BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

IED SIMULATOR WITH A SUPPORT OF INDUSTRIAL COMMUNICATION GOOSE

SIMULÁTOR IED S PODPOROU KOMUNIKACE GOOSE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TOMÁŠ LÁDR**
AUTOR PRÁCE

SUPERVISOR **Ing. PETR MATOUŠEK, Ph.D., M.A.**
VEDOUCÍ PRÁCE

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Ládr Tomáš**
Program: Informační technologie
Název: **Simulátor IED s podporou komunikace GOOSE**
IED Simulator with a Support of Industrial Communication GOOSE
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se fungováním komunikace SCADA a průmyslovým protokolem IEC 61850 (GOOSE).
2. Podívejte se na dostupné softwarové klienty/servery pro standard IEC 61850. Popište jejich chování, konfiguraci a možnosti použití.
3. Prostudujte specifikaci vybraného zařízení IED (Intelligent Electronic Device) a navrhněte softwarový simulátor tohoto zařízení.
4. Implementujte simulátor IED včetně komunikace GOOSE.
5. Ověřte chování simulátoru oproti reálnému systému za použití dostupných datasetů.
6. Zhodnoťte své výsledky a použití vytvořeného simulátoru.

Literatura:

- David Hanes, Gonzalo Salqueiro, Patrick Grossetete, Rob Barton, and Jereme Henry. IoT Fundamentals. Networking Technologies, Protocol and Use Cases for the Internet of Things. Cisco Press, 2017.
- Eric D. Knapp, Joel T. Langill: Industrial Network Security. Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems, Syngress, Elsevier Inc, 2015.
- MATOUŠEK Petr. Description of IEC 61850 Communication. FIT-TR-2018-01, Brno: Fakulta informačních technologií VUT v Brně, 2018.
- H. León, C. Montez, M. Stemmer and F. Vasques, "Simulation models for IEC 61850 communication in electrical substations using GOOSE and SMV time-critical messages," *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, Aveiro, 2016, pp. 1-8.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Matoušek Petr, Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 25. října 2018

Abstract

The goal of this bachelors thesis is to create simulator of IED working as publisher of the protocol IEC 61850-GOOSE. For achieving of this goal the library libIEC61850 has been used. During the work on this project an emulator of GOOSE publisher was implemented. There are experiments validating the emulated communication in the report. The merit of this thesis is creation of an open-source emulator for operating system Linux, because other existing solutions are commercial.

Abstrakt

Cílem této bakalářské práce je vytvoření simulátoru IED pracujícího jako publisher protokolu IEC 61850-GOOSE. Pro dosažení tohoto cíle byla použita knihovna libIEC61850. V rámci práce bylo naimplementován emulátor zařízení komunikujícího pomocí protokolu GOOSE. Zpráva obsahuje experimenty validující emulovanou komunikaci. Přínosem této práce je vytvoření open-source tohoto typu emlátoru pro operační systém Linux, protože ostatní existující řešní jsou komerční.

Keywords

SCADA, IEC 61850, IEC 61850-GOOSE, IED, emulation, RTU, OT, industrial networks, IoT,libIEC61850

Klíčová slova

SCADA, IEC 61850, IEC 61850-GOOSE, IED, emulace, RTU, OT, průmyslové sítě, IoT, libIEC61850

Reference

LÁDR, Tomáš. *IED Simulator with a Support of Industrial Communication GOOSE.* Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr Matoušek, Ph.D., M.A.

IED Simulator with a Support of Industrial Communication GOOSE

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Petra Matouška Ph.D., M.A. The supplementary information was provided by Ing. Petr Dittrich Ph.D. and Dr. Stéphane Mocanu. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Tomáš Ládr
July 31, 2019

Acknowledgements

I would like to thank Ing. Petr Matoušek Ph.D. for his academic support, guidance, feedback and advice required for the creation of this thesis. Furthermore I would like to thank Dr. Stéphane Mocanu for provided dataset and my friends at GreyCortex for their advice and suggestions.

Contents

Chapter 1

Introduction

The security of industrial networks became important topic in the recent past. This topic is very wide, but there is a lack of open-source software usable for testing and education purposes. The goal of this thesis is to create a solution usable for simulation or emulation of a device communicating using IEC 61850-GOOSE protocol.

To achieve this goal we have to learn how do networks using IEC 61850 standard work. We have to understand how does the GOOSE protocol work, how does the GOOSE message look like, what data does it contain and how are these data obtained from the hosting physical devices. We have to understand how are GOOSE messages transmitted through the network. These information are contained in the next chapter.

In Chapter 3, we introduce already existing solutions for this problem. We will look for applications prepared created for simulation of devices using IEC 61850 standard. We will also look for software libraries, which would help us wit implementation of our solution.

We will design and implement a solution of this problem. This process is described in Chapter 4. For this we will use the knowledge obtained during the previous work. In the Chapter 5, we will propose and realize experiments to find out if the implemented solution is corresponding to the standard. For these experiments we will have to obtain datasets based on a communication from a real network.

Chapter 2

Introduction to the IEC 61850-GOOSE protocol

The goal of this chapter is to describe the theory that is important for an understanding of this thesis. The chapter is divided into three sections. The first part briefly describes the IEC 61850 [2, 3, 4, 5, 1] standard, because the GOOSE [5] protocol is a part of the standard. The second section describes an information model of IEC 61850 standard and data types used. The model is important because the GOOSE messages contain application data which are extracted from this model. It also describes relations between IEC 61850 and ISO 9506 information models. The last part describes the GOOSE protocol in more details. It also contains a depiction of configuration possibilities of GOOSE exports using Structured Control Language (SCL) configuration language.

2.1 IEC 61850 standard

The purpose of the IEC 61850 standard is to provide means for interoperability between functions performed in substation by physical devices produced by different suppliers [3]. This goal is achieved by defining a common information model used by devices, a mapping of several communication protocols on this model and defining common configuration possibilities for intelligent electronic devices (IEDs).

Currently, the functions defined by the standard are mapped onto three communication protocols [10]:

- Manufacturing Message Specification (MMS) protocol supports two different types of communication [6]:
  - Client/server communication model which is used for calling certain service primitives of objects from the data model described in Section 2.2.
  - Conditioned reporting by a server without any previous client request. For example, these reports can be conditioned by an occurrence of a specific event or by elapsing defined specified time.
- Sampled Measured Values (SMV) protocol periodically transfers time-critical data such as currents and voltages. The protocol uses Ethernet-based multicast [8]. The data to be transmitted are defined in datasets defined on logical node “LLN0” [4].

- Generic Object Oriented Substation Event (GOOSE) protocol exchanges measured data among IEDs in the network. These data can be used for tripping and interlocking circuits [10]. The dataset and export itself is defined for any logical node within a logical device [5].

2.2 IEC 61850 information model

This section contains a description of an information model used by the IEC 61850 standard. The model is important for an understanding of the format of data sent using the GOOSE messages.

The first subsection describes, how data and metadata are stored in an IED and how are they represented in an IEC 61850 network. This description is based on the ISO 9506 [6] information model and shows how the IEC 61850 standard extends this information model [5, 9].

The second subsection explains the naming practice used by the IEC 61850 data model.

The third part of this section shows how the data are represented and depicts the data types used by the IEC 61850 model which are based on Abstract Syntax Notation One (ASN.1) [4].

2.2.1 Information model

The information model of IEC 61850 is based on a model defined by the ISO 9506 standard. This model uses an object-oriented modelling method [9, 12], see in Figure 2.1.

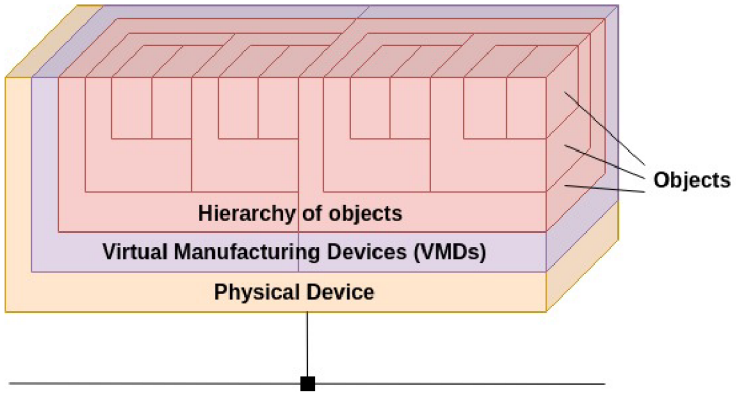

Figure 2.1: ISO 9506 information model

In Figure 2.1 we can see a physical device. The device is supposed to represent one IED in an industrial network. According to the information model of the ISO 9506 standard, this device contains 1 to n Virtual Manufacturing Devices (VMDs). A VMD is a software entity, which provides externally visible behaviour of the IED. A VMD contains an abstract representation of resources and functionality provided by the physical device as well as service primitives (methods) used for processing data of these resources. The resources are usually real physical devices connected to an IED (such as voltmeter, relay, etc.) [6].

Data in a VMD are stored as a hierarchy of objects. Objects are instances of several predefined classes (named variable, domain, ...). All objects can contain data with ASN.1 data types or other objects. They contain service primitives used for work with the data

as well. The number of objects in one layer or maximal level of nesting is not limited by the ISO 9506 standard [12].

The top-level object in the hierarchy can be of a type domain. A domain is to be viewed as a subset of capabilities of its VMD. Each domain is implemented in one VMD only and therefore domains are not distributed. An object which is subordinate of a domain is called domain-specific object [6].

Service primitives are software procedures stored in a VMD used for processing data stored by the VMD. A service primitive might be used for writing or reading data of VMD or its objects, exploration data structure of VMD or simple computation using the data [6].

The information model was used as a template for creation of the information model used by the IEC 61850 standard. The IEC 61850 information model is extended and mapped on the model defined by ISO 9506.

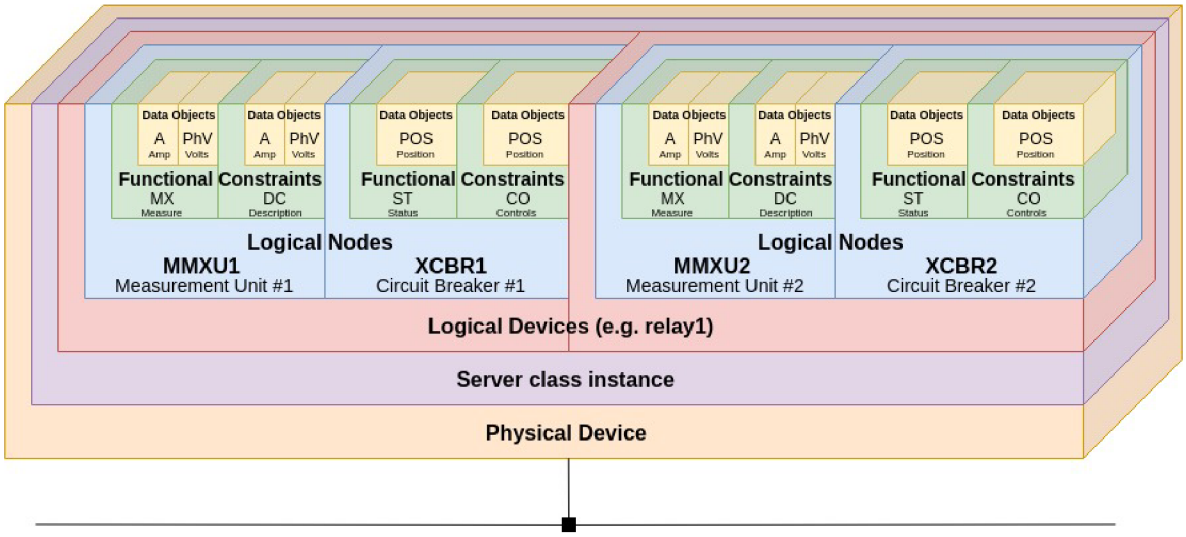

Figure 2.2: IEC 61850 information model [11]

In Figure 2.2 we can see an example of an IEC 61850 information model. We can see that physical device contains Server class instance which is mapped one to one on a VMD. The instance contains more than zero logical devices which are mapped on domains from the ISO 9506 information model [4, 5].

A logical device stands for all information produced and consumed by a group of domain-specific application functions. Each logical device contains three attributes:

- LDName - the name of the instance. This attribute unambiguously identifies its logical node in a subnetwork.
- LDRef - contains a reference of this logical device.
- a list of entities called logical nodes. Each logical device has to contain three or more logical nodes. Two of these mandatory logical nodes are related to common issues of logical device:
  - logical-node-zero (LLN0) which contains common data of logical device
  - logical-node-physical-device (LPHD) which represents common data of physical device hosting this logical device

The rest of logical nodes are used for the description of substation functions [3, 4, 10].

A logical node is defined as the smallest entity within the substation which is able to exchange data. A logical node is a virtual representation of substation equipment. This entity groups together data and service primitives related to one substation function. All logical nodes consist of many attributes, but only those described below are important for our project [4, 10]:

- LNName - unambiguous identifier of the logical node within its logical device.
- LNRef - unique path to the logical node (LDName/LNName).
- DataObject - a list of all data objects contained in the logical node.
- DataSet - a list of lists of data objects. Its elements are used for the generation of reports and some service primitives.
- GOOSEControlBlock - an attribute which holds a definition of GOOSE export.

