come outlines examples of how the IEC 61850 process bus concept is to be deployed in future for not only GOOSE messaging, but also inclusive of measurement distribution through sampled value messaging. This scenario is depicted in Fig.4.4 below.

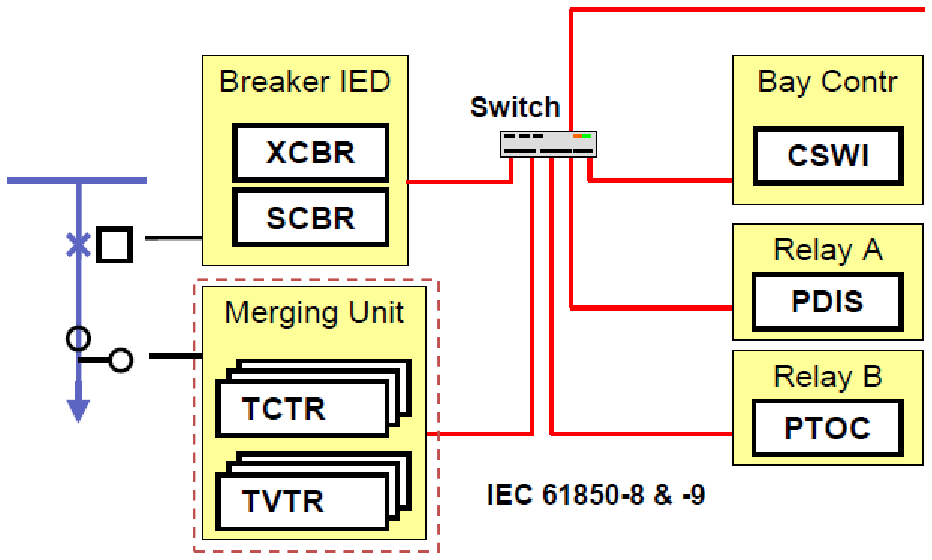


Figure 4.4: Process bus concept (Andersson et al., 2003)

In Fig. 4.4 above, the sensor node demarcated for measurement distribution is highlighted (red dashed line). This sensor node is widely referred to as the Merging Unit. Merging units are specified in the IEC 61850 standard as a means to interface to conventional and non-conventional instrument transformers. The advantages of merging unit deployment are (Kanabar, 2011):

- Raw, unprocessed measurement values: the sampling rate and distribution rate for the data acquisition node defined in the IEC 61850 standard is sufficient for power system devices requiring raw measurements.
- Support for both Unicast and Multicast published measurements.
- Substation automation systems are not only based on substation events, but on actual raw measurements.
- Presents a standardized interface to non-conventional instrument transformers.
- Improving the performance in systems with conventional transformers: the A/D conversion is done close to the source and much of the burden of processing can be removed from the bay level devices.
- Reduced wiring and reduced cabling costs.

Merging Units can also be used in other applications within the substation automation environment. The deployment of IEC 61850-9-2 Merging Units can be used to replace currently deployed stand-alone Phasor Measurement Units (PMUs) (Abdolkhalig et al., 2013) and can provide a solution for power quality measurements, such as harmonic measurements (Tholomier and Chatrefou, 2008).

An example of how measurement distribution can be accomplished using merging units is shown in Fig. 4.5 – in this example, both a sampled value transmitter and receiver are shown, accomplishing through processing of the stream of data what would currently be accomplished through event-driven GOOSE messaging. GOOSE messaging works exceedingly well for the tasks that it has been designed for, but there are other tasks over and above those associated with GOOSE messaging which would be better accomplished through processing of raw data streams.

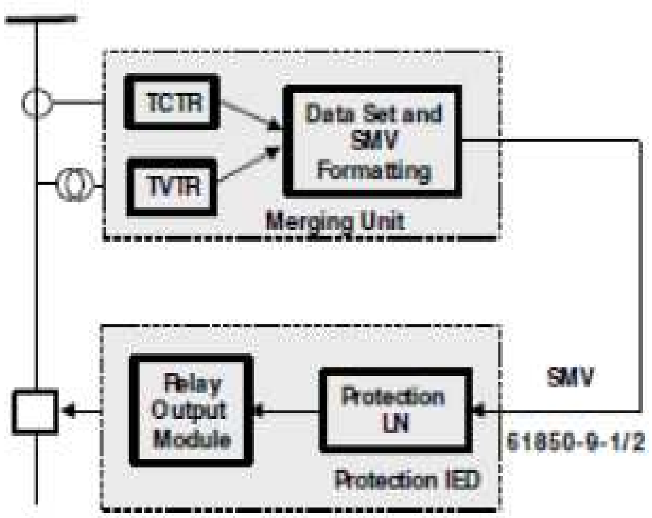


Figure 4.5: IEC 61850-9 -2 Sampled Values based IED (Apostolov et al., 2006)

.

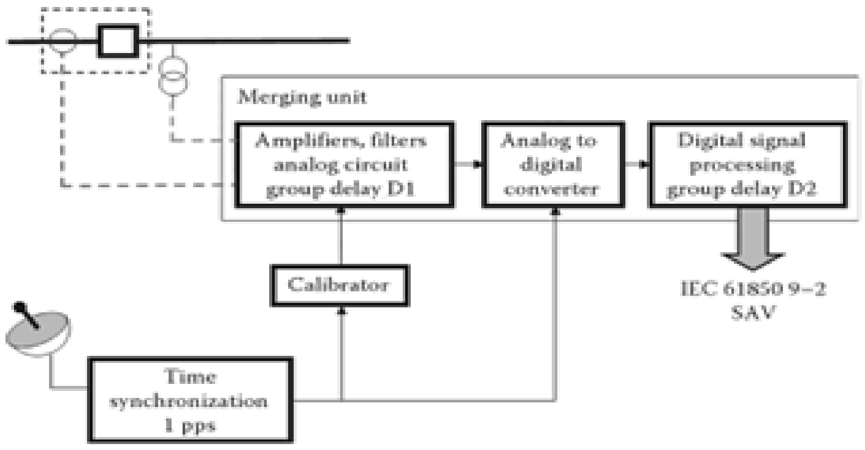
The function of a Merging Unit is to receive analogue measurements from the plant, digitize the analogue values, encode, and publish the raw data in a sampled value format onto the process-bus as shown in Fig. 4.5. The detailed structure of a merging unit is described in the next section.

### 4.3.1. Detail and structure of a merging unit

Figure 4.6 below describes the general structure of a merging unit. For a full implementation, once the three current and voltages together with their associated neutrals from the instrument transformer are coupled into the merging unit - the main components for subsequent processing and packaging consists of four major functional blocks (Weiss et al., 2011), namely:

- Signal conditioning unit and filtering

- The Analogue to Digital Converter (ADC)
- Time synchronization module
- Processing and computing unit (shown as Digital Signal Processing Unit in Fig. 4.6.)


Merging unit
Amplifiers, filters analog circuit group delay D1
Analog to digital converter
Digital signal processing group delay D2
IEC 61850 9-2 SAV
Calibrator
Time synchronization 1 pps
Figure 4.6: Concept of merging unit (Apostolov and Janssen, 2008)

As shown in Fig. 4.6 a Merging Unit receives analogue inputs signals from current and voltage transformers. The received analogue input signals are then amplified, filtered using anti-aliasing filters, and passed onto an analogue to digital conversion system. The received data on the analogue to digital converters is then digitized and accurately time stamped using the Global Positioning System (GPS) or some other precision time synchronization mechanism. Finally the digitized values are then processed, encoded, packaged into an IEEE 802.1Q frame in a manner that is compliant with the IEC 61850-9-2 part of the standard, with the appropriate VLAN tag, and ASN.1 tagging and published onto the process bus.

The four major functional blocks mentioned in the paragraph above are further elaborated upon in the section to follow.

### 4.3.2. Signal conditioning and filtering

Current and voltage transformers are primary-plant devices. They are typically installed outdoors in the high voltage yard, and are typically subjected to significant sources of noise. The 3-phase (3ϴ) current and voltage + 1 neutral current and voltage analogue signals received from these devices generally need to be conditioned and filtered using anti-aliasing filters before being inputted into the Analogue to Digital Converter (ADC). The aim of the anti-aliasing filters is to limit certain frequencies and harmonic interference, so that the downstream processing and computations can take place appropriately. The conditioned and filtered signals are then captured and converted by the analogue-to-digital converters - see Fig. 4.7 below and elaborated upon in the following section.

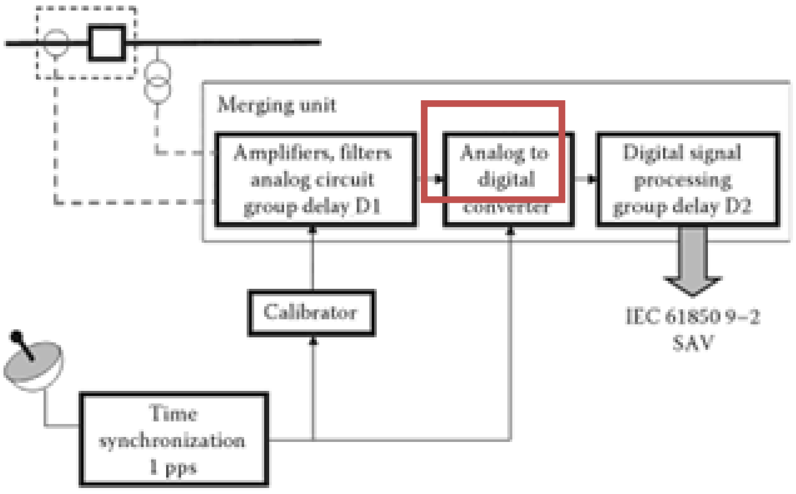
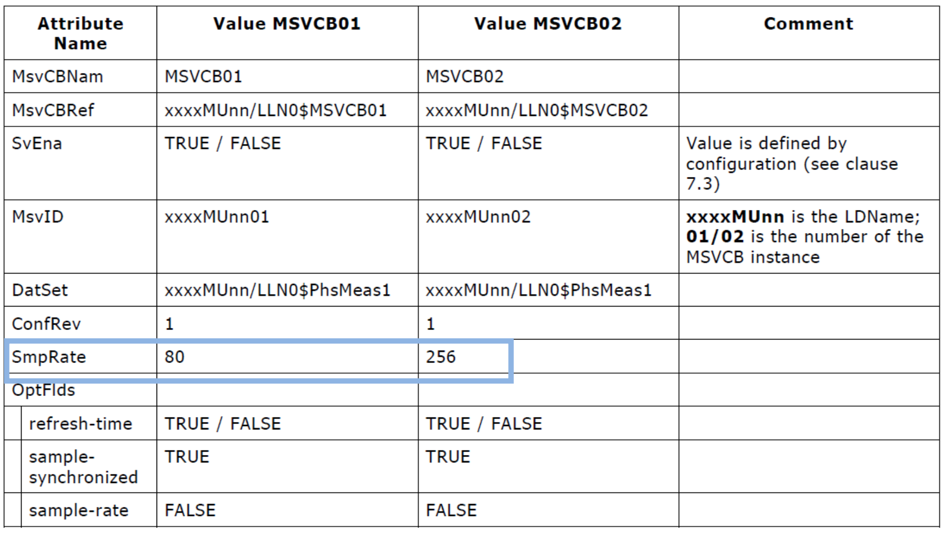

Merging unit
Amplifiers, filters analog circuit group delay D1
Analog to digital converter
Digital signal processing group delay D2
IEC 61850 9-2 SAV
Calibrator
Time synchronization 1 pps

Figure 4.7: Analogue to Digital conversion in IEC 61850 "MU" (Apostolov and Janssen, 2008)

### 4.3.3. Analogue to digital converter

The analogue signal received from the signal conditioning and filtering unit must be converted into digital data that can be processed to obtain magnitudes of the three phase currents and voltages + the neutral currents and voltages. The current release of the IEC 61850-9-2, 2004 does however not define the sampling rate and resolution of the Analogue to Digital Converter (ADC) with any great detail, nor does it define the signal conditioning unit discussed above. This has presented a challenge with the standardized implementation of merging units, and also with ensuring interoperability between developed merging units and bay-level devices. The UCA user grouper has since addressed this issue by releasing the IEC 61850-9-2LE implementation guideline. Appendix C on pages 20 and 21 of the implementation guideline document (IEC 61850-9-2LE) specifies the accuracy requirement. Table 3 on page 9 of the IEC 61850-9-2LE document defines two sampling rates, one for protection applications and a second for metering applications.  A sampling rate of 80 samples/cycle for protection applications, and a sampling rate of 256 samples/cycle metering and quality measurement data is suggested, as depicted in Figure 4.8 (see blue box).



| Attribute Name | Value MSVCB01 | Value MSVCB02 | Comment |
|---|---|---|---|
| MsvCBNam | MSVCB01 | MSVCB02 | |
| MsvCBRef | xxxxMUnn/LLN0$MSVCB01 | xxxxMUnn/LLN0$MSVCB02 | |
| SvEna | TRUE / FALSE | TRUE / FALSE | Value is defined by configuration (see clause 7.3) |
| MsvID | xxxxMUnn01 | xxxxMUnn02 | **xxxxMUnn** is the LDName; **01/02** is the number of the MSVCB instance |
| DatSet | xxxxMUnn/LLN0$PhsMeas1 | xxxxMUnn/LLN0$PhsMeas1 | |
| ConfRev | 1 | 1 | |
| SmpRate | 80 | 256 | |
| OptFlds | | | |
| refresh-time | TRUE / FALSE | TRUE / FALSE | |
| sample-synchronized | TRUE | TRUE | |
| sample-rate | FALSE | FALSE | |

Figure 4.8: Multicast sampled value control block (IEC 61850-9-2)

The IEC 61850-9-2LE implementation guideline (pages 22 through 24) requires a specified resolution for the measurement of the currents and voltages and to accommodate the dynamic range requirements. This measurement resolution, which is required translates to the number of bits on the analogue-to-digital converter for the measurement of a particular rating of current or voltage, i.e. for a particular rated current or voltage, a particular bit-size is required for the ADC (fault current levels also need to be taken into account). A manufacturer would then have to produce a merging unit with a particular rated voltage or current in mind. See Appendices E and F for further information.  The implementation by most vendors that have developed merging units at the present moment does not cater for all practical cases. Table 4.1 below shows the current implementation by two of the vendors that have commercial merging unit devices on the market.

Table 4-1: Analogue to Digital Converter (ADC) resolution from two commercial units

| SPECIFICATION | ALSTOM (MU Agile AMU) | VIZIMAX (AMU) |
|---|---|---|
| ADC resolution | 24 bit | 16 bit |

### 4.3.4. Time Synchronization

The IEC 61850-5 on page 48 specifies that time synchronization used to synchronize the internal clock of an IED in a substation automation system is dependent on the application used. The standard IED synchronization and accuracy requirements are specified in clause 13.7.6.2 as depicted in Table 4.2 below.

Table 4-2: IEC 61850-9-2 Time synchronization specification for MU

| Time performance class | Accuracy (µs) | Reference | | Phase angle (') 50 Hz | Phase angle (') 60 Hz | Fault location (m) |
|---|---|---|---|---|---|---|
| T3 | ± 25 | P1 | | 27 | 32 | 7 500 |
| T4 | ± 4 | P2 | M1 | 4 | 5 | 1 200 |
| T5 | ± 1 | P3 | M2/3 | 1 | 1 | 300 |

The IEC 61850-9-2LE implementation guideline document specifies the time performance class T4 (orange box) on table 4.2 above (IEC 61850-9-2LE). T4 allows for ± 4 µs error, and of the ± 4 µs, IEC 61850-9-2LE document allocates ± 2 µs jitter to the merging unit. The merging unit itself has to be synchronized from a time source with better than ± 1 µs accuracy, as shown in Fig. 4.9

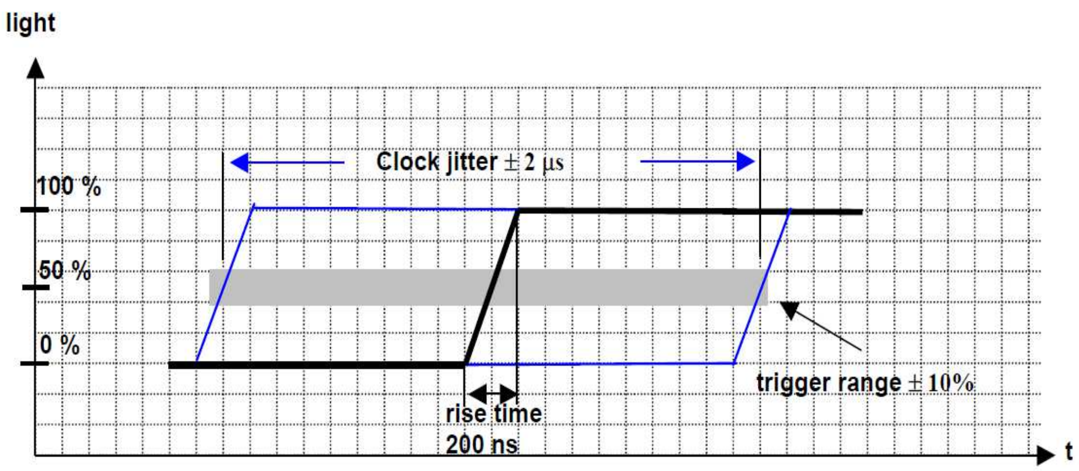


Figure 4.9: Definition of the maximum clock jitter and rise time at the merging unit clock input (IEC 61850-9-2LE)

The IEC 61850-9-2LE document on page 11 recommends the use of a Global Positioning System (GPS) clock for time synchronization. The issue of time synchronization has been an intensive area of research studies of late, as indicated in the literature search in chapter 2; with the IEEE 1588 Precision Time Protocol version 2 (PTPv2) proposed for time synchronization in substation automation systems and specifically for merging units (Ingram et al., 2012).

### 4.3.5. Processing and computing unit

All digitized signals from the merging unit are then processed in the merging unit logical nodes by the process and computing unit highlighted in green in Fig. 4.10 below.

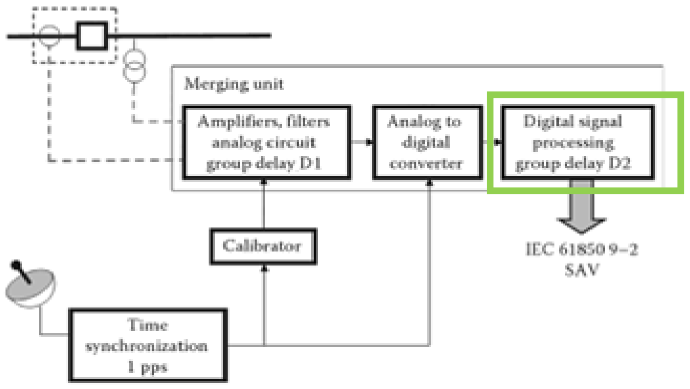
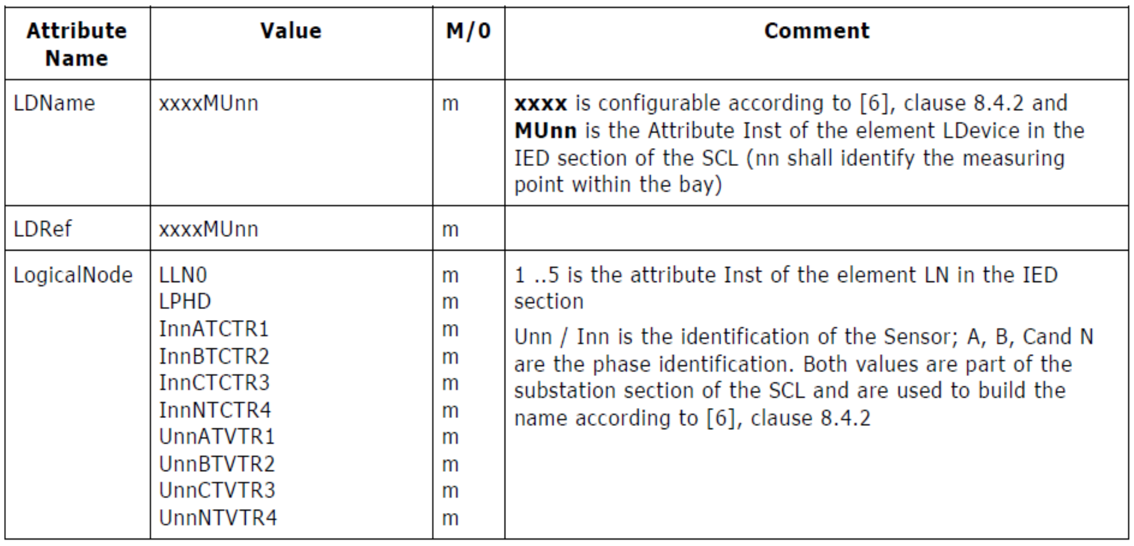

Merging unit
Amplifiers, filters analog circuit group delay D1
Analog to digital converter
Digital signal processing group delay D2
Calibrator
IEC 61850 9-2 SAV
Time synchronization 1 pps

Figure 4.10: Processing and computation, conversion in IEC 61850 “MU” (Apostolov and Janssen, 2008)

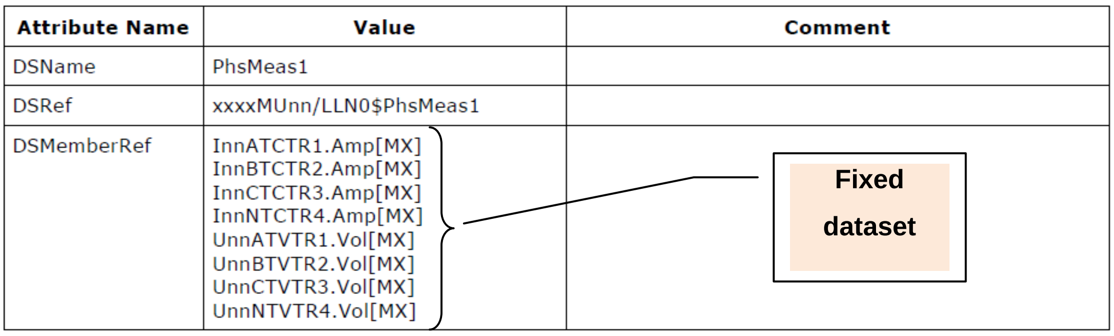
The logical nodes used in the merging unit logical device are defined in the IEC 61850-7-4, 2004 page 61 as TCTR for current transformers and TVTR for voltage transformers on page 62. The IEC 61850-7-4 standard is referenced in the IEC 61850-9-2LE document, and the Merging Unit logical device instance is defined on page 7 of the IEC 61850-9-2LE document as depicted in Fig. 4.11 below.



| Attribute Name | Value | M/O | Comment |
|---|---|---|---|
| LDName | xxxxMUnn | m | **xxxx** is configurable according to [6], clause 8.4.2 and **MUnn** is the Attribute Inst of the element LDevice in the IED section of the SCL (nn shall identify the measuring point within the bay) |
| LDRef | xxxxMUnn | m | |
| LogicalNode | LLN0 | m | 1 ..5 is the attribute Inst of the element LN in the IED section |
| | LPHD | m | Unn / Inn is the identification of the Sensor; A, B, Cand N are the phase identification. Both values are part of the substation section of the SCL and are used to build the name according to [6], clause 8.4.2 |
| | InnATCTR1 | m | |
| | InnBTCTR2 | m | |
| | InnCTCTR3 | m | |
| | InnNTCTR4 | m | |
| | UnnATVTR1 | m | |
| | UnnBTVTR2 | m | |
| | UnnCTVTR3 | m | |
| | UnnNTVTR4 | m | |

Figure 4.11: Logical device instance “MU” (IEC 61850-9-2LE)

Four instances (3ϴ + 1 neutral) of each the current (TCTR) and voltage (TVTR) logical nodes are required (mandatory requirement) within a logical device merging unit. This is due to the fact that primary equipment is typically done via a single phase representation. As part of the processing and computation, the data contained within the logical node has to be packaged into a dataset, encoded and published at the Datalink layer. The IEC 61850-9-2LE standard defines a fixed data set for the transmission of sampled values as shown in Fig. 4.12.



| Attribute Name | Value | Comment |
|---|---|---|
| DSName | PhsMeas1 | |
| DSRef | xxxxMUnn/LLN0$PhsMeas1 | |
| DSMemberRef | InnATCTR1.Amp[MX]<br>InnBTCTR2.Amp[MX]<br>InnCTCTR3.Amp[MX]<br>InnNTCTR4.Amp[MX]<br>UnnATVTR1.Vol[MX]<br>UnnBTVTR2.Vol[MX]<br>UnnCTVTR3.Vol[MX]<br>UnnNTVTR4.Vol[MX] | |

Fixed dataset

Figure 4.12: PhsMeas1 dataset (IEC 61850-9-2LE)

The fixed data set is then encoded and packaged into the sample value frame structure, with the suitable source and destination address, VLAN tag, Application protocol data unit and Abstract Syntax Notation (ASN.1) tagged Application Protocol Data Unit (APDU). The sampled value frame structure is discussed in detail in chapter 5 section 5.3. The generated sampled value message is then published at the Datalink layer, every 250µs for a 50Hz system.

## 4.3. Research project description and scope

In section 4.3 the complete merging unit device is discussed and ideally what needs to be designed and integrated for the full real-time implementation of a merging unit. The scope of this research project however is a delimited subset of the complete set of functionalities of a merging unit.

At the present moment there are only a limited number of suppliers who have successfully implemented devices that publish sampled values in a format compliant with IEC 61850-9-2LE guideline, and this has been mainly in vendor-based devices.

The limited number of IEC 61850-9-2LE devices in the market has resulted in quite a few research projects being initiated in research facilities around the globe. This research project investigates the development of a sensor node on a PC platform that can be used to simulate the publishing of sampled values. The intent of this project is to provide the building blocks for a future real-time, with real data implementation of a sensor node that is compliant with IEC 61850-9-2LE.

This project seeks to build the foundation for the creation of a virtual sensor node, to contribute to, and assist with the understanding of the process-bus concepts in an education environment. “Virtual” meaning that a PC based program will be simulating the sensor node and the sensor node is not actually implemented. The virtualised sensor node represents a limited functional implementation of the merging unit that will represent a virtualised data acquisition node capable of synthetic generation of waveforms through synthesis data generation, encoding of the data and transmitting the data in a format compliant with the IEC 61850 protocol.

The role of this project in an academic environment is to build the knowledge base required to assist with the understanding of the functioning and implementation and to lay the groundwork for a future real-time implementation.

Fig. 4.13a below represents a full implementation of a merging unit as specified by the standard. For a full implementation, once the three current and voltages together with their associated neutrals from the instrument transformer are coupled into the merging unit - the main components for subsequent processing and packaging are:

- Signal conditioning unit and anti-aliasing filters
- The Analogue to Digital Converter (ADC)
- Time synchronization module
- Processing and computing unit (shown as Digital signal Processing Unit.)

The full implementation will not be attempted in this proposed project. The research project will attempt to achieve the required functional behaviour for merging unit operation, but it is not within the scope of this project to achieve the temporal behaviour as stated within the standard. The scope and delimitation of the project is shown on the right-hand side, i.e. Fig. 4.13b below. The aspects of the merging unit architecture not directly implemented within this project, (Fig. 4.13b red perimeter) are:

- Interfacing to real-time primary plant devices (current and voltage transformers).
- The Analogue to Digital Converter (ADC)
- Time synchronization from an external source

The virtual sensor node development (Fig. 4.13b green perimeter) will include:

- Synthetic data generation through computation of a sine wave that is sampled 4000 times per cycle using the PC's system clock for time synchronization.
- Sample value frame structure generation in an object oriented environment
- Processing and encoding of the data generated into a sampled value frame structure
- ASN.1 encoding for the sampled value frame
- VLAN tagging of the sampled value message generated
- Time tagging of the sample value message using the PC's system clock

- Publishing of the sampled value frame generated onto the network.

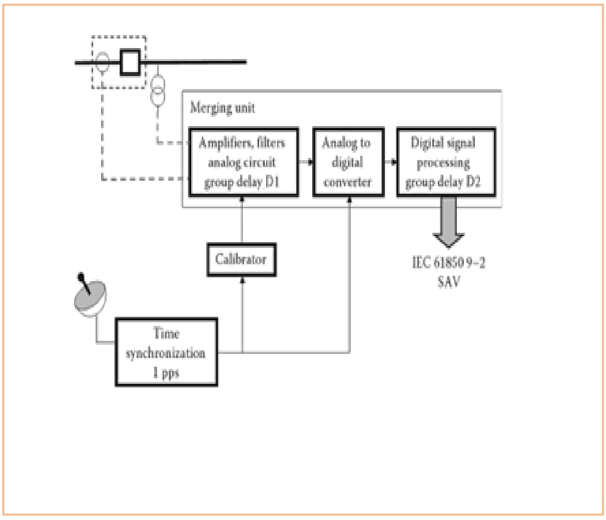

Merging unit
Amplifiers, filters analog circuit group delay D1
Analog to digital converter
Digital signal processing group delay D2
Calibrator
IEC 61850 9-2 SAV
Time synchronization 1 pps

Figure 4.13a: Structure of merging unit (Apostolov and Janssen, 2008)

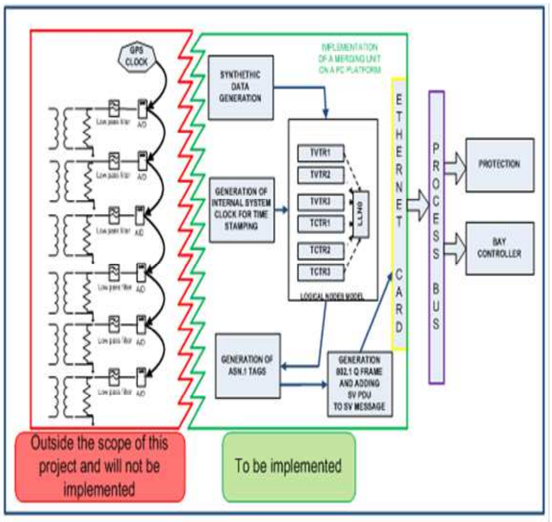

GPS CLOCK
Low pass filter
A/D
IMPLEMENTATION OF A MERGING UNIT ON A PC PLATFORM
SYNTHETHIC DATA GENERATION
GENERATION OF INTERNAL SYSTEM CLOCK FOR TIME STAMPING
TVTR1
TVTR2
TVTR3
TCTR1
TCTR2
TCTR3
LLN0
LOGICAL NODES MODEL
GENERATION OF ASN.1 TAGS
GENERATION 802.1 Q FRAME AND ADDING SV PDU TO SV MESSAGE
ETHERNET CARD
PROCESS BUS
PROTECTION
BAY CONTROLLER
Outside the scope of this project and will not be implemented
To be implemented

Figure 4.13b: PC based sensor capable of sampled value outputting

In the sensor node depicted in Fig. 4.13b the various parts required for a merging unit are simulated. The inputs from the current and voltage transformer, the signal conditioning and analogue-to-digital converter outputs are all replaced with the synthetic data generator. The time synchronization provided by a GPS clock in merging units is simulated using the PC’s internal clock. The processing and computing unit is substituted with a PC platform using the object oriented programming environment provided by Python. A Python program is used for 802.1Q packet crafting, priority tagging, data modelling, encoding and publishing of sampled values in a format compliant with IEC 61850-9-2LE.

With reference to Fig. 4.13b, the output of the virtual sensor node, once the data is encoded and packaged; it is required to publish these sample value messages at the datalink layer level (Konke et al., 2012) - see Fig. 4.14.

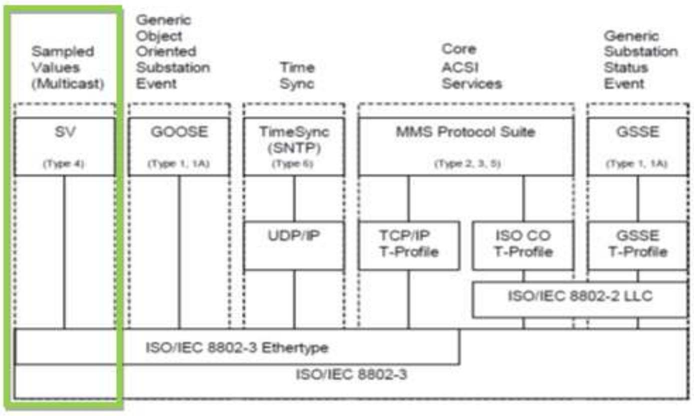


Figure 4.14: Overview of functionality and profiles, with emphasis on SV (IEC 61850-8-1, 2004)

The sampled value frame structure was introduced in chapter 3 in section 3.5.2.3.2 and will be further elaborated upon in chapter 5 section 5.3. In-order to select the hardware and software platform for the project development, an in-depth investigation into PC based hardware and software platforms was conducted. These are elaborated upon in more detail in the following section.

## 4.4. Development tools and development platform

For the development of the virtual sensor node proposed, an in-depth investigation into layer 2 networking on PC-based platforms had to be conducted. The two operating systems investigated are Windows and Linux Ubuntu using Python as the programming language as is shown in Fig. 4.15.

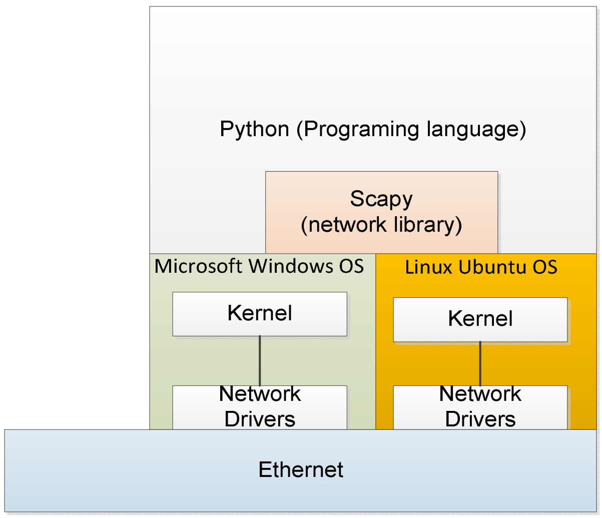


Figure 4.15: Low level networking on PC based platforms using MS Windows and Linux Ubuntu

The tools used for the development are Python as the programming language - initially using raw sockets (discussed later in this chapter), and also using a Python network library called Scapy. Python as a programming language is briefly introduced in section 4.5.1 and Scapy as a Python networking library is discussed in section 4.5.2.

### 4.4.1. Programming language: Python

Python is an object-oriented, programming language that provides built-in data structures that make Python attractive for Rapid Application Development (www.Python.org). Python is a relatively simple-to-learn programming language with easily readable syntax. The Python user website (www.Python.org) provides examples and tutorials to fast-track Python beginners into intermediate programmers. Additionally, Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and Python libraries are mostly available in source or binary form without any charge.  Python was selected for this project because:

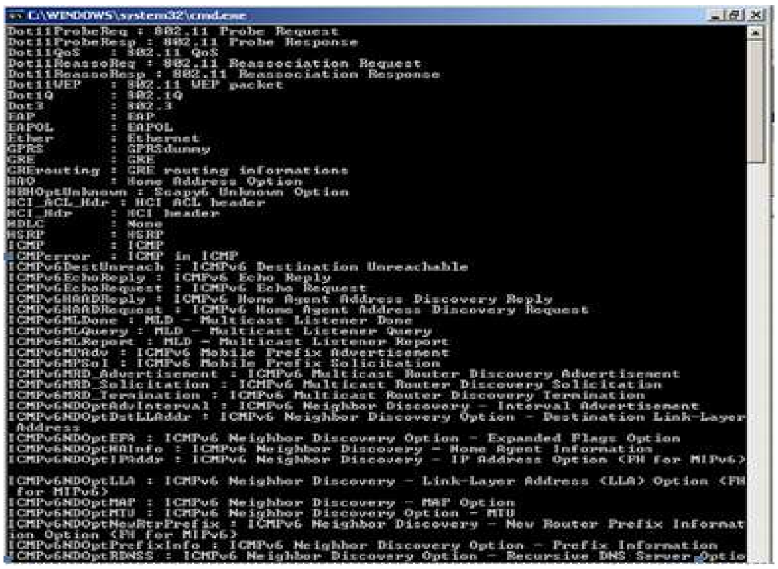
- It is an open source general-purpose language.
- It provides an object orientated environment
- Python is supported on a wide range on operating systems
- It has an intuitive, interactive environment for developers
- It has excellent data networking libraries such as SCAPY, and TWISTED Python
- Python has libraries that support the development of Abstract Syntax Notation (ASN.1) encoding such as  PyASN

### 4.4.2. Networking libraries – Scapy

Scapy is a Python networking library that is capable of encoding and decoding data for Ethernet frames and Internet protocol packets for a wide number of protocols, and is capable of transmitting the generated packets on through the network (www.scapy.org). Scapy uses Python classes to provide a framework for the creation of packets or frames (Biondi, 2003). Through objected oriented programming, Scapy provides support for the creation of custom packets by interacting with the operating system kernel using sockets such as:

- Socket.SOC_STREAM - Socket stream
- Socket.SOCK_DGRAM  - Socket Datagram
- Socket.SOCK - Raw sockets
- Socket.SOCK_SEQPACKET - Sequence sockets

The listed of supported protocols can be checked on Scapy by using the list () function as shown in Fig. 4.16 below.

