Master Thesis

Assessing the Security of IEC 60870-5-104 Implementations using Automata Learning

UNIVERSITY OF TWENTE.

Max Kerkers

April 2017

Max Kerkers: *Assessing the Security of IEC 60870-5-104 Implementations using Automata Learning*, April 2017

Supervisors:

prof. dr. ir. B.R.H.M. (Boudewijn) Haverkort - University of Twente
dr. A.K.I. (Anne) Remke - University of Twente
J.J. (Justyna) Chromik MSc - University of Twente
D. (Dennis) Waalewijn MSc - KPMG
M.J. (Martijn) Sprengers MSc - KPMG

Abstract

Industrial Control Systems (ICS) that monitor and control (critical) infrastructures have become more connected and therefore easier to reach from the internet. As a result of this, it has become easier for attackers to perform an attack on an ICS from a remote location. A protocol, that is used in such an ICS for the control of power distribution, is IEC 60870-5-104. In this thesis, a tool is presented that can be used to infer automata, i.e. finite state machines, from implementations of this protocol. This tool is used to learn automata from three simulators and two real devices that all implement IEC 60870-5-104. These automata are compared with each other and with the specification in the IEC 60870-5-104 standard. The real devices follow the specification more closely then the simulators. However, for both real devices, specific sequences of messages have still been found that make the implementation deviate from the specification. These differences could be used in attacks against this protocol or to fingerprint devices.

Acknowledgements

First of all, I would like to express my gratitude to Juystyna Chromik from the University of Twente, for the many discussions about my research, arranging meetings with Stedin and Cogas so that I could perform research there, the thorough reading through everything I wrote and the really helpful feedback. I would also like to thank Boudewijn Haverkort, as my first supervisor, and Anne Remke for joining my graduation committee.

I am very grateful that I could write my thesis as part of the KPMG Cyber team. Particularly, I would like to thank Dennis Waalewijn as my main supervisor from KPMG and Martijn Sprengers for helping me to come up with the subject of my research. Also, I would like to express my appreciation for my co-interns, who provided the occasional necessary distraction.

Furthermore, I would like express my gratitude to Joey Godefrooi and Pascal Everste from Stedin, for providing me access to one of their testing stations, where I could perform research. I would also like to thank Gerard Geist from Cogas and Ronald Robbertsen from Datawatt, for proving me with an RTU to perform tests on.

Lastly, I would like to thank my friends and family for their support throughout this research and the rest of my study.

Contents

List of Figures

List of Tables

List of Abbreviations

APCI Application Protocol Control Information. 30, 53

APDU Application Protocol Data Unit. 36, 53, 54

ARP Address Resolution Protocol. 27

ASDU Application Service Data Unit. 30, 34, 52–54, 59, 64, 73, 74

DCS Distributed Control Systems. 23

DFA Deterministic Finite Automaton. 40, 41, 43, 47

DNP3 Distributed Network Protocol. 28, 54

FSM Finite State Machine. 3, 39, 40

HMI Human-Machine Interface. 24, *Glossary:* HMI

ICS Industrial Control Systems. 3, 17–19, 21, 23, 25–28, 36, 56, 66, 72, 73

IEC International Electrotechnical Commission. 3, 18, 19, 26–28, 30, 36, 56, 66, 69–71, 73, 74

IEC-101 IEC 60870-5-101. 28, 30, 54, 60

IEC-104 IEC 60870-5-104. 21, 27, 28, 30, 48, 51–54, 56, 60, 61, 63, 65, 70–74

IED Intelligent Electronic Device. 24, *Glossary:* IED

IoT Internet of Things. 25

IP Internet Protocol. 23, 25, 27, 28, 30

LAN Local Area Network. 21

MAC Media Access Control. 27

MMS Manufacturing Message Specification. 27, 73

MTU Master Terminal Unit. 24, *Glossary:* MTU

PLC Programmable Logic Controller. 24, 56, *Glossary:* PLC

RTU Remote Terminal Unit. 21, 23, 24, 28, 33, 55, 56, 63–67, 70, 71, 73, 74, *Glossary:* RTU

SCADA Supervisory Control And Data Acquisition. 15, 17, 23, 28

SSH Secure Shell. 40

SUT Subject Under Test. 42, 46, 51, 54

TCP Transmission Control Protocol. 23, 25, 27, 28, 30, 36, 39

WAN Wide Area Network. 24, 25

Glossary

control server Host of the supervisory control software. 24

historian Database with records of all process information. 24

HMI (human-machine interface) Interface to provide human operators with information and control. 24

IED (intelligent electronic device) Sensor or actuator that is able to acquire data, communicate to other devices and perform local processing and control. 24

implementation The realisation of a specification into an actual system. 37

MTU (master terminal unit) Device acting as master for the communication in SCADA system. 24

PLC (programmable logic controller) Special purpose data acquisition and control unit provided with a communication interface. 24

protocol "A formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information." [20]. 15, 37

RTU (remote terminal unit) Special purpose data acquisition and control unit provided with a communication interface. 21, 24

specification The complete formal description of a protocol. 15, 37

standard "Document, established by consensus and approved by a recognised body, that provides, for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context." [19]. 37

Chapter 1

Introduction

In 2007, the Idaho National Labratories conducted an experiment that demonstrated that, when having access to the control network, it is possible to physically damage components that are used in power grids [35, 40]. This experiment was referred to as the Aurora generator test and it showed that it was possible to physically destroy a power generator by attacking its process control system. This control system is also used in operational power plants.