All data from substation equipment are stored in data objects, which provide means for the definition of typed data in logical node. A data object has to have one of the data types described in section 2.1.3. Data objects are divided into multiple predefined groups according to their specific usage. These groups are called functional constraints and they are defined in IEC 61850 standard [4].

In Table 2.1, we can see a simplified mapping of IEC 61850 standard on ISO 9506 information model.

IEC 61850	ISO 9506
Server Class	Virtual Manufacturing Device
Logical Device class	domain
Logical Node class	named variable
Data Object class	named variable
DataSet class	named variable list

Table 2.1: Simplified mapping of IEC 61850 on ISO 9506 information model [9]

2.2.2 Naming practise

The IEC 61850 is defines a standardized way for addressing of data objects. Figure 2.3 shows the default addressing scheme.

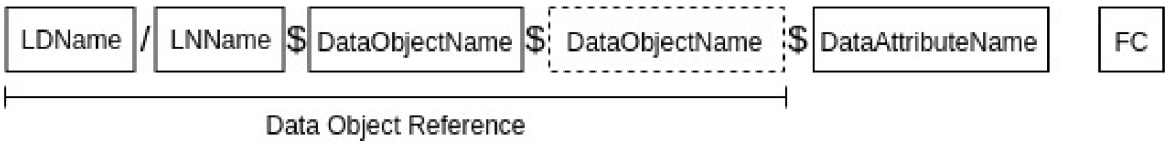

Figure 2.3: Default naming scheme of IEC 61850 [3, 9]

Each object reference is a path in the hierarchy of objects existing in the VMD. The reference consists of ObjectName attributes of instances in the path. These are the logical device name, logical node name, data object name and data attribute name. The functional

constraint (FC) is not shown in the object reference and the information about it mapped differently. The 8-1 of the IEC 61850 standard maps functional constraints between logical node name and data object name [10, 4, 5].

2.2.3 Data types

The IEC 61850-7-2 there are defines several primitive and complex data types for application data of each mapped protocol. We can see an overview of primitive types in Table 2.2. The maximal length of string data types is not defined in the standard. The length of each string attribute is declared by its definition [4].

Name	Value range
BOOLEAN	
INT8	-128 to 127
INT16	-32 768 to 32 767
INT32	-8 388 608 to 8 388 607
INT128	-2**127 to (2**127)-1
INT8U	Unsigned integer, 0 to 255
INT16U	Unsigned integer, 0 to 65 535
INT24U	Unsigned integer, 0 to 16 777 215
INT32U	Unsigned integer, 0 to 4 294 967 295
FLOAT32	Range of values and precision as specified by IEEE 754 single-precision floating-point
FLOAT64	Range of values and precision as specified by IEEE 754 double-precision floating point
ENUMERATED	Ordered set of values, defined where type is used
CODED ENUM	-128 to 127
OCTED STRING	Maximal length shall be defined where type is used
VISIBLE STRING	Maximal length shall be defined where type is used
UNICODED STRING	Maximal length shall be defined where type is used

Table 2.2: Primitive data types

The standard contains complex data types as well: the array and the structure. Both complex types consist of one or more elements which are primitive or complex data type. The only difference between array and structure is that an array contains elements of the same type. The structure can hold elements of different types. We can see an overview of complex data types in Table 2.3. [4]

Name	Description
ARRAY	Collection of elements with the same data type
STRUCTURE	Collection of elements with different data types

Table 2.3: Complex data types

2.3 Description of GOOSE protocol

In this section, we clarify basic facts about GOOSE protocol. We describe containings of packets and application data sent by the protocol as well as the messaging pattern used by the protocol and types of devices which use the protocol.

The usual purpose of GOOSE protocol is the exchange of time-sensitive measured or status information of physical equipment of substation among IEDs in the network. These measurements are used for protection or informative purposes in the substation. The communication is usually transmitted as multicast or broadcast on a local network using IEEE 802.3 Ethernet [10].

The export is defined in a SCL file. Each export uses user-defined dataset. Both dataset and export are declared on any logical node in the network. The number of exports is not limited [5].

2.3.1 Configuration of GOOSE export

All GOOSE exports are configured using SCL files. The System Configuration description Language (SCL) is an XML based language used for configuration of IEDs. GOOSE exports are defined on logical nodes as well as datasets used by GOOSE exports. Listing 2.1 shows a simplified example of a logical node definition. We can see that GSEControl and DataSet elements are defined in the Listing. We can see that the export is defined in GSEControl element and has a dataset to be user-defined as well as its AppID and name. The dataset to be used is defined as DataSet_2 in DataSet element. Also there has to be an export type defined in GSEControl element. In our case, the defined type is "GOOSE" [2].

```
1  <LN0 lnClass="LLN0" inst="" lnType="SIPROTEC5_LNType_LLN0_Application" desc="General
       ">
2    <DataSet name="DataSet_2">
3      <Private type="Siemens-GUID">39a7b23bd383477bb0bb3d14cb48c734</Private>
4      <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="SPC" daName
           ="stVal" fc="ST" />
5      <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="SPC" daName
           ="q" fc="ST" />
6      <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="DPCCB"
           daName="stVal" fc="ST" />
7      <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="DPCCB"
           daName="q" fc="ST" />
8    </DataSet>
9    <GSEControl datSet="DataSet_2" confRev="20001" appID="ASNERIES1_CAL/Application/
         LLN0/Control_DataSet_2" name="Control_DataSet_2" type="GOOSE" />
10 </LN0>
```

Listing 2.1: An example of the SCL configuration file example

2.3.2 GOOSE message format

The GOOSE protocol is related to three layers of the ISO OSI model: the physical layer, the data link layer and the application layer. In Figure 2.4, we can see an example of GOOSE packet encapsulated into IEEE 802.3 Ethernet frame. In Figure, we can see that GOOSE has four fields on the link layer. Each of these fields is two bytes long. The field

called APPID is an attribute that allows identification of application association of received GOOSE message. The Length field contains information about the count of bytes of APDU. There also are two reserved fields, where only one bit is used for identification of simulated communication [10, 4].

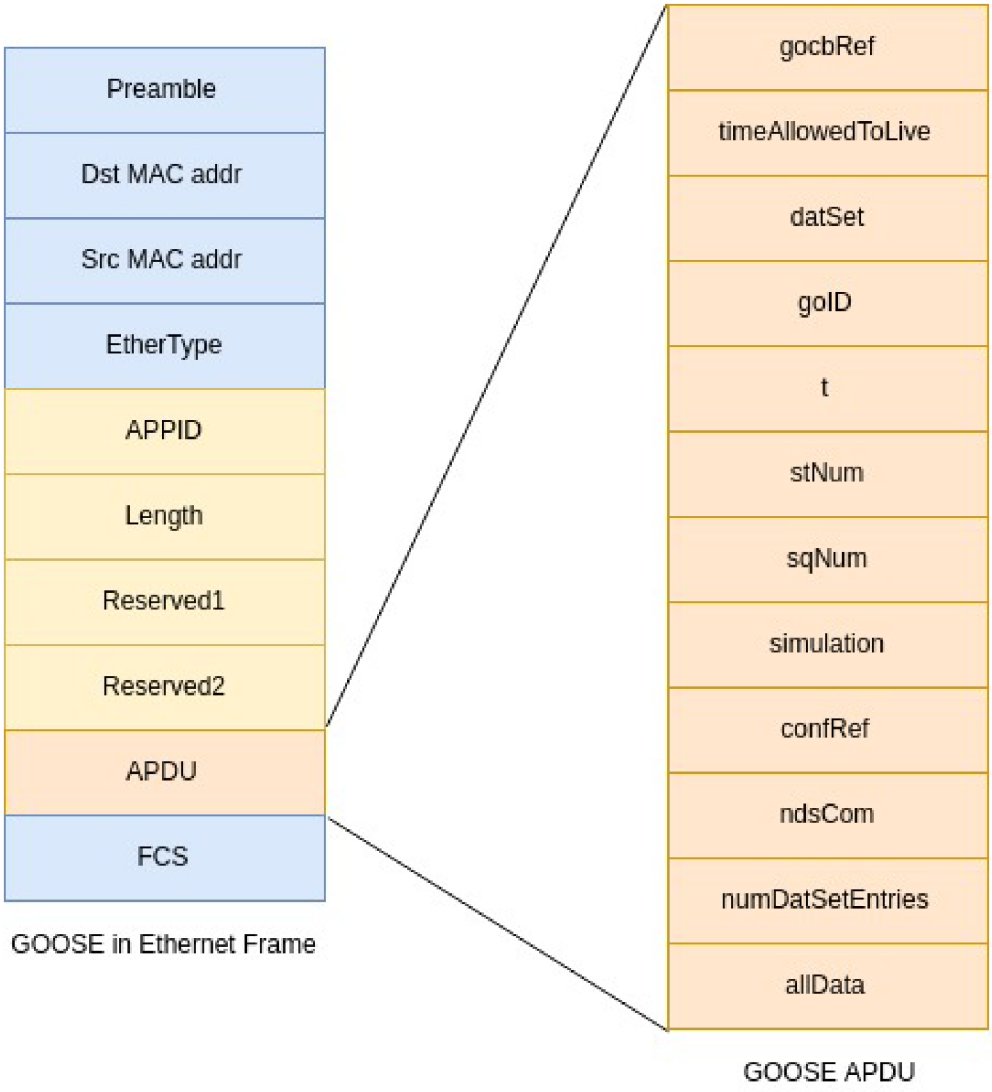

Figure 2.4: GOOSE message format [10]

On application layer the frame has an Application Protocol Data Unit (APDU), which contains GOOSE application data. The APDU and uses BER encoding, which is one of encodings defined by ASN.1. In the APDU, we can find twelve fields:

- GocbRef field contains GOOSE control block reference. GOOSEControlBlock is part of the logical node and contains the configuration of a GOOSE export.
- timeAllowedtoLive field contains "time at which the attribute StNum was incremented. It informs subscribers of how long to wait for the next repetition of the message" [10].
- DatSet keeps a reference on the dataset used for the export.

- GoID field contains a reference on application association of the export.
- StNum and SqNum are used for identification of a specific frame in a flow. The field StNum is incremented whenever the application data contained in AllData holds different values than the previous frame. The field SqNum is incremented with each frame sent. When StNum is incremented SqNum is set to zero.
- Simulation flag is used to indicate frames which are generated during simulation.
- ConfRef contains a count of changes, which have been done on dataset referenced by DatSet.
- NdsCom flag contains TRUE value when GOOSE Control Block requires further configuration. For example, this occurs when DatSet value is set to NULL.
- NumDatSetEntries field contains a count of elements of AllData.

The AllData field contains application data. These data are values of objects, which are part of dataset referenced by DatSet value. Each DatSet is defined on the logical node using an SCL file. The data have to have data types listed in Subsection 2.2.3. Figure 2.5 shows an example of AllData field from a GOOSE message. In the Figure, we can see that AllData section contains four elements. Three of these elements have data type bit-string (octet string) and one is boolean. The messages were decoded using BER encoding. The Basic Encoding Rules (BER) is a way of encoding of ASN.1 structures. [10, 7, 4].

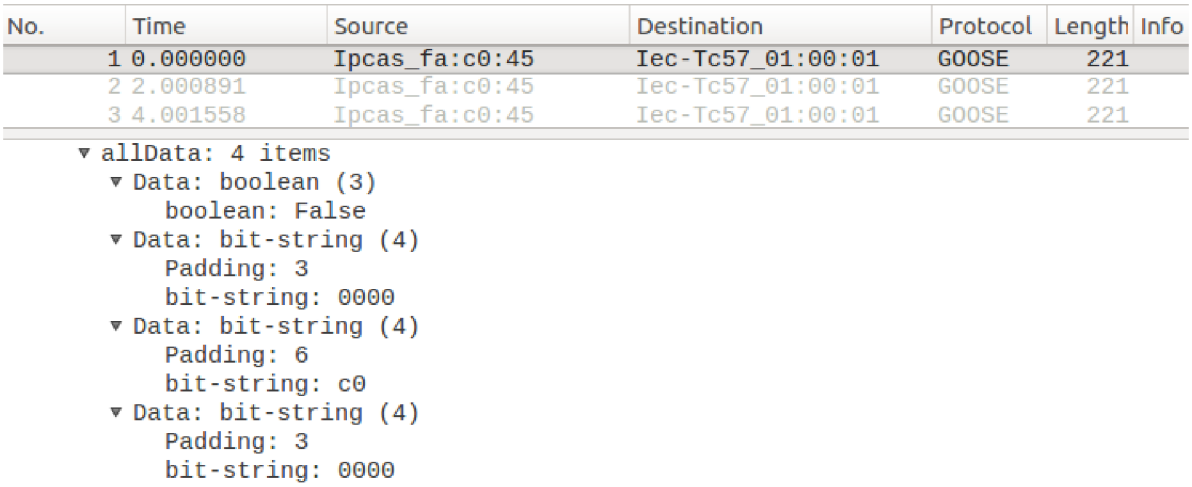

Figure 2.5: AllData example

2.3.3 Communication pattern

GOOSE messages are usually transmitted using multicast or broadcast. For this purpose publisher-subscriber messaging pattern is used. The pattern defines two possible behaviours of communicating devices: the publisher and the subscriber. Each physical device can behave as a publisher or subscriber or both. A publisher is a device which produces messages and sends them via the network. A subscriber is a device which processes the information

obtained from messages produced by a publisher and invokes some inner action based on the information. Figure 2.6 shows an example of the publisher-subscriber communication. We can see that there might be multiple subscribers processing one stream of published messages.