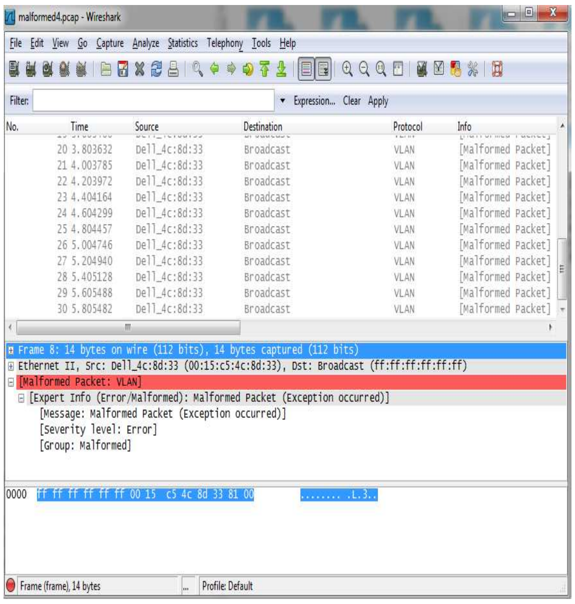


```
Dot11ProbeReq : 802.11 Probe Request
Dot11ProbeResp : 802.11 Probe Response
Dot11QoS     : 802.11 QoS
Dot11ReassoReq : 802.11 Reassociation Request
Dot11ReassoResp : 802.11 Reassociation Response
Dot11WEP     : 802.11 WEP packet
Dot1Q        : 802.1Q
Dot3         : 802.3
EAP          : EAP
EAPOL        : EAPOL
Ether        : Ethernet
GPRS         : GPRSdummy
GRE          : GRE
GRErouting : GRE routing informations
HAO          : Home Address Option
HBHOptUnknown : Scapy6 Unknown Option
HCI_ACL_Hdr : HCI ACL header
HCI_Hdr      : HCI header
HDLC         : None
HSRP         : HSRP
ICMP         : ICMP
ICMPerror    : ICMP in ICMP
ICMPv6DestUnreach : ICMPv6 Destination Unreachable
ICMPv6EchoReply : ICMPv6 Echo Reply
ICMPv6EchoRequest : ICMPv6 Echo Request
ICMPv6HAADReply : ICMPv6 Home Agent Address Discovery Reply
ICMPv6HAADRequest : ICMPv6 Home Agent Address Discovery Request
ICMPv6MLDone : MLD - Multicast Listener Done
ICMPv6MLQuery : MLD - Multicast Listener Query
ICMPv6MLReport : MLD - Multicast Listener Report
ICMPv6MPAdv : ICMPv6 Mobile Prefix Advertisement
ICMPv6MPSol : ICMPv6 Mobile Prefix Solicitation
ICMPv6MRD_Advertisement : ICMPv6 Multicast Router Discovery Advertisement
ICMPv6MRD_Solicitation : ICMPv6 Multicast Router Discovery Solicitation
ICMPv6MRD_Termination : ICMPv6 Multicast Router Discovery Termination
ICMPv6NDOptAdvInterval : ICMPv6 Neighbor Discovery - Interval Advertisement
ICMPv6NDOptDstLLAddr : ICMPv6 Neighbor Discovery Option - Destination Link-Layer
 Address
ICMPv6NDOptEFA : ICMPv6 Neighbor Discovery Option - Expanded Flags Option
ICMPv6NDOptHAInfo : ICMPv6 Neighbor Discovery - Home Agent Information
ICMPv6NDOptIPAddr : ICMPv6 Neighbor Discovery - IP Address Option (FH for MIPv6)

ICMPv6NDOptLLA : ICMPv6 Neighbor Discovery - Link-Layer Address (LLA) Option (FH
 for MIPv6)
ICMPv6NDOptMAP : ICMPv6 Neighbor Discovery - MAP Option
ICMPv6NDOptMTU : ICMPv6 Neighbor Discovery Option - MTU
ICMPv6NDOptNewRtrPrefix : ICMPv6 Neighbor Discovery - New Router Prefix Informat
ion Option (FH for MIPv6)
ICMPv6NDOptPrefixInfo : ICMPv6 Neighbor Discovery Option - Prefix Information
ICMPv6NDOptRDNSS : ICMPv6 Neighbor Discovery Option - Recursive DNS Server Optio
```

Figure 4.16: List of supported protocols on Scapy

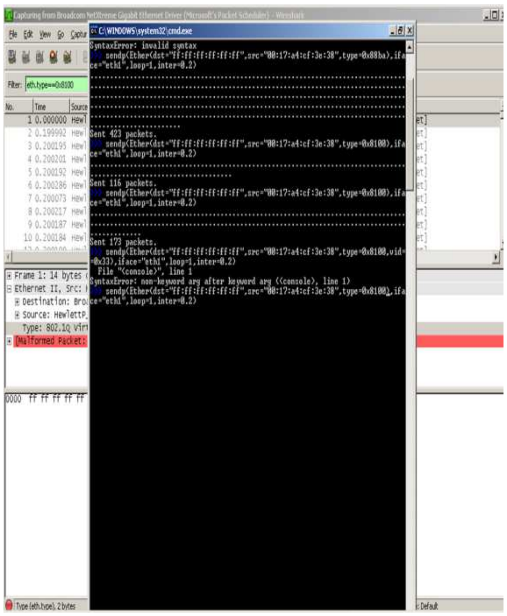
### 4.4.3. Crafting of custom packets utilising the Scapy library

Networking tools and libraries are generally built for specific functions and cannot be extended, or users cannot deviate much from the specific function the author had in mind when developing the library. Scapy differs from most networking libraries in this regard as it allows the user to create custom networking protocols by extending the Scapy library (Biondi, 2003). Chapter 9, page 41 on the Scapy documentation manual discusses the process of custom development on Scapy.


malformed4.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: Expression... Clear Apply

```
No.  Time      Source          Destination  Protocol  Info
20 3.803632    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
21 4.003785    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
22 4.203972    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
23 4.404164    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
24 4.604299    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
25 4.804457    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
26 5.004746    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
27 5.204940    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
28 5.405128    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
29 5.605488    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]
30 5.805482    Dell_4c:8d:33   Broadcast    VLAN      [Malformed Packet]

Frame 8: 14 bytes on wire (112 bits), 14 bytes captured (112 bits)
Ethernet II, Src: Dell_4c:8d:33 (00:15:c5:4c:8d:33), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
[Malformed Packet: VLAN]
  [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
    [Message: Malformed Packet (Exception occurred)]
    [Severity level: Error]
    [Group: Malformed]

0000  ff ff ff ff ff ff 00 15  c5 4c 8d 33 81 00        ........ .L.3..

Frame (frame), 14 bytes   ...   Profile: Default
```

Figure 4.17a: Custom packet generated with Scapy that does not conform to any standard [Linux]


Capturing from Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet Scheduler) - Wireshark
C:\WINDOWS\system32\cmd.exe
Filter: eth.type==0x8100

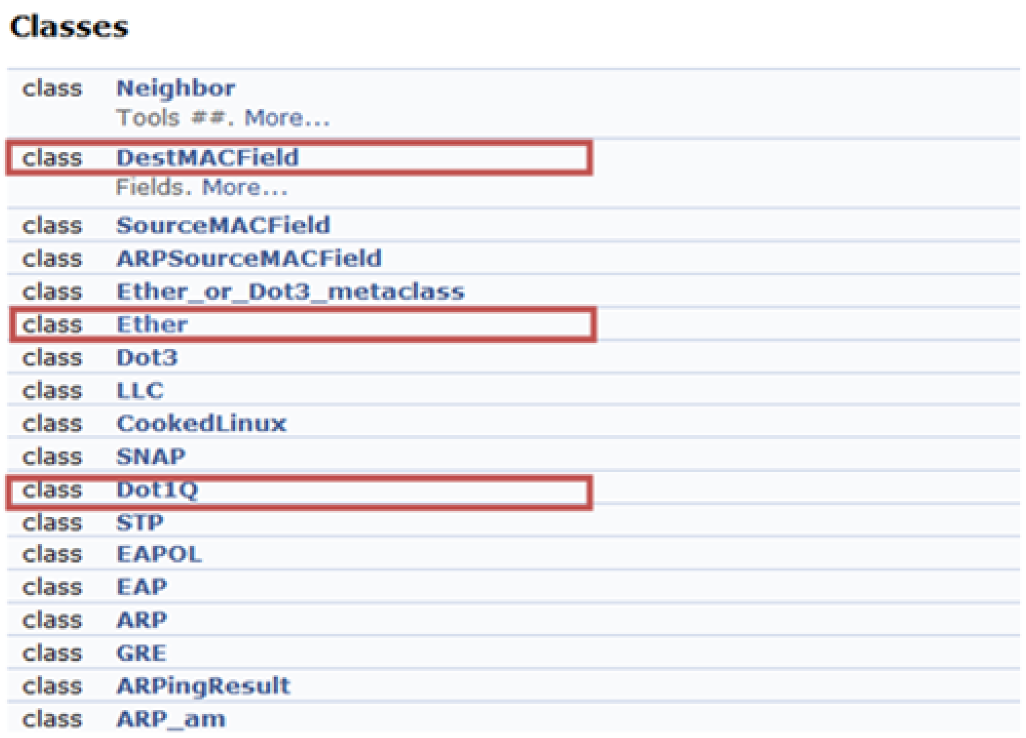
```
SyntaxError: invalid syntax
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:17:a4:cf:3e:38",type=0x88ba),ifa
ce="eth1",loop=1,inter=0.2)
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
.....................
Sent 423 packets.
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:17:a4:cf:3e:38",type=0x8100),ifa
ce="eth1",loop=1,inter=0.2)
................................................................................
.................................
Sent 116 packets.
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:17:a4:cf:3e:38",type=0x8100),ifa
ce="eth1",loop=1,inter=0.2)
................................................................................
................................................................................
............
Sent 173 packets.
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:17:a4:cf:3e:38",type=0x8100,vid=
=0x33),iface="eth1",loop=1,inter=0.2)
  File "<console>", line 1
SyntaxError: non-keyword arg after keyword arg (<console>, line 1)
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:17:a4:cf:3e:38",type=0x8100),ifa
ce="eth1",loop=1,inter=0.2)
```

```
Frame 1: 14 bytes
Ethernet II, Src:
  Destination: Bro
  Source: HewlettP
  Type: 802.1Q Vir
[Malformed Packet:
0000  ff ff ff ff ff
```

Figure 4.17b: Custom packet generated with Scapy that does not conform to any standard [Microsoft Windows]

Fig. 4.17a shows an example of a custom packet generated with Scapy that does not conform to any standard. The VLAN tag in the packet generated has been deliberately tampered with, and as a result, a protocol analyser such as Wireshark (discussed later in this document) identifies the packet as "Malformed" as it does not conform to any standard. Scapy provides the framework and allows for the transmission of custom layer 2 and layer 3 packets (Biondi).

For this project, the IEEE 802.1Q framework provided on Scapy is leveraged, and expanded upon. On Scapy, the three classes highlighted in red in Fig. 4.18 below, are already available as a foundation for custom layer 2 packet crafting.



**Classes**

```
class   Neighbor
        Tools ##. More...
class   DestMACField
        Fields. More...
class   SourceMACField
class   ARPSourceMACField
class   Ether_or_Dot3_metaclass
class   Ether
class   Dot3
class   LLC
class   CookedLinux
class   SNAP
class   Dot1Q
class   STP
class   EAPOL
class   EAP
class   ARP
class   GRE
class   ARPingResult
class   ARP_am
```

Figure 4.18:: Scapy supported classes for layer 2 networking (Biondi, 2009)

What is not available, and what needed to be developed for this project are classes for the following fields in the sampled value frame (Details about the sampled value frame structure have been covered in chapter 3):

- Application protocol Identifier (APPID)
- Reserved bytes
- Variable length
- Sampled value Protocol Data Unit (svPDU)
- ASN.1 encoding for the sampled value frame generated
- Sampled value dataset
- Sampled value counter

### 4.4.4. Using Scapy on Microsoft Windows and Linux Operating Systems

Scapy is supported and can be used in combination with a wide range of operating systems (Biondi, 2009). In this project Scapy networking was tested on both Microsoft Windows and Linux operating systems. The software packages required for the use of Scapy on Microsoft Windows Operating Systems (OS) are catalogued below.

- Python 2.5 or higher
- Pywin32
- WinPcap,

Fig. 4.19 below details how low-level networking is achieved on Microsoft Windows operating systems using Scapy.

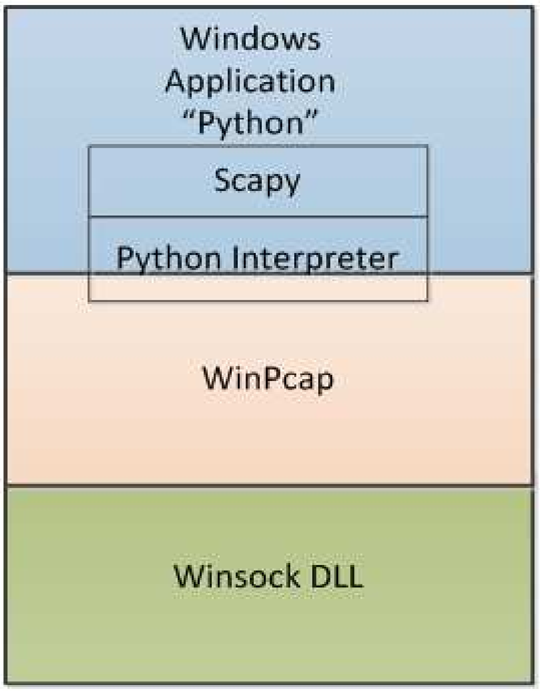


Figure 4.19: Using Scapy on Microsoft Windows

Accessing the physical layer on a Microsoft Windows platform involves interfacing with the Winsock Data Link Library (DLL). The Microsoft Winsock data link library is discussed later in this chapter. Scapy interfaces to the Winsock DLL on Windows using the Winpcap library - also detailed later in this document (Biondi, 2009). The challenge with Scapy on a Microsoft Windows system is that Scapy has been primarily developed for Unix-like systems and works best on those platforms (Biondi, 2009). Although the experiment with Scapy on Microsoft Windows was successful, limited functionality on a Microsoft Windows platform was experienced by the author. Limited functionality in the sense that Scapy would not work on any of the Integrated Development Platforms (IDEs) tested - such as PyWin, Not Just Another IDE (NINJA) and ECLIPSE.

The next phase was to test Scapy on a Linux operating system. The software packages listed below are required for Scapy on a Linux Operating System (OS) (Biondi, 2003):

- Python 2.5 or higher
- LibPcap
- PyPcap
- Pyreadline
- Libdnet

Fig. 4.20 below details how low-level networking is achieved on a Linux operating system using Scapy. Accessing the physical layer on a Linux Ubuntu platform involves interfacing with the Berkley Packet Filter (BPF). The Berkeley Packet Filter (BPF), on UNIX systems such as Linux Ubuntu provides an interface to the data link layer. The Berkley packet filters are accessed via the LibPcap library. LibPcap is a Linux equivalent of the WinPcap library discussed later in this document.

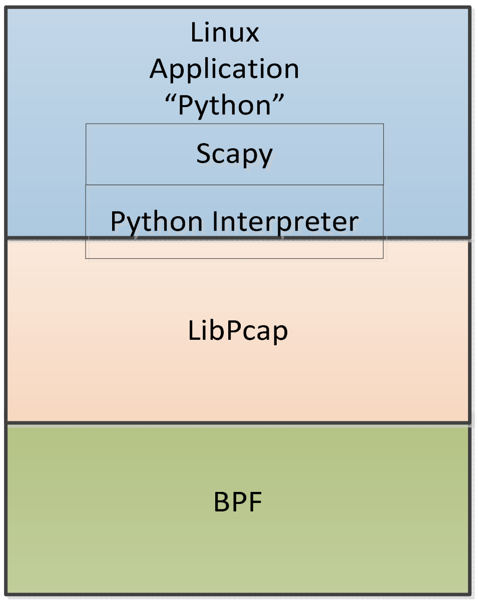
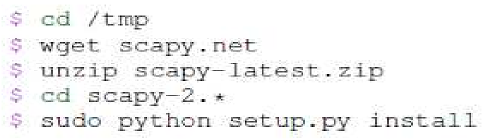


Figure 4.20: Using Scapy on Linux Ubuntu

Scapy can be installed on Linux from the terminal window using root user access credentials. The technique used to gain Root user access level on Linux operating systems is shown in Appendix B. Installing Scapy on a Linux operating system generally installs the required packages if they are not already installed on the user’s PC. Fig. 4.21 shows the installation process followed for installing Scapy on Linux Ubuntu.



```
$ cd /tmp
$ wget scapy.net
$ unzip scapy-latest.zip
$ cd scapy-2.*
$ sudo python setup.py install
```

Figure 4.21: Installing Scapy on Linux Ubuntu (Biondi, 2009).

## 4.5. Datalink layer networking

The Datalink is the layer just above the physical layer as shown in Fig. 4.22 below. The data link layer (layer 2) uses the services of the physical layer to send and receive data over the communication network and also provides a service to the network layer above.

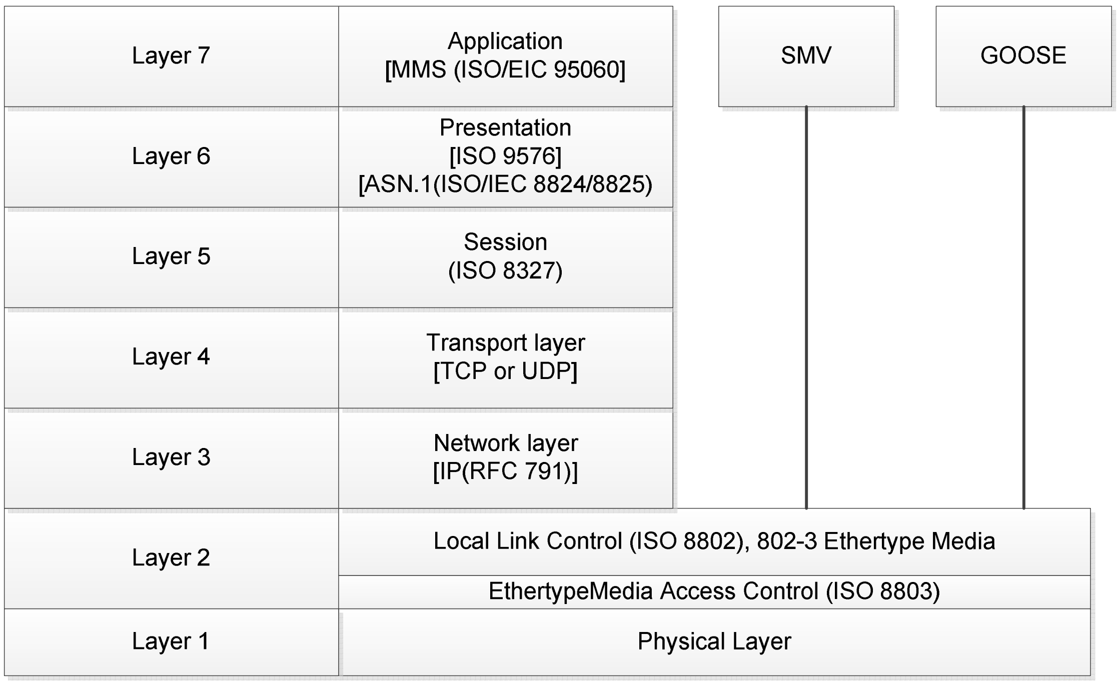


Figure 4.22: OSI reference model (Horalek and Sobeslav, 2014)

The Datalink layer is utilised by protocols such as Address Resolution Protocol (ARP), Point-to-Point protocol (PPP), Ethernet, etc. to provide fast messaging between two systems. GOOSE and sampled values also map directly to the Datalink layer to provide fast communication in IEC 61850 substation automation systems.

The major challenge with this project is that Data link layer frames are more difficult to program as opposed to the higher level protocols such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Data link layer frames use raw socket and raw packet structures as opposed to Datagram and Stream sockets utilised for higher level protocols. Sockets are discussed later in this chapter.

Higher level protocols in this context refer to the protocols using the TCP or UDP as the transport layer. These protocols are widely used in applications such as the Internet, and Supervisory Control and Data Acquisition (SCADA) in the substation environment. As a result, lots of work has already been done to create libraries/modules for scripting high level networking programs.

Using programming languages such as C, C++, Java, and Python libraries/modules to easily program transport layer protocols are readily available. A software developer often has the easy task of defining the data to be transmitted, and attaching it into the already erected transport mechanism. For this project though, these types of modules or libraries do not exist and methods to script data link messages using raw sockets had to be investigated and the use of raw sockets on the various operating systems had to be investigated.

### 4.5.1. PC-based platform

The publishing of data-link frames is investigated on two platforms, namely Microsoft Windows and Linux. Most of the initial work was performed on Microsoft Windows as this is a widely used commercial platform. Later Linux operating systems were investigated as Linux offers an open-source platform.

The objective of this investigation was to establish which kernel would allow generation, binding, and outputting of layer 2 frames. Fig. 4.23 below shows an example of how user applications interact with the physical devices in a PC environment.

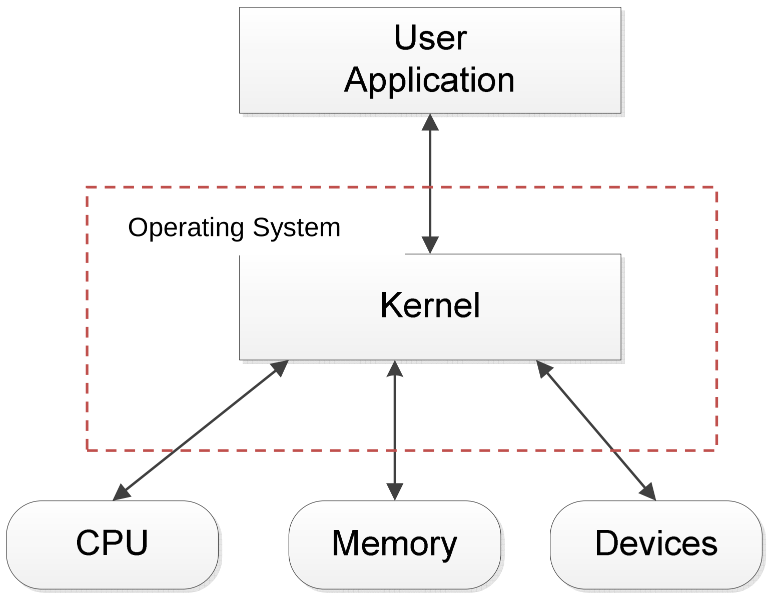


Figure 4.23: interaction between application layer and devices (Bovet and Cesati, 2000)

As shown in Fig. 4.23 above, the kernel is the main component of a personal computer’s operating system (Bovet and Cesati, 2000). The kernel provides the bridge between applications and the actual data processing done at the hardware level. In-order to implement low-level network programming applications, Microsoft Windows and Linux kernels and the user Application Programming Interface (API) interfaces to the network card for the transmission of data link frames has to be thoroughly understood. Fig. 4.24 below shows an example of a Linux kernel.

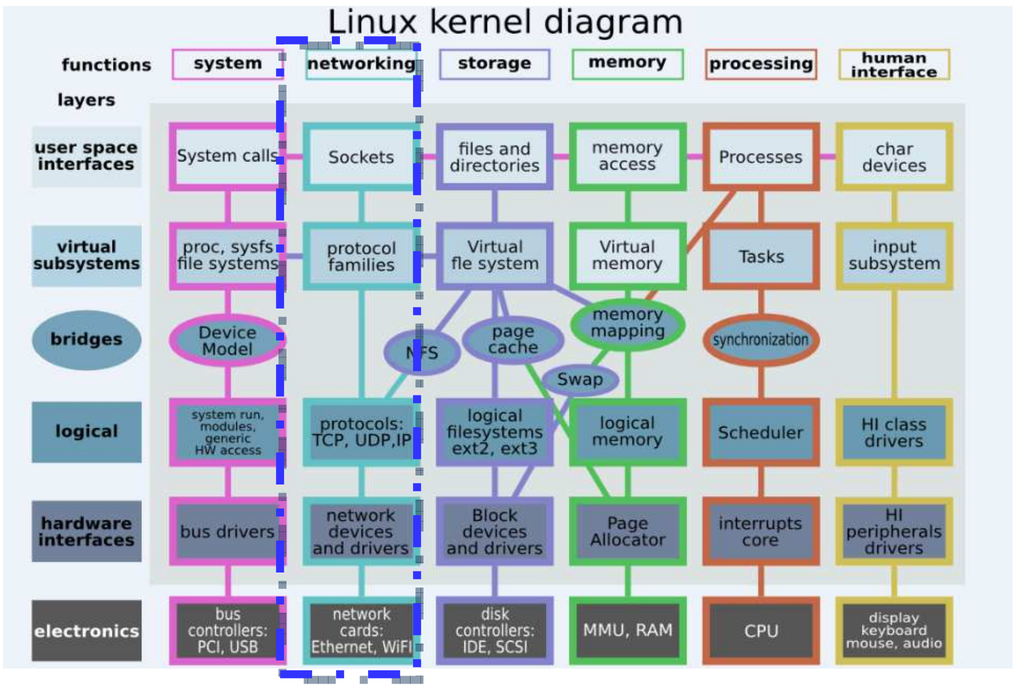


Figure 4.24: Linux kernel (Shuylupin, 2009)

The path of a packet through the kernel is highlighted (dashed blue line) in Fig. 4.24 above. The Application Program Interface (API) interface to the network card is provided through the use of sockets. Sockets are discussed in section 4.5.1.1 below.

#### 4.5.1.1. Network sockets

The API interface to the network card is generally provided through the use of sockets. **Definition**: “A socket is an endpoint of communication to which a name may be bound” (Leffer et al). The paper by Tiponut, 2001 “Python Network Programming” provides a good overview of using sockets in Python.

A socket is an Application Programming Interface (API) provided by Microsoft Windows and Linux operating systems. The socket API is also provided on other operating systems not covered in this document. The socket API which is sometimes referred to by its original name - Berkeley Socket API was originally released as the Berkeley-Based Code in 1983 (Hosmer, 2014), and is a set of standard function calls made by the application layer to the operating system. These function calls allow networking programmers to provide communication capabilities in the developed products.

Since the introduction of Berkeley-Based Code, other socket APIs have since been developed. Winsock (Microsoft Windows Sockets) developed in 1993 for the Microsoft Windows platform, and TLI (Transport Layer Interface) developed by AT&T, are two of the most common socket APIs.

There are four socket types available to application users and their functions are briefly discussed in section 4.5.1.2 below. These sockets types are supported on both Microsoft Windows and Linux environments.

### 4.5.1.2. Socket Types

There are four types of sockets available to the user. The stream and datagram sockets described below are the most commonly utilized socket types. Raw sockets and sequenced packet sockets are the less used sockets. Stream sockets are utilised when guaranteed delivery of a networked packet is required. This type of socket is typically used for TCP applications. An example of a TCP communication is shown in Fig. 4.25 below.

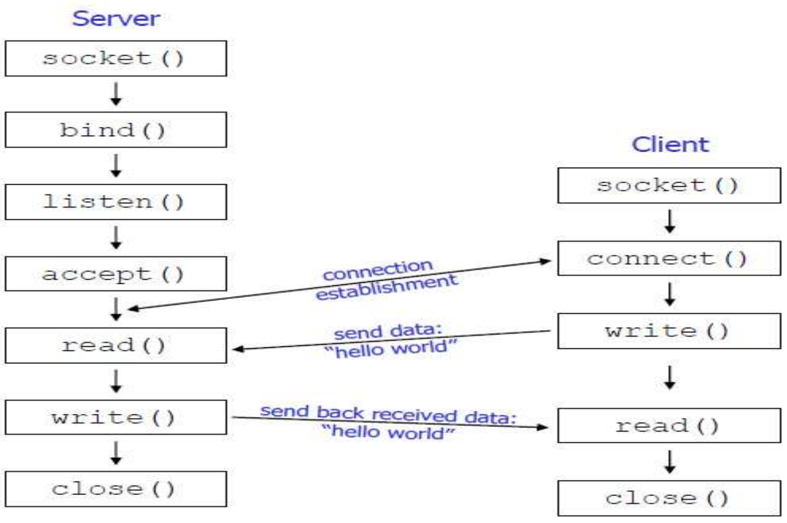


Figure 4.25: Socket Functions for TCP Client-Server connection (Halvorsen, 2005)

In Fig. 4.25 a connection is initiated from the client side using the socket connection function. A connection is then established with the server side device. The socket write function is then used to send data and an appropriate acknowledgement is received. If no acknowledgement is received the packet is retransmitted (Halvorsen, 2005).

When network delivery is required, but not necessarily guaranteed, Datagram sockets can be utilised. This type of socket is typically used for applications using the User Datagram protocol (UDP) as the transport. An example of a UDP communication is shown in Fig. 4.26 below.

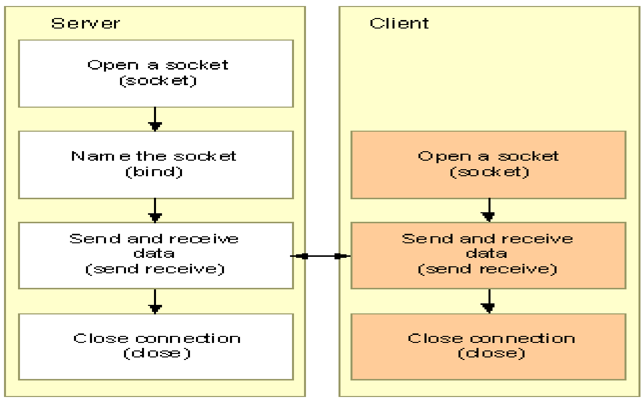


Figure 4.26: Socket Functions for UDP Client-Server connection (Halvorsen, 2005)

Datagram sockets are connectionless. An open connection does not need to be established as in Stream Sockets. A packet is crafted and transmitted to the destination without prior knowledge of whether the receiving device is still connected to the network and no acknowledgement is required.

Sequence packet sockets are similar to a stream socket, they are connection orientated, and differ in the sense that record boundaries are preserved. These types of sockets are only provided as part of the Network Systems (NS) socket abstraction, and are mentioned in this document simply for completeness.

Another type of socket is the raw socket. “Raw Sockets” provide users access to the underlying communication protocols (Jorgensen, 2012). Raw sockets are provided mainly for those interested in developing new communication protocols, or gaining access to some of the more obscure facilities of an existing protocol.

For the development of custom protocols, e.g. sampled value transmission at the data link layer, raw sockets have to be utilized. An investigation into data link layer message transmission on Microsoft Windows and Linux Operating Systems is conducted in sections 4.6.1.3 and 4.6.1.4 respectively.

#### 4.5.1.3. Using raw sockets on Microsoft Windows

Winsock is an application programming interface for networking applications of Microsoft Windows systems. It's called Winsock because it's an adaptation of the Berkeley socket interface which is provided for Unix/Linux systems. The Winsock API provides the interface between the user application program and network interface layers 2 or layer 3 protocols on a computer. An illustration of how a user application such as a Python networking script interfaces to the hardware level through the use of Winsock API is shown in Fig. 4.27.

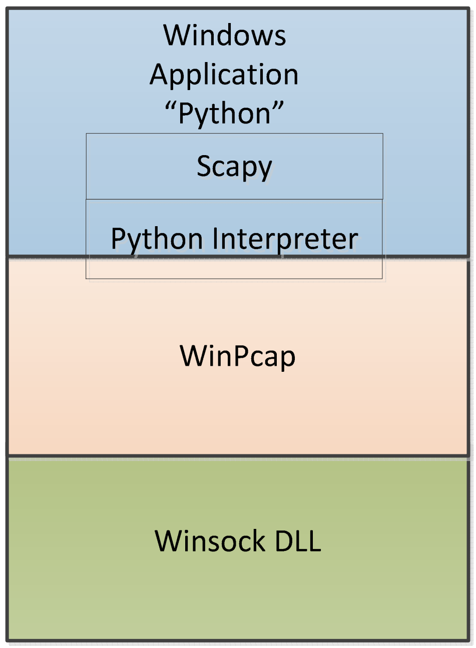


Figure 4.27: Using raw sockets on Microsoft Windows

As shown in Fig.4.27 above, the user application - in this case being a Python script - gains access to low-level networking through the use of Winsock API. The Winsock API can be accessed through the use of another library such as Winpcap (discussed later in this chapter), or through direct interaction with the Winsock Data Link Library (DLL). The test setup to examine the use of raw sockets in Microsoft Windows was implemented and is shown in Fig. 4.28.

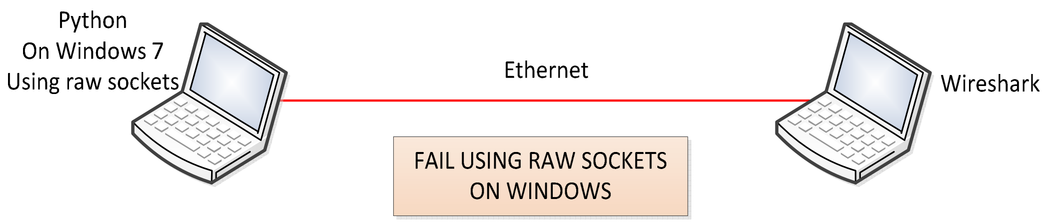


Figure 4.28: Test setup for using raw sockets on Microsoft Windows

The test setup shown in Fig. 4.28 above was used for testing the use of raw sockets on Microsoft Windows 7 was configured and a Python script to initiate/use raw sockets for data transmission was developed. In this case a Python script is used to gain access to low-level networking through the use direct socket calls to the Winsock API. As indicated in Fig. 4.28 this case study implementation was unsuccessful. The reason for this failure was that the use of raw sockets of the Winsock 2.0 data link library has been discontinued within the Microsoft Windows operating system (since Windows XP service pack 1). The System Administrator, Audit, Networking, and Security (SANS) paper on “The Raw and the Uncooked: The Microsoft Windows XP Raw Sockets Saga, Final Words” discusses why raw sockets support for security reasons has been discontinued on Winsock 2.0.

#### 4.5.1.4. Using raw sockets on a Linux operating system

Linux is an open-source operating system. It is a platform derived from UNIX to facilitate research and development. Raw sockets are still supported on Linux operating systems. The method by which Python applications can use raw sockets on Linux is illustrated in Fig. 4.29.

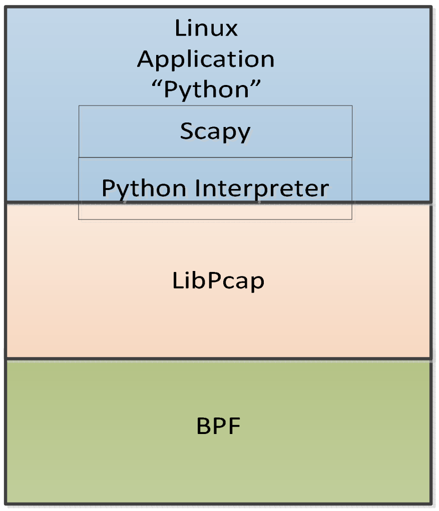


Figure 4.29: Using raw sockets on Linux

The Python interface to the UNIX system call and library interface for sockets is a direct transliteration to Python’s object-oriented style (Online: https://docs.python.org/2/library/socket.html). Many examples of how this is can be implemented can be found at www.docs.Python.org.