These Industrial Control Systems (ICS), such as Supervisory Control And Data Acquisition (SCADA) systems, are used to control and monitor (critical) infrastructures. These SCADA systems often control geographically distributed facilities such as power generation and distribution, water treatment, and oil and gas distribution. Also, ICS are found in systems for, e.g., heating, ventilation and air conditioning, as well as lighting or physical security.

Most of these systems were designed decades ago as standalone systems and were not connected to other networks. However, as these systems have become more connected over the years, threats from outside the network cannot be disregarded anymore. Even if the network of some industry, e.g., a power plant, is not connected to the internet, it can still be infected with malicious software by a device from outside the network. For example, this happened to an Iranian nuclear power plant, when a computer that was infected with the Stuxnet virus was connected to the trusted network of the nuclear power plant [26].

ICS are widely used, especially for controlling critical infrastructures; unfortunately, in many cases the protocols used for the communication in ICS, lack a formal proof that they satisfy the specification. For example, there could be flaws in implementations of protocols used in ICS, which may affect the security of the entire system. Such a flaw could cause a device to crash or become unresponsive, but this often only happens under irregular conditions, e.g., when multiple unusual messages are received in a specific order. This can

be a serious threat to the process, if the device with the implementation flaw is important to the system.

This research investigates implementations of the IEC 60870-5-104 protocol, which is an ICS protocol that is mainly used in power distribution. This protocol was chosen, because it is crucial for the communication between the control stations and distribution stations in Europe, even when these distribution station use newer protocols internally. Although this protocol is widely used in Europe, not much research has been performed on its security [28]. In the research for this thesis a tool is developed that is capable of inferring a formal representation from an IEC 60870-5-104 implementation. The source code of this tool can be found on GitHub [24].

The goal of this research is to provide a method to formally represent implementations of IEC 60870-5-104, as well as to provide means to analyse and compare the obtained representations and use them to assess the security attributes of IEC 60870-5-104. To achieve these mentioned goals, the following research questions are addressed in this master thesis:

RQ1: How can implementations of IEC 60870-5-104 be represented formally?

The goal of this research question is to propose a method to generate formal representations from the implementations of IEC 60870-5-104. To achieve this goal, two sub-questions need to be addressed first:

a) Which methods exist to generate formal representations?

b) What method is suitable for formally representing protocols?

RQ2: To what extent do implementations of IEC 60870-5-104 comply to their standard?

The goal of this research question is to provide an analysis of implementations of the IEC 60870-5-104 protocol and to compare them to the standard. The method that is obtained from the first research question can be used to represent these implementations formally. In order to be able to achieve this goal, these sub-questions need to be addressed first:

a) What implementations of IEC 60870-5-104 are available?

b) How can the IEC 60870-5-104 standard be formally represented?

c) How to compare these formal representations?

d) What do the findings from these questions tell about the IEC 60870-5-104 standard?

If these comparisons yield differences between implementations of the IEC 60870-5-104 specification, this indicates that the specification contains ambiguities and obscurities. These differences are described and their significance is examined by, e.g., investigating if a transition that can be circumvented in one of the implementation could cause incorrect behaviour.

RQ3: What information security attributes are violated in the implementations that were researched in the second research question and how?

The goal of this research question is to perform an assessment of the information security attributes based on the findings from the second research question. Before violations of these security attributes can be found, the following sub-question needs to be addressed:

a) What information security attributes are relevant in ICS?

The first contribution of this thesis is the described tool that can be used to infer formal representations automatically from any implementation of IEC 60870-5-104. The second contribution is the comparison and analysis of these formal representations that this tool generates. Another contribution is the assessment of the information security attributes of the implementations. Finally, the last contribution is the analysis of the specification itself.

The rest of this thesis consists of the following: Chapter 2 provides a background on ICS and their development and security aspects. In Chapter 3, methods on how to learn state machines from protocol implementations are given. Chapter 4 describes the methodology and the tools that were used and Chapter 5 describes the results. Finally, Chapter 6 concludes this thesis with a discussion about the research, ideas for future work and recommendations for vendors and operators.

Chapter 2

Industrial Control Systems

This chapter provides background information about Industrial Control Systems (ICS) and their security. First, section 2.1 provides some historical background on the development of ICS. Section 2.2 defines the components of ICS, as they are used in the rest of this thesis. Then, background information about the security of ICS is given in section 2.3. Section 2.4 describes the protocol IEC 60870-5-104 (IEC-104) that is examined in the rest of this research. Finally, this chapter is concluded with several final remarks in section 2.5.