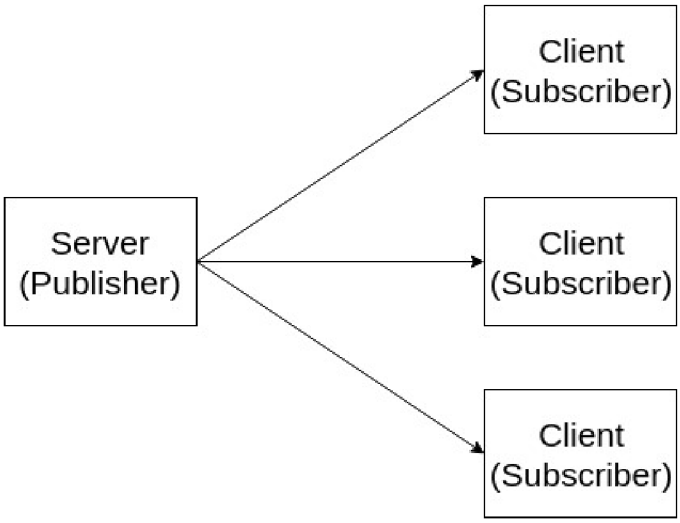

Figure 2.6: Publisher subscriber pattern [4]

2.3.4 Types of devices using GOOSE protocol

According to Subsection 2.3.3 we might split devices into two groups. First group is consisted of devices which are subscribing for GOOSE communication. The second group contains all publishing devices. Each physical device might be in both of these groups. Subscribing IEDs process the data received on application layer, but they do not generate any GOOSE response. We disregard this group of devices in this thesis. The focus of this thesis is the group of publishing devices. These devices generate all the GOOSE traffic, which is visible on the network.

2.4 Summary

In this chapter we depicted basic theoretical knowledge which is necessary for understanding of function of the emulator created for this project. We described the information model used by devices communicating using the IEC 61850-GOOSE protocol as well as its origin. We also described how data are stored and how are they represented in IEDs. We depicted how the data are transmitted using GOOSE, what is the message format and we split all devices using the protocol into two logical groups, where only one can be simulated by the software produced for this thesis.

Chapter 3

Existing tools and libraries for SCADA simulation

The goal of this chapter is to describe software which can be used for emulation of network with IEC 61850 devices. In the first section, we describe software emulators which support the IEC 61850 standard. We depict their capabilities and requirement for their usage. In the second section we briefly introduce existing software libraries which can be used for the implementation of IEC 61850-GOOSE simulator.

3.1 Existing solutions for IED simulation

This section provides information about existing tools for simulation of intelligent devices communicating by protocols of the IEC61850 standard. We look for a tool which is able to generate GOOSE messages on a real network interface. The tool should also be able to generate this communication in real-time. The tool has to be configurable to let the user define the export well enough to make the communication look like communication generated by a real IED. We consider the platform the tool based on as well.

In this section, we can find descriptions of existing solutions. At the end of the section, we can find a brief comparison.

3.1.1 OMNeT++

The OMNeT++[1] is a simulation library and framework used primarily for building discrete event network simulators. Protocol-specific functionality has to be provided by plugins, which are usually developed as independent projects. The framework provides its own Eclipse-based IDE and runtime user interface.

The OMNeT++ library uses an INET framework as a standard protocol model library. The INET contains simulation models for Internet stack and other popular protocols. The INET framework is open-source and heavily relies on the community around it with its maintenance and development. The framework is developed by OpenSim Ltd. company. It is released under Academic Public License for academic and educational usage. Commercial application requires purchase of different license.

The OMNeT++ simulator itself does not generate any output usable for simulation of GOOSE communication in a real network. All of the simulations run in a software

[1] https://omnetpp.org/

environment of the tool. There is an existing module for generation of PCAP traces of the communication. One of the flaws of the packet captures is obsolesce of timestamps in frames during replaying of the communication on the physical network interface. The simulator also does not support GOOSE protocol in default. We found one existing module for OMNET++ which makes the simulation of GOOSE possible. There are two major flaws of this module. The first is dependence on its platform which is Windows 7. The second flaw is that all of the documentation is written in Portuguese.

3.1.2 61850 Test Suite Pro

61850 Test Suite Pro[2] a tool-set for testing and troubleshooting networks which depend on devices which use IEC 61850 standard. This tool-set is developed for the Windows platform. For example, it provides tools for validation of configuration SCL files, description of data flow in the network and interface for display of application data form all the network.

One of these tools is called IED simulator. This tool is able to load the information model from the SCL file. The application is able to generate GOOSE messages and process them. The communication generated by the tool is sent to a physical interface.

This tool-box is commercial, but an evaluation license is provided after registration on the company website. The company Triangle MicroWorks, Inc. is the maintainer of this project.

3.1.3 IEDExplorer

This software[3] is an open-source project focused on testing of IEC 61850 IEDs. Its main purposes are testing and education. This tool can connect to an existing IED over the MMS communication protocol. It can be used for reading and writing values into Data Objects, capture and inspect MMS packets, inspect the SCL files and also explore and send GOOSE messages. It only runs in .NET environment on Windows. This tool has no Linux support. It is released under GNU General Public License.

3.1.4 SmartGridware® IEC 61850 IED Simulator

This application[4] is browser-based IEC 61850 IED simulator. It runs as a web server and provides support for all protocols of IEC 61850 including GOOSE. It is configured using SCL files. The software is able to simulate multiple IEDs at once. There is a possibility to generate exports in SCL or JSON implemented.

This tool is commercial but some evaluation version is available on producers website. It is implemented in JAVA and supports Windows, Linux and MacOS platforms.

3.1.5 SimFlex™ IEC 61850 Client Simulator

SimFlex™ IEC 61850 Client Simulator[5] is a tool used for verification of configuration of IEC 61850 based IEDs. This software comes with an implementation of the test cases defined in IEC 61850-10. It is designed to perform tests of IEC 61850 IEDs but it is also able to publish GOOSE messages. This software is commercial and there is only a free

[2] http://www.trianglemicroworks.com/products/testing-and-configuration-tools/61850-test-suite-pro-pages/overview

[3] https://sourceforge.net/projects/iedexplorer/

[4] http://www.smartgridware.com/java_iec61850_ied_simulator.html

[5] http://www.gridclone.com/p/simflextm-iec-61850-client-simulator

trial license available. It is developed by GridClone B.V. company and it is supported on Windows platform.

3.1.6 IEC 61850 TesT Software

This software[6] is used for simulation, monitoring and testing IEDs on an existing substation network. It is able to generate and process SV and GOOSE messages for tests of the IEDs functionality. The tool allows scripting to support more complex testing scenarios. The output of this software is generated in COMTRADE format. IEC 61850 TesT Software is configurable using a SCL file. It is also able to generate SCL files that comply with IEC 61850 parts 6 and 7. This software is commercial as well and it is developed by Doble Engineering Company. It is supported on Windows platform only. No evaluation licence is offered by the company.

3.1.7 Summary

There are multiple exiting software solutions capable of simulation of devices generating IEC61850-GOOSE messages described. All of the solutions described in this chapter have this support implemented. All of the discovered solutions are supported only on Windows with one exception. All of these applications are able to generate real-time GOOSE messages except OMNET++ which is able to generate only packet captures. The OMNET++ is an only open-source solution found. All of these tools are configurable using SCL or graphical user interface.

3.2 SCADA communication libraries

The purpose of this section is to describe existing libraries usable for simulation of devices communicating by protocols of IEC 61850 standard. The comparison of libraries is done by considering their implementation language, support of IEC 61850-GOOSE protocol and the licences the libraries are released under.

3.2.1 openIEC61850

OpenIEC61850[7] is an open-source library. It is written in Java language and is a part of OpenMUC framework which implements multiple communication standards (IEC 60870-5-104, ASN.1). The library is licensed under the Apache 2.0 license and is maintained by OpenMUC department of Fraunhofer Institute for Solar Energy Systems (ISE) in Freiburg, Germany. Because of programming language chosen by the authors, the library can be used for fast development and is easily deployable on any platform. The main drawback of this library is that it contains IEC61850-MMS client/server only. This means the library is not usable for this thesis.

3.2.2 libIEC61850

This library[8] provides an implementation of IEC61850-MMS client/server, IEC61850-GOOSE publisher/subscriber and IEC61850-SV publisher/subscriber. It is designed according to

[6] https://www.doble.com/product/software-61850-test/
[7] https://www.openmuc.org/iec-61850/
[8] https://libiec61850.com/libiec61850/

the second edition of the standard. The library is written in C, but it contains .NET and Python wrappers allowing the library to be used in high-level languages. The last release contains JAVA API. It is released under GPLv3 license. The library is maintained by the company MZ Automation GmbH.

3.2.3 rapid61850

The goal of this software[9] is to automatically generate C/C++ code for sending and receiving IEC61850-GOOSE and IEC61850-SV communication. Passed SCL file is used for the generation. The library also validates the SCL file. There is also the possibility to use SWIG for creation of high-level languages wrappers. Author of this library is Steven Blair and is published under GPLv2 license. The Eclipse IDE with the Eclipse Modeling Framework is the mandatory environment for development and running this library. The API of this library is complicated and hardly understandable.

3.2.4 IEC-61850 Library

IEC-61850 Library[10] is a commercial library and it is maintained by JPEmbedded company. The company offers a free evaluation version of the software. For licensing purposes, the royalty-free licensing model is used. This software is implemented in C++ and uses the object-oriented paradigm. The library supports all protocols defined in IEC61850 standard.

3.2.5 IEC 61850 Source Code Library

This commercial library[11] is produced by Triangle MicroWorks, Inc. The producer offers a free evaluation licence for this product. The Library supports all protocols of IEC 61850 including GOOSE. All components included are implemented in C/C++ or .NET.

3.2.6 Comparison

We were able to find five existing libraries which are usable for simulation of IEC 61850 communication. In Table 3.1 we can see the comparison between all of these libraries.

Name	License	GOOSE support	Language
openIEC61850	Apache 2.0	no	JAVA
libIEC61850	GPLv3	yes	C
rapid61850	GPLv2	yes	C/C++
IEC-61850 Library	commercial	yes	C/C++
IEC 61850 Source Code Library	commercial	yes	C/C++ or .NET

Table 3.1: Libraries comparison

The crucial trait for consideration is actual support of IEC 61850-GOOSE protocol. All libraries without this support are not usable for this thesis. All commercial libraries are not suitable as well. There are two libraries, which meet these conditions. Both of these are implemented in C/C++, so there is no difference for us. The difference is in the

[9] https://github.com/stevenblair/rapid61850
[10] http://www.jpembedded.eu/en/tab/iec-61850-library/
[11] http://www.trianglemicroworks.com/products/source-code-libraries/iec-61850-scl-pages

purpose and implementation of libraries. Rapid61850 is supposed to automatically generate C/C++ code, which is supposed to generate and read GOOSE communication. This code is generated from SCL files. This is not convenient for the purpose of this thesis. On the other hand, libIEC61850 is tool prepared for the implementation of GOOSE publisher or subscriber. That is the reason why find the library more suitable. For the implementation part of this thesis, we chose the libIEC61850 library.

3.3 Summary

In this chapter, we described existing software whose purpose is the simulation of devices communicating using IEC 61850 standard. In the first part of the chapter, we introduced already existing software capable of this simulation. We described the requirements and drawbacks of these applications. In the second part, we depicted existing libraries usable for implementation of IEC 61850-GOOSE simulator. We described their capabilities, compared them and explained the choice of the library for this thesis.

Chapter 4

Design and implementation of the emulator

This chapter contains two parts. The first part contains a description of the design of the emulator. There is written how the emulator should work and what will it emulate. In the section, there also is a description of blocks which compose the emulator. In the second part, we describe the implementation of the emulator. We clarify the choice of programming languages used and describe how did we proceed during the implementation. We describe how the emulator can be configured and how the configuration can be created from an SCL file. We also describe the fixes we had to make to the library used for the implementation.

4.1 Design

This part is about the design of the emulating system. This system has to be able to generate GOOSE communication just like a GOOSE publisher would. The input of the emulator should be a capture file which contains GOOSE communication. The emulator should also be configurable with a different type of input, which would let the user start an emulation without need for any existing capture. The emulation should generate some kind of output, which will enable the user to check the course of emulation.

4.1.1 Block scheme of the emulator

The system is supposed to consist of multiple separate blocks. The purpose of each of these blocks is described in this part. In Figure 4.1 we can see the design of the emulator. There are three blocks in the scheme. Each of these blocks has its purpose in the final emulator. We can see that there are two types of arrows going in and out of the blocks. The blue arrows denote inputs and red arrows denote outputs. The square labelled as “NIC” is not meant to be a block.