The socket() function on Linux operating systems returns a socket object whose methods implement the various socket system calls. On Python, buffer allocation on receive operations is automatic, and buffer length is implicit on send operations.

The test setup to examine the use of raw sockets in Microsoft Windows was implemented and is shown in Fig. 4.30.

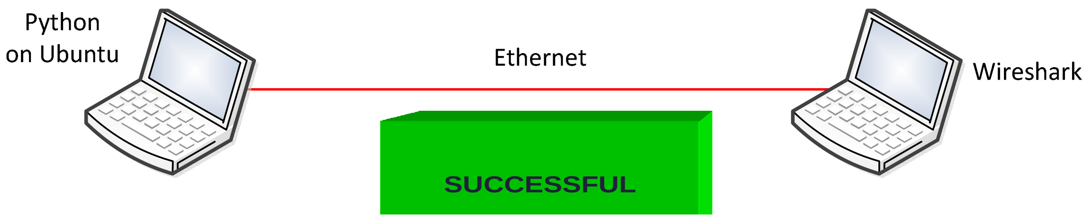


Figure 4.30: Using raw sockets on Linux

As shown in Fig. 4.30 this case study implementation was successful. Linux supports the use of raw sockets.  Python on a Linux operating system such as Ubuntu also has extensive support and there is a wide variety of low level networking libraries that can be utilised to ease the process of low-level network programming on Linux operating systems.

A number of these libraries were examined in this project so as to fast-track the development process. The libraries itemised below were examined for different functions which had to be performed in this project.

i) PyASN1 – for Abstract Syntax Notation One (ASN.1) encoding
ii) Scapy – for packet crafting
iii) Twisted Library – for packet crafting
iv) Scapy ASN – for ASN.1 encoding

From the libraries listed above, Scapy was selected for the sampled value packet generation and transmission. The Scapy library provides a powerful Python framework for crafting and transmitting packets. Scapy users are able to assemble and disassemble arbitrary frames/packets and send them out on to the network.

## 4.6. Conclusion

In this chapter the project context and the project scope and the delimitation of the research project undertaken, is outlined. Low-level networking on Microsoft Windows and Linux operating systems are investigated, for the development of the virtual sensor node. From the two platforms investigated Linux Ubuntu has been selected as the development platform. Ubuntu as a Linux operating system allows the use of raw sockets for custom protocol development. The Python programming language and Scapy as the networking libraries have been selected as the tools that are used in the implementation of the sensor node which is discussed in the chapter to follow.

# CHAPTER 5

# Implementation, Testing and Verification

## 5.1. Introduction

Sampled Value (SV) messages discussed in chapter 3 section 3.5.2.3.2.5 in this document are associated with the real-time distribution of data in substation automation systems. The sampled value messages are transmitted on a periodic basis from a sensor node referred to as a merging unit in substation automation systems. The sampled value message structure is shown in chapter 3, section 3.5.2.3 and consists of various fields, starting with the preamble and Start of Frame Delimiter (SFD) at the hardware level, and ending with the sampled value payload consisting of a fixed dataset containing the measured values.

In chapter four, the context and outline for the research project is established.

In this chapter, the implementation of the virtualized sensor node that can be used to publish sampled value messages is described, and two testing scenarios - one using a commercial Omicron CMC test injection device for power systems, and another using the Personal Computer (PC) based sensor node are discussed. The sampled value messages published from the two devices are captured using a commercial protocol analyser tool called Wireshark and a comparative analysis of the message structure obtained from both the commercial vendor device (CMC) and PC-based simulation are compared to each other, and to the sampled value message structure as specified by the IEC 61850-9-2 standard and the IEC 61850-9-2LE implementation guideline document. The comparative analysis of message structure will serve as verification of functional behaviour. Temporal performance will also be commented upon.

The chapter is structured into five sections. Section 5.1 is the introduction. In section 5.2 the implementation in terms of modular structure of the virtualised sensor node that can be used to simulate and publish sampled value messages is discussed. In section 5.3, Wireshark as the protocol analyser in section 5.4 through to section 5.5 is discussed. A practical experiment is conducted in section 5.4, first using the developed PC-based sensor node is performed in section 5.4.1 and using a practical a commercial unit, an Omicron CMC injection test set in section 5.4.2. In section 5.5 the structure of the sampled value messages published from both the developed PC-based sensor node and the commercial unit is compared to each other and to the format as stipulated in the IEC 61850-9-2LE guideline document.

## 5.2. Software development for the PC-based sensor node

In this section the software development for the Personal Computer (PC) based sensor node is discussed. The PC-based sensor node developed is shown in modular form in Fig. 5.1 below.

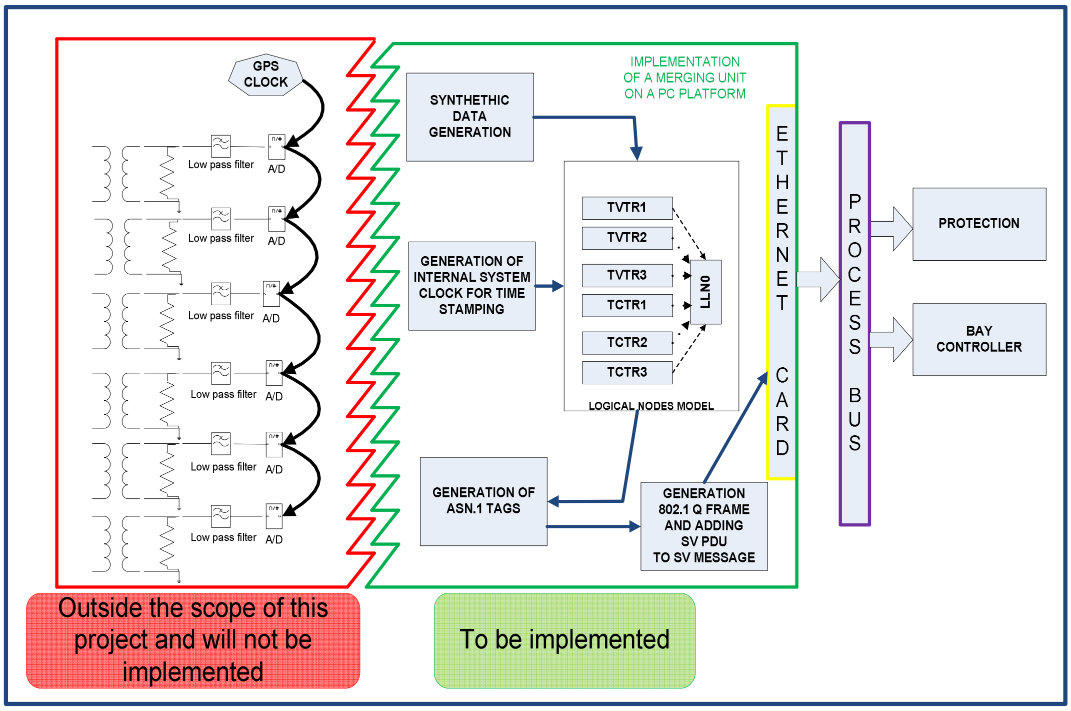

GPS CLOCK
Low pass filter
A/D
IMPLEMENTATION OF A MERGING UNIT ON A PC PLATFORM
SYNTHETHIC DATA GENERATION
GENERATION OF INTERNAL SYSTEM CLOCK FOR TIME STAMPING
TVTR1
TVTR2
TVTR3
TCTR1
TCTR2
TCTR3
LLN0
LOGICAL NODES MODEL
GENERATION OF ASN.1 TAGS
GENERATION 802.1 Q FRAME AND ADDING SV PDU TO SV MESSAGE
ETHERNET CARD
PROCESS BUS
PROTECTION
BAY CONTROLLER
Outside the scope of this project and will not be implemented
To be implemented

Figure 5.1: PC based Sensor Node capable of sampled value publishing

The PC based sensor node is developed in an Objected Orientated environment provided by the Python programming language and is capable of synthetic data generation, time stamping using the PC's internal clock, ASN.1 tagging of the generated sampled value frames and outputting of a sampled value at the datalink layer.

A simplified flowchart of the solution that can be used to simulate the functional behaviour of the IEC 61850 process level sensor node is shown in Fig. 5.2 below.

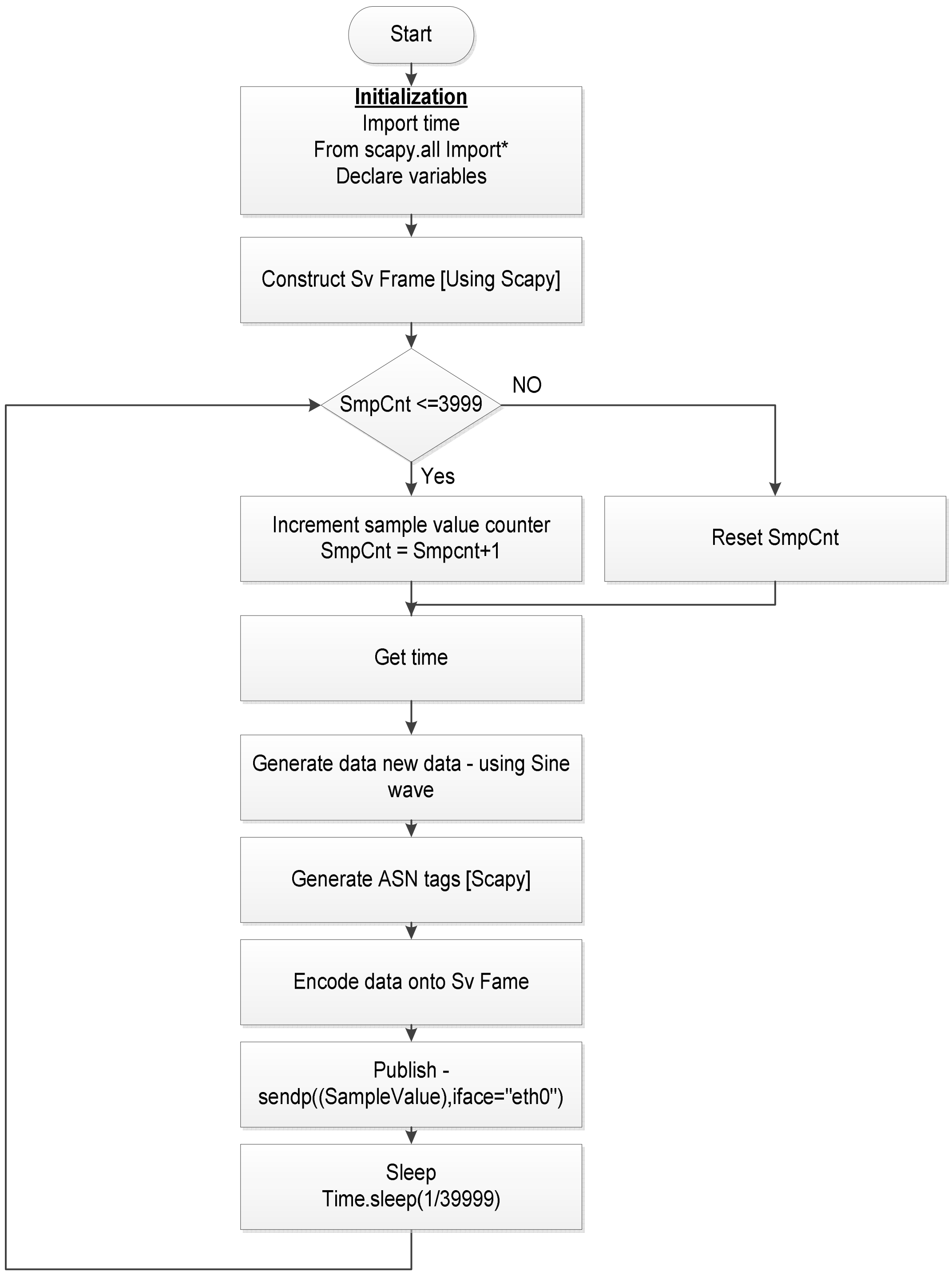
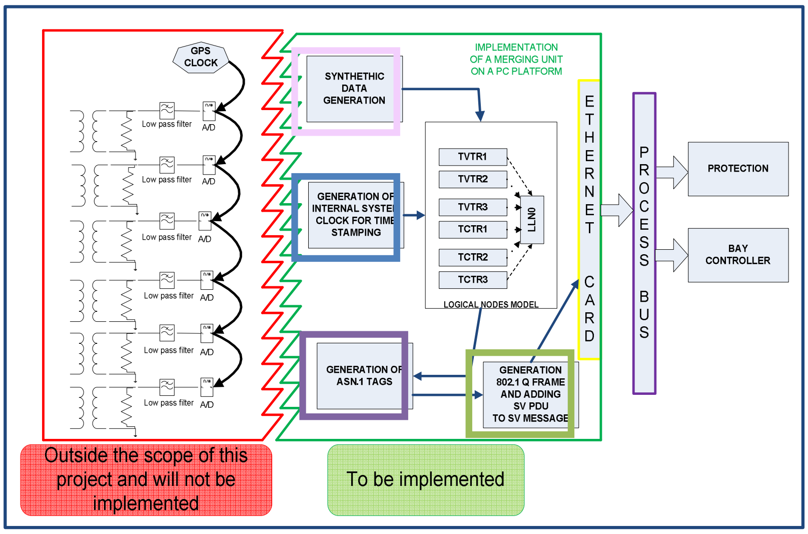


Figure 5.2: Simplified flow chart of the developed solution

The solution starts by importing the various Python libraries required for the development and finishes off by publishing a sampled value frame. The various class, class methods, and attributes that are utilised for synthetic data generation, time synchronization, ASN.1 encoding and sampled value frame structure - with Virtual Local Area Network (VLAN) tagging, are discussed in the sections to follow.

### 5.2.1. Synthetic data generation

In chapter 4 section 4.3, the design and integration for a full merging unit implementation was outlined. In the full implementation, a merging unit receives analogue inputs from instrument transformers. These analogue inputs are then conditioned, filtered, and digitized before they are processed. In this project, the acquisition of analogue input data (typically received from instrument transformers) and digitised with the use of analogue to digital converters does not fall within the scope, but is simulated through the programming of a discretised sinusoidal wave for the synthetic generation of the data. In Fig. 5.3 below the block responsible for the synthetic data generation is highlighted (pink block).



GPS CLOCK
Low pass filter
A/D
IMPLEMENTATION OF A MERGING UNIT ON A PC PLATFORM
SYNTHETHIC DATA GENERATION
GENERATION OF INTERNAL SYSTEM CLOCK FOR TIME STAMPING
TVTR1
TVTR2
TVTR3
TCTR1
TCTR2
TCTR3
LLN0
LOGICAL NODES MODEL
GENERATION OF ASN.1 TAGS
GENERATION 802.1 Q FRAME AND ADDING SV PDU TO SV MESSAGE
ETHERNET CARD
PROCESS BUS
PROTECTION
BAY CONTROLLER
Outside the scope of this project and will not be implemented
To be implemented

Figure 5.3: PC based Sensor Node synthetic data generation

The sine wave used to simulate the data for the sampled value transmission is discretised using the sample count value. The Python time module discussed in section 5.2.2 below is utilised to return current time in seconds, and this time is stored in a predefined variable. The "*ticks*" variable shown in Fig. 5.4 is used to store the "*n.t*" (sec) and is used to calculate the current and voltage magnitude values as shown in equation 5.1 below.

The three-phase voltages and currents are computed from equation 5.1 below, and the neutral values for voltage and current are computed from the three-phase values.

The IEC 61850-9-2LE guideline specifies a sampling rate of 80 samples per cycle when operating at 50Hz. This translates to a frequency of 4000 Hz, which is used to generate a sample count number of 0 – 3999 inclusive. The parameter *n* in the equations below, take on these values sequentially.

$$Va = Vin_a * Sin(\omega nt + 0°) \quad (5.1)$$

$$Vb = Vin_b * Sin(\omega nt + 120) \quad (5.2)$$

$$Vc = Vin_c * Sin(\omega nt + 240°) \quad (5.3)$$

$$Vn = Va + Vb + Vc \quad (5.4)$$

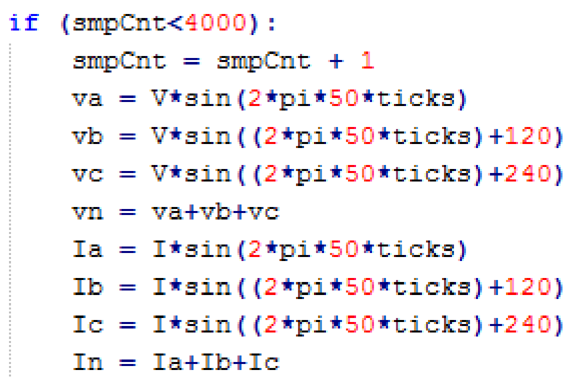
$$Ia = Iin_a * Sin(\omega nt + 0°) \quad (5.5)$$

$$Ib = Iin_b * Sin(\omega nt + 120°) \quad (5.6)$$

$$Ic = Iin_c * Sin(\omega nt + 240°) \quad (5.7)$$

$$In = Ia + Ib + Ic \quad (5.8)$$

In this simulated environment, the parameter *t* is picked up from the built-in time module in Python (as discussed in section 5.2.2) at each increment of the sample count. The peak of the three-phase voltages and currents are computed from the Root Mean Square (RMS) values which the user is initially prompted to input. The neutral voltage and current values are calculated as the sum of the computed phase voltage or current values respectively. **Note**: It is assumed that the system is not faulted.



```
if (smpCnt<4000):
    smpCnt = smpCnt + 1
    va = V*sin(2*pi*50*ticks)
    vb = V*sin((2*pi*50*ticks)+120)
    vc = V*sin((2*pi*50*ticks)+240)
    vn = va+vb+vc
    Ia = I*sin(2*pi*50*ticks)
    Ib = I*sin((2*pi*50*ticks)+120)
    Ic = I*sin((2*pi*50*ticks)+240)
    In = Ia+Ib+Ic
```

Figure 5.4: Sample code of the data generated

### 5.2.2. Time synchronization

In a full real-time implementation of a merging unit, a synchronization signal would be obtained from a 1 Pulse Per Second (1PPS) global time synchronization source such as a Global Positioning System (GPS). The reason for this signal is to mitigate against errors due to drift in the time signal - hence the system is re-synchronized every second, as illustrated in Fig. 5.5 below.

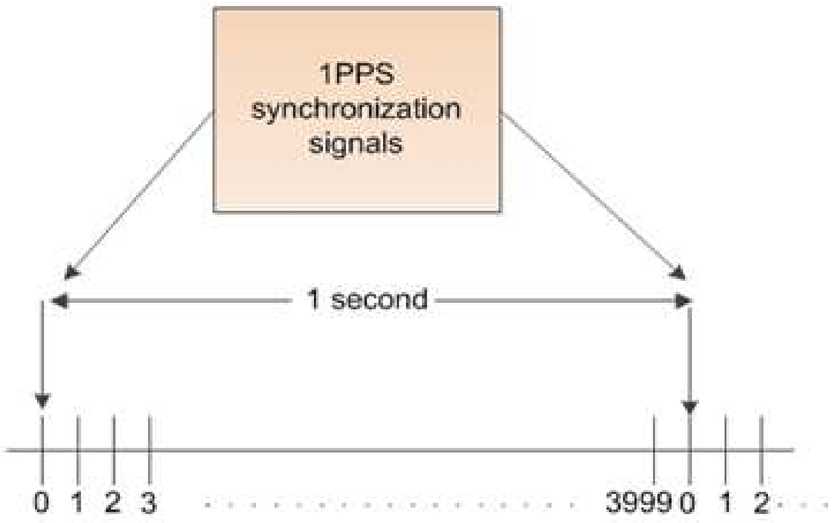


Figure 5.5: Time synchronization for a full real-time implementation of merging unit

Furthermore, the synchronization pulse sets/resets a counter to 0. The sample count is allowed to count up to 3999. The reason why the values are from 0 - 3999 is given in section 5.2.1 above.

**Note:** It is the sample count value (0-3999) that is transmitted within the message data, as it is assumed that the receiving device is also synchronized to the global 1PPS pulses. i.e. entire Wide Area Networks (WANs) are synchronized. This allows for "snapshots" to be taken for the wide area network and data for events within it can be processed or correlated if required.

The above description, as mentioned, is if a full real-time implementation were to be built. In this research project however the generation of the sample count values has been simulated through generating and incrementing an integer *"n"* (referred to in section 5.2.1 above) each time the *"t"* variable increments by a set amount i.e. 1/4000 seconds. The value for the time variable *"t"* is obtained from in-built Python functions which are described below.

The Python time and date module provides various functions related to time. An explanation of the terminology, functions and conversion method utilised in this module can be found on the Python website (http://docs.Python.org). Some of the functions and items that have been used during the various phases of the project development are catalogued in Table 5.1 below with a brief explanation provided for each.

Table 5-1: Example of Python's supported time functions (http://docs.Python.org).

| Function | Explanation |
|---|---|
| Time () | Returns the time as a floating point number expressed in seconds since the 1 June 1970. |
| Clock() | Returns the current CPU time as a floating point number expressed in seconds. This function is mainly used for benchmarking Python or timing algorithms. |
| Gmtime (secs) | Converts a time expressed in seconds since the epoch (1 June 1970) to a time tuple in UTC in which the Daylight Savings Time (DST) flag is always zero. Fractions of a second are ignored. |
| Sleep (secs) | Suspends execution for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time. |

The time module can be imported into any Python code and instantiated whenever it is required. An example using the sleep function is shown below.

```
#       Import time
#       TimeToSleep = time.sleep (number of seconds)
```

In this example, the program will go into sleep mode for the predefined number of seconds. The practical use case is shown in the flowchart in Fig. 5.2, where the program is paused for a number of seconds before resuming. The sleep function can be instantiated with both integer and float numbers. For example if the program has to go to sleep for milliseconds or microseconds this is fractions of a second and has to be detailed as such.

In the next section the Application Protocol Data Unit (APDU) section of the sampled value frame structure as specified in the IEC 61850-9-2 standard and IEC 61850-9-2LE implementation guideline document is discussed, with specific emphasis on the Abstract Syntax Notation (ASN.1) encoding and its practical implementation for the simulated environment.

### 5.2.3. Implementation: Abstract Syntax Notation (ASN.1) tag encoding

From the sampled value frame structure previously discussed in chapter 3 section 3.5.2.3.2 and also referred to in Fig. 5.6 below, we are now concentrating on the data in the Application Protocol Data Unit (APDU) section of the sampled value message frame, with specific emphasis on the encoding for the data. The various fields within the sampled value APDU are encoded through the use of ASN.1 as highlighted in Fig. 5.6 below.

Abstract Syntax Notation One (ASN.1) is an international standard used to define protocols. ASN.1 encoding essentially, refers to the use of tags to convey/indicate information about how the data is sectionalised, and what type of data and the length of the data for each section in the frame. This information is especially important so that the receiving device can decode the data appropriately (Kaliski Jr., 1993). The ASN.1 standard is introduced in chapter 3. The various fields within the sampled value protocol data unit are encoded through the use of Abstract Syntax Notation (ASN.1) as highlighted in Fig. 5.6 below (Maroon box in Fig. 5.6).

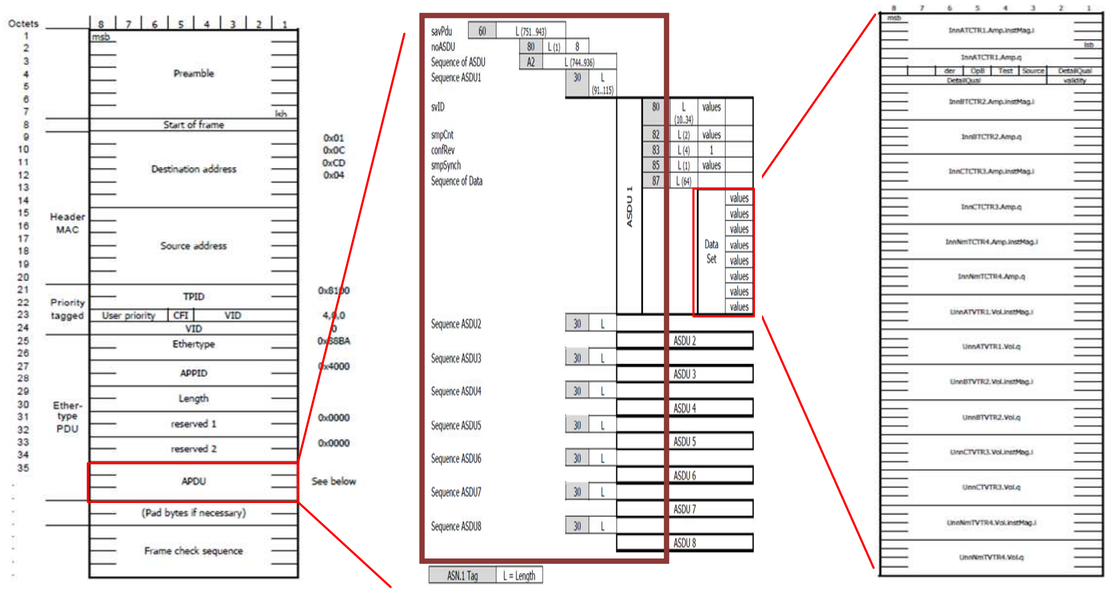


Figure 5.6: Sampled Value (SV) application service data unit (IEC 61850-9-2)

A summary table of the commonly utilised encoding values for sampled value messages is shown in table 5.2 and table 5.3 below.

The ASN.1 tags used to indicate the start of the savPdu is a hexadecimal (Hex) value = 0x60 and the start of the ASDUs is indicated by the value = 0x80 as can be seen on Table 5.2 below.

Table 5-2: Tag indicating the start of the savPdu

| Value octet<br>(Hexadecimal) | Value octet<br>(Binary) | Meaning<br>(Interpretation) |
|---|---|---|
| 60 | 01100000 | Class APPLICATION, Constructed. Tag no. = [0]. IMPLICIT SavPdu Identifier octet. Fieldname “savPdu” |
| 80 | 10000000 | Class CONTEXT-SPECIFIC, Primitive, Tag no = [0]. ]. IMPLICIT INTEGER.  Identifier octet. Fieldname “sv.noASDU” Number of ASDUs which will be concatenated into one APDU. |

The start of the ASDU is then followed by an optional field, the security field which is encoded with the value =0x81 (IEC 61850-9-2LE). The rest of the Hex value tags commonly used for encoding further information in the savPdu section can be seen on Table 5.3 (Falk and Burns, 2001).

Table 5-3: Tags commonly used for the encoding of further information in the savPdu

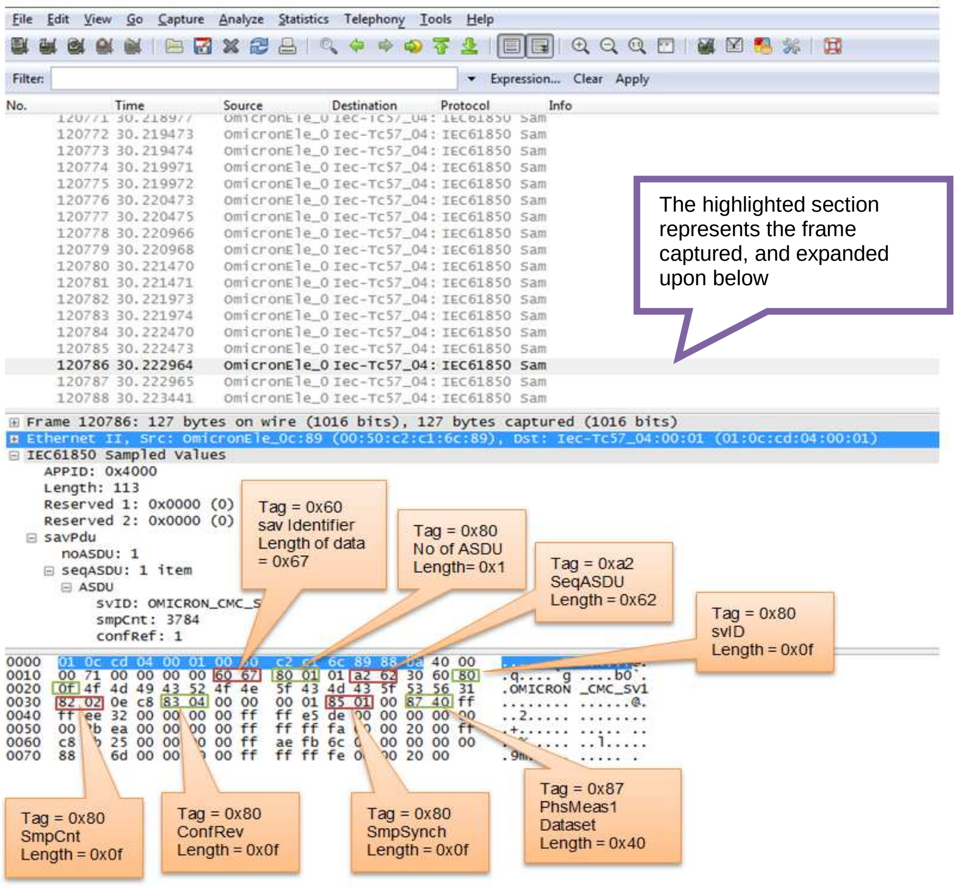
| Value octet<br>(Hexadecimal) | Value octet<br>(Binary) | Meaning<br>(Interpretation) |
|---|---|---|
| 81 | 10000001 | Class CONTEXT-SPECIFIC, Primitive, Tag no. = [1] Security ANY OPTIONAL |
| A2 | 10100010 | Class CONTEXT-SPECIFIC. Constructed Tag no.=[2]. IMPLICIT SEQUENCE OF ASDU. Identifier octet. Fieldname “sv.sequence of ASDU” |
| 30 | 00110000 | Class UNIVERSAL. Constructed. Tag no.=[16]. NOT LISTED IN THE STANDARD 61850-9-2, according to SISCO, “MMS and ASN.1 Encoding”. IA5STRING. |
| 80 | 10000000 | Class CONTEXT-SPECIFIC, Primitive, Tag no. = [0]. IMPLICIT Visible String Identifier octet. Fieldname “sv.ID” |

| | | |
|---|---|---|
| 81 | 10000001 | Class CONTEXT-SPECIFIC, Primitive, Tag no.=[1]. IMPLICIT Visible String Comment: Value form the MSVCB or USVCB |
| 82 | 10000010 | Class CONTEXT-SPECIFIC, Primitive, Tag no.=[2]. IMPLICIT INTEGER. Identifier octet. Fieldname “sv.SmpCnt” |
| 83 | 10000011 | Class CONTEXT-SPECIFIC, Primitive, Tag no.=[3]. IMPLICIT INTEGER. Identifier octet. Fieldname “sv.conRef” |
| 84 | 10000100 | Class CONTEXT-SPECIFIC, Primitive, Tag no. = [4]. IMPLICIT UTC TIME. Comment: RefrTm contains the refresh time of the SV buffer |
| 85 | 10000101 | Class CONTEXT-SPECIFIC, Primitive, Tag no. =[5]. IMPLICIT BOOLEAN. Identifier octet. Fieldname “sv.smpSynch” |
| 86 | 10000110 | Class CONTEXT-SPECIFIC, Primitive, Tag no.=[6]. IMPLICIT OCTECT STRING. |
| 87 | 10000111 | Class CONTEXT-SPECIFIC, Primitive, Tag no.=[7]. IMPLICIT FLOATING POINT DATA. Identifier octet. Fieldname “sv.deqData” |

In Table 5.2 and 5.3 above the meaning of the tag values is provided. The type tags used in the IEC 61850 sampled value protocol data unit enable the receiving device to accurately decode the received message. The ASN.1 tags consist of three elements (Cassel et al., 2000):

- Class and class number
- Implicit or explicit class definition
- Type being tagged.

As shown in Table 5.2 and Table 5.3 two class objects are used within most sampled value messages, a universal class and context-specific class. The universal class is distinguished from other data types as it defines an application independent data type (Cassel et al., 2000). The context-specific class denotes members of a sequence class. The class behaviour for the ASN.1 tags is defined by specifying whether the class is implicitly defined or explicit. The class behaviour defines the means by which the class can be aggregated. A practical example of ASN.1 encoding for sampled value messages is shown in Fig. 5.8 below.


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply
The highlighted section represents the frame captured, and expanded upon below
Tag = 0x60 sav Identifier Length of data = 0x67
Tag = 0x80 No of ASDU Length= 0x1
Tag = 0xa2 SeqASDU Length = 0x62
Tag = 0x80 svID Length = 0x0f
Tag = 0x80 SmpCnt Length = 0x0f
Tag = 0x80 ConfRev Length = 0x0f
Tag = 0x80 SmpSynch Length = 0x0f
Tag = 0x87 PhsMeas1 Dataset Length = 0x40

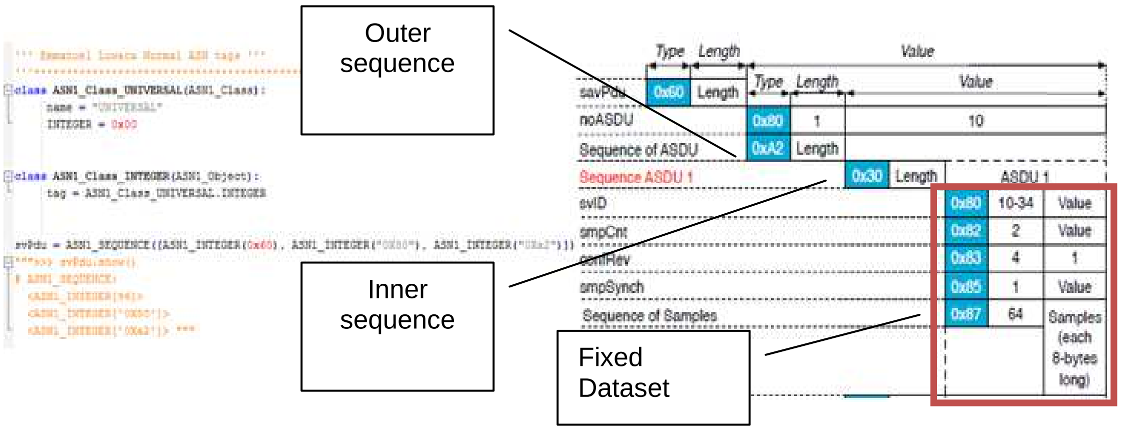
```
No.          Time        Source          Destination     Protocol    Info
120771 30.218977   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120772 30.219473   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120773 30.219474   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120774 30.219971   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120775 30.219972   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120776 30.220473   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120777 30.220475   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120778 30.220966   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120779 30.220968   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120780 30.221470   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120781 30.221471   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120782 30.221973   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120783 30.221974   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120784 30.222470   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120785 30.222473   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120786 30.222964   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120787 30.222965   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam
120788 30.223441   OmicronEle_0 Iec-Tc57_04: IEC61850 Sam

⊞ Frame 120786: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits)
⊞ Ethernet II, Src: OmicronEle_0c:89 (00:50:c2:c1:6c:89), Dst: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu
      noASDU: 1
    ⊟ seqASDU: 1 item
      ⊟ ASDU
          svID: OMICRON_CMC_S
          smpCnt: 3784
          confRef: 1

0000  01 0c cd 04 00 01 00 50  c2 c1 6c 89 88 ba 40 00  ........ ......@.
0010  00 71 00 00 00 00 60 67  80 01 01 a2 62 30 60 80  .q....`g ....b0`.
0020  0f 4f 4d 49 43 52 4f 4e  5f 43 4d 43 5f 53 56 31  .OMICRON _CMC_SV1
0030  82 02 0e c8 83 04 00 00  00 01 85 01 00 87 40 ff  ........ ......@.
0040  ff ee 32 00 00 00 00 ff  ff e5 de 00 00 00 00 00  ..2..... ........
0050  00 [illegible]b ea 00 00 00 00 ff  ff ff fa [illegible] 00 20 00 ff  .+...... ..... ..
0060  c8 [illegible] 25 00 00 [illegible] 00 ff  ae fb 6c [illegible] 00 00 00 00  ........ ..l.....
0070  88 [illegible] 6d 00 00 [illegible] 00 ff  ff ff fe [illegible] [illegible] 20 00     .9m..... ..... .
```

Figure 5.7: Example of ASN.1 tags used for sampled value messages

Highlighted in Fig. 5.7 are the ASN.1 tags for the sampled value protocol data unit. Explanation for each tag highlighted in Fig. 5.7 is provided in Table 5.2 and Table 5.3 above.