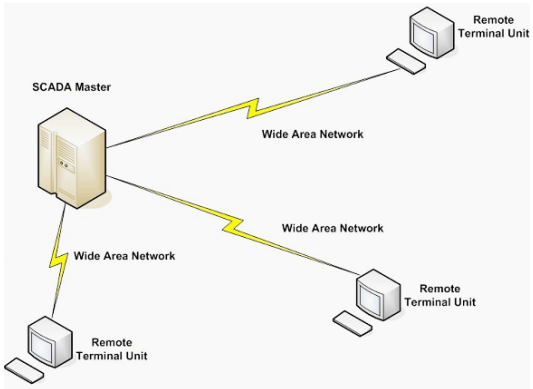
2.1 History of ICS

Thakur [44] divides the historical development of ICS into three generations: (i) the monolithic generation (Figure 2.1a), (ii) the distributed generation (Figure 2.1b) and (iii) the networked generation (Figure 2.1c).

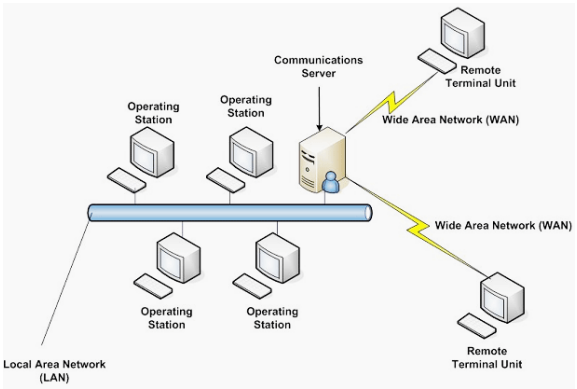
In the monolitic generation, ICS consisted of a mainframe computer with a dedicated serial line to each remote terminal unit (RTU) in the system. Each ICS vendor had developed their own devices that were using their own proprietary protocols. This resulted globally in about 200 proprietary protocols that were not interoperable [22].

In the distributed generation, ICS were not controlled by a single mainframe anymore. Instead, the mainframe was replaced by a local area network (LAN) of multiple systems. In this LAN each of the systems had a specific task. Communication between this LAN and the RTUs still happened over dedicated lines, but now there was one single computer in the LAN with the task of communicating with the RTUs. On the LAN itself the devices became connected using standard protocols, while for the communication with the RTUs proprietary protocols were still used.

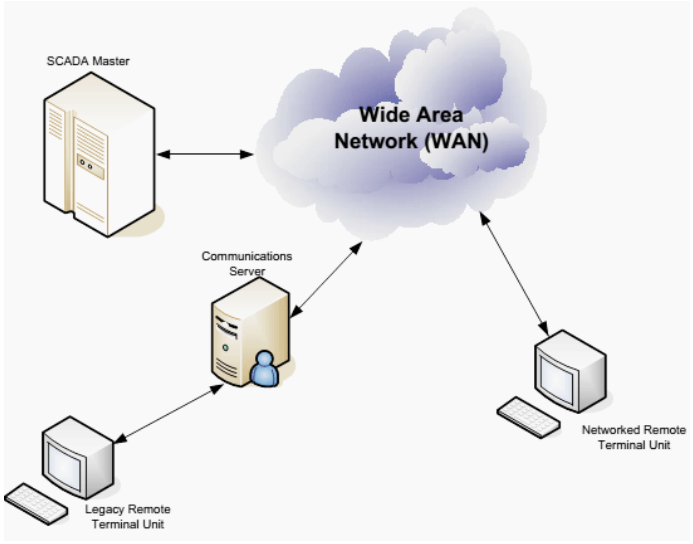
In the networked generation, the proprietary protocols and architecture were replaced by open standards. In this generation all devices (that are capable of it), communicate



(A) First generation: monolithic



(B) Second generation: distributed



(C) Third generation: networked

FIGURE 2.1: Generations in development of ICS/SCADA [12]

over packet switched networks using standard protocols like, e.g., TCP/IP [11]. Between the legacy RTUs, that are not capable of communicating over the standard protocols, a communication server is placed, which translates the communication. As the protocols in this architecture are based on open standards, interoperability between systems of different vendors became possible.

2.2 ICS components

ICS is the most comprehensive term for several types of control systems, including Supervisory Control And Data Acquisition (SCADA) systems and Distributed Control Systems (DCS) [39]. In Figure 2.2 the used terminology is visualised. The term SCADA is used to describe systems spanning a large geographical area, whereas DCS describes systems controlling a single location. This is denoted in Figure 2.2 by presenting DCS as a subsystem of SCADA. Note, that DCS do not have to be a part of a SCADA system; it can also be a standalone system.

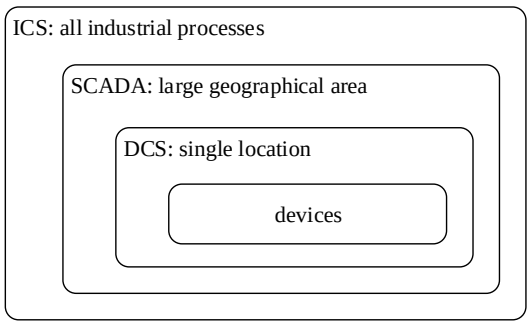


FIGURE 2.2: Overview scopes ICS, SCADA and DCS

ICS or SCADA systems generally can be divided intro three parts: (i) field network, (ii) control network and (iii) communication link [5]. For each of these parts several general components can be identified, as can be seen in Figure 2.3. How these components cooperate is described next.