The first block is the control block. The first purpose of this block is to be the interface between the rest of the blocks and the user. The block has two types of input. One of these is a capture file. This file is used as a template for the emulation. The second possible input of this block is a file containing a prepared transcript of the emulation. The output of this block is a capture file containing the output of the emulation block.

The second block is the configuration block. The objective of this block is to automatically generate a configuration. The input of the block is a capture file which contains the

communication to be emulated. An automatically generated configuration file is the output of this block. The block is controlled by the control block.

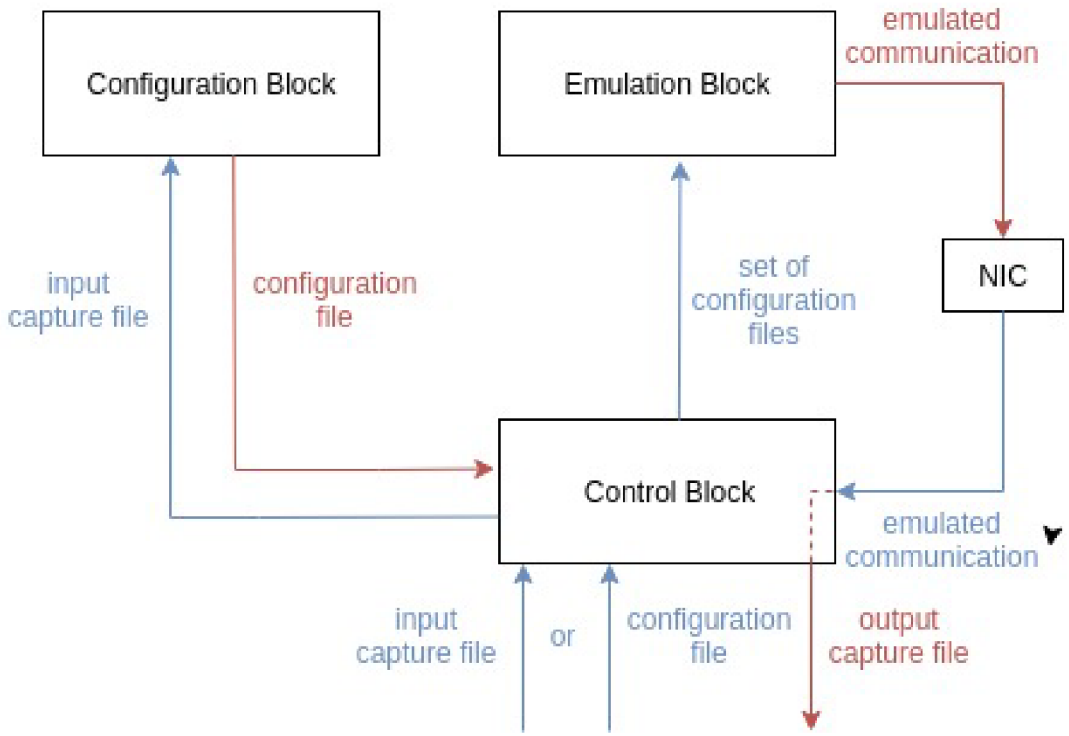

Figure 4.1: Emulator design diagram

4.1.2 Configuration file format

In this work we sometimes substitute a configuration file for a "transcript of emulation". The emulator needs to get a transcript of emulation, which enables to run emulation according to user-defined parameters. For this purpose, we chose to use an XML format for these configuration files. The reasons for this choice are that it is simple to manually create or edit XML files and there is a lot of existing tools which enable to easily work with this format of files.

In Listing 4.1 we can see a simple example of the configuration file. Each configuration file can contain only one transcript of emulation. In Listing we can see that *configuration* element is the root element of the file. All other elements have to be nested in this element. Next we can see an element named *headersValues*. This element contains information about one sequence of GOOSE messages. In a configuration file generated by the configuration block, this element would contain all packets which have the same values in the GOOSE header and have the same application data values. When any change would be found a new element *headersValues* would appear in the configuration file. There are four elements nested in the *headersValues* element. The *timeCount* element holds the count of packets sent in this series. The elements *timeStart* and *timeEnd* hold the information about the start and end of the stream of packets in seconds in relation to start of the whole emulation.

The element *gooseValues* holds information about the GOOSE header and the application data that are supposed to be in the messages. The elements contained in the *gooseValues* element follow:

- *src* - Holds source MAC address. The data-type of the value is a string.
- *dst* - Holds source MAC address. The data-type of the value is a string.
- *goID* - Keeps the reference on the application association of the export. The data-type of the value is a string.
- *goCb* - Keeps the reference on the GOOSEControlBlock of this export. The data-type of the value is a string.
- *datSet* - Holds the reference on the DataSet used by this export. The data-type of the value is a string.
- *confRef* - Keeps the count of changes, that has been done to the exported DataSet. The data-type of the value is an integer.
- *needsComm* - When the export needs to be reconfigured, the non-zero value is contained by this field. Else zero is contained.
- *allowedTTL* - Informs subscribers when should the next message arrive. The data-type of the value is an integer.
- *appID* - Holds information which enables a subscriber to identify application association of received message. The value is string, which represents a number in hexadecimal.
- *vlanID* - Contains ID of VLAN. If no VLAN is used the element contains -1. The data-type of the value is an integer.
- *vlanPrio* - Contains priority of VLAN. If no VLAN is used the element contains -1. The data-type of the value is an integer.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuraton>
    <headersValues>
        <gooseValues>
            <src>00:00:00:00:00:01</src>
            <dst>00:00:00:00:00:02</dst>
            <goID>LLN0/Control_DataSet_2</goID>
            <goCb>LLN0$GO$Control_DataSet_2</goCb>
            <datSet>LLN0$DataSet_2</datSet>
            <confRev>1</confRev>
            <needsComm>0</needsComm>
            <allowedTTL>3000</allowedTTL>
            <dataValues>
                <goose.boolean>0</goose.boolean>
            </dataValues>
            <appID>0x01</appID>
            <vlanID>4</vlanID>
            <vlanPrio>0</vlanPrio>
        </gooseValues>
        <timeCount>10</timeCount>
        <timeStart>0</timeStart>
        <timeEnd>10</timeEnd>
    </headersValues>
</configuraton>
```

Listing 4.1: XML configuration file example

The last type of element contained by *headersValues* is *gooseValues*. This element can appear multiple times in *headersValues* and each of these elements contains one element of the application data. These elements contain one element each. This element differs because of type of value contained in it.

4.2 Implementation

In this section, we describe how the blocks from Figure 4.1 are implemented. We depict how the emulation is set up. We also depict how the important parts of the emulator are programmed. We explain how the emulation transcript generated automatically and how we can generate it manually from an SCL file. There is a description of enhancements that we did to libIEC61850 library to make it work satisfyingly.

4.2.1 Control block

The purpose of this block is to check the format of input and make proper steps to get valid configuration files from this input. There are two possible types of input. The first of these is a directory containing prepared configuration files. Each of these files is used as a transcript of emulation for one stream of packets. With one stream of packets, we mean communication between one source and one destination device.

The second type of input is the capture file. When a file in PCAP format is passed, this block passes the file to configuration block. The configuration block prepares transcripts of emulation into specified directory.

When configuration files are ready, the control block passes the directory containing the configuration files to the emulation block and starts the emulation. During the emulation,

the control block captures GOOSE packets on the specified network interface. The emulation happens on the same interface. The result of this is output capture file, which contains the GOOSE communication generated during the emulation.

This block is implemented in script *Control.rb*. This script is programmed in Ruby language. We chose this language because its syntax is very simple. In the language, it is easy to call system utilities as well. Ruby also offers a wide variety of modules, which can be used for work with serialization formats. It is simple and reliable to use standard ways for installing these modules offered by Ruby.

4.2.2 Configuration block

This block is responsible for the generation of configuration files from the passed capture file. The block uses *tshark* utility to read all GOOSE packets from the capture file. After that *tshark* generates meta-data about all of GOOSE packets in JSON format. This meta-data is processed for each packet. The statistics for each of packet flows is taken. After processing all of the meta-data for each packet flow the new configuration file is generated. These files are generated into a directory. The path to this directory is defined as a parameter.

This block is also programmed in the Ruby language. The reasons for picking this language are the same as for the control block from section 4.2.1.

4.2.3 Emulation block

This block is responsible for the emulation of GOOSE communication. The input of this block is a path to a directory, which contains a set of configuration files. These files have been in a format defined by the DTD description from Annex A. The data from these files are parsed and stored into an internal data structure. For each of the stream of packets, a separate thread is created. These threads generate GOOSE packets according to metadata read from the configuration. Every configuration file has its own thread.

```
1  void* replayCommunication(publisherParameters* params) {
2      usleep(params->startOfThisFlow - params->absoluteStartOfEmulation);
3      for (int j = 0; j < params->gooseValueListCount; j++) {
4          timeParameters* tParams = params->timeParams[j];
5          int pktDelay = (tParams->endTime - tParams->startTime) / tParams->packetCount;
6          for (int i = 0; i < tParams->packetCount; i++) {
7              usleep(pktDelay);
8              if (GoosePublisher_publish(params->pub, params->gooseValueList[j]) == -1) {
9                  \\ prints error message
10             }
11         }
12     }
13     return NULL;
14 }
```

Listing 4.2: Pseudocode of transmitting part of the emulator

In Listing 4.2 we can see simplified pseudocode of part of the emulator. This part is used for transmission of GOOSE packets. This pseudocode runs in its own thread. This function gets a parameter *publisherParameters* params*, which is a pointer on a structure of type *publisherParameters*. This structure contains all data necessary for running the emulation of GOOSE publisher. In the second line, we can see how the synchronization of

starts of different packet flows is done. The thread sleeps for the difference between the start of the first packet flow and the packet flow which is supposed to be emulated by this thread.

There is two for cycles in Listing. The first of them iterates through all of the sets of packets which are supposed to be transmitted. By the set of packets, we mean a series of packets, which have the same values in headers and the same application data. In the inner representation of the packet series is always a count of packets and their values. For each series of packets is counted a delay between packets. It can be seen on the fifth line.

The nested cycle iterates through the count of packets in one packet stream. For each of these packets, the thread sleeps for the previously counted delay. After the sleep, the function for publishing of GOOSE messages is called. The parameters for the packet to be sent are the same for each message in the stream. If the GOOSE message can not be sent, the error message is printed and the transmitting process continues.

There are multiple flaws to the implementation. The first of them is that the timing of transmitted packets might not be the same if packet capture input is used. The messages from one flow are sent with the same delay. If GOOSE messages in the packet capture were not spread linearly the timing would be different. The second flaw is that the emulator does not support complex data types in application data of messages.

We can find the implementation of this block is in the *goose_publisher_emulator.c* file. This code is written in the C language. We chose this language because it is the programming language used for the implementation of libIEC61850 library. This is the library which was used for the emulation process.

4.2.4 Enhancements of libIEC61850

During the implementation, we found multiple imperfections in the libIEC61850 library. We had to fix these flaws in order to be able to make the emulation work as it should. The first flaw was that the library did not let the user set the source MAC address of GOOSE messages. A fixed MAC address was filled into messages. We fixed this by adding a parameter, which keeps the MAC address to the necessary functions.

The second flaw was that the library statically added IEEE 802.1Q tag. Even messages without specification of this had VLAN priority and VLAN ID fields in Ethernet frame. We fixed this with adding a condition which adds these field into Ethernet frame when requested.

4.2.5 Generation of configuration from SCL file

The configuration file can be created manually. We can partially create configuration file from an SCL file. The first missing part of configuration is the values of application data to be sent. To fill this information user should know datatypes defined in IEC 61850-7-2. The second part is the information about the number of packets and the time information about the emulation. The third part is the source MAC address of the communication.

In Annex B we can see a very simplified SCL file. This SCL file contains a definition of a GOOSE export. We created a transcript of emulation from this SCL file. The transcript can be found in Annex C. The data for the configuration file were obtained by the following steps:

- *src* - contains source MAC address. This depends on Network Interface Card of source device in the network. User can choose any MAC address and the emulation will work the same.
- *dst* - contains destination MAC address. This address can be found in GSE block from Listing B.1.
- *goID* + *confRef* - can be found in the GSEControl block defining the GOOSE export. The element *goID* is supposed to contain the value of appID. The element *confRef* is supposed to contain the value of confRev. These elements are in the Listing B.1.
- *goCb* - contains the reference on the GOOSEControlBlock in the IEC 61850 information model. This reference is consisted of:
  - name attribute of IED element
  - inst attribute of LDevice element
  - lnClass attribute of LN0 element
  - name attribute of GSEControl element

  The first three are separated with a slash sign. The last two are separated with a dollar sign. All of these elements are in Listing B.1.
- *datSet* - contains the reference on the dataset in the IEC 61850 information model. This reference is consisted of:
  - name attribute of IED element
  - inst attribute of LDevice element
  - lnClass attribute of LN0 element
  - name attribute of DataSet element

  The first three are separated with a slash sign. The last two are separated with a dollar sign. All of these elements are in Listing B.1.
- *needsComm* - is usually generated by the publishing IED. The meaning of the element is to inform if the export needs to be reconfigured. Can be user-defined depending on users goal with the emulation.
- *allowedTTL* - is filled by the publishing IED as well. The user has to choose the value.
- *dataValues* - depends on dataset which is used for the GOOSE export. We can see its name in GSEControl element in datSet attribute in Listing 4.3. This dataset is defined in the same logical node in DataSet element. In the DataSet element, we can see there are four elements defined.