During the development of this project various methods were tested to try and encode the ASN.1 tags onto the sampled value messages. Two libraries that can be used for sampled value encoding were examined to provide the ASN.1 tagging for the sampled value messages published from the sensor node. The PyASN library and Scapy ASN.1 tagging module were examined. The challenge with both libraries was the limited documentation that is provided. The Scapy library, as the library that is utilised for the sampled value frame construction discussed later in this chapter, was persisted with.

The biggest challenge with ASN.1 tags on the savPdu (see Chapters 3 and 4 for sampled value frame structure) is that there is encoding for the outer and the inner fields of the sampled value protocol data unit. As an example, there is the outer ASN.1 tag encoding for the savPdu and an inner ASN.1 tag for the Application Service Data Unit (ASDU), and possibly another inner ASDU, depending on how many application services data units are implemented in the sampled value frame. Fig. 5.8 shows an example of ASN.1 encoding using the Scapy library.



```
''' Emmanuel Lowara Normal ASN tags '''
''***************************************
class ASN1_Class_UNIVERSAL(ASN1_Class):
    name = "UNIVERSAL"
    INTEGER = 0x00

class ASN1_Class_INTEGER(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.INTEGER

svPdu = ASN1_SEQUENCE([ASN1_INTEGER(0x60), ASN1_INTEGER("0X80"), ASN1_INTEGER("0XA2")])
"""">>> svPdu.show()
# ASN1_SEQUENCE:
  <ASN1_INTEGER[96]>
  <ASN1_INTEGER['0X80']>
  <ASN1_INTEGER['0XA2']> """
```

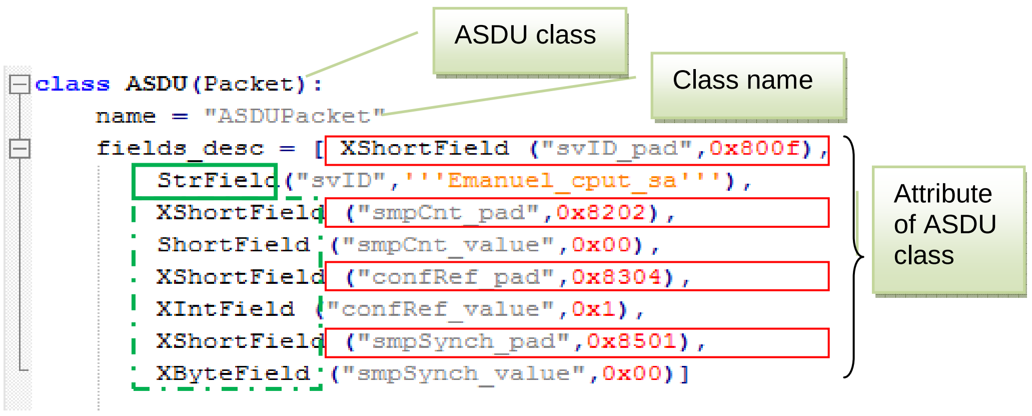
Figure 5.8: An example of the ASN.1 encoding using the Scapy library

As shown in Fig. 5.8, Scapy provides a way to encode and decode ASN.1 tags.  Scapy uses an object oriented programming engine “Scapy ASN.1 engine” to link objects and their tags. Scapy however is much more lax than what an ASN.1 parser should be (Biondi, 2009), and leeway is given with respect to certain constraints.

#### 5.2.3.1. Abstract Syntax Notation (ASN.1) encoding: challenges encountered

The main challenge encountered with using the Scapy ASN.1 engine for sampled value encoding involved the use of ASN1_Sequences within sequences. The aggregation of one sequence class to another sequence class was not properly handled by the Scapy library. This problem was escalated by the fact that limited Scapy ASN.1 engine documentation was available. The alternative was to hard-code the ASN.1 tags into the various fields that required tagging. This was a possibility open to us because the dataset recommended in the implementation guideline documentation is for a fixed dataset. The packet class model used for frame generation within Scapy provides a good platform for this kind of hard-coding of ASN.1 tags. Fig. 5.9 below shows an example of how the ASN.1 encoding is performed for the Application Service Data Unit (ASDU) class.

Highlighted in red in Fig. 5.9 are the ASN.1 tags for the various fields. For this development the data structure framework provided within Scapy is leveraged. Highlighted in green (dashed box) in Fig. 5.10 are the field data types.



```
class ASDU(Packet):
     name = "ASDUPacket"
     fields_desc = [ XShortField ("svID_pad",0x800f),
         StrField("svID",'''Emanuel_cput_sa'''),
         XShortField ("smpCnt_pad",0x8202),
         ShortField ("smpCnt_value",0x00),
         XShortField ("confRef_pad",0x8304),
         XIntField ("confRef_value",0x1),
         XShortField ("smpSynch_pad",0x8501),
         XByteField ("smpSynch_value",0x00)]
```

Figure 5.9: An example of a hard-coded ASN.1 encoding using the Scapy library

Two different data type structures are shown in Fig. 5.9. The difference between the two types is that one is the standard type and others are hexadecimal representations e.g. XByteField and XShortField. The length of a ByteField and that of a XByteField are exactly the same - the representation is simply different. Some of the data types supported and used in this development are catalogued in table 5.4 below.

Table 5-4: Data types supported in Scapy (Biondi, 2009)

| Simple Type | Enumerations | Strings |
|---|---|---|
| ByteField | EnumField | StrField |
| XByteField | CharEnumField | StrLenField |
| ShortField | BitEnumField | StrFixedLenField |
| LEShortField | ShortEnumField | StrNullField |
| XShortField | LEShortEnumField | StrStopField |
| X3BytesField | ByteEnumField | |
| IntField | IntEnumField | |
| SignedIntField | SignedIntEnumField | |
| LEIntField | LEIntEnumField | |
| LESignedIntField | XShortEnumField | |
| XIntField | | |
| LongField | | |
| XLongField | | |
| LELongField | | |
| IEEEFloatField | | |
| IEEEDoubleField | | |

| | | |
|---|---|---|
| BCDFloatField<br>BitField<br>XBitField | | |

The next section will discuss the implementation of the complete sampled value frame and its transmission.

### 5.2.4. Implementation of Sampled Value (SV) Frame Structure and it's transmission

The dashed green box highlighted in Fig. 5.10 below, represents the functions required for the implementation of the sampled value frame and its transmission – inclusive of the integration of the ASN.1 tags for the various fields within the sampled value protocol data, and the packaging of the data.

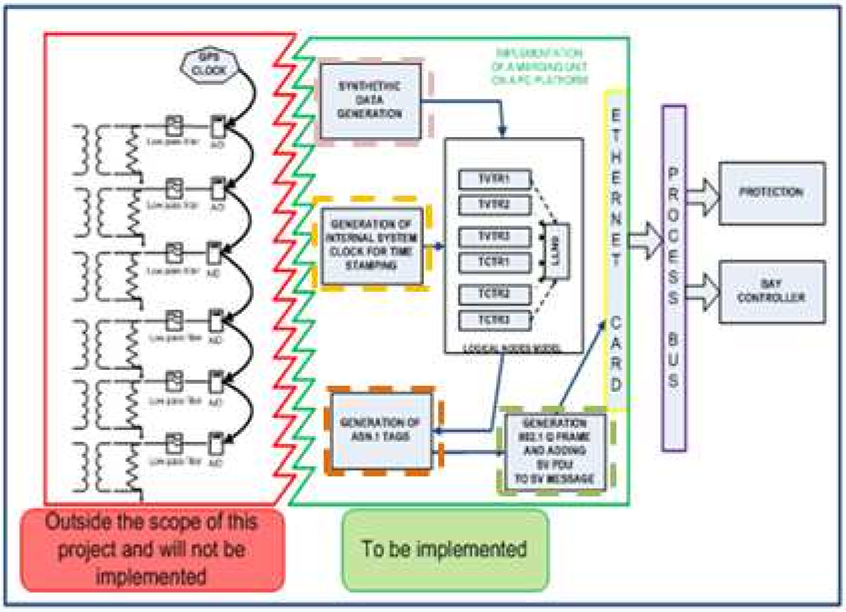

GPS CLOCK
Low pass filter
A/D
IMPLEMENTATION OF A MERGING UNIT ON A PC PLATFORM
SYNTHETIC DATA GENERATION
GENERATION OF INTERNAL SYSTEM CLOCK FOR TIME STAMPING
TVTR1
TVTR2
TVTR3
TCTR1
TCTR2
TCTR3
LLN0
LOGICAL NODES MODEL
GENERATION OF ASN.1 TAGS
GENERATION 802.1 Q FRAME AND ADDING SV PDU TO SV MESSAGE
ETHERNET CARD
PROCESS BUS
PROTECTION
BAY CONTROLLER
Outside the scope of this project and will not be implemented
To be implemented

Figure 5.10: PC based Sensor Node generation of SV frame

The structure in the SV Frame is defined in the IEC 61850-9-2 and discussed in section 5.2.3. The IEC 61850 sampled value sending device has to define the following fields: [Source and Destination, Virtual Area Network (VLAN), EtherType, Application protocol data unit (APPID), Length field, reserved fields, APDU]. The object model of the developed sampled value frame generator is depicted in Fig. 5.11.

**Note**: The left-hand of Fig. 5.11 represents the theoretical frame structure as dictated by the IEC 61850-9-2 standard and also the IEC 61850-9-2LE guideline. The right-hand side of Fig. 5.11 represents the classes that were utilised to implement the different sections of the frame structure. Certain of the classes are available as part of

Scapy (represented as the Ether and the dot1Q classes in Fig. 5.11). The other classes on the right-hand side of the figure do not exist in Scapy, and had to be developed specifically for this research project. These classes are those associated with implementing the following sections of the frame format:

i. Packet
ii. Gen
iii. PDU
iv. Length
v. Reserved Field x2
vi. APPID

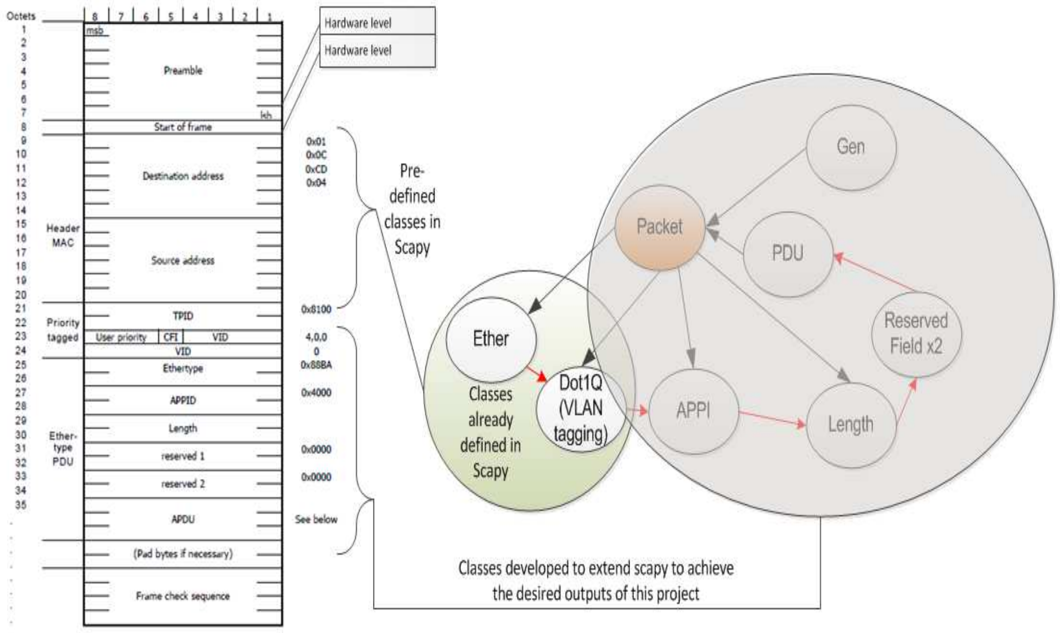


Figure 5.11: Developed Sampled Value object model

The Scapy networking libraries on the objected oriented programming platform is used to model the ISO/IEC 8802-3 sampled value frame structure as defined in the IEC 61850-9-2 standard. The Scapy library provides support for low-level networking and provides an object oriented framework for layer 2 custom packet crafting using certain classes. Each layer in a crafted layer 2 frame is a subclass of the Packet class. As shown in Fig. 5.11 each field in the sampled value frame that is not performed at the hardware level is represented as a class. These classes have different attributes and methods.

Scapy provides a hierarchical, layered packet class modelling structure, with the packet class as the base class.  Primarily all the logic behind layer manipulation in a crafted frame is held by the Packet class as shown in Fig. 5.12 below (Biondi, 2009).

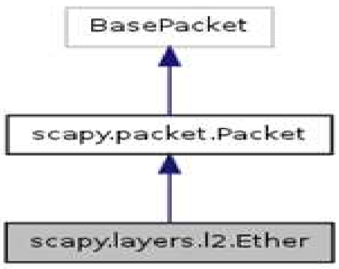


Figure 5.12: UML diagram of the development of a layer 2 packet (Biondi, 2009)

A layer 2 Packet class is shown in Fig. 5.12 above. The packet class inherits from the layer 2 Ethernet class. The layer 2 Ethernet class has a source address, destination address and a type - all inherited from different subclasses as shown in Fig. 5.13. Fig.5.13 below shows the Unified Modelling Language (UML) diagram of the source and destination addresses.

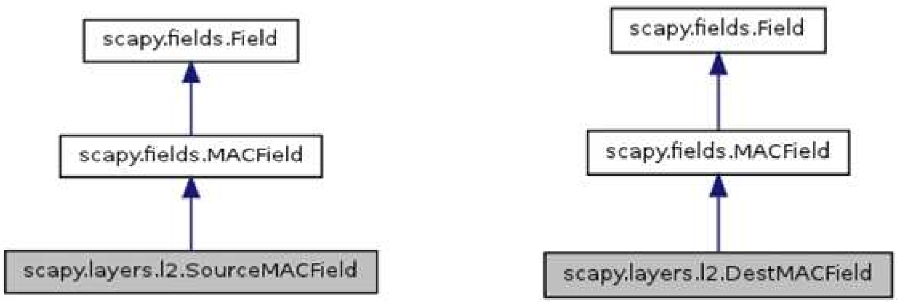
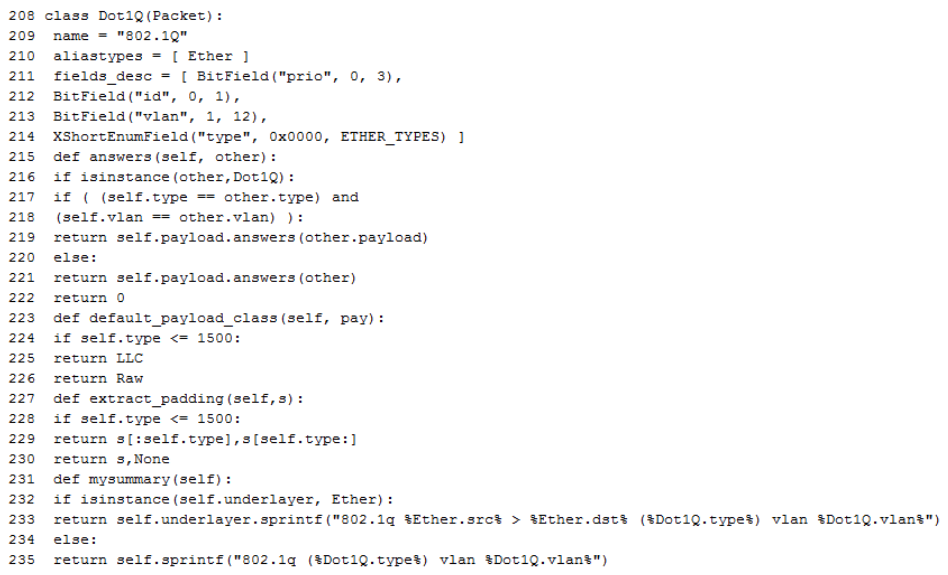


Figure 5.13: UML diagram of the source and destination classes (Biondi, 2009)

The next field in the ISO/IEC 8802-3 sampled value frame is the Virtual Local Area Network (VLAN) field.

#### 5.2.4.1. Sampled value VLAN tagging

Virtual Local Area Network (VLAN) and VLAN tagging is defined in the IEEE 802.1Q standard and is used to aid in the process of traffic routing in networks (Zereneh, 2006). VLAN tagging is applied by the sending node in an IEC 61850 sampled value transmission, so that sampled value traffic can be prioritised over less critical information in routers and switches. VLAN tagging is supported on Scapy and is defined in the Dot1Q Packet class as shown in Fig. 5.14.



```
class Dot1Q(Packet):
name = "802.1Q"
aliastypes = [ Ether ]
fields_desc = [ BitField("prio", 0, 3),
BitField("id", 0, 1),
BitField("vlan", 1, 12),
XShortEnumField("type", 0x0000, ETHER_TYPES) ]
def answers(self, other):
if isinstance(other,Dot1Q):
if ( (self.type == other.type) and
(self.vlan == other.vlan) ):
return self.payload.answers(other.payload)
else:
return self.payload.answers(other)
return 0
def default_payload_class(self, pay):
if self.type <= 1500:
return LLC
return Raw
def extract_padding(self,s):
if self.type <= 1500:
return s[:self.type],s[self.type:]
return s,None
def mysummary(self):
if isinstance(self.underlayer, Ether):
return self.underlayer.sprintf("802.1q %Ether.src% > %Ether.dst% (%Dot1Q.type%) vlan %Dot1Q.vlan%")
else:
return self.sprintf("802.1q (%Dot1Q.type%) vlan %Dot1Q.vlan%")
```

Figure 5.14: Scapy Dot1Q Class with the defaults values (Biondi, 2009)

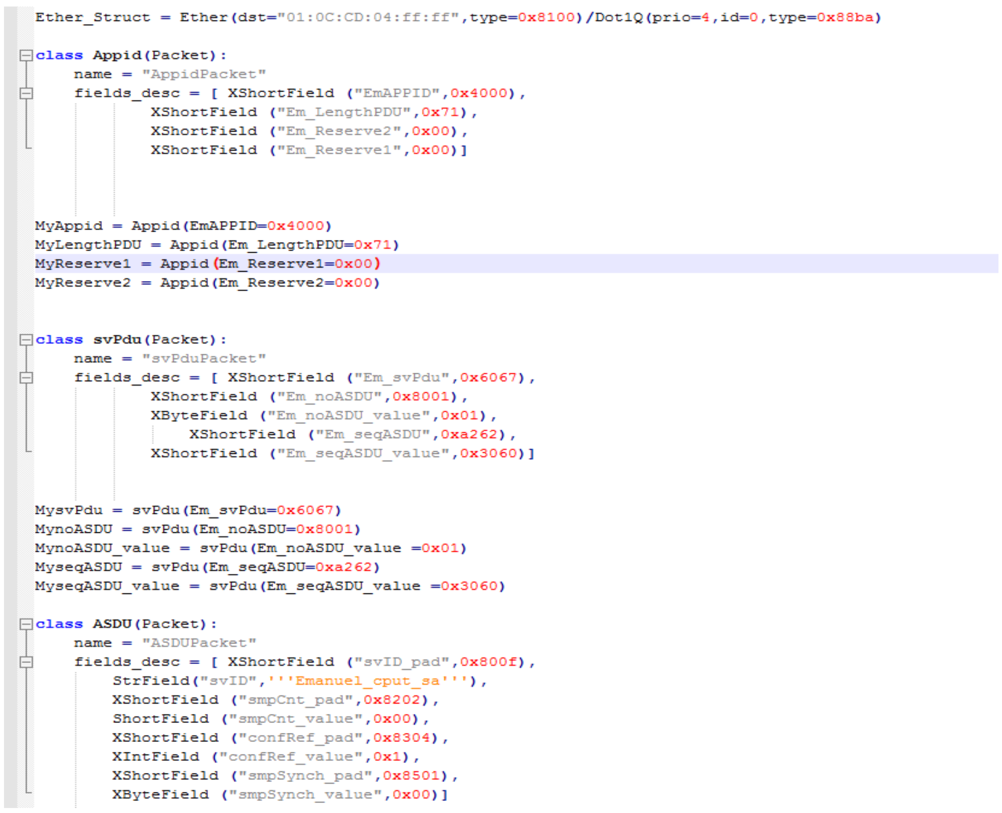
VLAN tagging is an Ethernet layer function and is specified as such in the class definition. The VLAN tagging class has to be instantiated with the appropriate priority, VLAN identifier and VLAN type. An example of how the VLAN tagging class can be instantiated and aggregated to the packet class is shown in Fig.5.15 below:

```
Test = Ether(dst="01:0C:CD:04:ff:ff",type=0x8100)/Dot1Q(prio=4,id=0,type=0x88ba)
```

Figure 5.15: Instantiating the Dot1Q class and aggregating into the class packet

#### 5.2.4.2. Completing the SV packet structure

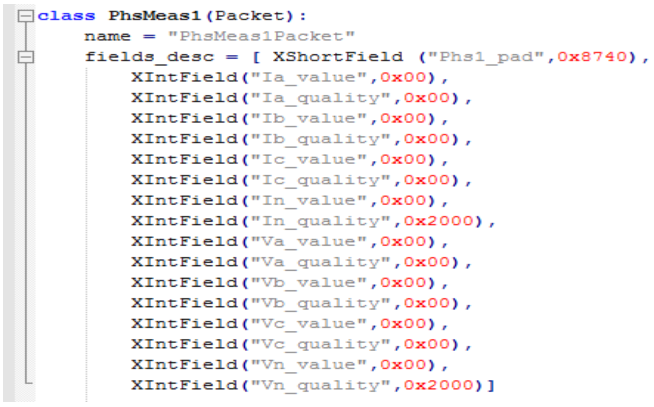
The section that follows the source address, destination address and VLAN tagging in the sampled value structure is the Application Protocol Identifier (APPID), Length, two reserved fields, and the Application Protocol Data Unit (APDU) of the sampled value structure. None of these classes are pre-defined in Scapy and had to be scripted with assistance from the chapter on custom development as referenced in the Scapy Manual (Biondi, 2009). The data types and data structures supported in Scapy and the Python programming language are used in the different classes to define the various fields required in the classes scripted, as shown in Fig. 5.16.



```
Ether_Struct = Ether(dst="01:0C:CD:04:ff:ff",type=0x8100)/Dot1Q(prio=4,id=0,type=0x88ba)

class Appid(Packet):
    name = "AppidPacket"
    fields_desc = [ XShortField ("EmAPPID",0x4000),
                    XShortField ("Em_LengthPDU",0x71),
                    XShortField ("Em_Reserve2",0x00),
                    XShortField ("Em_Reserve1",0x00)]


MyAppid = Appid(EmAPPID=0x4000)
MyLengthPDU = Appid(Em_LengthPDU=0x71)
MyReserve1 = Appid(Em_Reserve1=0x00)
MyReserve2 = Appid(Em_Reserve2=0x00)


class svPdu(Packet):
    name = "svPduPacket"
    fields_desc = [ XShortField ("Em_svPdu",0x6067),
                    XShortField ("Em_noASDU",0x8001),
                    XByteField ("Em_noASDU_value",0x01),
                        XShortField ("Em_seqASDU",0xa262),
                    XShortField ("Em_seqASDU_value",0x3060)]


MysvPdu = svPdu(Em_svPdu=0x6067)
MynoASDU = svPdu(Em_noASDU=0x8001)
MynoASDU_value = svPdu(Em_noASDU_value =0x01)
MyseqASDU = svPdu(Em_seqASDU=0xa262)
MyseqASDU_value = svPdu(Em_seqASDU_value =0x3060)

class ASDU(Packet):
    name = "ASDUPacket"
    fields_desc = [ XShortField ("svID_pad",0x800f),
                StrField("svID",'''Emanuel_cput_sa'''),
                XShortField ("smpCnt_pad",0x8202),
                ShortField ("smpCnt_value",0x00),
                XShortField ("confRef_pad",0x8304),
                XIntField ("confRef_value",0x1),
                XShortField ("smpSynch_pad",0x8501),
                XByteField ("smpSynch_value",0x00)]
```

Figure 5.16: Example of the sampled value classes developed

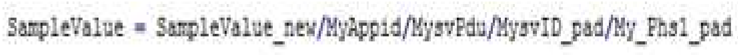
The SV PhsMeas1 class (which represents the full dataset inclusive of each measurement's quality attribute) is also implemented using object oriented programming, as is shown in Fig. 5.17 below.



```
class PhsMeas1(Packet):
    name = "PhsMeas1Packet"
    fields_desc = [ XShortField ("Phs1_pad",0x8740),
        XIntField("Ia_value",0x00),
        XIntField("Ia_quality",0x00),
        XIntField("Ib_value",0x00),
        XIntField("Ib_quality",0x00),
        XIntField("Ic_value",0x00),
        XIntField("Ic_quality",0x00),
        XIntField("In_value",0x00),
        XIntField("In_quality",0x2000),
        XIntField("Va_value",0x00),
        XIntField("Va_quality",0x00),
        XIntField("Vb_value",0x00),
        XIntField("Vb_quality",0x00),
        XIntField("Vc_value",0x00),
        XIntField("Vc_quality",0x00),
        XIntField("Vn_value",0x00),
        XIntField("Vn_quality",0x2000)]
```

Figure 5.17: PhsMeas1 frame

The classes are then instantiated with the correct values and concatenated with other classes to implement the complete sampled value frame structure as shown in Fig. 5.18.



```
SampleValue = SampleValue_new/MyAppid/MysvPdu/MysvID_pad/My_Phs1_pad
```

Figure 5.18: sampled value frame concatenation

The complete sampled value frame developed can be viewed using the *show* function in Python and is presented in Fig. 5.19.

```
SampleValue.show()
###[ Ethernet ]###
  dst= 01:0C:CD:04:ff:ff
  src= 00:00:00:00:00:00
  type= 0x8100
###[ 802.1Q ]###
     prio= 4
     id= 0
     vlan= 1
     type= 0x88ba
###[ Appid ]###
        EmAPPID= 0x4000
        Em_LengthPDU= 0x71
        Em_Reserve2= 0x0
        Em_Reserve1= 0x0
###[ svPdu ]###
           Em_svPdu= 0x6067
           Em_noASDU= 0x8001
           Em_noASDU_value= 0x1
           Em_seqASDU= 0xa262
           Em_seqASDU_value= 0x3060
###[ ASDU ]###
              svID_pad= 0x800f
              svID= 'Emmanuel_cput_e'
              smpCnt_pad= 0x8202
              smpCnt_value= 0
              confRef_pad= 0x8304
              confRef_value= 0x1
              smpSynch_pad= 0x8501
              smpSynch_value= 0x0
###[ PhsMeas1Dataset ]###
                 Phs1_pad= 0x8740
                 Ia_value= 0x0
                 Ia_quality= 0x0
                 Ib_value= 0x0
```

```
Ib_quality= 0x0
Ic_value= 0x0
Ic_quality= 0x0
In_value= 0x0
In_quality= 0x2000
Va_value= 0x0
Va_quality= 0x0
Vb_value= 0x0
Vb_quality= 0x0
Vc_value= 0x0
Vc_quality= 0x0
Vn_value= 0x0
Vn_quality= 0x2000
```

Figure 5.19: Presentation of the developed sampled value structure

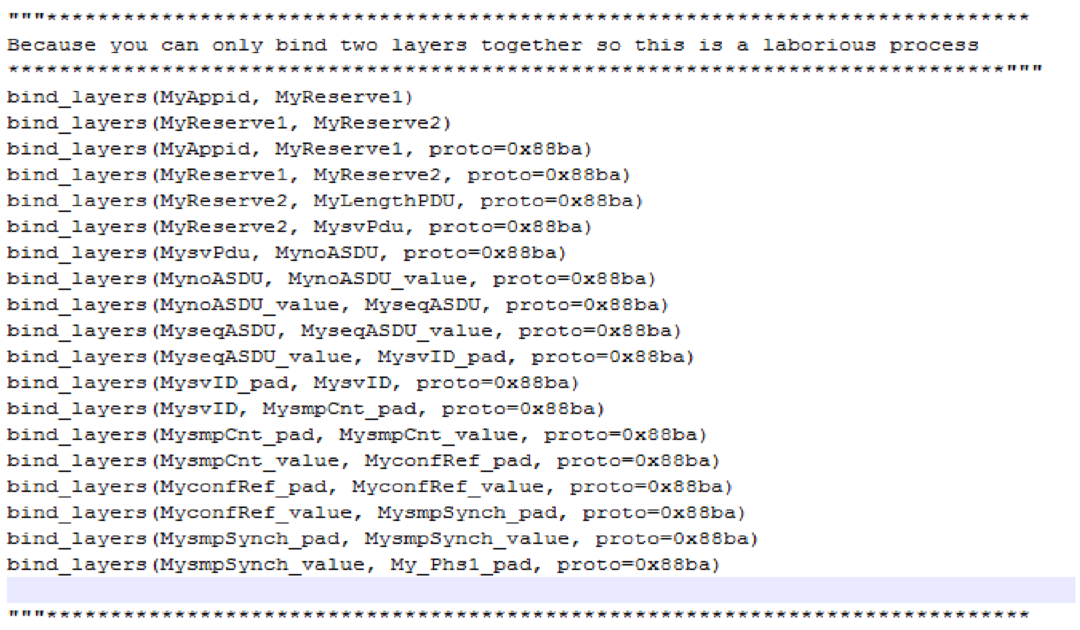
The PhsMeas1Dataset shown in Fig. 5.19 is used to store the measurement values and is used during the transmission process to publish the measurements. The PhsMeas1 class is instantiated and populated with the appropriate values as shown below.

*SampleValue=SampleValue_new/MyAppid/MysvPdu/ASDU(smpCnt_value =smpCnt)/PhsMeas1(Ia_value =Ia, Ib_value = Ib, Ic_value =Ic, In_value = In, va_value =va, vb_value = vb, vc_value =vc, vn_value = vn)*

#### 5.2.4.3. Binding the fields of the generated sampled value frame

After a socket has been created, it is nameless, even if it has been associated to a descriptor. Before the socket can be utilised, it must be associated with the appropriate protocol address so that the receiving device can process it accordingly (Tiponut, 2001)

On Scapy, each of the generated classes for the various fields of the sampled value frame have to be connected together for the generated packet to be communicated properly and associated with the appropriate protocol as shown in Fig. 5.20. It is worth mentioning that the Scapy library's binding function only allows two fields to be passed to it at a time.



```
"""*******************************************************************************
Because you can only bind two layers together so this is a laborious process
*******************************************************************************"""
bind_layers(MyAppid, MyReserve1)
bind_layers(MyReserve1, MyReserve2)
bind_layers(MyAppid, MyReserve1, proto=0x88ba)
bind_layers(MyReserve1, MyReserve2, proto=0x88ba)
bind_layers(MyReserve2, MyLengthPDU, proto=0x88ba)
bind_layers(MyReserve2, MysvPdu, proto=0x88ba)
bind_layers(MysvPdu, MynoASDU, proto=0x88ba)
bind_layers(MynoASDU, MynoASDU_value, proto=0x88ba)
bind_layers(MynoASDU_value, MyseqASDU, proto=0x88ba)
bind_layers(MyseqASDU, MyseqASDU_value, proto=0x88ba)
bind_layers(MyseqASDU_value, MysvID_pad, proto=0x88ba)
bind_layers(MysvID_pad, MysvID, proto=0x88ba)
bind_layers(MysvID, MysmpCnt_pad, proto=0x88ba)
bind_layers(MysmpCnt_pad, MysmpCnt_value, proto=0x88ba)
bind_layers(MysmpCnt_value, MyconfRef_pad, proto=0x88ba)
bind_layers(MyconfRef_pad, MyconfRef_value, proto=0x88ba)
bind_layers(MyconfRef_value, MysmpSynch_pad, proto=0x88ba)
bind_layers(MysmpSynch_pad, MysmpSynch_value, proto=0x88ba)
bind_layers(MysmpSynch_value, My_Phs1_pad, proto=0x88ba)

"""*******************************************************************************
```

Figure 5.20: Binding the various fields of the generated sampled value message

### 5.2.4.4. Transmitting the sampled value frame

Scapy has two methods of sending packets, depending on whether the packet being transmitted as a layer 2 or a layer 3 packet (Biondi, 2003). The Send () function is used for transmitting layer 3 packets and sendp () is used for sending layer 2 packets (Online__www.secdev.org_projects_scapy_doc_usage) such as those packets for sampled values. These two functions are used for simplex transmission, but other functions have to be used when half-duplex and full-duplex is required. The last part of this process is the transmission of the packet. This is achieved by inserting the sampled value packet (i.e. the sections that have been bound and concatenated) into the sendp () function and specifying the interface (*iface* – see Appendix G) across which the packet will be sent. At the completion of this insertion, the transmission of the packet happens automatically. The Python code required to achieve the above is the following:

Sendp(sampledvalue, iface= “eth0”)

In the sections to follow, practical experimentation is done which necessitates the capture of frames from the data network. The captures will be used to verify functional behaviour and to compare the commercial device output and the PC platform output against each other, and against the frame format as specified in the IEC 61850-9-2 and IEC 61850-9-2LE. The capturing of the data networking packets will require the use of Wireshark. Wireshark and the experimentation which was done, is commented on in the sections to follow.

## 5.3. Wireshark

Wireshark is an open-source data network packet analyser, which was originally developed by Gerald Combs, a computer science graduate of the University of Missouri at Kansas City (Sanders, 2000). The first version of Wireshark was released in 1998, as Ethereal under the GNU Public License (GPL). Wireshark monitors and identifies network traffic on a specified network interface card and displays the captured packet data with as much detail as possible.

Wireshark utilises the Winpcap/Libpcap link library to interface the user application level and the lower level layers such as Network Interface Card (NIC) in a personal computer, and Winpcap provides its own networking driver that allows connection from the user application level to the physical layer as shown in Fig. 5.21.

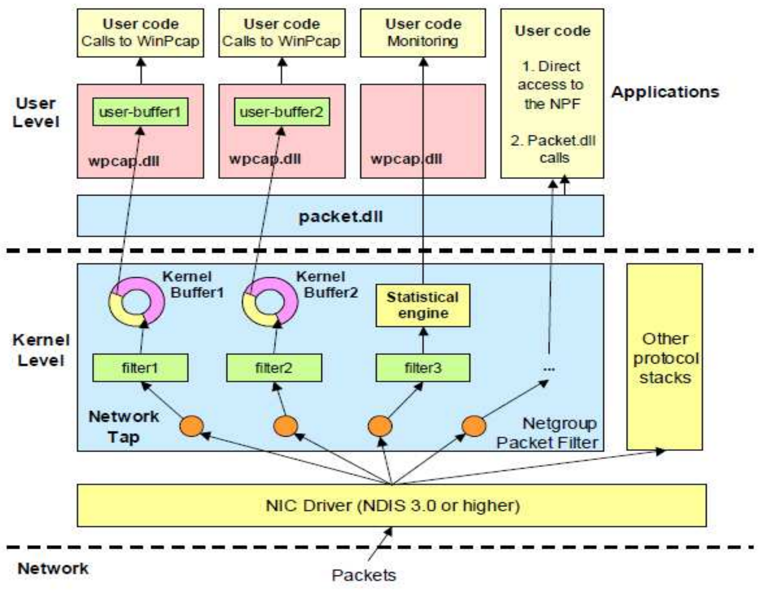
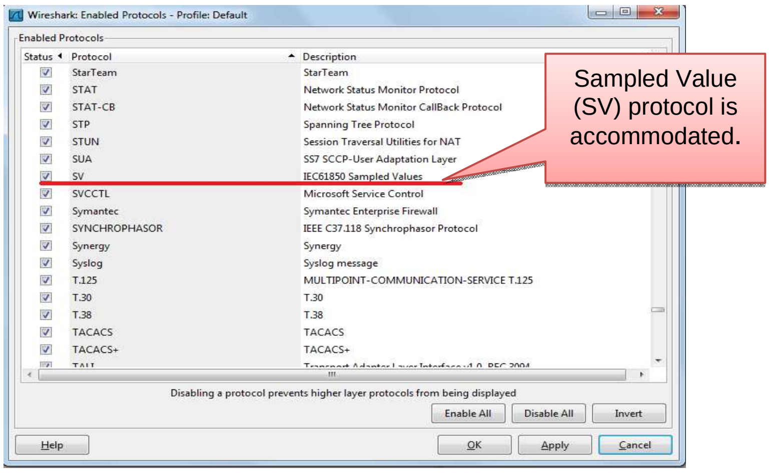


Figure 5.21: Winpcap and Netgroup Packet Filter (Risso and Degioanni, 2005)

### 5.3.1. Utilising Wireshark

Wireshark supports and is able to dissect a multitude of protocol (Sanders, 2012), however support for the protocol monitored should always be verified and enabled on the Wireshark version which is being utilised. If the protocol(s) being monitored are not supported or enabled on the Wireshark version utilised, Wireshark will not be able to provide a through decomposition of the captured packets. In Fig. 5.23 below the Wireshark supported protocols are catalogued, and the fact that the sampled value protocol is one of those accommodated is highlighted by underlining the entry in Fig. 5.22 below.