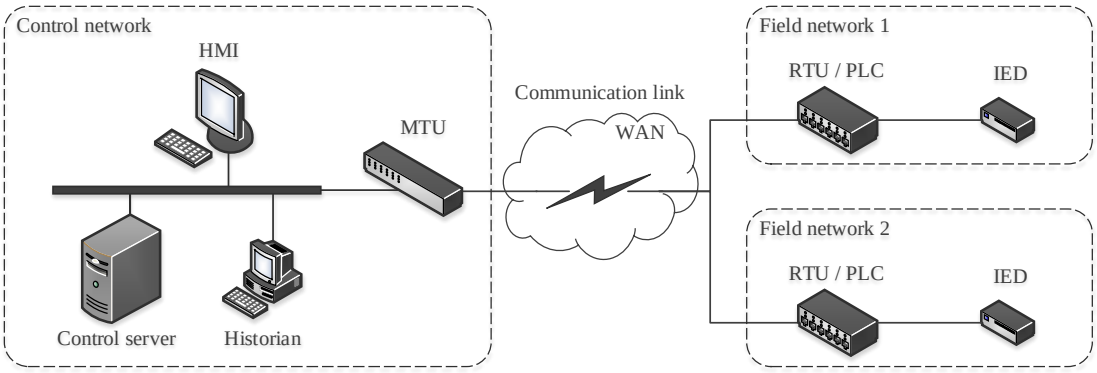


FIGURE 2.3: Typical Industrial Control System

Field network

The field network comprises the remote location that is controlled. Components that generally can be found in a field network are: a remote terminal unit (RTU), a programmable logic controller (PLC) and an intelligent electronic device (IED) [5, 33, 14]. The primary function of the RTU is to communicate with the MTU in the control network. Furthermore, an RTU can also act as a PLC in the field network. Attached to a PLC there can be several IEDs, from which the PLC collects data and to which it can send commands.

Control network

The control network comprises the location from where the entire system is controlled. Components that generally can be found in control networks are: a control server, a human-machine interface (HMI), a master terminal unit (MTU) and a historian. The control server controls this network and hosts the control software. The MTU sends control data to the RTUs and presents gathered data to the HMI. This HMI can be used by human operator to obtain information and to control the system. Lastly, the historian contains a database with all prior process information.

Communication link

The communication link can be any link, both wired or wireless that connects a field network to the control network. Most communication links are using a wide area network

(WAN).

2.3 Security of ICS

As described in section 2.1, ICS historically were structured so that they mainly depended on two forms of protection [5]: (i) the so-called *air gap* that isolated the systems from other networks, and (ii) the reliance of vendors on *security through obscurity*. This latter means that vendors believed that their systems would be safe as long as they would keep the information about them secret.

However, in the latest generation, ICS have become more interconnected, as a result of which protecting ICS became an even greater challenge [11]. Before this generation, an attacker had to acquire physical access in order to intrude the ICS, because there were only dedicated serial lines. Nowadays, the attacker would be able to attack the ICS remotely by compromising only a single machine that has access to the ICS' network.

Because of the standardisation of protocols and the increase in use of generic network devices, *security through obscurity* cannot be considered as a protection mechanism. In the endeavour of reducing cost and increasing efficiency, more low cost commercial off-the-shelf devices, like Windows computers, are used in ICS [5]. Also, the universal TCP/IP protocol stack is adapted more often.

Additionally, with the current tendency where all devices become increasingly connected to the internet, resulting in the so-called Internet of Things (IoT), the attack surface became larger for ICS [21]. Also, with the development of search engines that index both ICS and IoT devices, these devices have become easier to find over the internet. Examples of these are Shodan [38] and Censys [7], in which a protocol name can be entered, which results in all indexed devices that are accessible through the internet and recognised to use that protocol.

The rest of this section, first, describes the security attributes as they apply to ICS. Then, security weaknesses in ICS are addressed and, finally, an example attack scenario is described.

2.3.1 Security attributes

ICS face different security challenges than regular IT. Table 2.1 shows that the order of importance of security attributes in ICS is the opposite of the order in regular IT. Where in securing regular IT confidentiality is most important, in securing ICS, availability is most important [47]. The reason for this is that most ICS controlled processes are real time

processes that depend on immediate feedback. When these processes are not available, this could result in significant (financial) loss for the operator. On the contrary, confidentiality generally is less important in ICS, as the messages sent in those are relatively predictable and not containing much private information.

This difference is important, as this means that it would be a much bigger issue if an attacker would be able to shut down or disrupt a protocol in an ICS, than it would be for IT systems. Therefore, it is essential that the systems and protocols do not get disturbed.

Table 2.1: Order of importance of security attributes per sector

IT security
1. confidentiality
2. integrity
3. availability

ICS security
1. availability
2. integrity
3. confidentiality

2.3.2 Security weaknesses

The security weaknesses found in ICS are caused either by lack of sufficient security mechanisms or they origin in design [25]. The first ones often are known but not implemented either due to the costs of deploying security measures, or because of difficulties in applying them in legacy systems [28]. The second ones can have different causes, like weaknesses in the way a network is built or that there are security mechanisms available but not used. Even more particularly, these design vulnerabilities can origin in incorrect implementations of ICS specific protocols.