```
1 <LN0 lnClass="LLN0" inst="" lnType="SIPROTEC5_LNType_LLN0_Application" desc="
      General">
2   <DataSet name="DataSet_2">
3     <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="SPC"
          daName="stVal" fc="ST" />
4     <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="SPC"
          daName="q" fc="ST" />
5     <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="DPCCB
          " daName="stVal" fc="ST" />
6     <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="DPCCB
          " daName="q" fc="ST" />
7   </DataSet>
8   <GSEControl datSet="DataSet_2" confRev="20001" appID="ASNERIES1_CAL/
        Application/LLN0/Control_DataSet_2" name="Control_DataSet_2" type="
        GOOSE" />
9 </LN0>
```

Listing 4.3: SCL definiton of dataset and goose export

Each of these elements contains attribute lnClass. Using this attribute we can identify the element LNodeType, which belongs to this class of logical node. We can see LNodeType element in Listing 4.4. In LNodeType element, we can see that the logical node of this class contains two data objects. They are called *DPCCB* and *SPC*. These DO elements have an attribute called type. The type *DPCCB* is "DPC". In Listing 4.4 we can see elements DOType. The value of id attribute of DOType element and the value of type attribute of DO element have to match. If we take a look at DOType with id 'DPC', we can see data attributes of this data object.

These data attributes are important for the resolution of datatypes of *dataValues* in emulated GOOSE export. If we take a look at daName attributes of DataSet element of Listing 4.3 and resolve which DOType belongs to which element of the dataset we can find a proper order for datatypes in the dataset in the transcript of emulation. To determine the actual ASN.1 datatype, you have to search for the value of the bType attribute in the IEC 61850-6.

```
1 <LNodeType id="SIPROTEC5_LNType_USER_Universal" lnClass="USER">
2   <DO name="DPCCB" type="DPC" />
3   <DO name="SPC" type="SPC" />
4 </LNodeType>
5
6 <DOType id="SPC"">
7   <DA dchg="true" fc="ST" name="stVal" bType="BOOLEAN" />
8   <DA qchg="true" fc="ST" name="q" bType="Quality" />
9 </DOType>
10 <DOType id="DPC">
11   <DA dchg="true" fc="ST" name="stVal" bType="Dbpos" />
12   <DA qchg="true" fc="ST" name="q" bType="Quality" />
13 </DOType>
```

Listing 4.4: SCL definition of logical node related data objects

- *appID* + *vlanID* + *vlanPrio* - can be found in GSE block from Listing B.1.
- *timeCount* - was chosen by the user. Contains amount of packets to be sent. All of these packets would have values taken from *gooseValues*.

- *timeStart* + *timeEnd* - are chosen by the user as well, but user has to have in mind time restrictions from the GSE block. This block can be seen in Listing B.1 starting on the fifth line. The elements *MinTime* and *MaxTime* define minimal and maximal delay between packets. The values of *timeStart* and *timeEnd* have to be adjusted according to *MinTime* and *MaxTime*.

4.2.6 Summary

In this chapter, we described the design and the implementation of the emulator. In the description of the design, there is a block scheme of the emulator with an explanation of planned inputs, outputs and roles of each block. There is also the depiction of XML format used for the transcripts of emulation.

In the second part of this chapter, we described the implementation. In this part, we can find a depiction of possible usages, inputs and outputs of blocks the emulator. We also described software tools used for the implementation and we clarified why we chose them. We described changes we had to do to the library used for the implementation of the emulator. We also outlined the process which has to be used to create a transcript of emulation from an SCL file.

Chapter 5

Comparison of communication of emulator and real devices

In this chapter we describe the experiments done for validation of the emulator. First we define a methodology we use evaluation of experiment results. Next we do multiple experiments. In the first experiment we compare the result of emulation with a real dataset. In the second experiment we generate artificial transcripts of emulation, and compare the results with expected result.

One part of work on this project was an internship in GIPSA-lab[1], Grenoble, France. During the internship we obtained the dataset used for this project.

5.1 Methodology and requirements for emulator

During the experiments we record a packet capture of the communication generated by the emulator. We use this capture for comparison between the dataset used and the result of emulation. We check validity of headers and application data of the capture. We compare the time course of the capture in relation to the dataset. We visualize both, the dataset and the emulation capture, using the Wireshark[2]. We also use Wireshark for the analysis of packet captures.

The emulator is implemented to behave like an IED with a new GOOSE export. This means that *stNum* and *sqNum* behave according to the IEC 61850 standard. The *simulation* flag is set to "TRUE" because the simulation is not generated by a real device. This is the purpose of the flag. The timestamp (t) is set to the time of publishing of the message. The field *numDatSetEntrie* is automatically generated according to the count of elements in the *allData* list. The rest of GOOSE application data is set using the configuration file.

5.2 Comparison of emulated communication and replayed packet capture

In this experiment we used two datasets obtained in GIPSA-lab. Both datasets are stored as packet captures. We used these packet captures as inputs for the simulator. For each dataset a transcript of emulation was created.

[1] http://www.gipsa-lab.fr/
[2] https://www.wireshark.org/

5.2.1 Dataset with one GOOSE export

The dataset is contained in the file *goose_test_1.pcapng*. In the Table 5.1 there is information about length and timing of GOOSE messages. These messages have approximately two seconds delay between each other. In Figure 5.1 we can see the application data of the first dataset. There is ten GOOSE messages. All of the messages contain the same application data.

No.	Timestamp[s]	Length
0	0.000000	221
1	2.000891	221
2	4.001558	221
3	6.002258	221
4	8.002674	221
5	10.003350	221
6	12.003227	221
7	14.002880	221
8	16.003414	221
9	18.003278	221

Table 5.1: Time information of *goose_test_1.pcapng*

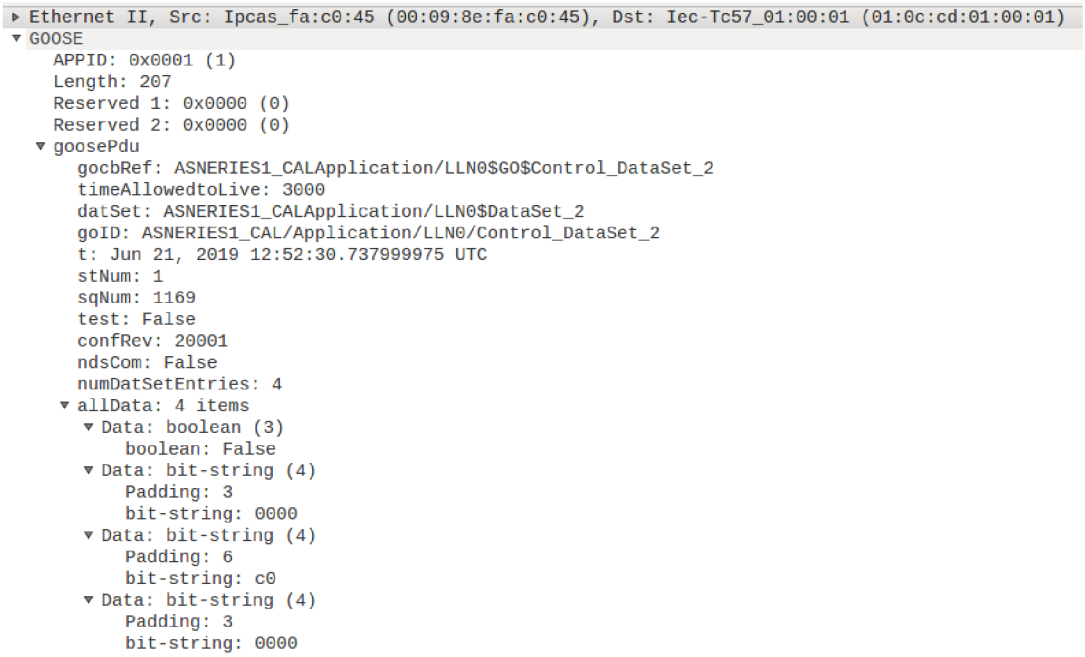

Figure 5.1: Containings of *goose_test_1.pcapng*

The result of emulation is visible in Table 5.2 and Figure 5.2. We can see the timing differs in tenths of milliseconds. For each message sent the time delay adds up. The lengths of messages are different because of the difference in value of *stNum* element. The dataset was not recorded since the start of communication. The emulator behaves like a device

which started the communication with the first emulated packet. The first packet of communication starts with *stNum* set to zero. The rest of application data is the same.

No.	Timestamp[s]	Length
0	0.000000	218
1	2.000585	218
2	4.001260	218
3	6.001847	218
4	8.002385	218
5	10.003009	218
6	12.003653	218
7	14.004324	218
8	16.005061	218
9	18.005763	218

Table 5.2: Timing of the emulation of the first dataset

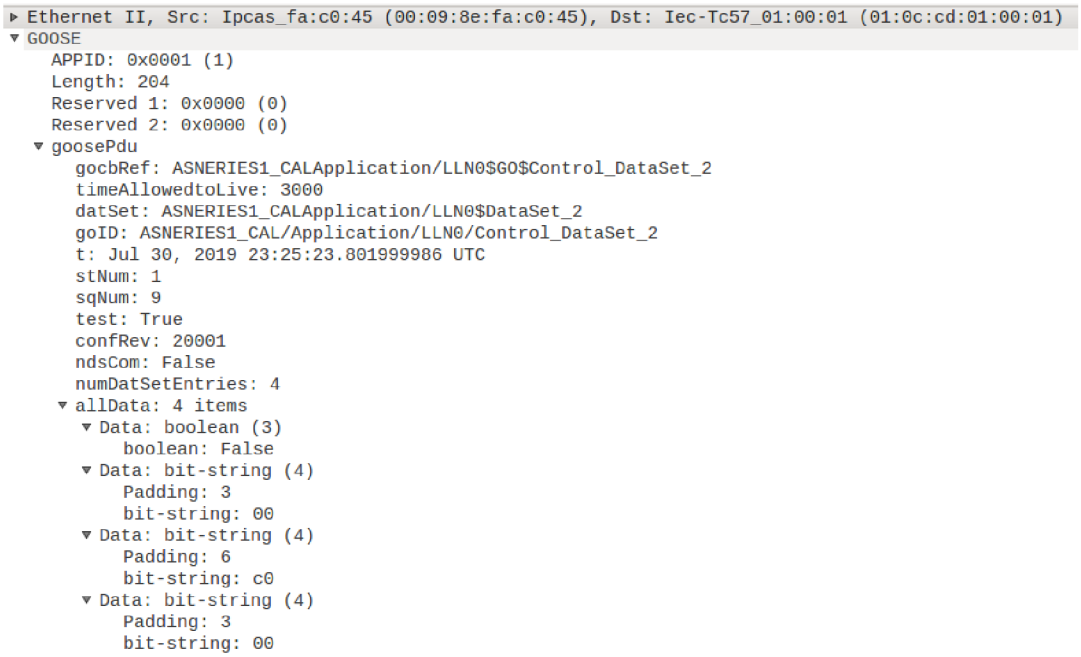

Figure 5.2: Application data from the emulation of the first dataset

5.2.2 Dataset with multiple GOOSE exports

The data set for this experiment is *goose_test_2.pcapng*. We can see the information about this dataset in Table 5.3, Figure 5.3 and Figure 5.4. In this dataset there are two GOOSE publishers defined. The first flow contains thirteen packets. All of them have the same application data, see Figure 5.3. They have two seconds delay between each other. There is one exception. Packets seven and eight have four seconds delay.

The second flow contains three packets. The flow starts approximately eight seconds after the first one. The packets of this flow have approximately ten seconds delay. The

packets contain also the same application data, see Figure 5.4. The application data between flows are different.

No.	Timestamp[s]	Length
0	0.000000	221
1	2.001065	221
2	4.001909	221
3	6.002377	221
4	8.002208	221
5	10.002700	221
6	12.002397	221
7	14.002397	221
8	18.002581	221
9	20.002511	221
10	22.003082	221
11	24.002913	221
12	26.003427	221

No.	Timestamp[s]	Length
0	8.602129	162
1	18.502551	162
2	28.403189	162

Table 5.3: Time information of *goose_test_2.pcapng*

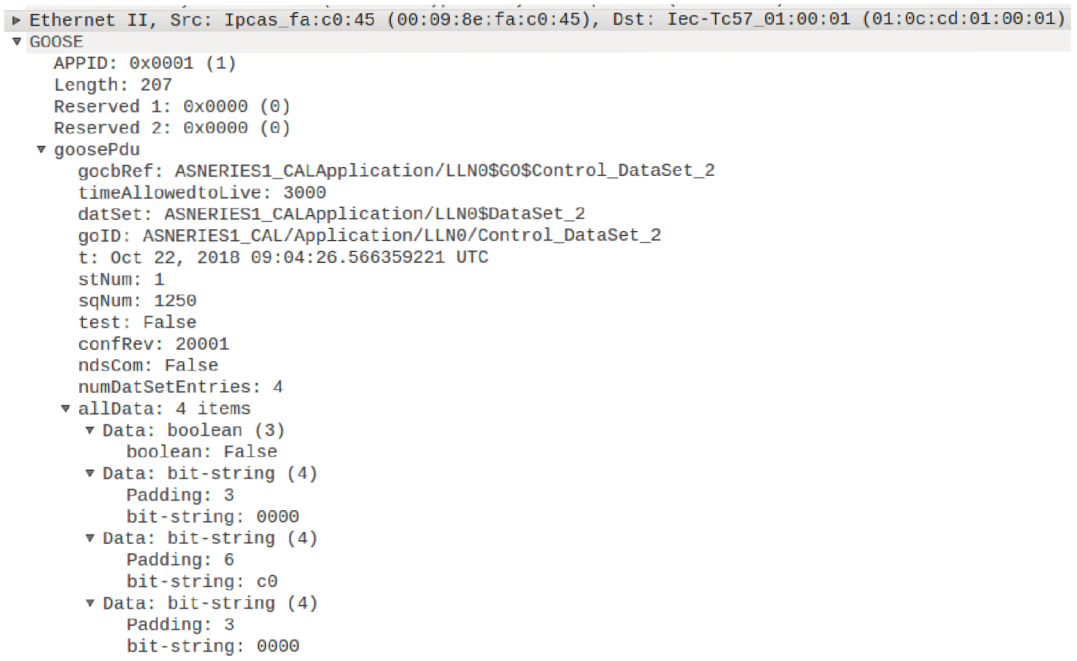

Figure 5.3: Containings of the first flow of *goose_test_2.pcapng*

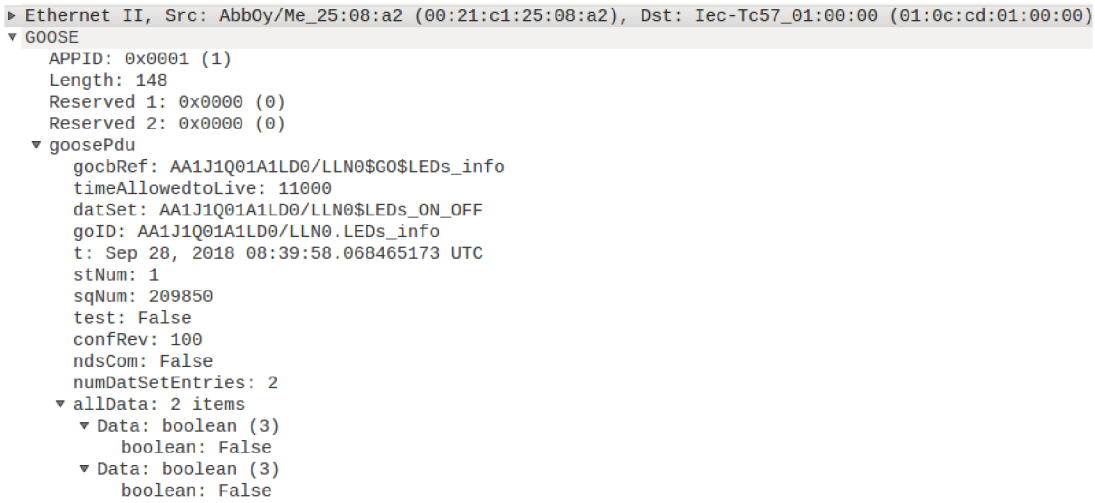

Figure 5.4: Containings of the second flow of *goose_test_2.pcapng*

We automatically created the configuration files from the dataset. We can see the results of emulation in Table 5.4, Figure 5.5 and Figure 5.6. There is a difference in timing of first flow between the dataset and the emulation. In the emulated communication the doubled delay disappeared. The automatically created configuration contains only metadata about the course of communication from the dataset. The emulator sends GOOSE messages with periodical delays. The timing of the second flow is the same as the timing in the dataset.

We can also notice the difference in lengths of messages. This difference is caused by the generation of new*sqNum* from zero again.

No.	Timestamp[s]	Length
0	0.000000	218
1	2.167143	218
2	4.334350	218
3	6.501651	218
4	8.668883	218
5	10.836058	218
6	13.003317	218
7	15.170571	218
8	17.337889	218
9	19.505177	218
10	21.672391	218
11	23.839641	218
12	26.007058	218

No.	Timestamp[s]	Length
0	8.602220	160
1	18.503015	160
2	28.403900	160

Table 5.4: Timing of the emulation of the second dataset

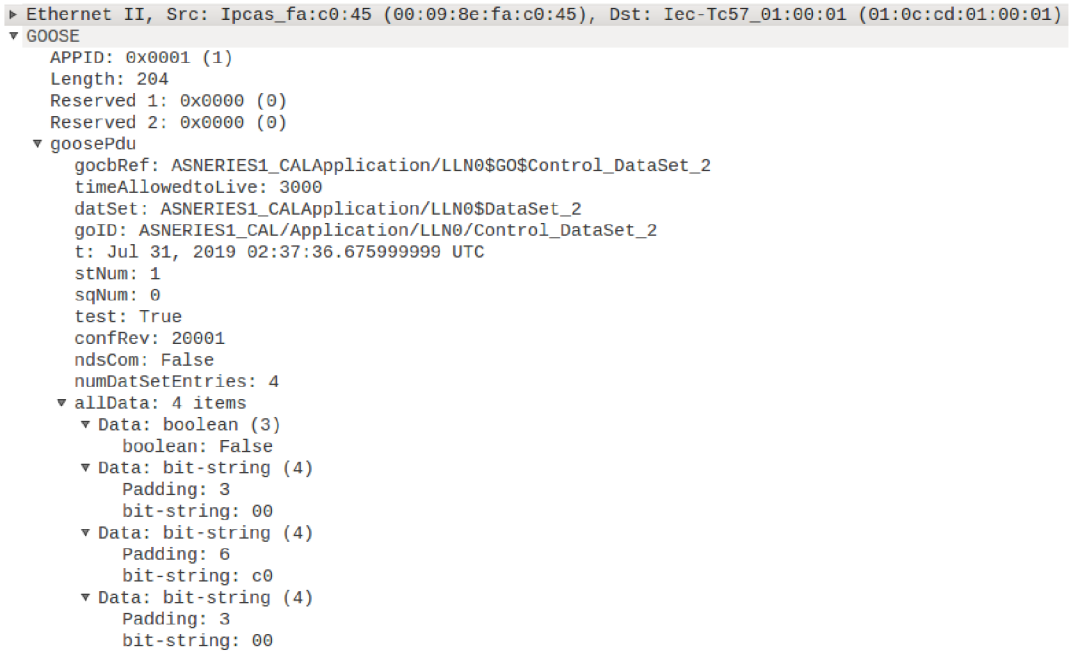

Figure 5.5: Application data from the emulation of the first flow of the second dataset

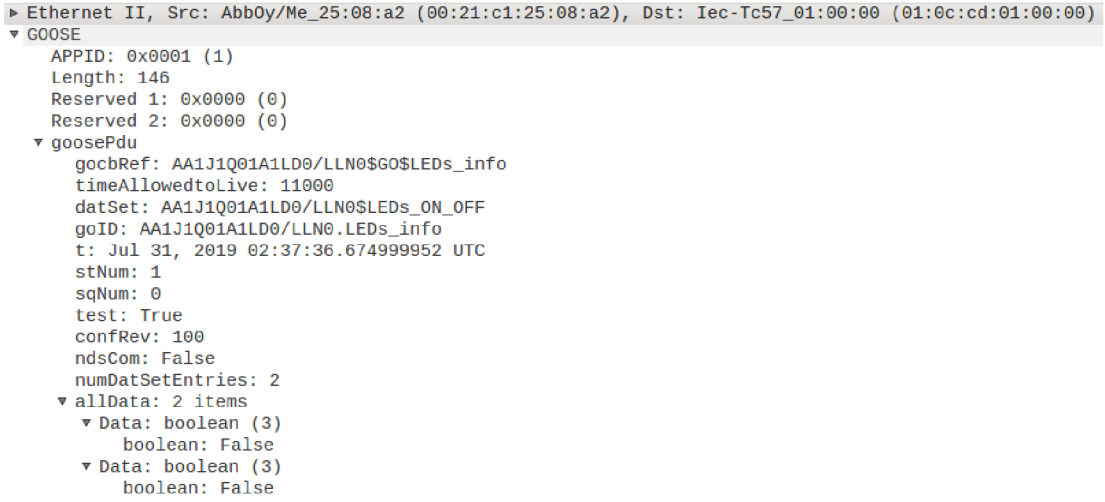

Figure 5.6: Application data from the emulation of the second flow of the second dataset

5.3 Result of emulation based on manually created configuration

In this experiment we used manually created configuration files. These files are visible in Appendix E. There are two flows declared. In one of the flows there is a change in application data during the emulation. The expected time course is to be seen in Table 5.5 and Table 5.6. For the estimate of time course following equation was used:

$$packet\_number * (flow\_end\_time - flow\_start\_time) = estimate$$

One flow should contain IEEE 802.1Q tag during the emulation. In this manually created configuration we have all supported datatypes.

No.	Estimate[s]
0	0.000000
1	0.444444
2	0.888888
3	1.333332
4	1.777776
5	2.222220
6	2.666664
7	3.111108
8	3.555552
9	3.999996

No.	Timestamp[s]
0	0.000000
1	0.444905
2	0.889669
3	1.334391
4	1.779154
5	2.223936
6	2.668693
7	3.113526
8	3.558333
9	4.003158

Table 5.5: Expected (left) and actual (right) timing (source MAC = 42:42:42:42:42:42)

No.	Timestamp[s]
0	0.000000
1	0.750000
2	1.500000
3	2.250000
4	3.000000
5	3.750000
6	4.500000
7	5.250000
8	6.000000
9	6.750000

No.	Timestamp[s]
0	0.000000
1	0.750402
2	1.500654
3	2.250911
4	3.001318
5	3.751718
6	4.502113
7	5.252417
8	6.002774
9	6.753194

Table 5.6: Expected (left) and actual (right) timing (source MAC = 00:09:8e:fa:c0:45)

In the Table 5.5 and Table 5.6 we can see a comparison between time courses of the two flows. There is a the small delay between messages. It is approximately four tenths of millisecond. Figure 5.7, Figure 5.8 and Figure 5.9 show application data from this communication. In Figure 5.7 in *allData* there is a octet-string. The data are malformed by Wireshark. The rest of application data of this flow corresponds to configuration from Listing E.2.

In Figure 5.8 and Figure 5.9 there is one flow. The flow is interesting because it has changed values in *allData* in the middle of flow. The floating-point is malformed by Wireshark as well. The rest of application data corresponds to Listing E.1. We can see the first flow does not have IEEE 802.1Q tag but the second flow does.