Wireshark: Enabled Protocols - Profile: Default

Enabled Protocols

| Status | Protocol | Description |
|---|---|---|
| ✓ | StarTeam | StarTeam |
| ✓ | STAT | Network Status Monitor Protocol |
| ✓ | STAT-CB | Network Status Monitor CallBack Protocol |
| ✓ | STP | Spanning Tree Protocol |
| ✓ | STUN | Session Traversal Utilities for NAT |
| ✓ | SUA | SS7 SCCP-User Adaptation Layer |
| ✓ | SV | IEC61850 Sampled Values |
| ✓ | SVCCTL | Microsoft Service Control |
| ✓ | Symantec | Symantec Enterprise Firewall |
| ✓ | SYNCHROPHASOR | IEEE C37.118 Synchrophasor Protocol |
| ✓ | Synergy | Synergy |
| ✓ | Syslog | Syslog message |
| ✓ | T.125 | MULTIPOINT-COMMUNICATION-SERVICE T.125 |
| ✓ | T.30 | T.30 |
| ✓ | T.38 | T.38 |
| ✓ | TACACS | TACACS |
| ✓ | TACACS+ | TACACS+ |
| ✓ | TALI | Transport Adapter Layer Interface v1.0, RFC 3094 |

Disabling a protocol prevents higher layer protocols from being displayed

Figure 5.22: The Wireshark supported and enabled protocols

Wireshark supports, and is capable of providing a complete breakdown of sampled value message structure, however on new installations it is often required for this protocol support to be enabled, as it is not enabled by default. For monitoring sampled values messages, Wireshark version 1.4 or higher is recommended as the various bug patches to the earlier Wireshark versions would have been applied (Online: https://wireshark.org). Catalogued below are some of the bugs related to ASN.1 decoding found in earlier versions of Wireshark that have been rectified in the later versions of Wireshark (Online: https://www.wireshark.org/lists/wireshark-bugs/). The earlier versions of Wireshark could not decode the ASN.1 tags in the sampled value messages for the fields listed below, hence causing the bugs mentioned.

- datSet (0x81), - Dataset ASN.1 tag
- refrTm (0x84), - Refresh Time ASN.1 tag
- smpRate (0x86), - Sample rate time ASN.1 tag
- smpMod (0x88) - Sample value modification ASN.1 tag
- Data/Payload (0x87) decoded as PhsMeas as recommended in the IEC61850-9-2 LE document.

In section 5.4.1 a practical experiment is conducted on the PC-based platform to simulate the sampled value messages from a merging unit. In section 5.4.2 a practical experiment is conducted using a commercial device. For the two experiments conducted, traffic from the data network during the experiments will be captured and analysed.

## 5.4. Practical experimentation

### 5.4.1. Practical experimentation: Using the PC-based sensor node which was developed

The test setup used for verification of sampled value message published from the developed sensor node, over the network is depicted Fig. 5.23 below.

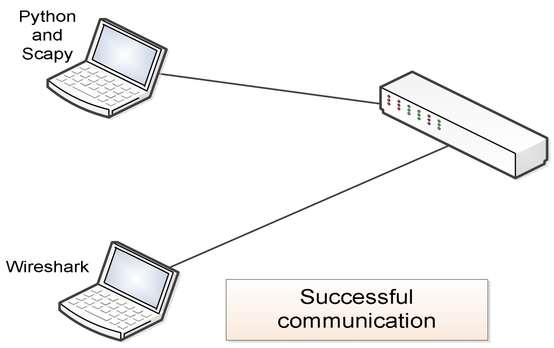
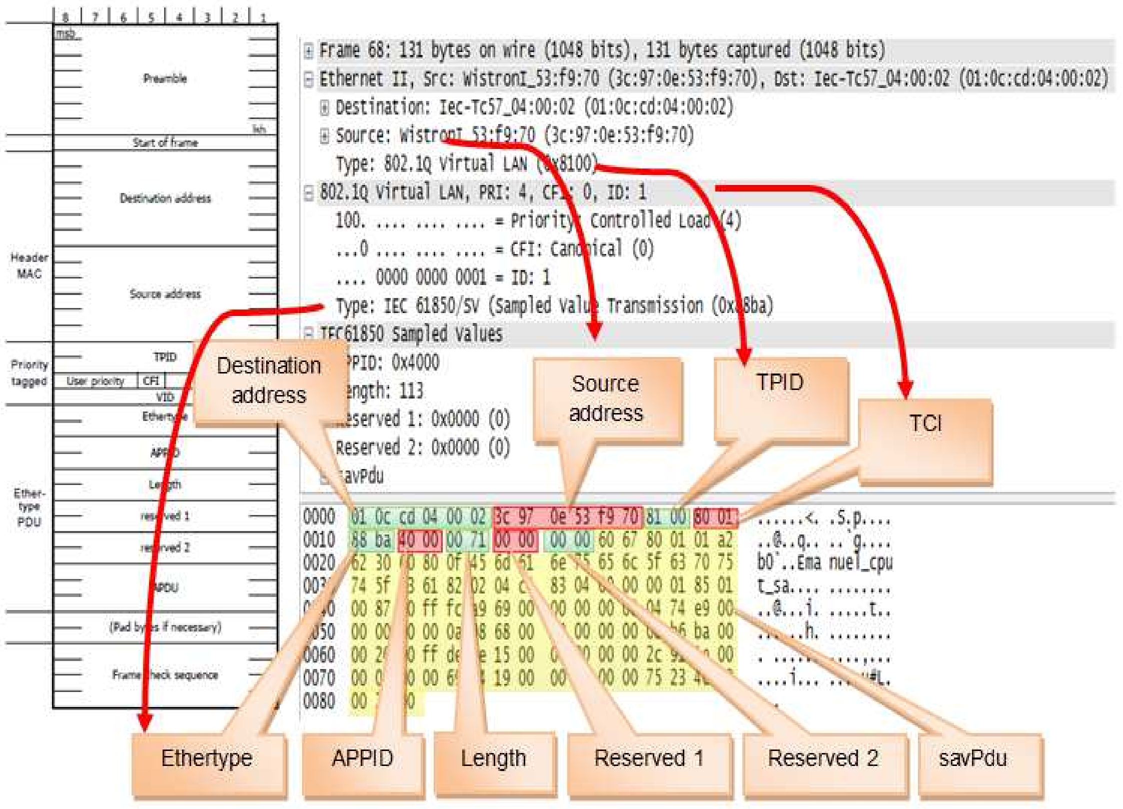


Figure 5.23: Setup for PC-based sensor node sampled values testing

On the test setup shown in Fig. 5.23, two personal computers are used. The first PC has a Linux Ubuntu operating system with the Python script used to publish sampled values. The second is a Microsoft Windows 7 and has the Wireshark program capturing the network packets. This test setup was successful in publishing sampled values and the frames captured on Wireshark are shown in Fig. 5.24 below.



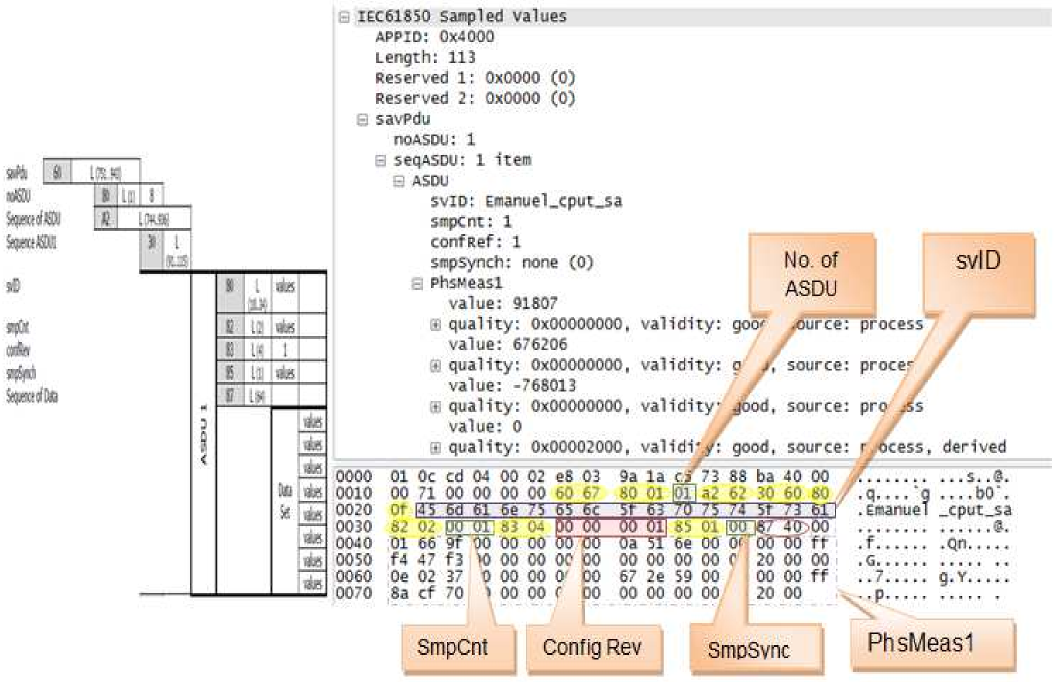
```
⊞ Frame 68: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
⊟ Ethernet II, Src: WistronI_53:f9:70 (3c:97:0e:53:f9:70), Dst: Iec-Tc57_04:00:02 (01:0c:cd:04:00:02)
  ⊞ Destination: Iec-Tc57_04:00:02 (01:0c:cd:04:00:02)
  ⊞ Source: WistronI_53:f9:70 (3c:97:0e:53:f9:70)
    Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 4, CFI: 0, ID: 1
    100. .... .... .... = Priority: Controlled Load (4)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 0001 = ID: 1
    Type: IEC 61850/SV (Sampled Value Transmission (0x88ba)
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu

0000  01 0c cd 04 00 02 3c 97  0e 53 f9 70 81 00 80 01   ......<. .S.p....
0010  88 ba 40 00 00 71 00 00  00 00 60 67 80 01 01 a2   ..@..q.. ..`g....
0020  62 30 [illegible] 80 0f 45 6d 61  6e [illegible] 65 6c 5f 63 70 75   b0`..Ema nuel_cpu
0030  74 5f [illegible] 61 82 02 04 [illegible]  83 04 [illegible] 00 00 01 85 01   t_sa.... ........
0040  00 87 [illegible] ff fc a9 69 00  00 00 00 [illegible] 04 74 e9 00   ..@...i. .....t..
0050  00 00 [illegible] 00 0a [illegible] 68 00  [illegible] 00 00 00 [illegible] b6 ba 00   [illegible]..h. ........
0060  00 2[illegible] [illegible] ff de [illegible] 15 00  [illegible] 00 00 2c [illegible] 00   . ...... ....,...
0070  00 0[illegible] [illegible] 00 6[illegible] [illegible] 19 00  00 [illegible] 00 00 75 23 4[illegible] [illegible]   ....i... ....u#L.
0080  00 [illegible]
```

Figure 5.24: PC-based sensor Sampled Value (SV) message structure confirmation

On the left-hand side of Fig. 5.24 is the sampled value frame structure as defined in the IEC 61850 standard and the IEC 61850-9-2LE guideline document. On the right-hand side of Fig. 5.24 is the captured packet containing, as can be seen, the data generated, published and captured from the PC-based sensor node.

The preamble and start of frame delimiter depicted on the left-hand side of Fig. 5.24 occurs at the hardware level and as a result it is not captured in the Wireshark frame displayed on the right-hand side of Fig. 5.24. The captured frame from the PC-based sensor node consists of source and destination addresses, VTPID, TCI, APPID, Length field, two reserved fields and the protocol data unit.  These are indicated on Fig. 5.24 and are in accordance with the IEC 61850 standard.

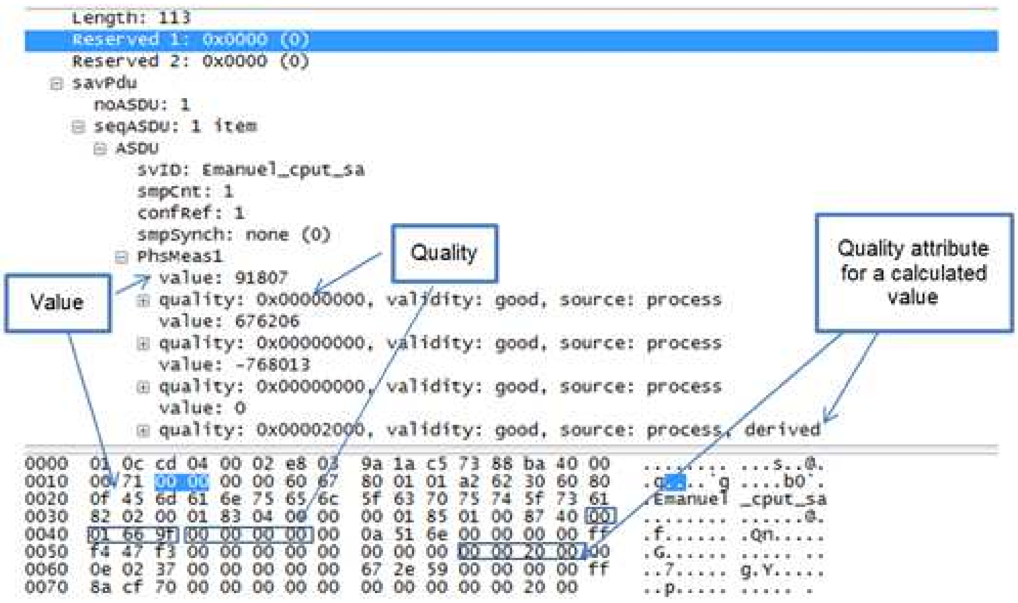
The next section in the sampled value frame is the Application Protocol Data Unit (APDU) shown in Fig. 5.25.



```
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu
      noASDU: 1
    ⊟ seqASDU: 1 item
      ⊟ ASDU
          svID: Emanuel_cput_sa
          smpCnt: 1
          confRef: 1
          smpSynch: none (0)
        ⊟ PhsMeas1
            value: 91807
          ⊞ quality: 0x00000000, validity: good, source: process
            value: 676206
          ⊞ quality: 0x00000000, validity: good, source: process
            value: -768013
          ⊞ quality: 0x00000000, validity: good, source: process
            value: 0
          ⊞ quality: 0x00002000, validity: good, source: process, derived

0000  01 0c cd 04 00 02 e8 03  9a 1a c3 73 88 ba 40 00   ........ ...s..@.
0010  00 71 00 00 00 00 60 67  80 01 01 a2 62 30 60 80   .q....`g ....b0`.
0020  0f 45 6d 61 6e 75 65 6c  5f 63 70 75 74 5f 73 61   .Emanuel _cput_sa
0030  82 02 00 01 83 04 00 00  00 01 85 01 00 87 40 00   ........ ......@.
0040  01 66 9f 00 00 00 00 00  0a 51 6e 00 00 00 00 ff   .f...... .Qn.....
0050  f4 47 f3 00 00 00 00 00  00 00 00 00 00 20 00 00   .G...... ..... ..
0060  0e 02 37 00 00 00 00 00  67 2e 59 00 00 00 00 ff   ..7..... g.Y.....
0070  8a cf 70 00 00 00 00 00  00 00 00 00 00 20 00      ..p..... ..... .
```

Figure 5.25: PC-based sensor SV protocol data unit

The published sampled value sampled value Protocol Data Unit is shown in Fig. 5.25 and compared to the IEC 61850-9-2 standard PDU. The protocol data unit consists of ASN.1 tags for the various fields in the sampled value PDU, the number of application service data units, sampled value identifier (svID), sampled value counter that increments from 0 to 3999 and resets back to zero as discussed in section 5.2.2. The other fields are the configuration revision, sampled value synchronization which is set to none, and data contained with the dataset as shown in Fig. 5.26 below.



```
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu
      noASDU: 1
    ⊟ seqASDU: 1 item
      ⊟ ASDU
          svID: Emanuel_cput_sa
          smpCnt: 1
          confRef: 1
          smpSynch: none (0)
        ⊟ PhsMeas1
            value: 91807
          ⊞ quality: 0x00000000, validity: good, source: process
            value: 676206
          ⊞ quality: 0x00000000, validity: good, source: process
            value: -768013
          ⊞ quality: 0x00000000, validity: good, source: process
            value: 0
          ⊞ quality: 0x00002000, validity: good, source: process, derived

0000  01 0c cd 04 00 02 e8 03  9a 1a c5 73 88 ba 40 00   ........ ...s..@.
0010  00 71 00 00 00 00 60 67  80 01 01 a2 62 30 60 80   .q....`g ....b0`.
0020  0f 45 6d 61 6e 75 65 6c  5f 63 70 75 74 5f 73 61   .Emanuel _cput_sa
0030  82 02 00 01 83 04 00 00  00 01 85 01 00 87 40 00   ........ ......@.
0040  01 66 9f 00 00 00 00 00  0a 51 6e 00 00 00 00 ff   .f...... .Qn.....
0050  f4 47 f3 00 00 00 00 00  00 00 00 00 00 20 00 00   .G...... ..... ..
0060  0e 02 37 00 00 00 00 00  67 2e 59 00 00 00 00 ff   ..7..... g.Y.....
0070  8a cf 70 00 00 00 00 00  00 00 00 00 00 20 00      ..p..... ..... .
```

Figure 5.26: PC-based sensor dataset

A phase measurement (Phsmeas1) dataset is transmitted with the phase shown in Fig. 5.25. The phase measurement dataset consists of 3 phase currents and voltages, and 1 neutral current and voltage value. The representation of the values within the dataset is modelled from the practical experiment conducted in the next section and will be discussed in detail in the next section. The same scaling as applied in the Omicron CMC, discussed in the next section has been applied. The values contained within the dataset have the associated quality attribute as indicated in Fig. 5.26. The neutral values quality attributes indicates that the value is calculated (derived) as shown in Fig. 5.26.

On examination, as illustrated by the text boxes in Fig. 5.25 and Fig. 5.26, and comparing the sampled value frame structure defined in the IEC 61850-9-2 standard, it is evident that the frame structure of the transmitted sampled value frame from the PC-based sensor node has all the subsections of the sampled value frame in place, and is recognised by a protocol analyser such as Wireshark. As discussed in Chapter 4 section 4.4.2, if the captured frame does not conform to any standard, Wireshark would indicate that the packet is “Malformed”.

**Note**: However, the rate at which the sampled values are being published is not compliant with the IEC 61850-9-2LE specification of SV rate = 80 samples/cycle required for protection applications. The functional behaviour is being achieved but the required temporal behaviour is not being achieved, i.e. that the “*smpCnt*” is not being updated every 250µS. The maximum speed at which the sampled value counter is incremented and the sampled values are being published currently is 980µS.

### 5.4.2. Practical experiment: Using Omicron CMC test injection device for power systems

To be able perform a comparative analysis of the developed sensor node against a commercial product utilised for sampled value testing; a practical experiment with an Omicron CMC is conducted in this section. The experiment was conducted using an Omicron CMC to publish Sampled Value messages and the packet analyser Wireshark is used to capture and display the published packet with as much detail as possible, as shown in Fig. 5.27.

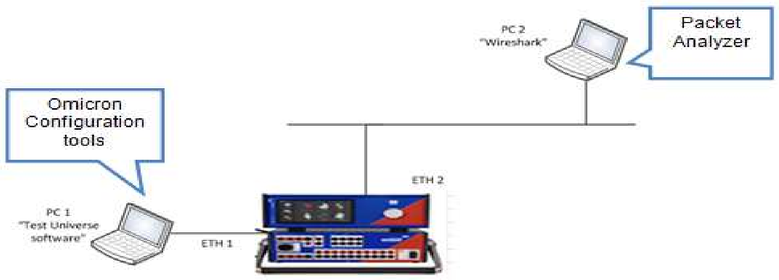
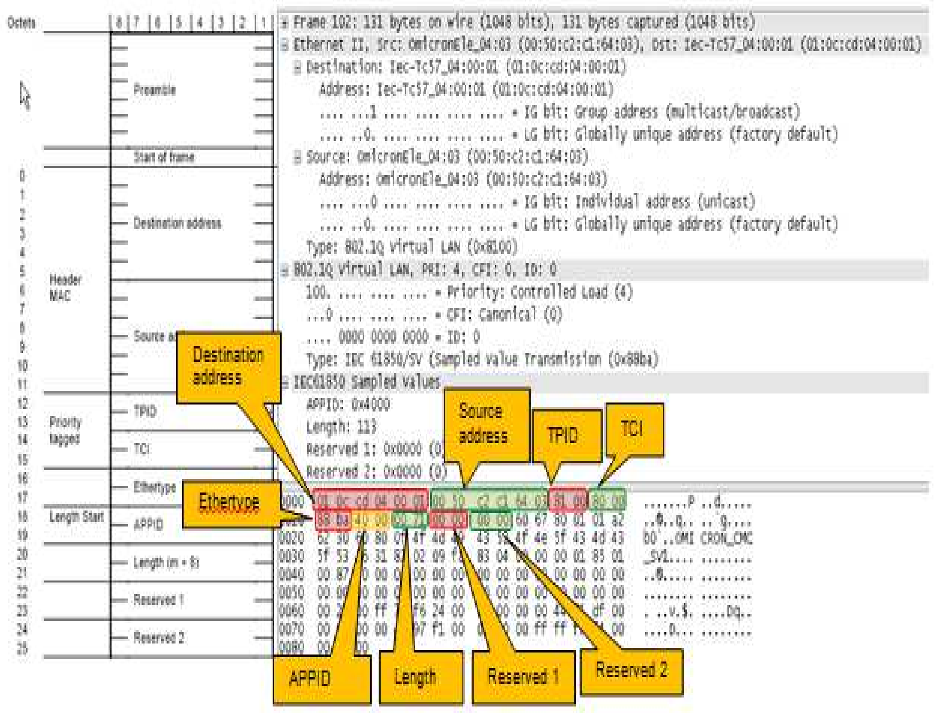


Figure 5.27: Test setup for the Sampled Value message investigation using an Omicron CMC and Wireshark

The test setup is depicted in Fig. 5.27 above with 2 personal computers used for communicating to the Omicron test set. PC1 in Fig. 5.27 has the Test Universe software installed and running on it to do the configuration of the Omicron CMC unit. PC2 is running Wireshark (MMS Ethereal) to capture the messages published from the Omicron CMC. The Omicron CMC test set is a commissioning and testing tool traditionally used to perform secondary injection during the commissioning phase of power system equipment. The newer versions of the Omicron CMC such as CMC 256, CMC 353, CMC 356, CMC 850, etc. provide support for IEC 61850 GOOSE and Sampled Values (Online: Omicron - IEC 61850 Testing Tool ).

The CMC's Sampled Values Configuration module can be set up to generate up to three sampled value streams in the test set (Online: Omicron - IEC 61850 Testing Tool ). It also provides the relevant communication parameters enabling the Sampled Values communication or outputting. The test set supports generation of sampled values at a rate of 80 samples per cycle and 256 samples per cycles at nominal frequencies of 50Hz and 60Hz as specified in the IEC 61850-9-2LE document for protection and metering applications. The Sampled Values published onto the network, contain information mirroring and corresponding to the physical analogue voltages and currents generated at the voltage and current terminals of the test set.

The data generated, published, and captured, from an OMICRON CMC device in the practical experiment is discussed below and is compared to the IEC 61850-9-2LE implementation guideline document for its functional and temporal behaviour. Fig. 5.28 below shows the sampled value packet captured from the Omicron CMC.

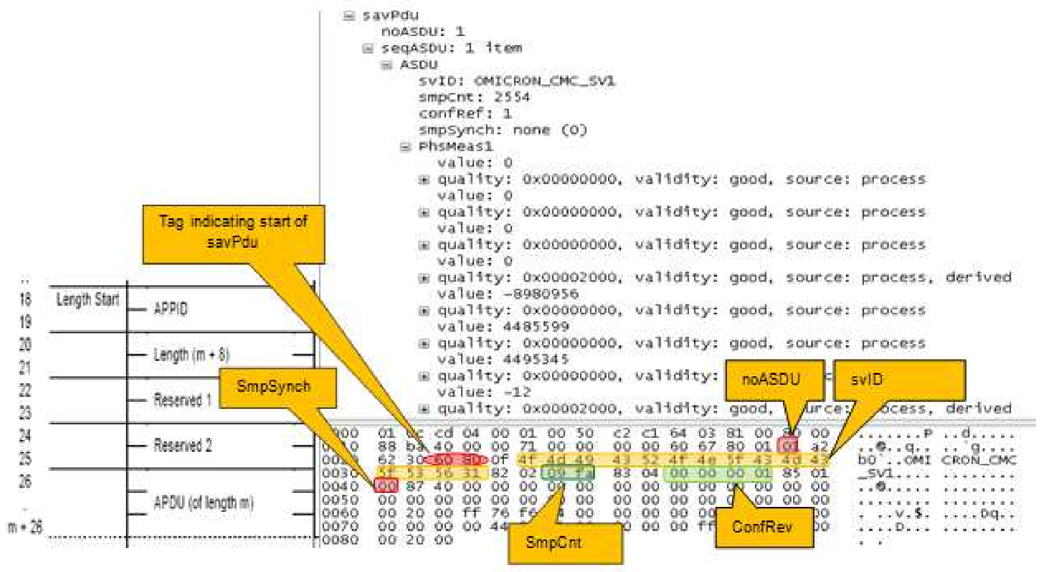


```
⊞ Frame 102: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
⊟ Ethernet II, Src: OmicronEle_04:03 (00:50:c2:c1:64:03), Dst: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
  ⊟ Destination: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
      Address: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  ⊟ Source: OmicronEle_04:03 (00:50:c2:c1:64:03)
      Address: OmicronEle_04:03 (00:50:c2:c1:64:03)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 4, CFI: 0, ID: 0
    100. .... .... .... = Priority: Controlled Load (4)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0000 0000 0000 = ID: 0
    Type: IEC 61850/SV (Sampled Value Transmission (0x88ba)
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)

0000  01 0c cd 04 00 01 00 50  c2 c1 64 03 81 00 80 00   .......P ..d.....
0010  88 ba 40 00 00 71 00 00  00 00 60 67 80 01 01 a2   ..@..q.. ..`g....
0020  62 30 [illegible] 80 0[illegible] 4f 4d 4[illegible]  43 [illegible] 4f 4e 5f 43 4d 43   b0`..OMI CRON_CMC
0030  5f 53 [illegible] 31 8[illegible] 02 09 f[illegible]  83 04 [illegible] 00 00 01 85 01   _SV1.... ........
0040  00 87 [illegible] 00 0[illegible] 00 00 00  [illegible] 00 00 [illegible] 00 00 00 00   ..@..... ........
0050  00 0[illegible] [illegible] 00 [illegible] 00 00 00  [illegible] 00 00 0[illegible] [illegible] 00 00 00   ........ ........
0060  00 [illegible] [illegible] ff [illegible] f6 24 00  [illegible] 00 00 4[illegible] [illegible] df 00   . ..v.$. ....Dq..
0070  00 [illegible] [illegible] 00 [illegible] 97 f1 00  [illegible] 00 ff ff [illegible] [illegible] 00   ....0... ........
0080  00 [illegible] 00
```

Figure 5.28: CMC Sampled Value (SV) message structure confirmation

On the left hand side of Fig. 5.28 is the sampled value frame structure as defined in the IEC 61850-9-2 standard, and on the right hand side of the Fig. 5.28 is the captured packet from the CMC. The preamble and start of frame delimiter depicted in the left hand side of Fig. 5.28 occurs at the hardware level and as a result it is not captured in the Wireshark frame displayed on the right hand side of Fig. 5.28.

The captured frame consists of source and destination addresses, TPID, TCI, APPID, Length field, two reserved fields and the protocol data unit as highlighted in Fig. 5.28 above. The next section in the sampled value frame is the protocol data unit shown in Fig. 5.29.



```
⊟ savPdu
    noASDU: 1
  ⊟ seqASDU: 1 item
    ⊟ ASDU
        svID: OMICRON_CMC_SV1
        smpCnt: 2554
        confRef: 1
        smpSynch: none (0)
      ⊟ PhsMeas1
          value: 0
        ⊞ quality: 0x00000000, validity: good, source: process
          value: 0
        ⊞ quality: 0x00000000, validity: good, source: process
          value: 0
        ⊞ quality: 0x00000000, validity: good, source: process
          value: 0
        ⊞ quality: 0x00002000, validity: good, source: process, derived
          value: -8980956
        ⊞ quality: 0x00000000, validity: good, source: process
          value: 4485599
        ⊞ quality: 0x00000000, validity: good, source: process
          value: 4495345
        ⊞ quality: 0x00000000, validity: [illegible]
          value: -12
        ⊞ quality: 0x00002000, validity: good, [illegible]urce: [illegible]ocess, derived
```

```
[illegible]00  01 0c cd 04 00 01 00 50  c2 c1 64 03 81 00 [illegible] 00   .......P ..d.....
[illegible]0  88 ba 40 00 00 71 00 00  00 00 60 67 80 01 01 a2   ..@..q.. ..`g....
00[illegible]  62 30 60 80 0f 4f 4d 49  43 52 4f 4e 5f 43 4d 43   b0`..OMI CRON_CMC
003[illegible]  5f 53 56 31 82 02 09 fa  83 04 00 00 00 01 85 01   _SV1.... ........
0040  00 87 40 00 00 00 0[illegible] 00  00 00 00 00 0[illegible] 00 00 00   ..@..... ........
0050  00 00 00 00 00 00 [illegible] 00  00 00 00 00 [illegible] 00 00 00   ........ ........
0060  00 20 00 ff 76 f6 [illegible] 00  00 00 00 00 [illegible] [illegible]0   . ..v.$. .....Dq..
0070  00 00 00 00 44 [illegible]  [illegible]0 00 00 ff [illegible] 00   ....D... ........
0080  00 20 00                                          . .
```

Figure 5.29: CMC Sampled Value (SV) Protocol data unit and ASN.1 tagging

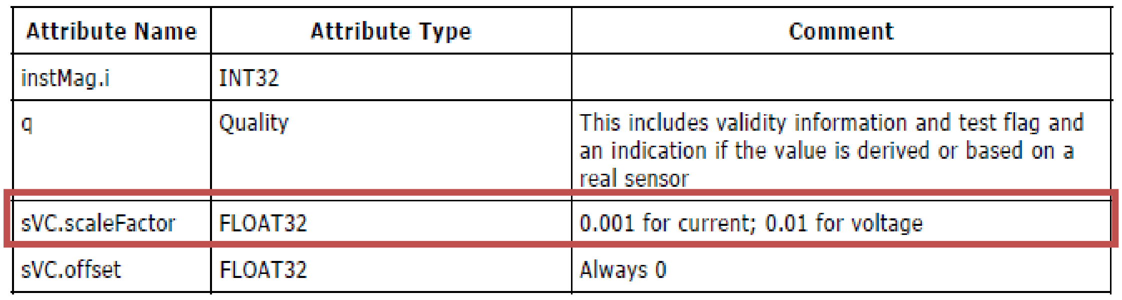
In Fig. 5.29 above the sampled value protocol data unit of a frame published by the CMC is depicted. The sampled value protocol data unit is tagged using the abstract syntax notation to indicate the start of a savPdu. All the other fields within the sampled value frame protocol data unit are encoded as specified in the IEC 61850-9-2 and IEC 61850-9-2LE implementation guideline document.

The published frame from the Omicron CMC contains data which is packaged within a dataset. The software environment associated with the CMC was set up to generate and publish voltages [set to primary RMS values of 63.508 kV (L-N)] in this experiment. From the values configured on the CMC, the peak value is expected to be as shown in equation 5.9.

$$63.508kV * \sqrt{2} = 89813.87V \qquad (5.9)$$

Theoretically the sinusoidal waveform varies between + 89813.87V and - 89813.87V. What is observed through the practical experiment however is that the values in the captured frames varied between -8980717 and +8980717. To verify that the correct values were being published from the CMC, the IEC 61850-9-2 standard was utilised.

This at first glance does not seem to relate at all to the value 89813.87V above. However, these values were confirmed to be correct and compliant with the IEC 61850-9-2LE document, because the IEC 61850-9-2LE document specifies a scaling factor of 0.01 for voltages and 0.001 for current as shown in Fig. 5.30 below.



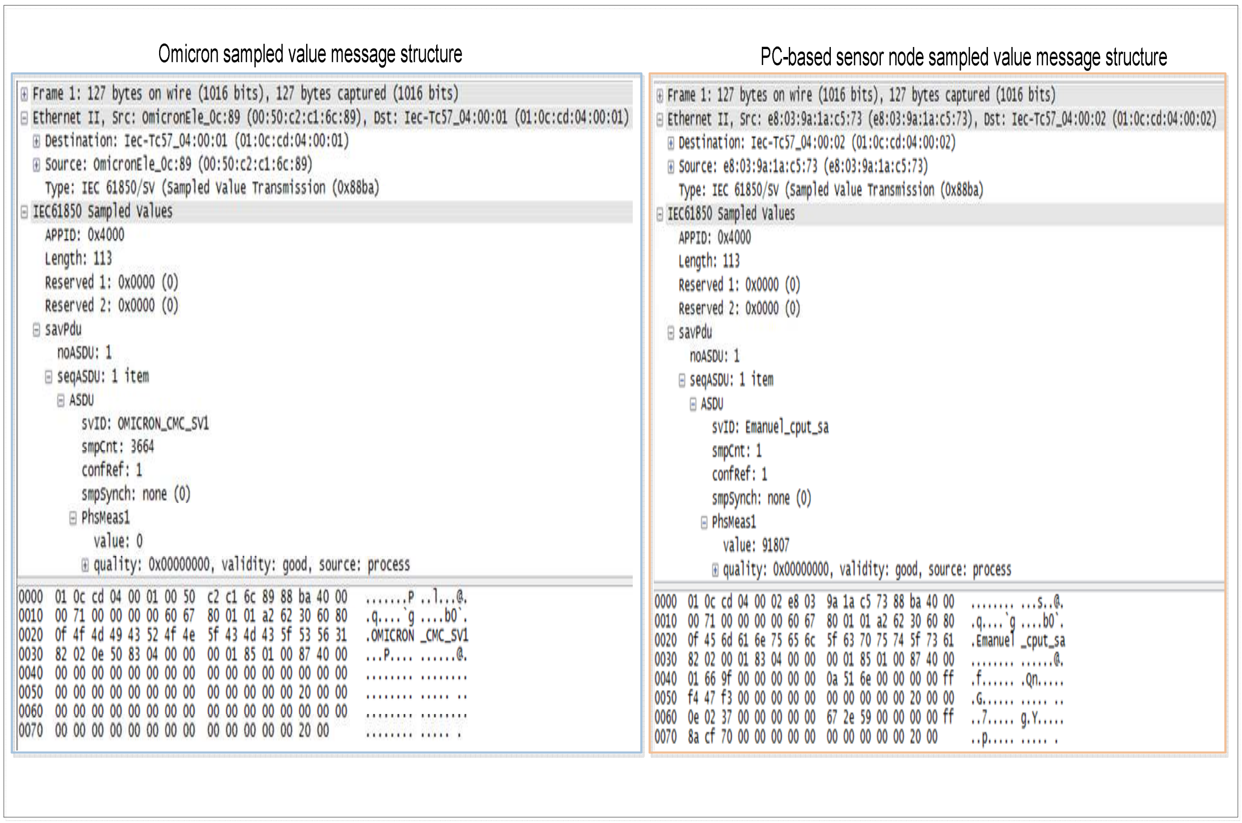
| Attribute Name | Attribute Type | Comment |
|---|---|---|
| instMag.i | INT32 | |
| q | Quality | This includes validity information and test flag and an indication if the value is derived or based on a real sensor |
| sVC.scaleFactor | FLOAT32 | 0.001 for current; 0.01 for voltage |
| sVC.offset | FLOAT32 | Always 0 |

Figure 5.30: Merging Unit current and voltage scaling factor (IEC 61850-9-2LE).