Most communication between ICS devices follow ICS specific protocols [35]. As seen in the last section, the availability of these protocols is crucial for the functioning of the ICS. That means that if there is a vulnerability in the implementation of such a protocol in a device in the ICS, this is a vulnerability for the entire ICS. Therefore, attacks on ICS protocols should be considered a significant threat to ICS. In the rest of this section several examples are given to show that attacks on ICS protocols are possible in practice. After that, the ICS protocol IEC 60870-5-104 is introduced in section 2.4 and examined in the rest of this research.

The most extreme objective of attacks on ICS can be to cause physical damage. An example proving that this is possible by manipulating an ICS protocol is the Aurora generator test that was described in the Introduction.

Most of the protocols used in ICS, such as also IEC-104, do not contain any form of cryptographic protection [35]. Also, most ICS protocols use weak or no authentication and integrity checks. This enables conducting several types of attacks on these protocols. Rrushi [35] describes three of these types of attacks:

man-in-the-middle attack where the attacker obtains access to the network and can intercept all messages between two nodes in the network;

malicious machine code upload attack where the attacker is able to use the ICS protocol to upload malicious machine code to a process control system;

denial of services attack where the attacker is able to disrupt connections to specific devices in the network.

For example, Kang et al. present an attack on an ICS protocol that is caused by the lack of authentication [23]. They describe a man-in-the-middle attack on the ICS protocol IEC 61850, which is also mainly used in power grids. In order to perform this attack they use the Address Resolution Protocol (ARP) that is used to resolve IP addresses into MAC addresses. By spoofing the table that is used to resolve these addresses, they are able to route all traffic in the ICS protocol via an attacker. Kang et al. use this ARP spoofing to man-in-the-middle IEC 61850's communication service Manufacturing Message Specification (MMS). Due to the lack of authentication, the attackers are able to intercept the sent MMS messages and alter them. As IEC-104 also misses authentication, it would also be possible to conduct a similar attack on IEC-104.

Furthermore, Zhu et al. [47] describe several attacks on the implementations of protocols used in ICS. For example, they describe several denial of service attacks that exploit vulnerabilities in the implementation of TCP/IP in Windows systems that were not patched. Also, they describe a vulnerability in an implementation of MMS, i.e. it does not handle all malformed packets correctly, which makes a remote denial of service attack possible.

More generally, for every protocols that runs on top of TCP/IP, the absence of adequate authentication makes it possible to spoof TCP packets and set wrong flags [35]. This could be used by an attacker to perform a denial of service attack, as wrong flags could cause a TCP connection to terminate.

As these examples show, various research has been performed that examine vulnerabilities in TCP. Therefore, these vulnerabilities are not further investigated in this research. This research is scoped to an ICS protocol on top of TCP, i.e. IEC-104, which is described in more detail in section 2.4.

2.3.3 Example attack scenario

Before the details on IEC-104 are given, this section first provides an example of an attack scenario that could be performed with the ICS security weaknesses that are investigated in this research. This attack scenario consists of the following steps:

1. Achieve (one-time) access to the control network (for example with an infected USB-stick).
2. Send messages to multiple RTUs to set unsafe conditions.
3. Make substations unreachable by sending a specific stream of messages that exploits the incorrect implementation.

When these steps are executed, the processes on multiple sites are in unsafe conditions and an operator would not be able to resolve that remotely. This could have serious consequences if the operator is not able to reach each of the sites before accidents occur.

These first two steps are not in the scope of the research in this thesis, as the first step can be performed in multiple different ways and the second step requires regular use of the protocol. Therefore, the focus of this research is on the third step of this example attack scenario.

2.4 IEC 60870-5-104

As stated in the Introduction, this research focuses on IEC 60870-5-104 [42]. This is one of the protocols in the IEC 60870 collection of standards that was developed by Technical Committee 57 (TC57) of the International Electrotechnical Commission (IEC) [28]. The standards in IEC 60870 were developed between 1988 and 2000 for the transmission of SCADA telemetry control data. These standards describe the protocols that are currently used in the European electrical industry [9]. Next to that, they are also used by some water and gas industries in Europe.

As Figure 2.4 shows, both IEC 60870-5-101 and -104 are used for the communication between the control centre and the substations. IEC-104 is the TCP/IP version of the older serial protocol, IEC 60870-5-101 (IEC-101) [30]. IEC-101 defines all functionality and data objects that are necessary for telecontrol applications over wide areas, such as communication between electrical control station and substation systems [9].

According to Maynard and Mclaughlin [28], most research on vulnerabilities in ICS protocols is not conducted on IEC-104, but on other protocols like DNP3 (i.e. a standard