```
▸ Ethernet II, Src: 42:42:42:42:42:42 (42:42:42:42:42:42), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▾ GOOSE
    APPID: 0x0002 (2)
    Length: 215
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ▾ goosePdu
      gocbRef: IED1/Application/LLN0$GO$Control_DataSet_2
      timeAllowedtoLive: 3000
      datSet: IED1/Application/LLN0$DataSet_2
      goID: IED1/Application/LLN0/Control_DataSet_2
      t: Jul 31, 2019 04:53:17.980999946 UTC
      stNum: 1
      sqNum: 0
      test: True
      confRev: 20001
      ndsCom: False
      numDatSetEntries: 4
    ▾ allData: 4 items
      ▾ Data: unsigned (6)
          unsigned: 100000
      ▾ Data: octet-string (9)
          octet-string: 303030303030
      ▾ Data: visible-string (10)
          visible-string: test_visible_string
      ▾ Data: mMSString (16)
          mMSString: test_mms_string
```

Figure 5.7: Application data from the emulation flow (source MAC = 42:42:42:42:42:42)

```
▸ Ethernet II, Src: Ipcas_fa:c0:45 (00:09:8e:fa:c0:45), Dst: Iec-Tc57_01:00:01 (01:0c:cd:01:00:01)
▸ 802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 1
▾ GOOSE
    APPID: 0x0001 (1)
    Length: 206
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ▾ goosePdu
      gocbRef: ASNERIES1_CALApplication/LLN0$GO$Control_DataSet_2
      timeAllowedtoLive: 3000
      datSet: ASNERIES1_CALApplication/LLN0$DataSet_2
      goID: ASNERIES1_CAL/Application/LLN0/Control_DataSet_2
      t: Jul 31, 2019 04:53:17.980999946 UTC
      stNum: 1
      sqNum: 0
      test: True
      confRev: 20001
      ndsCom: False
      numDatSetEntries: 4
    ▾ allData: 4 items
      ▾ Data: boolean (3)
          boolean: False
      ▾ Data: integer (5)
          integer: 42
      ▾ Data: floating-point (7)
          floating-point: 0842848400
      ▾ Data: bit-string (4)
          Padding: 3
          bit-string: 00
```

Figure 5.8: Application data from the emulated flow (source MAC = 00:09:8e:fa:c0:45) first part