The theoretical peak value when the voltage scaling factor of 0.01 is taken into account, as specified in page 8, table 2 of the IEC 61850-9-2LE document is 8980717 x 0.01 = 89807.17. This is close to the theoretical value of 89813.87 calculated above.

## 5.5. Comparative Analysis and Conclusion

In section 5.4.1 a practical experiment is performed using the developed PC-based sensor node, and in section 5.4.2 a practical experiment using a commercial unit (Omicron CMC) is discussed. Fig. 5.31 below shows the sampled value messages published from these devices side by side.



Omicron sampled value message structure

```
⊞ Frame 1: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits)
⊟ Ethernet II, Src: OmicronEle_0c:89 (00:50:c2:c1:6c:89), Dst: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
  ⊞ Destination: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
  ⊞ Source: OmicronEle_0c:89 (00:50:c2:c1:6c:89)
    Type: IEC 61850/SV (Sampled Value Transmission (0x88ba)
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu
      noASDU: 1
    ⊟ seqASDU: 1 item
      ⊟ ASDU
          svID: OMICRON_CMC_SV1
          smpCnt: 3664
          confRef: 1
          smpSynch: none (0)
        ⊟ PhsMeas1
            value: 0
          ⊞ quality: 0x00000000, validity: good, source: process

0000  01 0c cd 04 00 01 00 50  c2 c1 6c 89 88 ba 40 00   .......P ..l...@.
0010  00 71 00 00 00 00 60 67  80 01 01 a2 62 30 60 80   .q....`g ....b0`.
0020  0f 4f 4d 49 43 52 4f 4e  5f 43 4d 43 5f 53 56 31   .OMICRON _CMC_SV1
0030  82 02 0e 50 83 04 00 00  00 01 85 01 00 87 40 00   ...P.... ......@.
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 20 00 00   ........ ..... ..
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 20 00      ........ ..... .
```

PC-based sensor node sampled value message structure

```
⊞ Frame 1: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits)
⊟ Ethernet II, Src: e8:03:9a:1a:c5:73 (e8:03:9a:1a:c5:73), Dst: Iec-Tc57_04:00:02 (01:0c:cd:04:00:02)
  ⊞ Destination: Iec-Tc57_04:00:02 (01:0c:cd:04:00:02)
  ⊞ Source: e8:03:9a:1a:c5:73 (e8:03:9a:1a:c5:73)
    Type: IEC 61850/SV (Sampled Value Transmission (0x88ba)
⊟ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 113
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⊟ savPdu
      noASDU: 1
    ⊟ seqASDU: 1 item
      ⊟ ASDU
          svID: Emanuel_cput_sa
          smpCnt: 1
          confRef: 1
          smpSynch: none (0)
        ⊟ PhsMeas1
            value: 91807
          ⊞ quality: 0x00000000, validity: good, source: process

0000  01 0c cd 04 00 02 e8 03  9a 1a c5 73 88 ba 40 00   ........ ...s..@.
0010  00 71 00 00 00 00 60 67  80 01 01 a2 62 30 60 80   .q....`g ....b0`.
0020  0f 45 6d 61 6e 75 65 6c  5f 63 70 75 74 5f 73 61   .Emanuel _cput_sa
0030  82 02 00 01 83 04 00 00  00 01 85 01 00 87 40 00   ........ ......@.
0040  01 66 9f 00 00 00 00 00  0a 51 6e 00 00 00 00 ff   .f...... .Qn.....
0050  f4 47 f3 00 00 00 00 00  00 00 00 00 00 20 00 00   .G...... ..... ..
0060  0e 02 37 00 00 00 00 00  67 2e 59 00 00 00 00 ff   ..7..... g.Y.....
0070  8a cf 70 00 00 00 00 00  00 00 00 00 00 20 00      ..p..... ..... .
```

Figure 5.31: Comparative analysis of the developed sensor nodes sampled value structure with a commercial unit from Omicron

Comparatively the functional behaviour of the messages from these simulation devices is the same. The two devices can both publish sampled value messages with all the required subsections of the sampled value message structure as described in the IEC 61850-9-2 and IEC 61850-9-2LE standard.

The temporal behaviour is however vastly different. The Omicron CMC device publishes real-time sampled value messages that can be used for protection application testing, whereas the sensor node developed can only achieve functional behaviour but not the temporal behaviour.

In this chapter the implementation of the PC-based simulation for the merging unit has been discussed relative to the outputs from a commercial unit, and commentary has been given on both functional and temporal behaviour. The next chapter is the conclusion for the research project in its entirety and what future work could be derived from the foundation laid.

# CHAPTER 6

# Conclusion and Future Work

## 6.1. Introduction

The deployment of standardized substation automation systems has been steadily increasing in utilities since the introduction of the IEC 61850 standard for "Communication networks and systems in substations" in 2004. The deployed devices however are mainly for station-level applications, as discussed in chapter 4 section 4.2. The current number of devices available in the market to support the IEC 61850-9-2 process bus is however still very limited, and mainly resides in the vendor environment. The commercial units that are compliant IEC 61850-9-2LE generally tend to be expensive and are generally not accessible in an academic environment.

From an educational point of view, there are very few resources and tools available to support the establishment of a knowledge base with respect to the process-bus paradigm. The attempt in this project has been to establish this knowledge base through the development of a non-real-time simulation environment which would act as a precursor for future projects on real-time development. Through experimentation the validation of functional behaviour was observed and commentary on the temporal behaviour is also given.

The software development is on the Linux Ubuntu operating system version10.04 using the Python programming language. The evaluation of the results of the developed sensor node show that the virtual sensor node developed is capable of simulating the functional behaviour of an IEC 61850-9-2 standard device but is not capable of simulating the temporal behaviour using the Scapy networking library.

## 6.2. Research Aim and objectives

### 6.2.1. Aim

From the problem statement articulated in the first chapter, the research aim of this postgraduate project is to examine the standardization developments within the IEC 615850 substation automation environment, to study the substation automation framework presented in the IEC61850 standard, and measurement distribution methods proposed in the IEC 61850 standard. The knowledge gained has been used to implement certain aspects of a Merging Unit. This limited functional merging unit represents a virtualized data acquisition node capable of synthetic data generation, encoding of the data and transmitting the data in a format compliant with the IEC 61850-9-2 sampled value message structure.

### 6.2.2. Research objectives

The original research objectives as stated in chapter one and which have been achieved are as follows:

i. To identify current and emerging substation automation trends.
ii. To Investigation whether Microsoft Windows and Linux operating systems allow for the generation, binding and transmission of packets at the Datalink layer.
iii. To thoroughly understand the sampled value messages utilised in the IEC 61850 standard and to simulate sampled value messages from a personal computer (PC) platform.

The project objectives which have been achieved are listed as follows:

1. Literature review: Current and emerging substation automation trends have been identified
2. Literature review: A comparative analysis of the developments stated in the literature with respect to measurement distribution methods utilised in substation automation systems has been successfully undertaken.
3. An overview of the IEC 61850 standard with emphasis on sampled values and GOOSE messaging has been provided.
4. The sampled values structure breakdown and the other IEC 61850 related standards are outlined.
5. A low-level network programming Application Programming Interface (API) on both Microsoft Windows and Linux operating systems is presented.
6. Software development on Linux Ubuntu to simulate the functional behaviour of sampled value messaging has been implemented.
7. Experimentation via the development of a test scenario to verify the structure of the sampled value from the developed sensor node is performed, and a comparative analysis has been performed against a commercial IEC 61850-9-2 compliant product.

## 6.3. Project assumption and Thesis deliverables

During the development of the work on this project, the following assumptions and scope delimitation as in chapter one have been implemented:

### 6.3.1. Assumptions:

i. Modern computers have a fast enough processing speed and modern operating systems allow message generation and transmission at the Datalink layer, enabling accurate emulation of a merging unit.
ii. The use of custom-built networking libraries does degrade the performance of the generated program.

It should be observed that the second assumption does not hold true as stated in chapter one.

### 6.3.2. Delimitation:

i. The data acquisition node is explored on a PC platform only
ii. Verification of sampled values is performed using software tools only.
iii. The project is not integrated into the IEC 61850 architecture; no substation configuration language (SCL) files are produced for the developed device (MU); the publisher/subscriber mechanism is not be explored; the sampled value messaging is verified using only software tools such as Wireshark.
iv. The system is not implemented on the medium voltage (MV) network.
v. Current and voltage values are generated internally, packaged and transmitted as multicast messages - no sensors are integrated into the virtual device.
vi. Analogue values published by the sensor node are monitored using Wireshark.
vii. The software program is developed using the Python programming language.
viii. Time synchronization and time stamping of frames is performed using the personal computer’s internal clock (No integration of a GPS clock source).

### 6.3.3. Thesis deliverables:

A literature search and review, on the substation automation framework provided by the IEC 61850 standard, focusing on measurement distribution using IEC 60044-7, IEC 60044-8, IEC 1451 and IEC 61850 standards has been presented. The theoretical base and algorithms for software development has been formalised and applied to simulate the sampled value concept introduced in the IEC 61850 standard. Some of the achieved outputs are itemised below.

1. Experimentation with an Omicron CMC has been conducted to assist with the understanding and development of the proposed virtual sensor.
2. Low level networking software development in both Microsoft Windows and Linux operating systems has been investigated.

3. The concepts introduced in the IEC 61850-9-2, IEC 61850-9-2LE, IEC 60044-7 and IEC 60044-8 standards has been used to a model and a limited function sensor node capable of sampled values functional behaviour has been simulated.
4. A Python script running on a Linux Ubuntu operating system has been developed and used to simulate IEC 61850-9-2 sampled values.
5. Wireshark is used to captured the sampled value frames published from the virtual sensor node and the functional behaviour of the virtual device is verified.
6. The simulated sampled value message is transmitted through a Virtual Area Network (VLAN)- enabled switch and is captured on the receiving device with the correct VLAN tag.
7. The use of the Scapy networking Library even with the use of the fast-send function has however been found to degrade the performance of virtual sensor node developed. The 250µS required repetition rate for sending of packets, i.e. 80 samples per cycle as specified in the IEC 61850 standard when operating at 50Hz; was not achieved even when running just Scapy in a loop without even sending a layer 2 packet containing the sampled value data.

## 6.4. Challenges Encountered

There have been a substantial number of challenges that have been encountered during the course of this research project implementation. These include:

a) The IEC 61850 standard resides in the public domain, but that the transition from the specification within the standard to that of implementation is an extremely challenging one. This knowledge usually is proprietary and resides mainly within the vendor environment. Consequently very little implementation and experimentation knowledge in the public domain generally, and in the academic environment particularly. The situation has improved with respect to the knowledge base and equipment for GOOSE messaging, but there is still very little knowledge with respect to the process bus environment.
b) Very few platforms, tools and forums for the development or adaptation of IEC 61850 solutions in the public domain.
c) A multi-disciplinary environment – the IEC 61850 combines a lot of different disciplines e.g. data networking, Information Technology, protection etc. Software development to simulate IEC 61850 sampled value messages

requires detailed knowledge in multiple fields such as data networking, electrical protection, software modelling and protection and control testing methods.

d) Interpretation of the standard – a pre-requisite to development of an IEC 61850-compliant device requires detailed knowledge of various parts of the standard.

e) Transmission of layer 2 packets on Microsoft Windows and Linux Ubuntu operating systems

f) Fast network packet outputting using Scapy. Scapy is not designed for fast networking and as a result the only the functional behaviour of the merging unit could be simulated in this project and the temporal behaviour could not be achieved.

## 6.5. Future research work

This project investigated the development of a sensor node on a PC platform. The intention was to provide the building blocks for future real time implementation of a sensor node that is compliant with IEC 61850-9-2LE. The role of this project was to build and lay a foundation for IEC 61850 sampled value real-time implementation in the academic sphere. Future directions for research work could include the following:

1. Investigate the use an embedded development platform to achieve both the functional and temporal behaviour of a IEC 61850 merging unit.
2. The modelling of the sampled values logical nodes (TCTR and TVTR) instead of hard-coding the values into the sampled value dataset.
3. On the current development the ASN.1 tags are hard-coded into the various classes that represent the fields with the sampled value frame structure. An alternative method would be to implement an ASN.1 encoding algorithm.
4. The virtual sensor node device should be improved and adapted to receive realistic data from external sources.
5. In the sensor device developed, all the data is generated internally and is assumed to have good quality. When a real-time system is developed the data that is used by the unit will be from external sources, so an alternative method to validate the quality attribute for each measurement in the published dataset using the common data class enumeration method defined in the standard should be implemented in future.
6. Time synchronization should be provided by a device capable of providing accurate 1PPS time synchronization and should be integrated into the rest of the system functionality.

## 6.6. Publications

1. Luwaca. E., Petev P., Kriger C., and Behardien S., 2014. “Virtualization of a sensor node to enable the simulation of IEC 61850-based sampled value messages”. Submitted to the International Journal of Computers, Communications and Control (awaiting confirmation for publication).
2. Kriger C., Retonda J., Luwaca E., and Behardien S., 2011. “Analysis of GOOSE and Sampled Value Message Structure for Educational Purposes”, PAC World Conference, Dublin.

# APPENDICES

# APPENDIX A

*Scapy layer 3 source code*

```
## This file is part of Scapy
## See http://www.secdev.org/projects/scapy for more informations
## Copyright (C) Philippe Biondi <phil@secdev.org>
## This program is published under a GPLv2 license
```

```
"""
ASN.1 (Abstract Syntax Notation One)
"""

import random
from scapy.config import conf
from scapy.error import Scapy_Exception,warning
from scapy.volatile import RandField
from scapy.utils import Enum_metaclass, EnumElement

class RandASN1Object(RandField):
    def __init__(self, objlist=None):
        if objlist is None:
            objlist = map(lambda x:x._asn1_obj,
                          filter(lambda                                        x:hasattr(x,"_asn1_obj"),
ASN1_Class_UNIVERSAL.__rdict__.values()))
        self.objlist = objlist
        self.chars                                                                                    =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
    def _fix(self, n=0):
        o = random.choice(self.objlist)
        if issubclass(o, ASN1_INTEGER):
            return o(int(random.gauss(0,1000)))
        elif issubclass(o, ASN1_IPADDRESS):
            z = RandIP()._fix()
            return o(z)
        elif issubclass(o, ASN1_STRING):
            z = int(random.expovariate(0.05)+1)
            return o("".join([random.choice(self.chars) for i in range(z)]))
        elif issubclass(o, ASN1_SEQUENCE) and (n < 10):
            z = int(random.expovariate(0.08)+1)
            return o(map(lambda x:x._fix(n+1), [self.__class__(objlist=self.objlist)]*z))
        return ASN1_INTEGER(int(random.gauss(0,1000)))


##############
#### ASN1 ####
##############

class ASN1_Error(Scapy_Exception):
    pass

class ASN1_Encoding_Error(ASN1_Error):
    pass

class ASN1_Decoding_Error(ASN1_Error):
    pass

class ASN1_BadTag_Decoding_Error(ASN1_Decoding_Error):
    pass



class ASN1Codec(EnumElement):
    def register_stem(cls, stem):
        cls._stem = stem
    def dec(cls, s, context=None):
        return cls._stem.dec(s, context=context)
    def safedec(cls, s, context=None):
        return cls._stem.safedec(s, context=context)
    def get_stem(cls):
        return cls.stem


class ASN1_Codecs_metaclass(Enum_metaclass):
    element_class = ASN1Codec

class ASN1_Codecs:
    __metaclass__ = ASN1_Codecs_metaclass
    BER = 1
    DER = 2
    PER = 3
    CER = 4
    LWER = 5
    BACnet = 6
    OER = 7
    SER = 8
    XER = 9

class ASN1Tag(EnumElement):
    def __init__(self, key, value, context=None, codec=None):
        EnumElement.__init__(self, key, value)
        self._context = context
        if codec == None:
            codec = {}
        self._codec = codec
    def clone(self): # /!\ not a real deep copy. self.codec is shared
        return self.__class__(self._key, self._value, self._context, self._codec)
    def register_asn1_object(self, asn1obj):
        self._asn1_obj = asn1obj
    def asn1_object(self, val):
        if hasattr(self,"_asn1_obj"):
            return self._asn1_obj(val)
        raise ASN1_Error("%r does not have any assigned ASN1 object" % self)
    def register(self, codecnum, codec):
        self._codec[codecnum] = codec
    def get_codec(self, codec):
        try:
            c = self._codec[codec]
        except KeyError,msg:
            raise ASN1_Error("Codec %r not found for tag %r" % (codec, self))
        return c

class ASN1_Class_metaclass(Enum_metaclass):
    element_class = ASN1Tag
    def __new__(cls, name, bases, dct): # XXX factorise a bit with Enum_metaclass.__new__()
        for b in bases:
            for k,v in b.__dict__.iteritems():
                if k not in dct and isinstance(v,ASN1Tag):
                    dct[k] = v.clone()

        rdict = {}
        for k,v in dct.iteritems():
            if type(v) is int:
                v = ASN1Tag(k,v)
                dct[k] = v
                rdict[v] = v
            elif isinstance(v, ASN1Tag):
                rdict[v] = v
        dct["__rdict__"] = rdict

        cls = type.__new__(cls, name, bases, dct)
        for v in cls.__dict__.values():
            if isinstance(v, ASN1Tag):
                v.context = cls # overwrite ASN1Tag contexts, even cloned ones
        return cls


class ASN1_Class:
    __metaclass__ = ASN1_Class_metaclass

class ASN1_Class_UNIVERSAL(ASN1_Class):
    name = "UNIVERSAL"
    ERROR = -3
    RAW = -2
    NONE = -1
    ANY = 0
    BOOLEAN = 1
    INTEGER = 2
    BIT_STRING = 3
    STRING = 4
    NULL = 5
    OID = 6
    OBJECT_DESCRIPTOR = 7
    EXTERNAL = 8
    REAL = 9
    ENUMERATED = 10
    EMBEDDED_PDF = 11
    UTF8_STRING = 12
    RELATIVE_OID = 13
    SEQUENCE = 0x30#XXX 16 ??
    SET = 0x31 #XXX 17 ??
    NUMERIC_STRING = 18
    PRINTABLE_STRING = 19
    T61_STRING = 20
    VIDEOTEX_STRING = 21
    IA5_STRING = 22
    UTC_TIME = 23
    GENERALIZED_TIME = 24
    GRAPHIC_STRING = 25
    ISO646_STRING = 26
    GENERAL_STRING = 27
    UNIVERSAL_STRING = 28
    CHAR_STRING = 29
    BMP_STRING = 30
    IPADDRESS = 0x40
    COUNTER32 = 0x41
    GAUGE32 = 0x42
    TIME_TICKS = 0x43
    SEP = 0x80
```

```
class ASN1_Object_metaclass(type):
    def __new__(cls, name, bases, dct):
        c = super(ASN1_Object_metaclass, cls).__new__(cls, name, bases, dct)
        try:
            c.tag.register_asn1_object(c)
        except:
            warning("Error registering %r for %r" % (c.tag, c.codec))
        return c


class ASN1_Object:
   __metaclass__ = ASN1_Object_metaclass
    tag = ASN1_Class_UNIVERSAL.ANY
    def __init__(self, val):
        self.val = val
    def enc(self, codec):
        return self.tag.get_codec(codec).enc(self.val)
    def __repr__(self):
        return "<%s[%r]>" % (self.__dict__.get("name", self.__class__.__name__), self.val)
    def __str__(self):
        return self.enc(conf.ASN1_default_codec)
    def strshow(self, lvl=0):
        return ("  "*lvl)+repr(self)+"\n"
    def show(self, lvl=0):
        print self.strshow(lvl)
    def __eq__(self, other):
        return self.val == other
    def __cmp__(self, other):
        return cmp(self.val, other)

class ASN1_DECODING_ERROR(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.ERROR
    def __init__(self, val, exc=None):
        ASN1_Object.__init__(self, val)
        self.exc = exc
    def __repr__(self):
        return "<%s[%r]{{%s}}>" % (self.__dict__.get("name", self.__class__.__name__),
                                   self.val, self.exc.args[0])
    def enc(self, codec):
        if isinstance(self.val, ASN1_Object):
            return self.val.enc(codec)
        return self.val

class ASN1_force(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.RAW
    def enc(self, codec):
        if isinstance(self.val, ASN1_Object):
            return self.val.enc(codec)
        return self.val

class ASN1_BADTAG(ASN1_force):
    pass

class ASN1_INTEGER(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.INTEGER
```

```
class ASN1_STRING(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.STRING

class ASN1_BIT_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.BIT_STRING

class ASN1_PRINTABLE_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.PRINTABLE_STRING

class ASN1_T61_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.T61_STRING

class ASN1_IA5_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.IA5_STRING

class ASN1_NUMERIC_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.NUMERIC_STRING

class ASN1_VIDEOTEX_STRING(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.VIDEOTEX_STRING

class ASN1_IPADDRESS(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.IPADDRESS

class ASN1_UTC_TIME(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.UTC_TIME

class ASN1_GENERALIZED_TIME(ASN1_STRING):
    tag = ASN1_Class_UNIVERSAL.GENERALIZED_TIME

class ASN1_TIME_TICKS(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.TIME_TICKS

class ASN1_BOOLEAN(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.BOOLEAN

class ASN1_ENUMERATED(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.ENUMERATED

class ASN1_NULL(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.NULL

class ASN1_SEP(ASN1_NULL):
    tag = ASN1_Class_UNIVERSAL.SEP

class ASN1_GAUGE32(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.GAUGE32

class ASN1_COUNTER32(ASN1_INTEGER):
    tag = ASN1_Class_UNIVERSAL.COUNTER32

class ASN1_SEQUENCE(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.SEQUENCE
    def strshow(self, lvl=0):
        s = ("  "*lvl)+("# %s:" % self.__class__.__name__)+"\n"
        for o in self.val:
            s += o.strshow(lvl=lvl+1)
        return s

class ASN1_SET(ASN1_SEQUENCE):
    tag = ASN1_Class_UNIVERSAL.SET

class ASN1_OID(ASN1_Object):
    tag = ASN1_Class_UNIVERSAL.OID
    def __init__(self, val):
        val = conf.mib._oid(val)
        ASN1_Object.__init__(self, val)
    def __repr__(self):
        return    "<%s[%r]>"    %    (self.__dict__.get("name",    self.__class__.__name__),
conf.mib._oidname(self.val))
    def __oidname__(self):
        return '%s'%conf.mib._oidname(self.val)



conf.ASN1_default_codec = ASN1_Codecs.BER
```

# APPENDIX B

Scapy layer 2 source code

```
## This file is part of Scapy
## See http://www.secdev.org/projects/scapy for more informations
## Copyright (C) Philippe Biondi <phil@secdev.org>
## This program is published under a GPLv2 license

"""
Classes and functions for layer 2 protocols.
"""

import os,struct,time
from scapy.base_classes import Net
from scapy.config import conf
from scapy.packet import *
from scapy.ansmachine import *
from scapy.plist import SndRcvList
from scapy.fields import *
from scapy.sendrecv import srp,srp1
from scapy.arch import get_if_hwaddr




#################
## Tools      ##
#################


class Neighbor:
    def __init__(self):
        self.resolvers = {}

    def register_l3(self, l2, l3, resolve_method):
        self.resolvers[l2,l3]=resolve_method

    def resolve(self, l2inst, l3inst):
        k = l2inst.__class__,l3inst.__class__
        if k in self.resolvers:
            return self.resolvers[k](l2inst,l3inst)

    def __repr__(self):
        return "\n".join("%-15s -> %-15s" % (l2.__name__, l3.__name__) for l2,l3 in
self.resolvers)

conf.neighbor = Neighbor()

conf.netcache.new_cache("arp_cache", 120) # cache entries expire after 120s


@conf.commands.register
def getmacbyip(ip, chainCC=0):
    """Return MAC address corresponding to a given IP address"""
    if isinstance(ip,Net):
        ip = iter(ip).next()
    ip = inet_ntoa(inet_aton(ip))
    tmp = map(ord, inet_aton(ip))
    if (tmp[0] & 0xf0) == 0xe0: # mcast @
        return "01:00:5e:%.2x:%.2x:%.2x" % (tmp[1]&0x7f,tmp[2],tmp[3])
    iff,a,gw = conf.route.route(ip)
    if ( (iff == "lo") or (ip == conf.route.get_if_bcast(iff)) ):
        return "ff:ff:ff:ff:ff:ff"
    if gw != "0.0.0.0":
        ip = gw

    mac = conf.netcache.arp_cache.get(ip)
    if mac:
        return mac

    res = srp1(Ether(dst=ETHER_BROADCAST)/ARP(op="who-has", pdst=ip),
               type=ETH_P_ARP,
               iface = iff,
               timeout=2,
               verbose=0,
               chainCC=chainCC,
               nofilter=1)
    if res is not None:
        mac = res.payload.hwsrc
        conf.netcache.arp_cache[ip] = mac
        return mac
    return None



### Fields

class DestMACField(MACField):
    def __init__(self, name):
        MACField.__init__(self, name, None)
    def i2h(self, pkt, x):
        if x is None:
            x = conf.neighbor.resolve(pkt,pkt.payload)
            if x is None:
                x = "ff:ff:ff:ff:ff:ff"
                warning("Mac address to reach destination not found. Using broadcast.")
        return MACField.i2h(self, pkt, x)
    def i2m(self, pkt, x):
        return MACField.i2m(self, pkt, self.i2h(pkt, x))

class SourceMACField(MACField):
    def __init__(self, name):
        MACField.__init__(self, name, None)
    def i2h(self, pkt, x):
        if x is None:
            iff,a,gw = pkt.payload.route()
            if iff:
                try:
                    x = get_if_hwaddr(iff)
                except:
                    pass
            if x is None:
```

```
            x = "00:00:00:00:00:00"
        return MACField.i2h(self, pkt, x)
    def i2m(self, pkt, x):
        return MACField.i2m(self, pkt, self.i2h(pkt, x))

class ARPSourceMACField(MACField):
    def __init__(self, name):
        MACField.__init__(self, name, None)
    def i2h(self, pkt, x):
        if x is None:
            iff,a,gw = pkt.route()
            if iff:
                try:
                    x = get_if_hwaddr(iff)
                except:
                    pass
            if x is None:
                x = "00:00:00:00:00:00"
        return MACField.i2h(self, pkt, x)
    def i2m(self, pkt, x):
        return MACField.i2m(self, pkt, self.i2h(pkt, x))



### Layers


class Ether(Packet):
    name = "Ethernet"
    fields_desc = [ DestMACField("dst"),
                    SourceMACField("src"),
                    XShortEnumField("type", 0x9000, ETHER_TYPES) ]
    def hashret(self):
        return struct.pack("H",self.type)+self.payload.hashret()
    def answers(self, other):
        if isinstance(other,Ether):
            if self.type == other.type:
                return self.payload.answers(other.payload)
        return 0
    def mysummary(self):
        return self.sprintf("%src% > %dst% (%type%)")
    @classmethod
    def dispatch_hook(cls, _pkt=None, *args, **kargs):
        if _pkt and len(_pkt) >= 14:
            if struct.unpack("!H", _pkt[12:14])[0] <= 1500:
                return Dot3
        return cls


class Dot3(Packet):
    name = "802.3"
    fields_desc = [ DestMACField("dst"),
                    MACField("src", ETHER_ANY),
                    LenField("len", None, "H") ]
    def extract_padding(self,s):
        l = self.len
        return s[:l],s[l:]
    def answers(self, other):
        if isinstance(other,Dot3):
            return self.payload.answers(other.payload)
        return 0
    def mysummary(self):
        return "802.3 %s > %s" % (self.src, self.dst)
    @classmethod
    def dispatch_hook(cls, _pkt=None, *args, **kargs):
        if _pkt and len(_pkt) >= 14:
            if struct.unpack("!H", _pkt[12:14])[0] > 1500:
                return Ether
        return cls


class LLC(Packet):
    name = "LLC"
    fields_desc = [ XByteField("dsap", 0x00),
                    XByteField("ssap", 0x00),
                    ByteField("ctrl", 0) ]

conf.neighbor.register_l3(Ether, LLC, lambda l2,l3: conf.neighbor.resolve(l2,l3.payload))
conf.neighbor.register_l3(Dot3, LLC, lambda l2,l3: conf.neighbor.resolve(l2,l3.payload))


class CookedLinux(Packet):
    name = "cooked linux"
    fields_desc = [ ShortEnumField("pkttype",0, {0: "unicast",
                                                 4:"sent-by-us"}), #XXX incomplete
                    XShortField("lladdrtype",512),
                    ShortField("lladdrlen",0),
                    StrFixedLenField("src","",8),
                    XShortEnumField("proto",0x800,ETHER_TYPES) ]



class SNAP(Packet):
    name = "SNAP"
    fields_desc = [ X3BytesField("OUI",0x000000),
                    XShortEnumField("code", 0x000, ETHER_TYPES) ]

conf.neighbor.register_l3(Dot3, SNAP, lambda l2,l3: conf.neighbor.resolve(l2,l3.payload))


class Dot1Q(Packet):
    name = "802.1Q"
    aliastypes = [ Ether ]
    fields_desc =  [ BitField("prio", 0, 3),
                     BitField("id", 0, 1),
                     BitField("vlan", 1, 12),
                     XShortEnumField("type", 0x0000, ETHER_TYPES) ]
    def answers(self, other):
        if isinstance(other,Dot1Q):
            if ( (self.type == other.type) and
                 (self.vlan == other.vlan) ):
                return self.payload.answers(other.payload)
        else:
            return self.payload.answers(other)
        return 0
    def default_payload_class(self, pay):
        if self.type <= 1500:
            return LLC
        return conf.raw_layer
    def extract_padding(self,s):
        if self.type <= 1500:
            return s[:self.type],s[self.type:]
        return s,None
    def mysummary(self):
        if isinstance(self.underlayer, Ether):
            return  self.underlayer.sprintf("802.1q  %Ether.src%  >  %Ether.dst%  (%Dot1Q.type%)
vlan %Dot1Q.vlan%")
        else:
            return self.sprintf("802.1q (%Dot1Q.type%) vlan %Dot1Q.vlan%")


conf.neighbor.register_l3(Ether, Dot1Q, lambda l2,l3: conf.neighbor.resolve(l2,l3.payload))

class STP(Packet):
    name = "Spanning Tree Protocol"
    fields_desc = [ ShortField("proto", 0),
                    ByteField("version", 0),
                    ByteField("bpdutype", 0),
                    ByteField("bpduflags", 0),
                    ShortField("rootid", 0),
                    MACField("rootmac", ETHER_ANY),
                    IntField("pathcost", 0),
                    ShortField("bridgeid", 0),
                    MACField("bridgemac", ETHER_ANY),
                    ShortField("portid", 0),
                    BCDFloatField("age", 1),
                    BCDFloatField("maxage", 20),
                    BCDFloatField("hellotime", 2),
                    BCDFloatField("fwddelay", 15) ]


class EAPOL(Packet):
    name = "EAPOL"
    fields_desc = [ ByteField("version", 1),
                    ByteEnumField("type", 0, ["EAP_PACKET", "START", "LOGOFF", "KEY", "ASF"]),
                    LenField("len", None, "H") ]

    EAP_PACKET= 0
    START = 1
    LOGOFF = 2
    KEY = 3
    ASF = 4
    def extract_padding(self, s):
        l = self.len
        return s[:l],s[l:]
    def hashret(self):
        return chr(self.type)+self.payload.hashret()
    def answers(self, other):
        if isinstance(other,EAPOL):
            if ( (self.type == self.EAP_PACKET) and
                 (other.type == self.EAP_PACKET) ):
```

```
            return self.payload.answers(other.payload)
        return 0
    def mysummary(self):
        return self.sprintf("EAPOL %EAPOL.type%")


class EAP(Packet):
    name = "EAP"
    fields_desc = [ ByteEnumField("code", 4, {1:"REQUEST",2:"RESPONSE",3:"SUCCESS",4:"FAILURE"}),
                ByteField("id", 0),
                ShortField("len",None),
                ConditionalField(ByteEnumField("type",0, {1:"ID",4:"MD5"}), lambda pkt:pkt.code not in [EAP.SUCCESS, EAP.FAILURE])

                    ]

    REQUEST = 1
    RESPONSE = 2
    SUCCESS = 3
    FAILURE = 4
    TYPE_ID = 1
    TYPE_MD5 = 4
    def answers(self, other):
        if isinstance(other,EAP):
            if self.code == self.REQUEST:
                return 0
            elif self.code == self.RESPONSE:
                if ( (other.code == self.REQUEST) and
                     (other.type == self.type) ):
                    return 1
            elif other.code == self.RESPONSE:
                return 1
        return 0

    def post_build(self, p, pay):
        if self.len is None:
            l = len(p)+len(pay)
            p = p[:2]+chr((l>>8)&0xff)+chr(l&0xff)+p[4:]
        return p+pay


class ARP(Packet):
    name = "ARP"
    fields_desc = [ XShortField("hwtype", 0x0001),
                XShortEnumField("ptype",  0x0800, ETHER_TYPES),
                ByteField("hwlen", 6),
                ByteField("plen", 4),
                ShortEnumField("op", 1, {"who-has":1, "is-at":2, "RARP-req":3, "RARP-rep":4, "Dyn-RARP-req":5, "Dyn-RAR-rep":6, "Dyn-RARP-err":7, "InARP-req":8, "InARP-rep":9}),
                ARPSourceMACField("hwsrc"),
                SourceIPField("psrc","pdst"),
                MACField("hwdst", ETHER_ANY),
                IPField("pdst", "0.0.0.0") ]
    who_has = 1
    is_at = 2
    def answers(self, other):
        if isinstance(other,ARP):
            if ( (self.op == self.is_at) and
                 (other.op == self.who_has) and
                 (self.psrc == other.pdst) ):
                return 1
        return 0
    def route(self):
        dst = self.pdst
        if isinstance(dst,Gen):
            dst = iter(dst).next()
        return conf.route.route(dst)
    def extract_padding(self, s):
        return "",s
    def mysummary(self):
        if self.op == self.is_at:
            return self.sprintf("ARP is at %hwsrc% says %psrc%")
        elif self.op == self.who_has:
            return self.sprintf("ARP who has %pdst% says %psrc%")
        else:
            return self.sprintf("ARP %op% %psrc% > %pdst%")

conf.neighbor.register_l3(Ether, ARP, lambda l2,l3: getmacbyip(l3.pdst))

class GRErouting(Packet):
    name = "GRE routing informations"
    fields_desc = [ ShortField("address_family",0),
                    ByteField("SRE_offset", 0),
                    FieldLenField("SRE_len", None, "routing_info", "B"),
                    StrLenField("routing_info", "", "SRE_len"),
                    ]


class GRE(Packet):
    name = "GRE"
    fields_desc = [ BitField("chksum_present",0,1),
                    BitField("routing_present",0,1),
                    BitField("key_present",0,1),
                    BitField("seqnum_present",0,1),
                    BitField("strict_route_source",0,1),
                    BitField("recursion_control",0,3),
                    BitField("flags",0,5),
                    BitField("version",0,3),
                    XShortEnumField("proto", 0x0000, ETHER_TYPES),
                    ConditionalField(XShortField("chksum",None),                                    lambda
pkt:pkt.chksum_present==1 or pkt.routing_present==1),
                    ConditionalField(XShortField("offset",None),  lambda  pkt:pkt.chksum_present==1
or pkt.routing_present==1),
                    ConditionalField(XIntField("key",None), lambda pkt:pkt.key_present==1),
                    ConditionalField(XIntField("seqence_number",None),                              lambda
pkt:pkt.seqnum_present==1),
                    ]
    def post_build(self, p, pay):
        p += pay
        if self.chksum_present and self.chksum is None:
            c = checksum(p)
            p = p[:4]+chr((c>>8)&0xff)+chr(c&0xff)+p[6:]
        return p
```