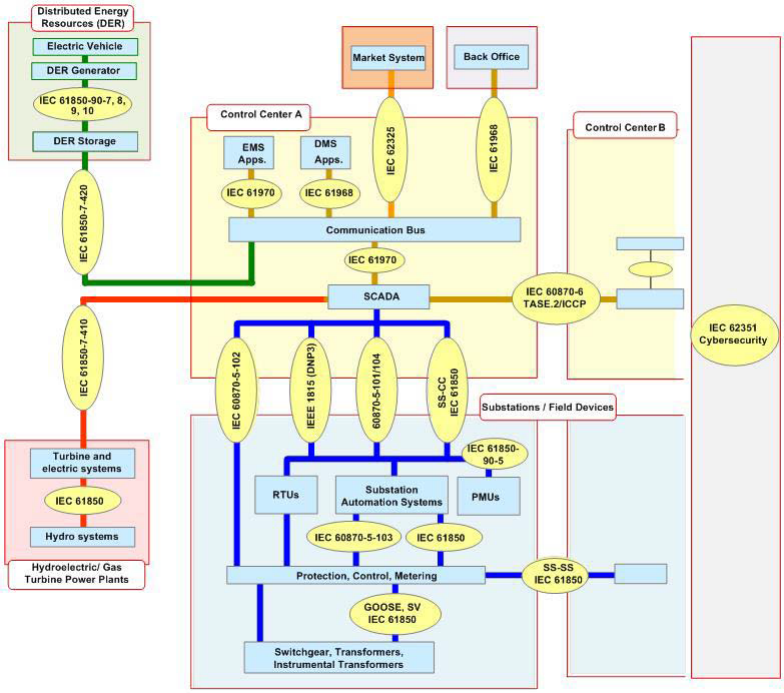


FIGURE 2.4: Architecture of IEC TC57 Information Exchange Standards [10]

that is derived from IEC-104 during its development and is used mainly in North-America). Therefore, more research has to be performed on vulnerabilities in IEC-104.

Technical specification IEC TS 60870-5-7 [43] defines an implementation of IEC 62351 to provide security and authentication to IEC-104. The purpose of this technical specification is to make it possible to verify that IEC-104 messages were sent by an authorised user and that they were not modified by a third party. However, due to implementation costs, this technical specification is not implemented in many systems.

2.4.1 Structure

IEC-104 describes two different layers that can be placed in the application layer (layer 7) of the OSI-model [42]. As shown in Table 2.2, the bottom one is the Application Protocol Control Information (APCI) layer. The APCI layer lays on top of the TCP layer and can has three different of message formats: Unnumbered control functions (U-format), numbered Supervisory functions (S-format) and the Information transfer format (I-format). On top of the APCI layer is the Application Service Data Unit (ASDU) layer. This layer is derived from the message structure defined in IEC-101 and it is only used in I-format messages. The purpose and structure of these two layers is examined next.

Table 2.2: OSI layers

7	Application	ASDU
		APCI
6	Presentation	
5	Session	
4	Transport	TCP
3	Network	IP
2	Link	Ethernet
1	Physical	

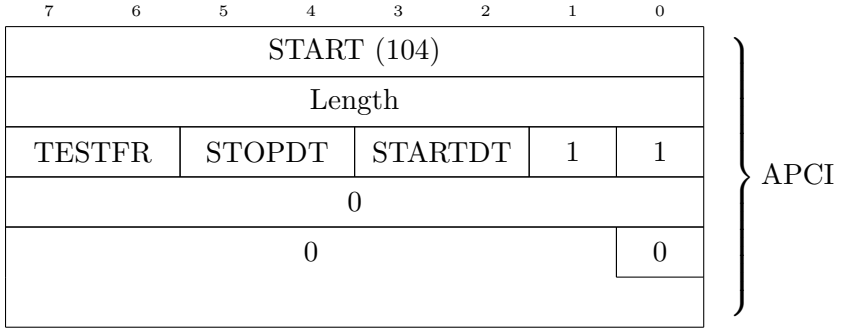
As can be seen in Figure 2.5, the first two bytes of each of the three IEC-104 message formats have the same purpose. The first byte indicates that it is an IEC-104 message and this byte always the decimal number 104. The second byte indicates the length of the rest of the message, i.e. the length of the message without the first two bytes. For U-format and S-format messages, this length should always be 4.

U-format messages (Figure 2.5a) either describe activation or confirmation and have three types: STARTDT, STOPDT and TESTFR. The STARTDT (start data transfer) type is to set up a connection, the STOPDT (stop data transfer) to tear down the connection and the TESTFR (test frame) is to test if the connection is still active.

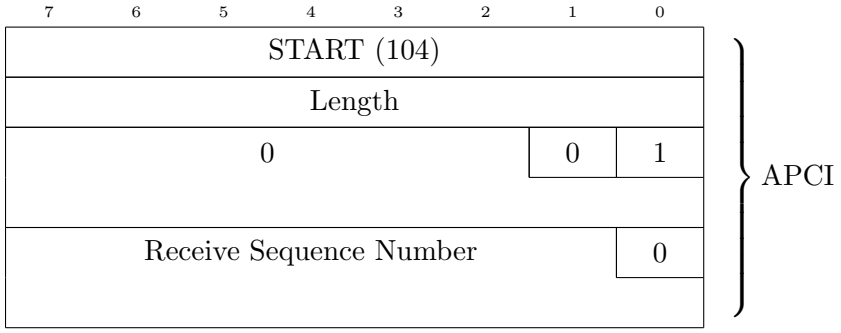
S-format messages (Figure 2.5b) are used to confirm up to which I-format message is received.