```
▸ Ethernet II, Src: Ipcas_fa:c0:45 (00:09:8e:fa:c0:45), Dst: Iec-Tc57_01:00:01 (01:0c:cd:01:00:01)
▸ 802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 1
▾ GOOSE
    APPID: 0x0001 (1)
    Length: 206
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ▾ goosePdu
      gocbRef: ASNERIES1_CALApplication/LLN0$GO$Control_DataSet_2
      timeAllowedtoLive: 3000
      datSet: ASNERIES1_CALApplication/LLN0$DataSet_2
      goID: ASNERIES1_CAL/Application/LLN0/Control_DataSet_2
      t: Jul 31, 2019 05:02:12.382999956 UTC
      stNum: 1
      sqNum: 5
      test: True
      confRev: 20001
      ndsCom: False
      numDatSetEntries: 4
    ▾ allData: 4 items
      ▾ Data: boolean (3)
          boolean: True
      ▾ Data: integer (5)
          integer: 24
      ▾ Data: floating-point (7)
          floating-point: 0842109000
      ▾ Data: bit-string (4)
          Padding: 6
          bit-string: 00
```

Figure 5.9: Application data from the emulated flow (source MAC = 00:09:8e:fa:c0:45) second part

5.4 Validation of generated communication with a third party software

To confirm that the communication generated by the emulator is valid according to the IEC 61850 standard, we used a third party software. The application GreyCortex Mendel[3] is used for network analysis including industrial networks.

The packet capture *goose_test_1.pcapng* was emulated on the application. In Figure 5.10 we can see statistics of link layer. In comparison with Table 5.2, there is approximately the same data. In Figure 5.11 we can see application data. When we compare the data with the application data from Figure 5.2 the data are the same.

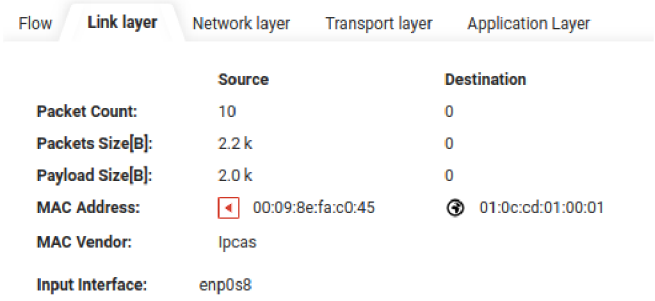

Figure 5.10: Link layer in GreyCortex Mendel

[3]https://www.greycortex.com/mendel

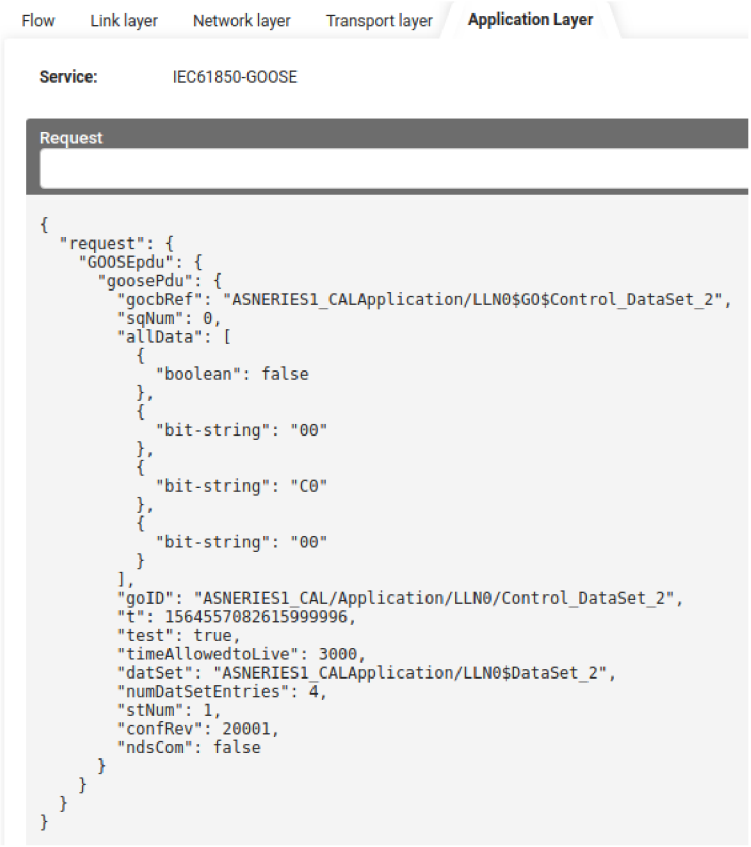

Figure 5.11: Application layer in GreyCortex Mendel

5.5 Summary

In this chapter we did multiple experiments. Their goal was to validate the behavior of the emulator. First we experimented with datasets we obtained in GIPSA-lab. We managed to generate the communication matching the requirements from Section 5.1. Then we manually generated transcripts of emulation with more complex communication. With this we confirmed the emulator can change application data value in the middle of flow and it can add IEEE 802.1Q tag to the Ethernet frame. With the last experiment we validated the emulator against third party software. This experiment confirmed that the communication is generated according to IEC 61850 standard.

Chapter 6

Conclusion

There is a lack of software applications for testing and simulation of devices which use protocols of IEC 61850 standard. The purpose of this work is to create an emulator of devices communicating using IEC 61850-GOOSE protocol. We fulfilled this purpose by implementation of the emulator of publisher devices of the GOOSE protocol. We validated the emulator with several experiments. We used datasets obtained during an internship in GIPSA-lab, Grenoble, France. These datasets contain communication from a real devices. We used these datasets as input of the emulator and then we compared these two communications. Also we manually generated a few transcripts of emulation for validation of support of extreme cases. We were not able to get a real dataset containing these cases. To validate the emulated communication we used a third party software GreyCortex Mendel which is able to parse GOOSE messages. The emulator is used in the GreyCortex company for testing purposes. The assignment was fulfilled.

In the future we are going to create the improve the application to make it even more useful for research and testing purposes. We would love to add support for MMS server and client. To make this possible, we will have to implement support for containing of IEC 61850 information model. Because of this, we need to add support for configuration using SCL files. When we will have the information model stored we can add support for SV protocol. We could also add GUI for making work with the emulator simpler.

Bibliography

[1] IEC-TC57: *Communication networks and systems for power utility automation – Part 9-2: Specific Communication Service Mapping (SCSM) – Sampled values over ISO/IEC 8802-3*. International Electrotechnical Commission. 2004.

[2] IEC-TC57: *Communication networks and systems for power utility automation – Part 6: Configuration description language for communication in power utility automation systems related to IEDs* . International Electrotechnical Commission. 2010.

[3] IEC-TC57: *Communication networks and systems for power utility automation – Part 7-1: Principles and models*. International Electrotechnical Commission. 2010.

[4] IEC-TC57: *Communication networks and systems for power utility automation – Part 7-2: Basic information and communication structure*. International Electrotechnical Commission. 2010.

[5] IEC-TC57: *Communication networks and systems for power utility automation – Part 8-1: Specific Communication Service Mapping (SCSM) – Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3*. International Electrotechnical Commission. 2010.

[6] ISO/TC184/SC5: *Industrial automation systems – Manufacturing Message Specification, Part 1: Service definition* . International Organization for Standardization. 2003.

[7] ITU-T: *OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)*. International Telecommunication Union. 2015.

[8] LEON, H.; MONTEZ, C.; STEMMER, M.; et al.: Simulation models for IEC 61850 communication in electrical substations using GOOSE and SMV time-critical messages. In *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. May 2016. pp. 1–8. doi:10.1109/WFCS.2016.7496500.
Retrieved from: https://ieeexplore.ieee.org/document/7496500

[9] MACKIEWICZ, R. E.: Overview of IEC 61850 and Benefits. In *2006 IEEE PES Power Systems Conference and Exposition*. Oct 2006. pp. 623–630. doi:10.1109/PSCE.2006.296392.
Retrieved from: https://ieeexplore.ieee.org/document/4075831

[10] MATOUŠEK, P.: Description of IEC 61850 Communication. Technical report. 2018.
Retrieved from: http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11832

[11] MOHAGHEGHI, S.; STOUPIS, J.; WANG, Z.: Communication protocols and networks for power systems-current status and future trends. In *2009 IEEE/PES Power Systems Conference and Exposition*. March 2009. pp. 1–9. doi:10.1109/PSCE.2009.4840174.
Retrieved from: https://ieeexplore.ieee.org/document/4840174

[12] SCHWARZ, K.: Manufacturing message specification-iso 9506 (mms). *Schwarz Consulting Company, Karlsruhe, Germany. Available from World Wide Web:< URL: http://www. nettedautomation. com/download/MMS-R1-2_2008-02-26. pdf*. 2008.
Retrieved from: https://www.semanticscholar.org/paper/Manufacturing-Message-Specification-%E2%80%93-ISO-9506-Schwarz-Schwarz/40e6f1ea9fafeb582750df7ae725d1dd836659f6

Appendices

Appendix A

DTD description for the XML configuration files

```
<!ELEMENT configuration (headersValues+)>
<!ELEMENT headersValues (gooseValues,timeCount,timeStart,timeEnd)>
<!ELEMENT gooseValues (src,dst,goID,goCb,datSet,confRev,needsComm,allowedTTL,dataValues+,
    appID,vlanID,vlanPrio)>

<!ELEMENT src (#PCDATA)>
<!ELEMENT dst (#PCDATA)>
<!ELEMENT goID (#PCDATA)>
<!ELEMENT goCb (#PCDATA)>
<!ELEMENT datSet (#PCDATA)>
<!ELEMENT confRev (#PCDATA)>
<!ELEMENT needsComm (#PCDATA)>
<!ELEMENT dataValues (#PCDATA)>
<!ELEMENT appID (#PCDATA)>
<!ELEMENT vlanID (#PCDATA)>
<!ELEMENT vlanPrio (#PCDATA)>
<!ELEMENT dataValues (goose.integer+, goose.unsigned+, goose.floating_point+, goose.boolean+,
    goose.octet_string+, goose.visible_string+, goose.mMSString+, goose.bit_string+)>

<!ELEMENT goose.integer (#PCDATA)>
<!ELEMENT goose.unsigned (#PCDATA)>
<!ELEMENT goose.floating_point (#PCDATA)>
<!ELEMENT goose.boolean (#PCDATA)>
<!ELEMENT goose.octet_string (#PCDATA)>
<!ELEMENT goose.visible_string (#PCDATA)>
<!ELEMENT goose.mMSString (#PCDATA)>
<!ELEMENT goose.bit_string (#PCDATA)>
```

Listing A.1: DTD description of the XML configuration files

Appendix B

Example of an SCL with GOOSE export

This SCL file was obtained from Stéphane Mocanu, Grenoble INP. The file contains simplified configuration of a substation with devices using IEC 61850.

```
<SCL>
  <Communication>
    <SubNetwork name="ProcessBusSubnet" type="8-MMS">
      <ConnectedAP iedName="ASNERIES1_CAL" apName="E">
        <GSE ldInst="Application" cbName="Control_DataSet_2">
          <Address>
            <P type="MAC-Address" xsi:type="tP_MAC-Address">01-0C-CD
                -01-00-01</P>
            <P type="VLAN-ID" xsi:type="tP_VLAN-ID">000</P>
            <P type="VLAN-PRIORITY" xsi:type="tP_VLAN-PRIORITY">4</P>
            <P type="APPID" xsi:type="tP_APPID">0001</P>
          </Address>
          <MinTime unit="s" multiplier="m">10</MinTime>
          <MaxTime unit="s" multiplier="m">2000</MaxTime>
        </GSE>
      </ConnectedAP>
    </SubNetwork>
  </Communication>

  <IED desc="ASNERIES1_CAL_6MD85" name="ASNERIES1_CAL" type="6MD85">
    <Services nameLength="64">
      <ClientServices goose="true" gsse="false" bufReport="false" unbufReport="false" readLog
          ="false" sv="false" supportsLdName="true" maxGOOSE="128">
        <TimeSyncProt sntp="true" />
      </ClientServices>
    </Services>
    <AccessPoint desc="Port E" name="E" router="false" clock="false">
      <Server timeout="0">
        <Authentication none="true" />
        <LDevice desc="Application" inst="Application">
          <LN0 lnClass="LLN0" inst="" lnType="SIPROTEC5_LNType_LLN0_Application"
              desc="General">
            <DataSet name="DataSet_2">
              <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="
                  SPC" daName="stVal" fc="ST" />
              <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="
                  SPC" daName="q" fc="ST" />
```