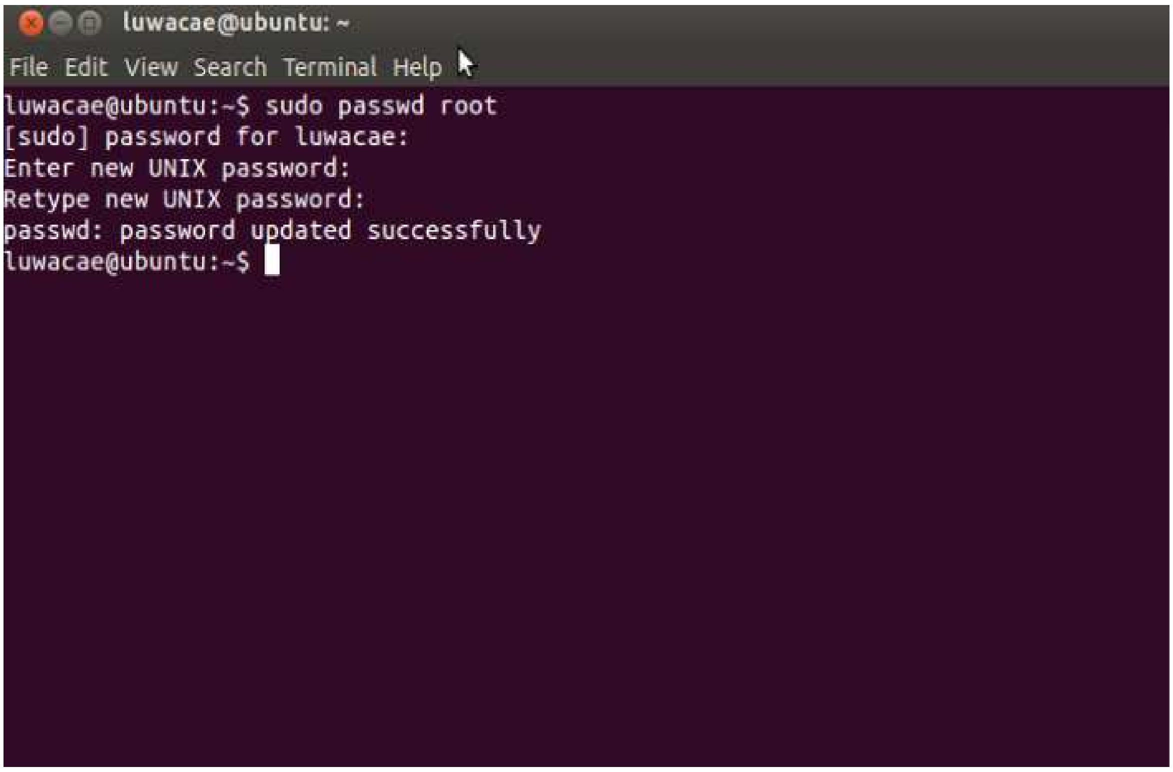
```
bind_layers( Dot3,          LLC,           )
bind_layers( Ether,         LLC,           type=122)
bind_layers( Ether,         Dot1Q,         type=33024)
bind_layers( Ether,         Ether,         type=1)
bind_layers( Ether,         ARP,           type=2054)
bind_layers( Ether,         EAPOL,         type=34958)
bind_layers( Ether,         EAPOL,         dst='01:80:c2:00:00:03', type=34958)
bind_layers( CookedLinux,   LLC,           proto=122)
bind_layers( CookedLinux,   Dot1Q,         proto=33024)
bind_layers( CookedLinux,   Ether,         proto=1)
bind_layers( CookedLinux,   ARP,           proto=2054)
bind_layers( CookedLinux,   EAPOL,         proto=34958)
bind_layers( GRE,           LLC,           proto=122)
bind_layers( GRE,           Dot1Q,         proto=33024)
bind_layers( GRE,           Ether,         proto=1)
bind_layers( GRE,           ARP,           proto=2054)
bind_layers( GRE,           EAPOL,         proto=34958)
bind_layers( GRE,           GRErouting,    { "routing_present" : 1 } )
bind_layers( GRErouting,    conf.raw_layer,{ "address_family" : 0, "SRE_len" : 0 })
bind_layers( GRErouting,    GRErouting,    { } )
bind_layers( EAPOL,         EAP,           type=0)
bind_layers( LLC,           STP,           dsap=66, ssap=66, ctrl=3)
bind_layers( LLC,           SNAP,          dsap=170, ssap=170, ctrl=3)
bind_layers( SNAP,          Dot1Q,         code=33024)
bind_layers( SNAP,          Ether,         code=1)
bind_layers( SNAP,          ARP,           code=2054)
bind_layers( SNAP,          EAPOL,         code=34958)
bind_layers( SNAP,          STP,           code=267)

conf.l2types.register(ARPHDR_ETHER, Ether)
conf.l2types.register_num2layer(ARPHDR_METRICOM, Ether)
conf.l2types.register_num2layer(ARPHDR_LOOPBACK, Ether)
conf.l2types.register_layer2num(ARPHDR_ETHER, Dot3)
conf.l2types.register(144, CookedLinux)  # called LINUX_IRDA, similar to CookedLinux
conf.l2types.register(113, CookedLinux)

conf.l3types.register(ETH_P_ARP, ARP)




@conf.commands.register
def arpcachepoison(target, victim, interval=60):
    """Poison target's cache with (your MAC,victim's IP) couple
arpcachepoison(target, victim, [interval=60]) -> None
"""
    tmac = getmacbyip(target)
    p = Ether(dst=tmac)/ARP(op="who-has", psrc=victim, pdst=target)
    try:
        while 1:
            sendp(p, iface_hint=target)
            if conf.verb > 1:
                os.write(1,".")
            time.sleep(interval)
    except KeyboardInterrupt:
        pass


class ARPingResult(SndRcvList):
    def __init__(self, res=None, name="ARPing", stats=None):
        SndRcvList.__init__(self, res, name, stats)

    def show(self):
        for s,r in self.res:
            print r.sprintf("%19s,Ether.src% %ARP.psrc%")



@conf.commands.register
def arping(net, timeout=2, cache=0, verbose=None, **kargs):
    """Send ARP who-has requests to determine which hosts are up
arping(net, [cache=0,] [iface=conf.iface,] [verbose=conf.verb]) -> None
Set cache=True if you want arping to modify internal ARP-Cache"""
    if verbose is None:
        verbose = conf.verb
    ans,unans = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=net), verbose=verbose,
                    filter="arp and arp[7] = 2", timeout=timeout, iface_hint=net, **kargs)
    ans = ARPingResult(ans.res)

    if cache and ans is not None:
        for pair in ans:
            conf.netcache.arp_cache[pair[1].psrc] = (pair[1].hwsrc, time.time())
    if verbose:
        ans.show()
    return ans,unans

@conf.commands.register
def is_promisc(ip, fake_bcast="ff:ff:00:00:00:00",**kargs):
    """Try to guess if target is in Promisc mode. The target is provided by its ip."""

    responses = srp1(Ether(dst=fake_bcast) / ARP(op="who-has", pdst=ip),type=ETH_P_ARP,
iface_hint=ip, timeout=1, verbose=0,**kargs)

    return responses is not None

@conf.commands.register
def promiscping(net, timeout=2, fake_bcast="ff:ff:ff:ff:ff:fe", **kargs):
    """Send ARP who-has requests to determine which hosts are in promiscuous mode
    promiscping(net, iface=conf.iface)"""
    ans,unans = srp(Ether(dst=fake_bcast)/ARP(pdst=net),
                    filter="arp and arp[7] = 2", timeout=timeout, iface_hint=net, **kargs)
    ans = ARPingResult(ans.res, name="PROMISCPing")

    ans.display()
    return ans,unans


class ARP_am(AnsweringMachine):
    function_name="farpd"
    filter = "arp"
    send_function = staticmethod(sendp)

    def parse_options(self, IP_addr=None, iface=None, ARP_addr=None):
        self.IP_addr=IP_addr
        self.iface=iface
        self.ARP_addr=ARP_addr

    def is_request(self, req):
        return (req.haslayer(ARP) and
                req.getlayer(ARP).op == 1 and
                (self.IP_addr == None or self.IP_addr == req.getlayer(ARP).pdst))

    def make_reply(self, req):
        ether = req.getlayer(Ether)
        arp = req.getlayer(ARP)
        iff,a,gw = conf.route.route(arp.psrc)
        if self.iface != None:
            iff = iface
        ARP_addr = self.ARP_addr
        IP_addr = arp.pdst
        resp = Ether(dst=ether.src,
                     src=ARP_addr)/ARP(op="is-at",
                                       hwsrc=ARP_addr,
                                       psrc=IP_addr,
                                       hwdst=arp.hwsrc,
                                       pdst=arp.pdst)
        return resp

    def sniff(self):
        sniff(iface=self.iface, **self.optsniff)

@conf.commands.register
def etherleak(target, **kargs):
    """Exploit Etherleak flaw"""
    return srpflood(Ether()/ARP(pdst=target),
                    prn=lambda (s,r): conf.padding_layer in r and hexstr(r[conf.padding_layer].load),
                    filter="arp", **kargs)
```

# APPENDIX C

## Root user access on Linux Ubuntu 10.04

Linux operating systems have a robust permissions system. This is often a very good thing, as it enables a clear separation of roles among users. Linux operating systems have 6 user levels from a single user all the way to a root user.  A root user account by default has access to all commands and files on a Linux or other Unix-like operating system

Root access is required to perform a lot of administrative actions on Linux Ubuntu, such as installing software. Root user access is locked by default on Linux Ubuntu. Commands are however provide temporary administrators access to root commands. For the installing of the required packages and to run Wireshark on Ubuntu 10.04, root user access is required. Root user access can be enabled on Linux Ubuntu as shown below

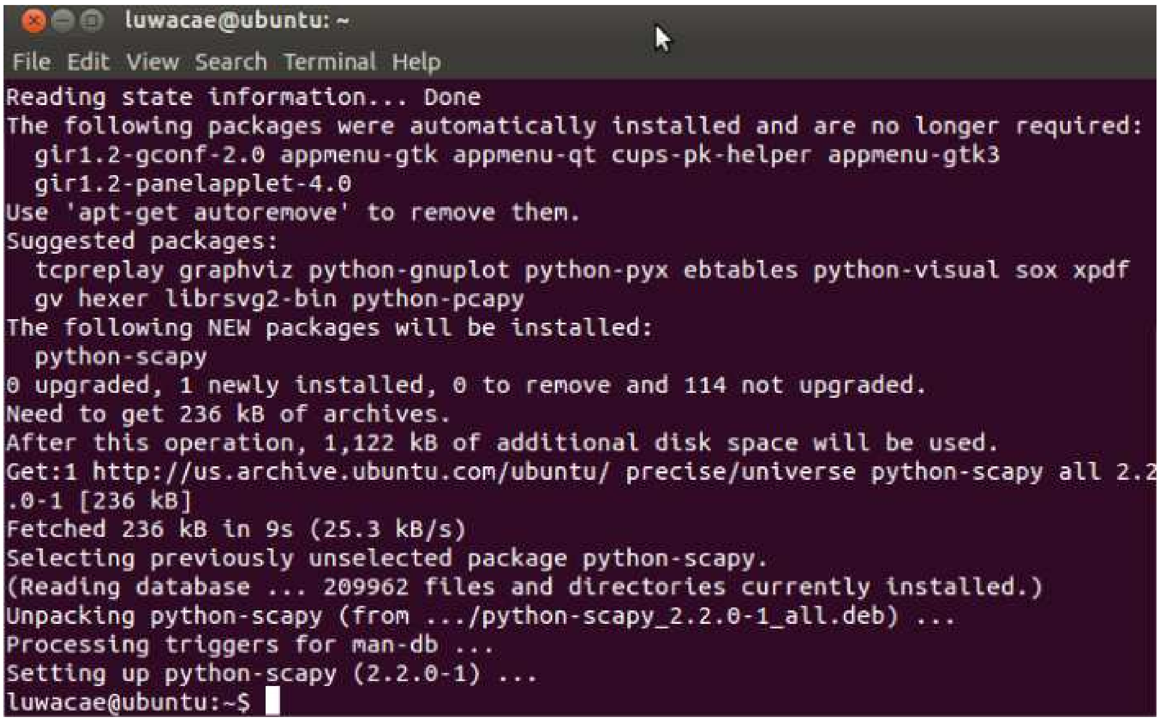


```
luwacae@ubuntu:~$ sudo passwd root
[sudo] password for luwacae:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
luwacae@ubuntu:~$
```

# APPENDIX D

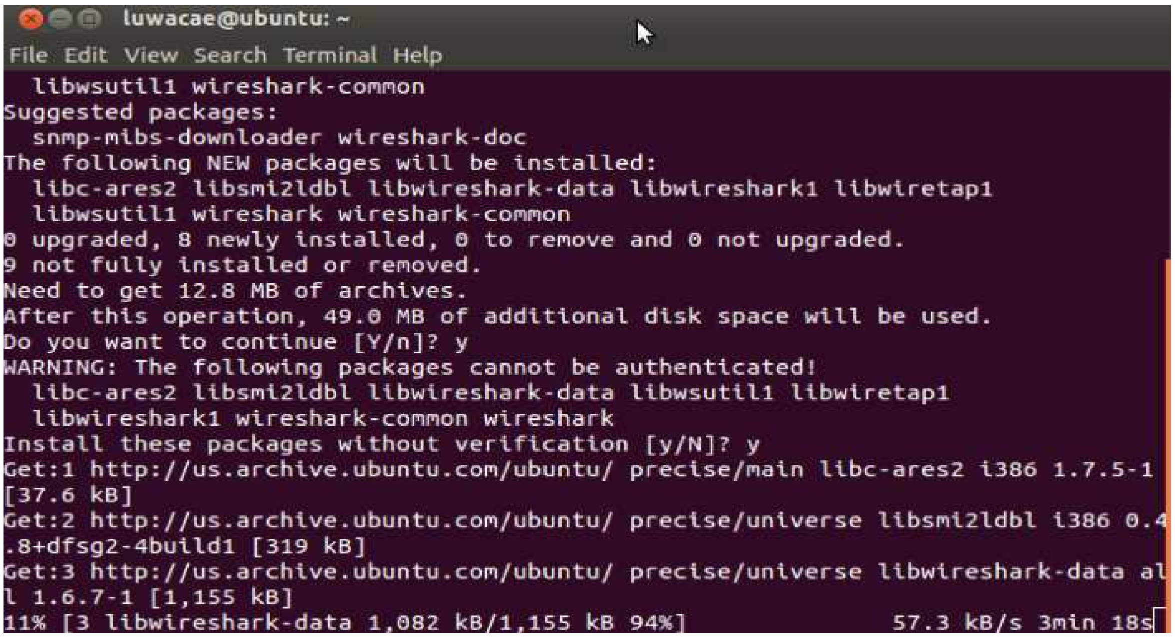
## Installing Scapy and the required packages on Linux Ubuntu

### Installing Scapy on Linux Ubuntu



```
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gir1.2-gconf-2.0 appmenu-gtk appmenu-qt cups-pk-helper appmenu-gtk3
  gir1.2-panelapplet-4.0
Use 'apt-get autoremove' to remove them.
Suggested packages:
  tcpreplay graphviz python-gnuplot python-pyx ebtables python-visual sox xpdf
  gv hexer librsvg2-bin python-pcapy
The following NEW packages will be installed:
  python-scapy
0 upgraded, 1 newly installed, 0 to remove and 114 not upgraded.
Need to get 236 kB of archives.
After this operation, 1,122 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise/universe python-scapy all 2.2
.0-1 [236 kB]
Fetched 236 kB in 9s (25.3 kB/s)
Selecting previously unselected package python-scapy.
(Reading database ... 209962 files and directories currently installed.)
Unpacking python-scapy (from .../python-scapy_2.2.0-1_all.deb) ...
Processing triggers for man-db ...
Setting up python-scapy (2.2.0-1) ...
luwacae@ubuntu:~$
```

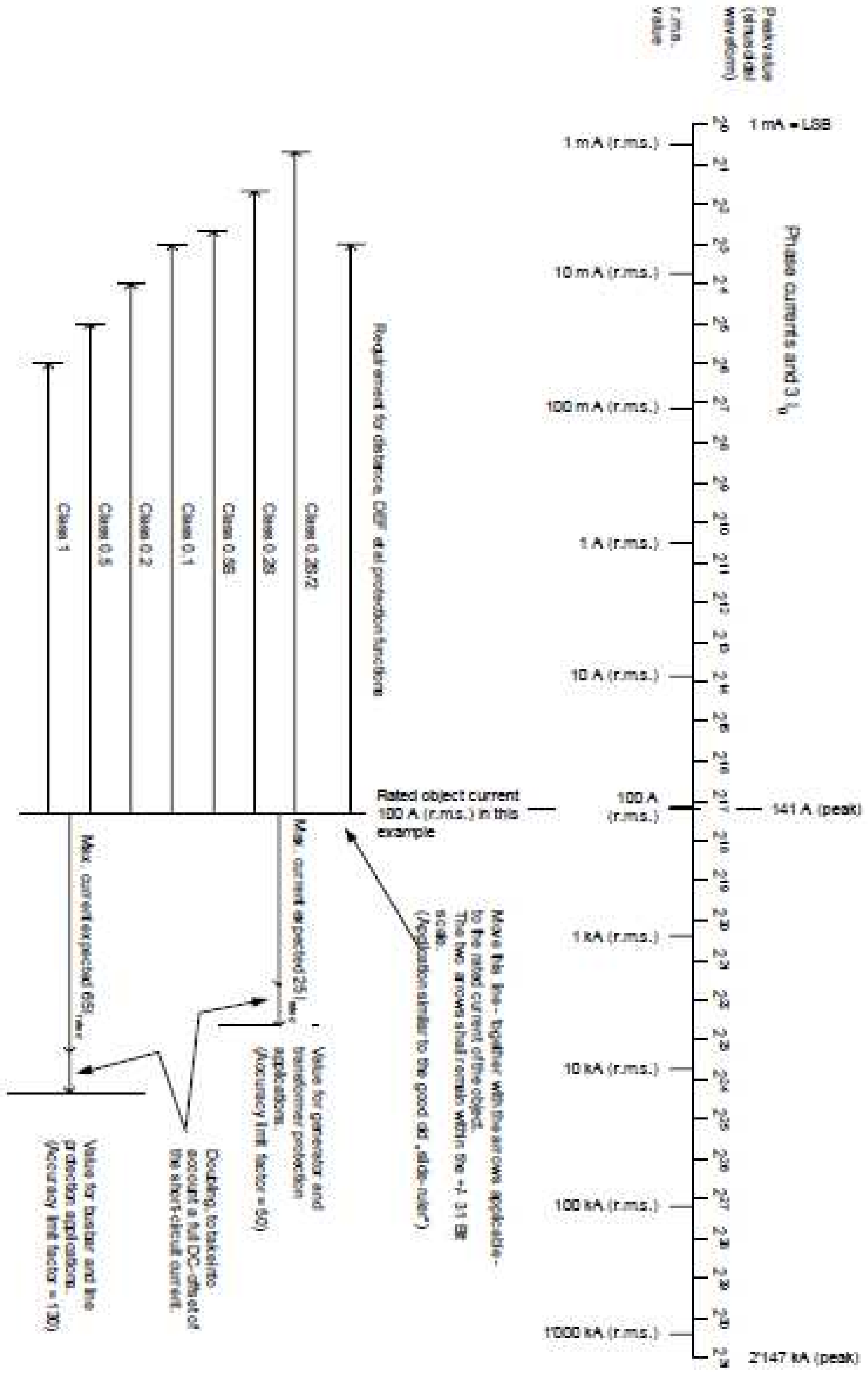
### Installing Scapy on Wireshark



```
  libwsutil1 wireshark-common
Suggested packages:
  snmp-mibs-downloader wireshark-doc
The following NEW packages will be installed:
  libc-ares2 libsmi2ldbl libwireshark-data libwireshark1 libwiretap1
  libwsutil1 wireshark wireshark-common
0 upgraded, 8 newly installed, 0 to remove and 0 not upgraded.
9 not fully installed or removed.
Need to get 12.8 MB of archives.
After this operation, 49.0 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
WARNING: The following packages cannot be authenticated!
  libc-ares2 libsmi2ldbl libwireshark-data libwsutil1 libwiretap1
  libwireshark1 wireshark-common wireshark
Install these packages without verification [y/N]? y
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise/main libc-ares2 i386 1.7.5-1
[37.6 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ precise/universe libsmi2ldbl i386 0.4
.8+dfsg2-4build1 [319 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu/ precise/universe libwireshark-data al
l 1.6.7-1 [1,155 kB]
11% [3 libwireshark-data 1,082 kB/1,155 kB 94%]            57.3 kB/s 3min 18s
```

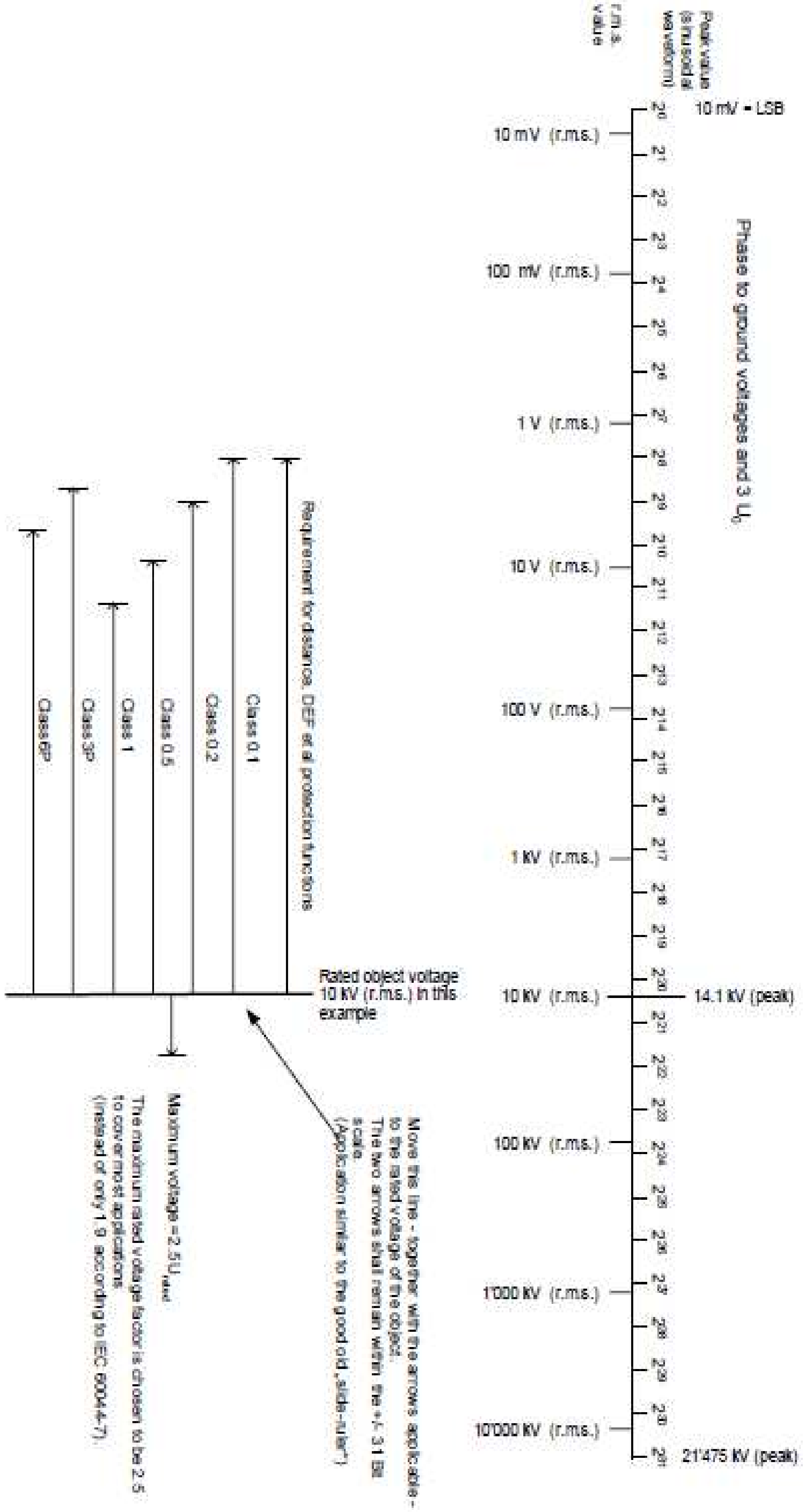
# APPENDIX E

## Nomogram for current


Peak value (sinusoidal waveform)
r.m.s. value
1 mA = LSB
Phase currents and 3 I0
1 mA (r.m.s.)
10 mA (r.m.s.)
100 mA (r.m.s.)
1 A (r.m.s.)
10 A (r.m.s.)
100 A (r.m.s.)
141 A (peak)
1 kA (r.m.s.)
10 kA (r.m.s.)
100 kA (r.m.s.)
1000 kA (r.m.s.)
2'147 kA (peak)
Requirement for distance, Diff. dial protection functions
Class 1
Class 0.5
Class 0.2
Class 0.1
Class 0.05
Class 0.2S
Class 0.2S/2
Rated object current 100 A (r.m.s.) in this example
Move this line - together with the arrows applicable - to the rated current of the object. The two arrows shall remain within the +/- 31 Bit scale. (Application similar to the good old „slide-ruler")
Max. current expected 65 I_rated
Max. current expected 25 I_rated
Value for generator and transformer protection applications. (Accuracy limit factor = 50)
Doubling, to take into account a full DC-offset of the short-circuit current.
Value for busbar and line protection applications. (Accuracy limit factor = 130)

# APPENDIX F

## Nomogram for voltage

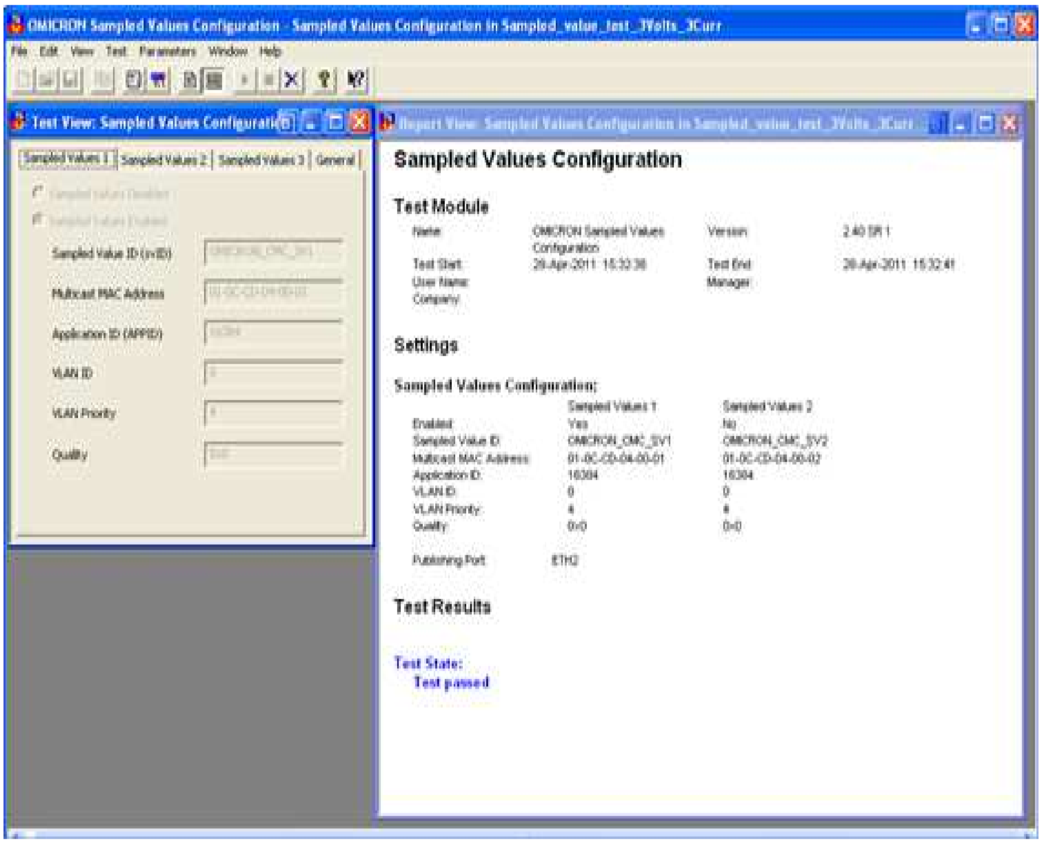


Peak value (sinusoidal waveform)
r.m.s. value
10 mV = LSB
10 mV (r.m.s.)
100 mV (r.m.s.)
1 V (r.m.s.)
10 V (r.m.s.)
100 V (r.m.s.)
1 kV (r.m.s.)
10 kV (r.m.s.)
14.1 kV (peak)
100 kV (r.m.s.)
1'000 kV (r.m.s.)
10'000 kV (r.m.s.)
21'475 kV (peak)
$2^0$ $2^1$ $2^2$ $2^3$ $2^4$ $2^5$ $2^6$ $2^7$ $2^8$ $2^9$ $2^{10}$ $2^{11}$ $2^{12}$ $2^{13}$ $2^{14}$ $2^{15}$ $2^{16}$ $2^{17}$ $2^{18}$ $2^{19}$ $2^{20}$ $2^{21}$ $2^{22}$ $2^{23}$ $2^{24}$ $2^{25}$ $2^{26}$ $2^{27}$ $2^{28}$ $2^{29}$ $2^{30}$ $2^{31}$
Phase to ground voltages and 3 $U_0$
Requirement for distance, DEF et al protection functions
Class 6P
Class 3P
Class 1
Class 0.5
Class 0.2
Class 0.1
Rated object voltage 10 kV (r.m.s.) in this example
Maximum voltage = 2.5 $U_{rated}$
The maximum rated voltage factor is chosen to be 2.5 to cover most applications (instead of only 1.9 according to IEC 60044-7).
Move this line - together with the arrows applicable - to the rated voltage of the object. The two arrows shall remain within the +/- 31 Bit scale. (Application similar to the good old „slide-ruler“)

# APPENDIX G

## Setting up Omicron CMC for sampled value publishing

The newer versions of the Omicron CMC such as CMC 256, CMC 353, CMC 356, CMC 850, etc. provide support for IEC 61850 GOOSE and Sampled Values (Yang et al., 2009).


OMICRON Sampled Values Configuration - Sampled Values Configuration in Sampled_value_test_3Volts_3Curr
File Edit View Test Parameters Window Help
Test View: Sampled Values Configuration
Sampled Values 1
Sampled Values 2
Sampled Values 3
General
Sampled Values Disabled
Sampled Values Enabled
Report View: Sampled Values Configuration in Sampled_value_test_3Volts_3Curr

Sampled Value ID (svID) OMICRON_CMC_SV1
Multicast MAC Address 01-0C-CD-04-00-01
Application ID (APPID) 16384
VLAN ID 0
VLAN Priority 4
Quality 0x0

**Sampled Values Configuration**

**Test Module**

| | | | |
|---|---|---|---|
| Name | OMICRON Sampled Values Configuration | Version | 2.40 SR 1 |
| Test Start | 20-Apr-2011 15:32:38 | Test End | 20-Apr-2011 15:32:41 |
| User Name | | Manager | |
| Company | | | |

**Settings**

**Sampled Values Configuration:**

| | Sampled Values 1 | Sampled Values 2 |
|---|---|---|
| Enabled | Yes | No |
| Sampled Value ID | OMICRON_CMC_SV1 | OMICRON_CMC_SV2 |
| Multicast MAC Address | 01-0C-CD-04-00-01 | 01-0C-CD-04-00-02 |
| Application ID | 16384 | 16384 |
| VLAN ID | 0 | 0 |
| VLAN Priority | 4 | 4 |
| Quality | 0x0 | 0x0 |

Publishing Port ETH2

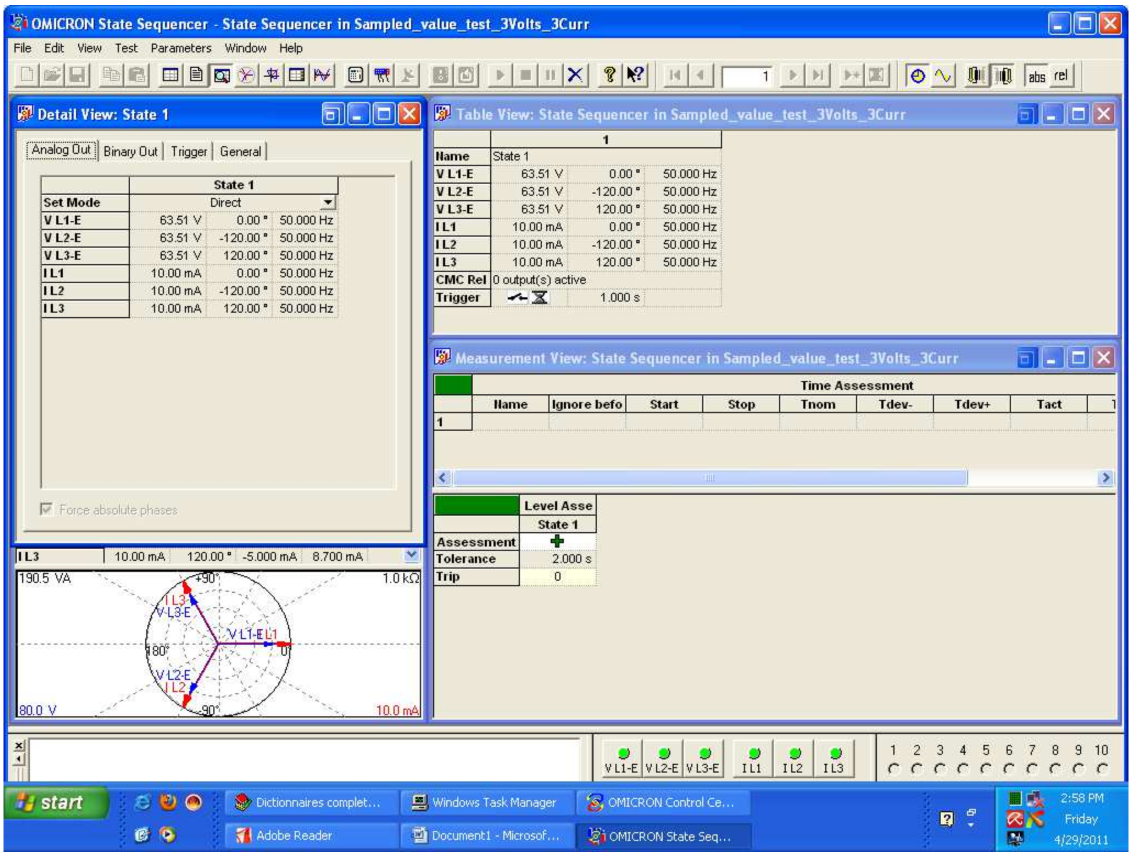
**Test Results**

**Test State:**
**Test passed**

Omicron CMC Sampled Values support

The procedure followed when setting up the CMC to allow for Sampled Value outputting is:

i) Associating the PC being used for configuring the CMC with the device
ii) Setting up the communication parameters
iii) Setting up The voltage and current magnitude and phase angles
iv) Setting up some trigger mechanisms, this is optional depending on the test being performed.
v) Initiating the publishing of sampled values



OMICRON State Sequencer - State Sequencer in Sampled_value_test_3Volts_3Curr

Detail View: State 1

|  | State 1 |  |  |
|---|---|---|---|
| Set Mode | Direct |  |  |
| V L1-E | 63.51 V | 0.00 ° | 50.000 Hz |
| V L2-E | 63.51 V | -120.00 ° | 50.000 Hz |
| V L3-E | 63.51 V | 120.00 ° | 50.000 Hz |
| I L1 | 10.00 mA | 0.00 ° | 50.000 Hz |
| I L2 | 10.00 mA | -120.00 ° | 50.000 Hz |
| I L3 | 10.00 mA | 120.00 ° | 50.000 Hz |