I-format messages (Figure 2.5c) are the messages that are used to send actual data. They use TypeIDs to define what kind of message is sent. These are numbers that range

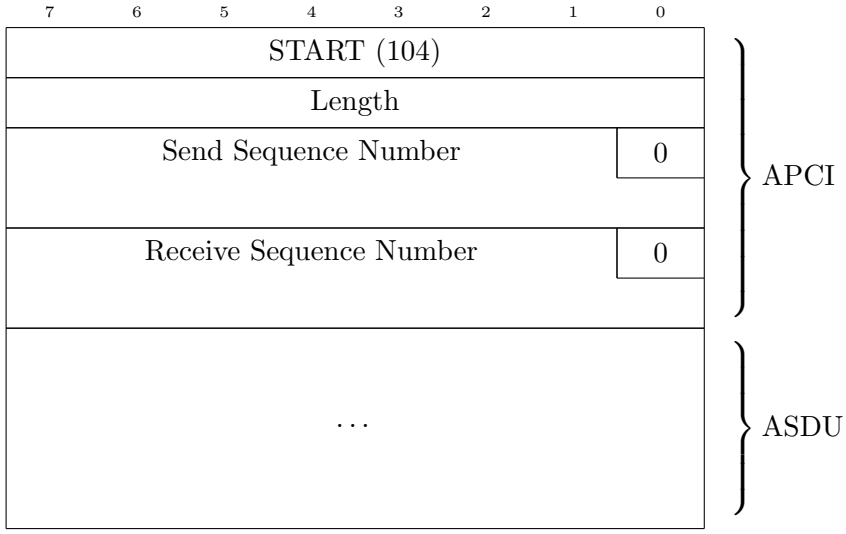
from 0 to 255, where TypeID 0 is reserved not to be used. The TypeIDs ranging between 1 and 127 mostly originate from the IEC 60870-5-101 standard, except for several changes and additions [41, 42]. This TypeID range from 1 to 127 is divided further into the groups that are shown in Table 2.3. The TypeIDs in the range from 128 up to 255 are not defined in the standard and are there to be used by a vendor to define its own custom private TypeIDs. Of this private range, the TypeIDs 128 up to 135 are reserved for routing of messages.



(A) Unnumbered control functions (U-format)



(B) Numbered Supervisory functions (S-format)



(C) Information transfer format (I-format)

FIGURE 2.5: Application Protocol Data Unit (APDU) formats

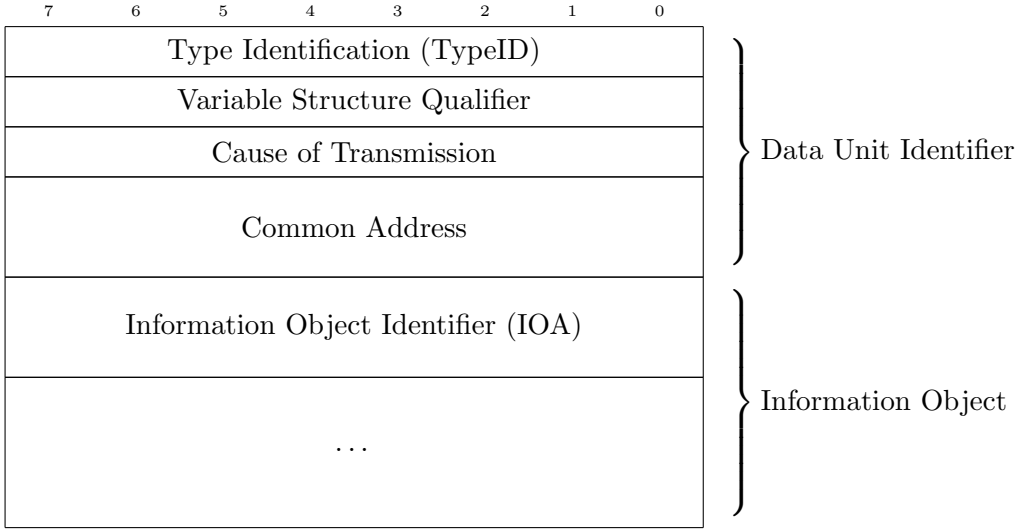


FIGURE 2.6: Application Service Data Unit (ASDU) structure

TABLE 2.3: ASDU types defined in IEC 60870-5-104

TypeID	
0..44	Process information in monitoring direction (from slave to master)
45..69	Process information in control direction (from master to slave)
70..99	System information in monitor direction
100..109	System information in control direction
110..119	Parameter in control direction
120..127	File transfer

Not all TypeIDs in all ranges are used. When there are TypeIDs missing they are reserved for new compatible definitions in that range. An example of a type of message from the first range is TypeID 1, which is also represented with the tag M_SP_NA_1 and is used to send single point formatted data in monitoring direction (from RTU to controlling station). An example of a message that is sent in control direction is the message with TypeID 100 (C_IC_NA_1). This is the general interrogation command that can be used to request information from all controlled stations. The full specification of the rest of these TypeIDs can be found in Appendix A.

Not every device that implements IEC-104 implements all of these different TypeIDs. As specified in the standard, every device should have an interoperability list, that specifies which ASDU types are supported, combined with which Causes of Transmission can be used for each type [42].