```
33                    <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="
                          DPCCB" daName="stVal" fc="ST" />
34                    <FCDA ldInst="Application" prefix="" lnClass="USER" lnInst="1" doName="
                          DPCCB" daName="q" fc="ST" />
35                 </DataSet>
36                 <GSEControl datSet="DataSet_2" confRev="20001" appID="ASNERIES1_CAL
                       /Application/LLN0/Control_DataSet_2" name="Control_DataSet_2" type
                       ="GOOSE" />
37              </LN0>
38              <LN lnClass="USER" inst="1" lnType="SIPROTEC5_LNType_USER_Universal"
                     desc="Novion Test" prefix="">
39                 <DOI name="DPCCB" desc="CB Position">
40                    <DAI name="dataNs">
41                       <Val>Siprotec5/user-defined</Val>
42                    </DAI>
43                 </DOI>
44                 <DOI name="SPC" desc="SPC">
45                    <DAI name="dataNs">
46                       <Val>Siprotec5/user-defined</Val>
47                    </DAI>
48                 </DOI>
49              </LN>
50           </LDevice>
51        </Server>
52     </AccessPoint>
53  </IED>
54
55  <IED name="MU_02" type="Siprotec-7SX8xx">
56     <Services nameLength="64">
57        <ConfDataSet max="5" maxAttributes="30" modify="true" />
58        <GSESettings cbName="Conf" datSet="Conf" appID="Conf" />
59        <SMVSettings cbName="Fix" datSet="Fix" svID="Conf" optFields="Conf" smpRate="Fix
                " samplesPerSec="false" pdcTimeStamp="false">
60           <SmpRate>80</SmpRate>
61        </SMVSettings>
62        <GOOSE max="3" />
63        <SMVsc max="1" delivery="multicast" deliveryConf="false" />
64        <ConfLNs fixPrefix="true" fixLnInst="true" />
65        <ClientServices goose="true" maxGOOSE="3" supportsLdName="false" maxAttributes="
                60" gsse="false" bufReport="false" unbufReport="false" readLog="false" sv="false"
                maxReports="0" maxSMV="0">
66           <TimeSyncProt sntp="true" c37_238="false" other="true" />
67        </ClientServices>
68        <SupSubscription maxGo="0" maxSv="0" />
69        <RedProt hsr="true" prp="true" rstp="true" />
70     </Services>
71     <AccessPoint name="P1">
72        <Server>
73           <Authentication none="true" />
74           <LDevice inst="MU01" desc="Mu01">
75              <LN inst="1" prefix="ASN" desc="ASNGGIO1" lnClass="GGIO" lnType="MU_02/
                      CTRL/ASNGGIO1">
76                 <DOI name="SPCSO3" desc="external Single Point ON/OFF (ExSP)">
77                    <Private type="Siemens-Dir">Rx</Private>
78                    <DAI name="ctlModel">
79                       <Val>status-only</Val>
80                    </DAI>
81                 </DOI>
82                 <DOI name="DPCSO1" desc="external double indication (ExDI)">
```

```
83                    <Private type="Siemens-Dir">Rx</Private>
84                    <DAI name="ctlModel">
85                       <Val>status-only</Val>
86                    </DAI>
87                 </DOI>
88                 <Inputs>
89                    <ExtRef doName="SPC" daName="stVal" intAddr="CTRL/ASNGGIO1/ST/
                         SPCSO3/stVal" serviceType="GOOSE" iedName="ASNERIES1_CAL"
                         ldInst="Application" lnClass="USER" lnInst="1" srcCBName="
                         Control_DataSet_2" srcLDInst="Application" srcLNClass="LLN0" />
90                    <ExtRef doName="SPC" daName="q" intAddr="CTRL/ASNGGIO1/ST/
                         SPCSO3/q" serviceType="GOOSE" iedName="ASNERIES1_CAL" ldInst
                         ="Application" lnClass="USER" lnInst="1" srcCBName="
                         Control_DataSet_2" srcLDInst="Application" srcLNClass="LLN0" />
91                    <ExtRef doName="DPCCB" daName="stVal" intAddr="CTRL/ASNGGIO1/
                         ST/DPCSO1/stVal" serviceType="GOOSE" iedName="
                         ASNERIES1_CAL" ldInst="Application" lnClass="USER" lnInst="1"
                         srcCBName="Control_DataSet_2" srcLDInst="Application" srcLNClass=
                         "LLN0" />
92                    <ExtRef doName="DPCCB" daName="q" intAddr="CTRL/ASNGGIO1/ST/
                         DPCSO1/q" serviceType="GOOSE" iedName="ASNERIES1_CAL" ldInst
                         ="Application" lnClass="USER" lnInst="1" srcCBName="
                         Control_DataSet_2" srcLDInst="Application" srcLNClass="LLN0" />
93                 </Inputs>
94              </LN>
95           </LDevice>
96        </Server>
97     </AccessPoint>
98  </IED>
99
100 <LNodeType id="SIPROTEC5_LNType_USER_Universal" lnClass="USER">
101    <DO name="SPC" type="SPC_ID" />
102    <DO name="DPCCB" type="DPC_ID" />
103 </LNodeType>
104
105 <DOType id="SPC_ID" cdc="SPC">
106    <DA dchg="true" fc="ST" name="stVal" bType="BOOLEAN" />
107    <DA qchg="true" fc="ST" name="q" bType="Quality" />
108 </DOType>
109 <DOType id="DPC_ID" cdc="DPC">
110    <DA dchg="true" fc="ST" name="stVal" bType="Dbpos" />
111    <DA qchg="true" fc="ST" name="q" bType="Quality" />
112 </DOType>
113 </SCL>
```

Listing B.1: Example of an SCL with GOOSE export

Appendix C

Configuration file generated from Apendix B

```
<configuraton>
   <headersValues>
      <gooseValues>
         <src>00:00:00:00:00:01</src>
         <dst>01:0c:cd:01:00:01</dst>
         <goID>ASNERIES1_CAL/Application/LLN0/Control_DataSet_2</goID>
         <goCb>ASNERIES1_CAL/Application/LLN0$Control_DataSet_2</goCb>
         <datSet>ASNERIES1_CAL/Application/LLN0$DataSet_2</datSet>
         <confRev>20001</confRev>
         <needsComm>0</needsComm>
         <allowedTTL>3000</allowedTTL>
         <dataValues>
            <goose.boolean>0</goose.boolean>
         </dataValues>
         <dataValues>
            <goose.bit_string>00:00</goose.bit_string>
         </dataValues>
         <dataValues>
            <goose.bit_string>c0</goose.bit_string>
         </dataValues>
         <dataValues>
            <goose.bit_string>00:00</goose.bit_string>
         </dataValues>
         <appID>0x00000001</appID>
         <vlanID>000</vlanID>
         <vlanPrio>4</vlanPrio>
      </gooseValues>
      <timeCount>10</timeCount>
      <timeStart>0</timeStart>
      <timeEnd>10</timeEnd>
   </headersValues>
</configuraton>
```

Listing C.1: Configuration file generated from SCL from Listing B.1

Appendix D

User manual

The root folder of this project contains:

- ConfigurationCreator.rb - script used for creation of configuration files from capture file - please, use Control.rb instead for this purpose
- Control.rb - script for directing od the emulation
- examples - folder containing an example dataset
  - pcap - contains packet captures usable for an emulation
  - xml - contains examples of configuration files
- goose_publisher_emulator.c - file containing implementation of the emulator
- libiec61850-1.3 - folder containing the libiec61850 folder
- Makefile - makefile
- README.md - the file containing this text
- scl - directory containing an example of SCL file (can not be an input of emulator)

For compilation and proper work, the project requires following packages:

- gcc-4.8.5
- ruby-2.0.0p648
- libxml2-devel-2.9.1-6

The installation process was tested on *Ubuntu 16.04*.
For installation please navigate into project root folder and use following commands:

- sudo apt-get install gcc ruby libxml2-devel
- gem install gyoku
- make

Please, always run the emulator using *Control.rb*. To run the emulation you can use following examples:

- to run emulation from capture file use following command:
  sudo ruby Control.rb -p <input_pcap> -r -o <output_file>
- to run emulation with a directory with prepared transcripts of emulation use following command:
  sudo ruby Control.rb -c <configuration_directory> -r -o <output_file>
- to run emulation with prepared configuration files on specific network interface:
  sudo ruby Control.rb -c <configuration_directory> -r -i <interface>
- to run configuration generation:
  ruby Control.rb -c <configuration_directory> -p <input_file>

Command line parameters of Control.rb :

- *-i* - output interface
- *-p* - input pcap file
- *-o* - output pcap file
- *-r* - run emulation
- *-c* - configuration directory

For manual creation of configuration files you can use following blank template:

```
<configuraton>
   <headersValues> <!--can be repeated-->
      <gooseValues>
         <src></src> <!--string format: "XX:XX:XX:XX:XX:XX"-->
         <dst></dst> <!--string format: "XX:XX:XX:XX:XX:XX"-->
         <goID></goID> <!--string-->
         <goCb></goCb> <!--string-->
         <datSet></datSet> <!--string-->
         <confRev></confRev> <!--integer-->
         <needsComm></needsComm> <!--integer-->
         <allowedTTL></allowedTTL> <!--integer-->
         <dataValues> <!--can be repeated-->
            <goose.boolean></goose.boolean> <!--integer-->
         </dataValues>
         <appID></appID> <!--string containing hexa integer in format "0x0001"-->
         <vlanID></vlanID> <!--integer-->
         <vlanPrio></vlanPrio> <!--integer-->
      </gooseValues>
      <timeCount></timeCount> <!--integer-->
      <timeStart></timeStart> <!--float-->
      <timeEnd></timeEnd> <!--float-->
   </headersValues>
</configuraton>
```

Listing D.1: Blank configuration file

Appendix E

Configuration files used for experiment from Section 5.3

In this appendix there are the configuration files used for the experiment from Section 5.3.

```
<configuraton>
    <headersValues>
        <gooseValues>
            <src>00:09:8e:fa:c0:45</src>
            <dst>01:0c:cd:01:00:01</dst>
            <goID>ASNERIES1_CAL/Application/LLN0/Control_DataSet_2</goID>
            <goCb>ASNERIES1_CALApplication/LLN0$GO$Control_DataSet_2</goCb>
            <datSet>ASNERIES1_CALApplication/LLN0$DataSet_2</datSet>
            <confRev>20001</confRev>
            <needsComm>0</needsComm>
            <allowedTTL>3000</allowedTTL>
            <dataValues>
                <goose.boolean>0</goose.boolean>
            </dataValues>
            <dataValues>
                <goose.integer>42</goose.integer>
            </dataValues>
            <dataValues>
                <goose.floating_point>42.42</goose.floating_point>
            </dataValues>
            <dataValues>
                <goose.bit_string>00</goose.bit_string>
            </dataValues>
            <appID>0x00000001</appID>
            <vlanID>1</vlanID>
            <vlanPrio>4</vlanPrio>
        </gooseValues>
        <timeCount>5</timeCount>
        <timeStart>0</timeStart>
        <timeEnd>3</timeEnd>
    </headersValues>
    <headersValues>
        <gooseValues>
            <src>00:09:8e:fa:c0:45</src>
            <dst>01:0c:cd:01:00:01</dst>
            <goID>ASNERIES1_CAL/Application/LLN0/Control_DataSet_2</goID>
            <goCb>ASNERIES1_CALApplication/LLN0$GO$Control_DataSet_2</goCb>
```

```
38            <datSet>ASNERIES1_CALApplication/LLN0$DataSet_2</datSet>
39            <confRev>20001</confRev>
40            <needsComm>0</needsComm>
41            <allowedTTL>3000</allowedTTL>
42          <dataValues>
43              <goose.boolean>1</goose.boolean>
44            </dataValues>
45            <dataValues>
46              <goose.integer>24</goose.integer>
47            </dataValues>
48            <dataValues>
49              <goose.floating_point>24.24</goose.floating_point>
50            </dataValues>
51            <dataValues>
52              <goose.bit_string>00</goose.bit_string>
53            </dataValues>
54            <appID>0x00000001</appID>
55            <vlanID>1</vlanID>
56            <vlanPrio>4</vlanPrio>
57        </gooseValues>
58        <timeCount>5</timeCount>
59        <timeStart>3</timeStart>
60        <timeEnd>6</timeEnd>
61    </headersValues>
62 </configuraton>
```

Listing E.1: Configuration file generated from SCL from Listing B.1

```
<configuraton>
    <headersValues>
        <gooseValues>
            <src>42:42:42:42:42:42</src>
            <dst>00:00:00:00:00:02</dst>
            <goID>IED1/Application/LLN0/Control_DataSet_2</goID>
            <goCb>IED1/Application/LLN0$GO$Control_DataSet_2</goCb>
            <datSet>IED1/Application/LLN0$DataSet_2</datSet>
            <confRev>20001</confRev>
            <needsComm>0</needsComm>
            <allowedTTL>3000</allowedTTL>
            <dataValues>
                <goose.unsigned>0100000</goose.unsigned>
            </dataValues>
            <dataValues>
                <goose.octet_string>000000</goose.octet_string>
            </dataValues>
            <dataValues>
                <goose.visible_string>test_visible_string</goose.visible_string>
            </dataValues>
            <dataValues>
                <goose.mMSString>test_mms_string</goose.mMSString>
            </dataValues>
            <appID>0x00000002</appID>
            <vlanID>-1</vlanID>
            <vlanPrio>-1</vlanPrio>
        </gooseValues>
        <timeCount>10</timeCount>
        <timeStart>0</timeStart>
        <timeEnd>4</timeEnd>
    </headersValues>
</configuraton>
```

Listing E.2: Configuration file generated from SCL from Listing B.1