Force absolute phases

I L3 10.00 mA 120.00 ° -5.000 mA 8.700 mA

Table View: State Sequencer in Sampled_value_test_3Volts_3Curr

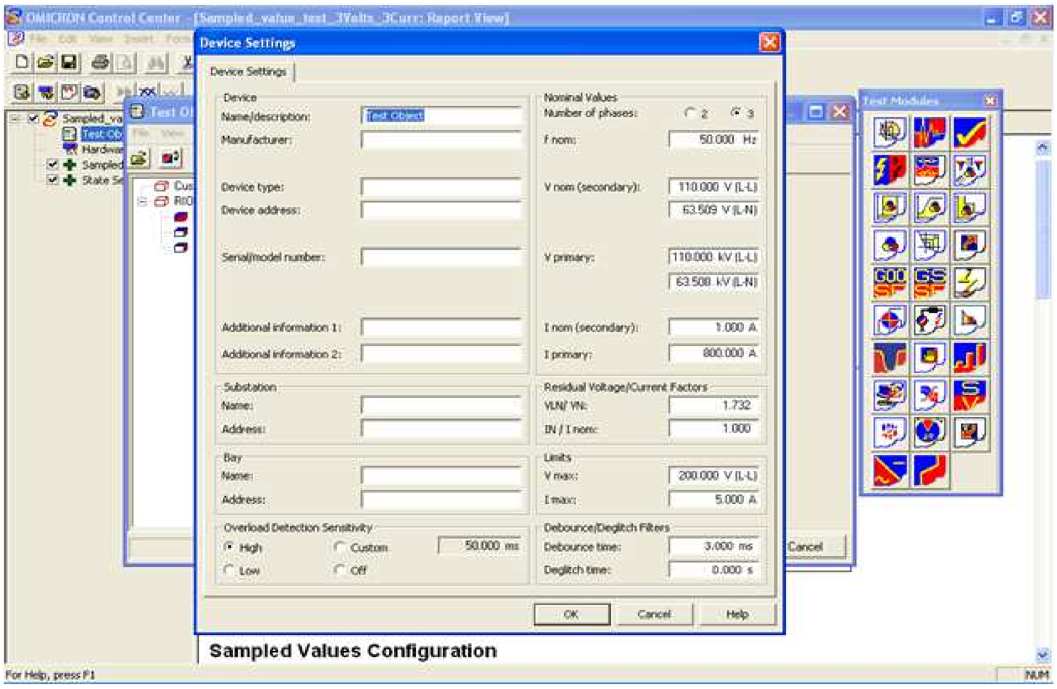
|  | 1 |  |  |
|---|---|---|---|
| Name | State 1 |  |  |
| V L1-E | 63.51 V | 0.00 ° | 50.000 Hz |
| V L2-E | 63.51 V | -120.00 ° | 50.000 Hz |
| V L3-E | 63.51 V | 120.00 ° | 50.000 Hz |
| I L1 | 10.00 mA | 0.00 ° | 50.000 Hz |
| I L2 | 10.00 mA | -120.00 ° | 50.000 Hz |
| I L3 | 10.00 mA | 120.00 ° | 50.000 Hz |
| CMC Rel | 0 output(s) active |  |  |
| Trigger |  | 1.000 s |  |

Measurement View: State Sequencer in Sampled_value_test_3Volts_3Curr

|  | Name | Ignore befo | Start | Stop | Tnom | Tdev- | Tdev+ | Tact |
|---|---|---|---|---|---|---|---|---|
| 1 |  |  |  |  |  |  |  |  |

|  | Level Asse |
|---|---|
|  | State 1 |
| Assessment | ✚ |
| Tolerance | 2.000 s |
| Trip | 0 |

Setting up the magnitude and phase angles on Omicron CMC (phase to neutral values



OMICRON Control Center - [Sampled_value_test_3Volts_3Curr: Report View]

Device Settings

| Device |  |
|---|---|
| Name/description: | Test Object |
| Manufacturer: |  |
| Device type: |  |
| Device address: |  |
| Serial/model number: |  |
| Additional information 1: |  |
| Additional information 2: |  |

| Nominal Values |  |
|---|---|
| Number of phases: | 2 [✓ 3] |
| f nom: | 50.000 Hz |
| V nom (secondary): | 110.000 V (L-L) |
|  | 63.509 V (L-N) |
| V primary: | 110.000 kV (L-L) |
|  | 63.508 kV (L-N) |
| I nom (secondary): | 1.000 A |
| I primary: | 800.000 A |

| Substation |  |
|---|---|
| Name: |  |
| Address: |  |

| Residual Voltage/Current Factors |  |
|---|---|
| VLN/ VN: | 1.732 |
| IN / I nom: | 1.000 |

| Bay |  |
|---|---|
| Name: |  |
| Address: |  |

| Limits |  |
|---|---|
| V max: | 200.000 V (L-L) |
| I max: | 5.000 A |

Overload Detection Sensitivity: [✓ High] Low Custom Off 50.000 ms

| Debounce/Deglitch Filters |  |
|---|---|
| Debounce time: | 3.000 ms |
| Deglitch time: | 0.000 s |

Sampled Values Configuration

Omicron CMC device setup (line –line values)

# APPENDIX H

## Code for the sampled value frame structure

```
NOTE: The code is not complete and represents on the frame structure developed
'' Emmanuel Luwaca Sampled Value frame structure '''
'''---------------------------------------------'''

SampleValue_new                                                               =
Ether(dst=""01:0C:CD:04:ff:ff"",type=0x8100)/Dot1Q(prio=4,id=0,type=0x88ba)

class Appid(Packet):
  name = "AppidPacket"
  fields_desc = [ XShortField ("EmAPPID",0x4000),
                  XShortField ("Em_LengthPDU",0x71),
                  XShortField ("Em_Reserve2",0x00),
                  XShortField ("Em_Reserve1",0x00)]



MyAppid = Appid(EmAPPID=0x4000)
MyLengthPDU = Appid(Em_LengthPDU=0x71)
MyReserve1 = Appid(Em_Reserve1=0x00)
MyReserve2 = Appid(Em_Reserve2=0x00)


class svPdu(Packet):
  name = "svPduPacket"
  fields_desc = [ XShortField ("Em_svPdu",0x6067),
                  XShortField ("Em_noASDU",0x8001),
                  XByteField ("Em_noASDU_value",0x01),
                XShortField ("Em_seqASDU",0xa262),
                  XShortField ("Em_seqASDU_value",0x3060)]


MysvPdu = svPdu(Em_svPdu=0x6067)
MynoASDU = svPdu(Em_noASDU=0x8001)
MynoASDU_value = svPdu(Em_noASDU_value =0x01)
MyseqASDU = svPdu(Em_seqASDU=0xa262)
MyseqASDU_value = svPdu(Em_seqASDU_value =0x3060)

class ASDU(Packet):
  name = "ASDUPacket"
```

```
    fields_desc = [ XShortField ("svID_pad",0x800f),
                    StrField("svID","'Emanuel_cput_sa'"),
                    XShortField ("smpCnt_pad",0x8202),
                    ShortField ("smpCnt_value",0x00),
                    XShortField ("confRef_pad",0x8304),
                    XIntField ("confRef_value",0x1),
                    XShortField ("smpSynch_pad",0x8501),
                    XByteField ("smpSynch_value",0x00)]


MysvID_pad = ASDU(svID_pad=0x800f)
MysvID = ASDU(svID = "'Emanuel_cput_sa'")
MysmpCnt_pad = ASDU(smpCnt_pad=0x8202)
MysmpCnt_value = ASDU(smpCnt_value=0x00)
MyconfRef_pad = ASDU(confRef_pad=0x8304)
MyconfRef_value = ASDU(confRef_value=0x1)
MysmpSynch_pad = ASDU(smpSynch_pad=0x8501)
MysmpSynch_value = ASDU(smpSynch_value=0x1)


class PhsMeas1(Packet):
    name = "PhsMeas1Packet"
    fields_desc = [ XShortField ("Phs1_pad",0x8740),
                    XIntField("Ia_value",0x00),
                    XIntField("Ia_quality",0x00),
                    XIntField("Ib_value",0x00),
                    XIntField("Ib_quality",0x00),
                    XIntField("Ic_value",0x00),
                    XIntField("Ic_quality",0x00),
                    XIntField("In_value",0x00),
                    XIntField("In_quality",0x2000),
                    XIntField("Va_value",0x00),
                    XIntField("Va_quality",0x00),
                    XIntField("Vb_value",0x00),
                    XIntField("Vb_quality",0x00),
                    XIntField("Vc_value",0x00),
                    XIntField("Vc_quality",0x00),
                    XIntField("Vn_value",0x00),
                    XIntField("Vn_quality",0x2000)]


My_Phs1_pad = PhsMeas1(Phs1_pad=0x8740)
```

```
My_Ia_value = PhsMeas1(Ia_value=0x00)
My_Ib_value = PhsMeas1(Ib_value=0x00)
My_Ic_value = PhsMeas1(Ic_value=0x00)
My_In_value = PhsMeas1(In_value=0x00)
My_Va_value = PhsMeas1(Va_value=0x00)
My_Vb_value = PhsMeas1(Vb_value=0x00)
My_Vc_value = PhsMeas1(Vc_value=0x00)
My_Vn_value = PhsMeas1(Vn_value=0x00)
"""*****************************************************************************
Because you can only bind two layers together so this is a laborious process
*****************************************************************************"""
bind_layers(MyAppid, MyReserve1)
bind_layers(MyReserve1, MyReserve2)
bind_layers(MyAppid, MyReserve1, proto=0x88ba)
bind_layers(MyReserve1, MyReserve2, proto=0x88ba)
bind_layers(MyReserve2, MyLengthPDU, proto=0x88ba)
bind_layers(MyReserve2, MysvPdu, proto=0x88ba)
bind_layers(MysvPdu, MynoASDU, proto=0x88ba)
bind_layers(MynoASDU, MynoASDU_value, proto=0x88ba)
bind_layers(MynoASDU_value, MyseqASDU, proto=0x88ba)
bind_layers(MyseqASDU, MyseqASDU_value, proto=0x88ba)
bind_layers(MyseqASDU_value, MysvID_pad, proto=0x88ba)
bind_layers(MysvID_pad, MysvID, proto=0x88ba)
bind_layers(MysvID, MysmpCnt_pad, proto=0x88ba)
bind_layers(MysmpCnt_pad, MysmpCnt_value, proto=0x88ba)
bind_layers(MysmpCnt_value, MyconfRef_pad, proto=0x88ba)
bind_layers(MyconfRef_pad, MyconfRef_value, proto=0x88ba)
bind_layers(MyconfRef_value, MysmpSynch_pad, proto=0x88ba)
bind_layers(MysmpSynch_pad, MysmpSynch_value, proto=0x88ba)
bind_layers(MysmpSynch_value, My_Phs1_pad, proto=0x88ba)

"""*****************************************************************************
SampleValue = SampleValue_new/MyAppid/MysvPdu/MysvID_pad/My_Phs1_pad
sendp((SampleValue),iface="eth0")
*****************************************************************************"""
```

# APPENDIX I

## Enabling Eth0 on Linux Ubuntu

By default eth0 is not setup automatically on Linux Ubuntu 10.04, and needs to be enabled and setup to start automatically. There are various ways of setting up the network interface to start-up automatically. As shown below, root users on Linux Ubuntu can access the network script and enable the network interface.

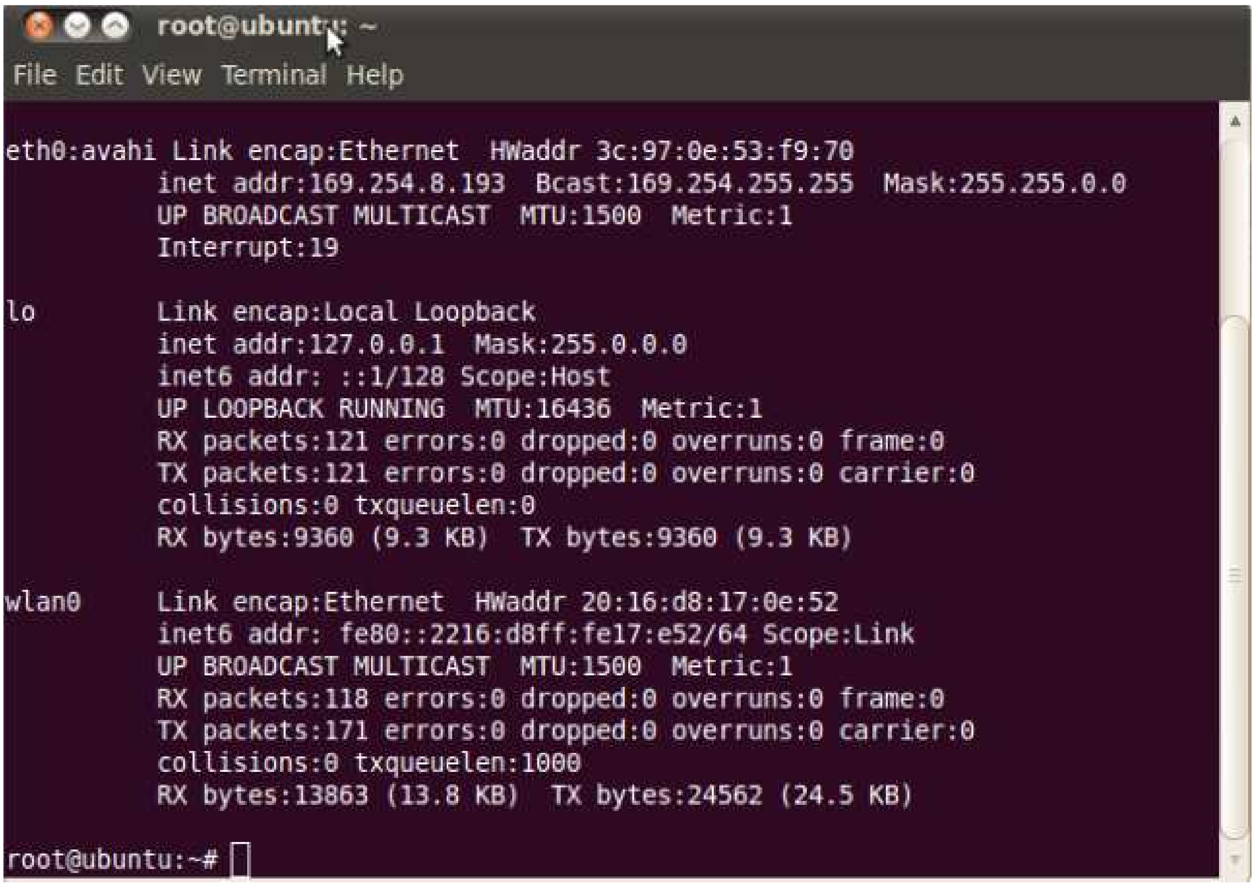
```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
```

After the script has been edited the network interface needs to be restarted. The script for the network driver is also shown below. The network driver being utilised is Eth0: Avahi

```
#!/bin/sh

set -e

# Description:	Remove routes to allow communication between machines which
#		only have an IPv4LL address assigned and those which only
#		have a routable address assigned. These were added by
#		/etc/network/if-up.d/avahi-autoipd.
#
#		See http://developer.apple.com/qa/qa2004/qa1357.html for
#		more information.


[ "$IFACE" != "lo" ] || exit 0
case "$ADDRFAM" in
  inet|NetworkManager) ;;
  *) exit 0
esac
case "$METHOD" in
	static|dhcp|NetworkManager) ;;
	*) exit 0
esac
```

```
if [ -x /bin/ip ]; then
        # route already present?
        ip route show | grep -q '^169.254.0.0/16[[:space:]]' && exit 0

        /bin/ip route del 169.254.0.0/16 dev $IFACE metric 1000 scope link || true
elif [ -x /sbin/route ]; then
        # route already present?
        /sbin/route -n | grep -q "^169.254.0.0[[:space:]]" && exit 0

        /sbin/route del -net 169.254.0.0 netmask 255.255.0.0 dev $IFACE metric 1000 || true
fi
```

After the network card has been restarted, the operation of the interface can be verified simple by typing ifconfig, as shown below



```
eth0:avahi Link encap:Ethernet  HWaddr 3c:97:0e:53:f9:70
          inet addr:169.254.8.193  Bcast:169.254.255.255  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Interrupt:19

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:121 errors:0 dropped:0 overruns:0 frame:0
          TX packets:121 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9360 (9.3 KB)  TX bytes:9360 (9.3 KB)

wlan0     Link encap:Ethernet  HWaddr 20:16:d8:17:0e:52
          inet6 addr: fe80::2216:d8ff:fe17:e52/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:118 errors:0 dropped:0 overruns:0 frame:0
          TX packets:171 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13863 (13.8 KB)  TX bytes:24562 (24.5 KB)

root@ubuntu:~# 
```

## REFERENCES

Abdolkhalig, A. and Zivanovic, R., “Performance Evaluation of Phasor Estimator within IEC 61850-9-2 Communication Network”, The International Conference on Electrical and Electronics Engineering, Clean Energy and Green Computing (EEECEGC 2013), Dubai, 2013, pp. 113-119.

Andersson, L., Brunner, C. & Engler, F., (2003), “Substation Automation based on IEC 61850 with newprocess-close Technologies”, IEEE Power Tech Conference Proceedings, Bologna, pp.1-6.

Andersson, L., and Brand K.P. (2000) “The benefits of the coming standard IEC 61850 for communication in substation” Southern African Power System Protection Conference, Johannesburg, 8-9 November 2000

Andersson, L., Brand, K.P., Wimmer, W., “The Impact of the coming Standard IEC61850 on the Life-cycle of Open Communication Systems in Substations” Distribution and Transmission – Brisbane, Australia, Nov 11-14, 2001. pp. 1-9

Apostolov, A., Ziegler, C., DeMicco, E., “Recording power-quality events in IEC 61850-based systems” , Power-quality 2005 conference, Baltimore, MD, October 2005

Apostolov, A. (2011), “Engineering the distribution system of the future” , CIRED 21st International Conference on Electricity Distribution, 2011, pp.1-4

Apostolov, A. (2010). “IEC 61850 Substation Configuration Language and Its Impact on the Engineering of Distribution Substation Systems” CIDAL Argentina 2010 pp.1-6

Apostolov, A. (2011), “Engineering the distribution system of the future” Power Systems Conference and Exposition, PSCE’06, Atlanta, pp.1053-1058.

Apostolov, A. & Tholomier, D. (2006), “Impact of IEC 61850 on Power System Protection”, IEEE PES Power Systems Conference and Exposition, PSCE’06, Atlanta, 1053-1058.

Apostolov, A., Auperrin, F., Passet, R., Guenego, M., and Gilles, F. (2006). “IEC 61850 Process Bus based distributed waveform recording.” IEEE Power Engineering Society General Meeting, June 2006 , pp. 1-6.

Apostolov, A. and Vandiver, B., (2010). "IEC 61850 process bus - principles, applications and benefits". In Protective Relay Engineers, 2010 63rd Annual Conference. College Station, TX, 2010. IEEE. pp.1 – 6

Baigent, D., Adamiak, M. and R. Mackiewicz. “IEC 61850 Communication Networks and Systems In Substations” An Overview for Users. SIPSEP, 2004.

Bertolottol, P., Faifer2, M. and R. Ottoboni2., (2007). “High Voltage Multi-Purpose current and voltage electronic.” Instrumentation and Measurement Technology Conference, 2007. IEEE. pp.1-5

Biondi, P., and the Scapy community, “Scapy Documentation Release” 2003

Biondi, P. and the Scapy community, “Scapy Documentation Release 2.0.1” 2009

Bovet, D.P. and Cesati, M., “Understanding the Linux Kernel” 2001 pp. 1- 684, O'Reilly & Associates, Inc.

Brand, K.P. (2004), “The Standard IEC 61850 as Prerequisite for Intelligent Applications in

Substations", IEEE Power Engineering Society General Meeting, 714-718.

Brunner, C., "Will IEEE 1588 Finally Leverage the IEe 61850 Process Bus?" 2011

Brunner, C., "IEC 61850 Process connection – A smart solution to connect the primary equipment to the substation automation system", ABB Switzerland, 2005.

Brunner, C., Antonova, G.S., "Smarter Time Sync: Applying the IEEE PC37.238 Standard to Power System Applications" IEEE. Protective Relay March 2011, pp.91-102

Brunner, C., Cottens,  E., Genier, M., Muller, D.,  Nibbio, N., Reuter, J., "Engineering approach for the end user in IEC 61850 applications", CIGRE 2010, August 23 – 27, 2010, Paris, France.

Brunner, C. "IEC 61850 for Power System Communication". IEEE/PES Transmission and Distribution Conference and Exposition, 2008. pp.1-6.

Brunner, C. "The Impact of IEC 61850 on Protection", IET 9th International Conference on Developments in Power System Protection, DPSP'08, 2008. pp.14-19.

Brunner, C. "IEC 61850 and Smart Grids", pacworld magazine, UT Innovation, Switzerland.

Cassel, L.N., "Computer Networks and Open Systems: An Application Development Perspective" Authors: Publication: Book 1$^{st}$ Jones and Bartlett Publishers, Inc. , USA 2000

Chen, A., "Requirements for Ethernet Networks in Substation Automation" Moxa white paper Released January 2008. pp. 1-12

Chatrefou, D., "Process bus application in NCIT experiments; impact on future architectures" Proc. 17th Conf. Elec. Pwr Sup. Ind. (CEPSI), Macau, China, 27–31 Oct. 2008

Christensen, L.H. (2005), "Design, construction, and test of a passive optical prototype high voltage instrument transformer." IEEE Transactions on Power Delivery, v. 10, No.3, July, 1995.

Elmenreich, W., Haidinger, W., Kopetz, H., Losert, T., Obermaisser, R., Paulitsch, M., Peti, P.,  "A standard for real-time smart transducer interface." 2005. pp.613-624

Falk, H. and Burns, M., "MMS and ASN.1 Encodings Simple Examples and Explanations on How to Crack an MMS PDU", Systems Integration Specialists Company, Inc.(SISCO), 1996 pp.1-31

Flynn, B, "Secure Substation Automation for Operations & Maintenance" GE Energy Whitepaper, 2007. pp. 1-10

Gers, J. M. and Holmes, E. J.  "Protection of Electricity Distribution Networks", 2nd ed. London: Institution of Engineering and Technology 2004.

Halvorsen, P., "Data Communication:Introduction to Berkeley Sockets" INF1060: Introduction to Operating Systems and Data Communication, 2005

Hosmer, C., "Python Forensics A Workbench for Inventing and Sharing Digital Forensic Technology"
Horalek, J. and Sobeslav, V., "Data networking Aspects of Power Substation Automation" Communication and Management in Technological Innovation and Academic Globalization, 2014. pp. 1-7

IEC 60044, “Instrument Transformers Part7: Electronic Voltage transformer” Ed.1., 2002.

IEC 60044, “Instrument Transformers Part8: Electronic Current transformer” Ed.1., 2002.

IEC 61850-1, 2003. Communication networks and systems in substations – Part 1: Introduction and overview. Standard. IEC.

IEC 61850-2, 2003. Communication networks and systems- Part 2: Glossary. Standard. IEC.

IEC 61850-2, 2004. Communication networks and systems in substations – Part 9-2:Specific Communication Service Mapping (SCSM) –Sampled values over ISO/IEC 8802-3. Standard. IEC.

IEC 61850-5, 2003. Communication networks and systems in substations- Part 5: Communication requirements for functions and device models. Standard. IEC.

IEC 61850-6, 2004. Communication networks and systems in substations – Part 6: Configuration description language for communication in electrical substations related to IEDs. Standard. IEC.

IEC 61850-7-1, 2003. Communication networks and systems in substations – Part 7-1: Basic communication structure for substation and feeder equipment – Principles and models. Standard. IEC.

IEC 61850-7-2, 2003. Communication networks and systems in substations – Part 7-2: Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI). Standard. IEC.

IEC 61850-7-4, 2003. Communication networks and systems in substations – Part 7-4: Basic communication structure for substation and feeder equipment – Compatible logical node classes and data classes. Standard. IEC.

IEC 61850-8-1, 2004. Communication networks and systems in substations – Part 8-1: Specific Communication Service Mapping (SCSM) – Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3. Standard. IEC.

IEC 61850-9-1, 2003. Communication networks and systems in substations – Part 9-1: Specific Communication Service Mapping (SCSM) – Sampled values over serial unidirectional multidrop point to point link

IEC 61850-9-2, 2004. Communication networks and systems in substations – Part 9-2: Specific Communication Service Mapping (SCSM) – Sampled values over ISO/IEC 8802-3. Standard. IEC.

IEC 61970: Energy management system application program interface (EMS-API)-part 301: Common information model (CIM) base, Aug. 2011. Edition 3.0

IEC 61968: Application integration at electric utilities-system interface for management- part 1: Interface architecture and general requirements, 2011. Edition 2.0 (Draft)

IEC: Substation Configuration Language” Summary August 2006

IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks, IEEE Std 802.1Q - 2005, 31 19 May 2006

IEEE Standard for Local and Metropolitan Area Networks – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer

Specifications , IEEE Std. 802.3- 2008, 26 Dec. 2008.

IEEE Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks, IEEE Std. 802.1Q-2011, 31 Aug. 2011.

Ingram, D., Schaub, P., and Campbell, D.A., "Multicast Traffic Filtering for Sampled Value Process Bus Networks" Proceedings of the 37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011), IEEE Crown Conference Centre, Melbourne, VIC, pp. 1-6.

Ingram, D., Schaub, P., Campbell, D.A., and Taylor, R.R., "Performance analysis of PTP components for IEC61850 process bus applications." 2012. IEEE Transactions on Industrial Informatics, pp. 1445- 1454.

Janssen, M.C., Apostolov, A., (2008), "IEC 61850 impact on substation design" Transmission and Distribution Conference and Exposition,2008 T& D IEEE/PES pp. 1-7, 2008.

Jorgensen, B., "Guide to Network Programming - Using Internet Sockets" 2012. pp. 1-109

Kanabar, M. (2011), "Investigating Performance and Reliability of Process Bus Networks for Digital Protective Relaying"

Kaliski Jr., B.S., " A layman's guide to a subset of ASN.1, BER, and DER", Online: http://luca.ntop.org/Teaching/Appunti/asn1.html. pp.1-27

Kezunovic, M., Perunicic, B., Levi, S., Soljanin, E., "Digital Signal Processing Algorithms for Power and Line Parameter Measurements with Low Sensitivity to Frequency Change," IEEE Transactions on Power Delivery, Vol. 5, April 1990. pp. 1209-1215

Kezunovic, M., Taylor, H. (2004),"New Solutions for Substation Sensing, Signal Processing and Decision Making" Proceedings of the 37th Hawaii International Conference on System Sciences – 2004. pp. 1-9

Kezunovic, M., Portillo, L., Karady, G., and Kucuksari, S. (2006) "Impact of Optical Instrument Transformer Characteristics on the Performance of Protective Relays and Power Quality Meters" IEEE PES Transmission and Distribution Conference and Exposition Latin America, Venezuela

Kezunovic, M. and Popovic, T. (2005), "Developing Future Substation Automation Strategies: Selecting Apprpriate IEDs and Developing New Applications", International Energy Journal, Special Issue: Cogeneration, Distributed Generation, Distribution System and Power Quality, and DSM and Energy Efficiency, Vol. 6, No. 1, Pt. 3, pp 3-151-3-162, June 2005.

Kim, J., Kim, D., Byun, H., Ham, H Jung, W., Han, D., Park, J., Lee, H., "The definition of basic TEDS of IEEE 1451.4 for sensors for an electronic tongue and the proposal of new template TEDS for electrochemical devices." 2006. pp.1642-1645

Kirkman, R. (2007), "Development in Substation Automation Systems," Proceedings of International Conference on Intelligent Systems Applications to Power Systems, Toki Messe, 5-8 November 2007, pp. 1-6. doi:10.1109/ISAP.2007.4441690

Kriger, C., Retonda, J., Luwaca, E., Behardien, S., "Analysis of GOOSE and Sampled Value Message Structure for Educational Purposes" Protection Automation and Control World conference, Dublin, Ireland, 2011

Konka, W., Authur, C.M., and Atkins, R.C., "Traffic Generator of IEC 61850 sampled values" IEEE. in Smart Grid Modelingand Simulation (SGMS), 2011 IEEE First International Workshop on, 17-17 Oct. 2011, pp. 43–48

Lee, K. & Schneeman, R., “Distributed Measurement and Control Based on the IEEE 1451 Smart Transducer Interface Standards. IEEE Transactions on Instrumentation and Measurement. 2000 pp. 621 - 627.

Lee, K. (2003), “UML Model for the IEEE 1451.1 Standard” IMTC 2003 - Instrumentation and Measurement Technology Conference, 2003. IEEE. pp.1588-1592

Lee, K., Song. E., “Object-Oriented Application Framework for IEEE 1451.1 Standard” Technology Conference, 2004. IMTC 2004 Instrumentation and Measurement
Technology Conference Como, Italy, May 18-20,2004. pp.1182-1187

Leffler, S. J., Robert S. F., William N.J., Lapsley, P., “An Advanced 4.4BSD Interprocess Communication Tutorial” Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

Leupp, P. and Rytoft, C. (2011), “Special Report IEC 61850” ABB document. 2011 pp.1-64

Liang, Y. and Campbell, R. H., “Understanding and Simulating the IEC61850 Standard,” 2008. [Online]. Available: https://www.ideals.illinois.edu/handle/2142/11457.
Mackiewicz, R.E., “Overview of IEC 61850 and Benefits” 2006. IEEE Power Systems Conference and Exposition, Atlanta, 29 October-1 November June 2006, pp.623-630

Mark, J. and Hufnagel, P., “The IEEE 1451.4 Standard for Smart Transducers,” IEEE. 2004. pp.1-13

Minkner, R., “Development Trends in Medium- and High Voltage Technologies for Measuring Systems, Filters and Bushings” 2005. IEEE. pp.1-8

Midance, R. and Iadonis, D., “Ethernet networks redundancy with focus on IEC 61850 applications. In:20th International Conference and Exhibition on Electricity Distribution. Prague: Curran, 2009, pp. 1-4.

Mohagheghi, S., Stoupis, J. & Wang, Z. (2009), “Communication Protocols and Networks for Power Systems- Current Status and Future Trends”, IEEE/PES Power Systems Conference and Exposition (PSCE), pp.1-9
.
Mohagheghi, S., Tournier, J. C., Stoupis, J., Guise, L., Coste, T., Andersen, C. A. & Dall, J. (2011), “Applications of IEC 61850 in distribution automation”, IEEE/PES Power Systems Conference and Exposition (PSCE) , pp.1-9.

Mohagheghi, S., Mousavi, M., Stoupis, J. & Wang, Z. (2009), “Modeling Distribution Automation System Components Using IEC 61850”, IEEE PES General Meeting (PES GM’09), Calgary, Alberta, Canada, 1-6.

Nordell, D.E, “Substation Communication History and Practice” 2008 pp.137-173

Pal, S., Rakshit, A. (2004), “Development of network capable smart transducer interface for traditional sensors and actuators” Online: Science direct - Sensors and Actuators A112 (2004). pp.381-387

PIIRAINEN, J., “Application of horizontal communication in industrial power networks” 2010. pp1-80

“Pan, F., Chen, R.,  Xiao,Y., and Sun, W.  (2012) “Electronic Voltage and Current Transformers Testing Device” www.mdpi.com/journal/sensors 2012 pp.1043-1052

Pofoud, D., “UCA and IEC 61850 for dummies.” distribuTech conference 2002. pp.1-37

Rahmatian, F., Chavez, P.P. “SF6-Free 550 kV Combined optical voltage and current transducer system,” IEEE Transmission and Distribution Conference, September 7-12, 2003, pp. 379-382.

Rahmatian, F., Chavez, P.P., and Jaeger, N.A., “138 kV and 345 kV Wide-Band SF6-Free Optical Voltage Transducers” Proceedings of 2002 IEEE Power Engineering Society Winter Meeting, Jan 28 – Feb 1, 2002.)

Risso, H. and Degioanni,L., “An, An Architecture for High Performance Network Analysis. Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001), Hammamet, Tunisia, July 2001.

Risso, H. and Degioanni,L. “An architecture for high performance network analysis.” 2005 pp.1-9

Sawa, T., Kurosawa, K., Kaminishi, T., Yokota,T., “Development of Optical Instrument Transformers.” IEEE Transactions on Power Delibery, Vol. 5, No. 2, April 1990 pp.884-891

Sanders, C. (2000), “Practical packet analysis.”

Menzies, T., “The Raw And The Uncooked: The Windows XP Raw Sockets Saga, Final Words (Hopefully)” SANS Institute 2002

Schaub, P. and Kenwrick, A., “An IEC 61850 Process Bus Solution for Powerlink's iPASS Substation Refurbishment Project”, Cigré Australia Panel B5, SEAPAC Conference - Powerlink Queensland, March 2009.

Sidhu, T. S., Kanabar, M.G., “Implementation Issues with IEC 61850 Based Substation Automation Systems” Fifteenth National Power Systems Conference (NPSC), IIT Bombay, December 2008

Sidhu, T. S., Gangadharan, P. K., “Control and Automation of Power System Substation using IEC 61850 Communication,” in Proc. IEEE Control and Applications, pp.1331-1336, Aug. 2005.

Sidhu, T. S., Yin, Y., “Modelling and Simulation for Performance Evaluation of IEC 61850 based Substation Communication Systems,” IEEE Trans. On Power Delivery, vol. 22, no. 3, pp. 1482-1489, Jul. 2007.

Shuylupin,C., (2009), “Embedded Linux Software, Device Drivers” Online:http://www.makelinux.net/kernel/diagram

Sonawane, Y., Patil, P., Sonawane, N.D., Sonawane, S. (2007), “Power System Real Time Simulation Testing Using IEC 61850.”

Skendzic, V., Guzmia, A., “Enhancing Power System Automation Through the use of Real-Time Ethernet.” pp.480-495

Skendzic, V., Ender, I., and Zweigle, G., “IEC 61850-9-2 Process Bus and Its Impact on Power System Protection and Control Reliability” Online: Schweitzer Engineering Laboratories, Inc.
Steinhauser, F., “Technology of the future – Sampled Values provide many benefits for the power systems of tomorrow” 2012. pp.1-4

Tiponut, S.V. (2001), “Python Network Programming.” Version 0.00, 16. July 2001

Tholomier, D. and Chatrefou, D. (2008), “IEC 61850 Process Bus – It is Real” PAC World Magazine, Winter 2008, pp.48-53

UCA International Users Group, “Implementation Guidance for Digital Interface to Instrument Transformers Using IEC 61850-9-2" 2004. pp.1-31

Udren, E., Kunsman, S., and Dolezilek, D. “Significant substation communication standardization development” In 2nd Annual Western Power Delivery Automation Conference (WPDAC) Proceedings (April 2000). pp.1-27

Weiss, S., Graeve, P., Andersson, A., “Benefits of conventional instrument transformer data into Smart Grid capable process data utilizing Merging unit” 21st International Conference on Electricity Distribution. 2011. pp.1-4

Wiczer, J. 2001. “Smart Interfaces for Sensors”. Proceeding Sensor Expo 2001, Chicago, IL. pp.10-13

Li Yang. "Impact evaluation of IEC 61850 process bus architecture on numerical protection systems", 2009 International Conference on Sustainable Power Generation and Supply, April 2009

Yang. L., "Performance assessment of a IEC 61850-9-2 based protection scheme for a transmission substation", 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies, December 2011

Zereneh, W. “Packet crafting using Scapy” 2006. pp.1-56 Bachelor of Science thesis, Toronto, 2006

Zheng, P., Pan, S., Chen, C., Ren., Y., “Research on A Networked Optical Fiber Temperature Sensor of Large Power transformer windings” 2006. IEEE International Conference on Industrial Informatics pp.515-518

Online: Omicron - IEC 61850 Testing Tool
https://www.omicron.at/fileadmin/user_upload/pdf/literature/IEC-61850-Testing-Tools-ENU.pdf

Online: www.Python.org

Online: http://www.pearlstreet.inc.com

Online: http://inventors.about.com

Online: http://www.fi.edu/learn/case-files/energy.html

Online: http://www.nettedautomation

Online: www.wireshark.org

Online: www.secdev.org

Online: http://en.wikipedia.org/wiki/Berkerly_Packet_Filter