2.4.2 State transition diagram

Figure 2.7 shows the state transition diagram for controlled stations, as it is presented in the IEC-104 standard. This diagram shows that in the initial (STOPPED) state, the device should only respond to U-frames and terminate the connection if any other type of message is received. Also, it shows that there is an UNCONFIRMED STOPPED state that is reached when a STOPDT command is send while not all I-frames are confirmed by an S-frame. Otherwise, if all I-frames are confirmed, a STOPDT should directly receive a STOPDT confirmation and enter the STOPPED state.

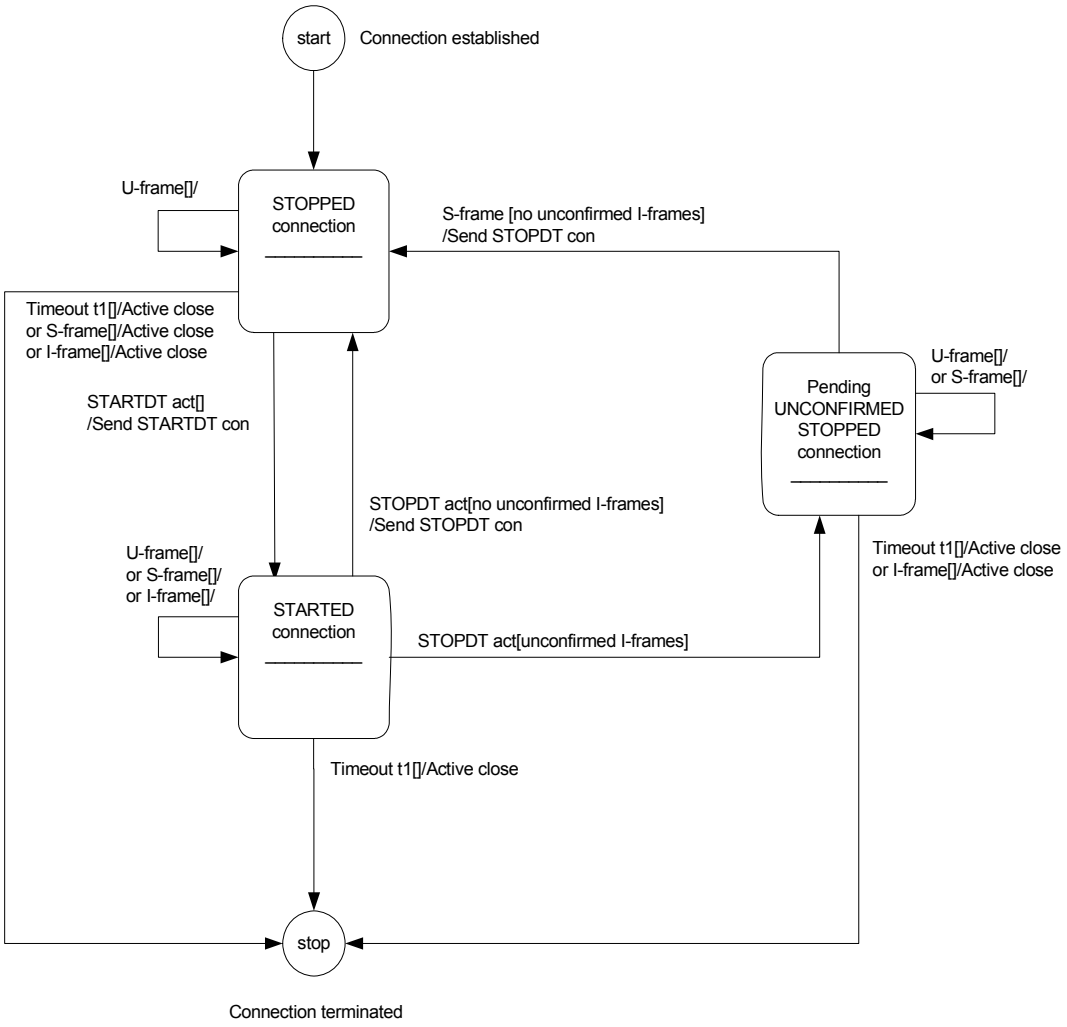


FIGURE 2.7: State transition diagram for Start/Stop procedure (controlled station) [42]

2.4.3 IEC-104 parameters

In IEC-104 several parameters are used, that have to be defined for each device. The TCP port number should be 2404 in all cases. In Table 2.4 the other parameters are explained and their default values are given.

Table 2.4: IEC-104 parameters and default values [42]

k	Maximum number of unacknowledged I-format APDUs before transmitter terminates	(default: 12 APDUs)
w	Maximum number of received I-format APDUs before receiver must acknowledge	(default: 8 APDUs)
t_0	Time-out of connection establishment	(default: 30s)
t_1	Time-out of receiving response to latest sent I-format/U-format APDUs	(default: 15s)
t_2	Time-out for sending acknowledgement S-format message ($t_2 < t_1$)	(default: 10s)
t_3	Time-out for sending TESTFR after not receiving any APDUs	(default: 20s)

2.5 Conclusion

This chapter described that ICS consist of a control network, a field network and a communication link. Each of these networks consists of different components. The historical development is described and shows the development of air-gapped ICS to networked ICS that are in use nowadays. These networked ICS are better controllable, but also introduce security issues. One of the reasons for that is that the protocols used in ICS are not secure by design. Finally, the end of this chapter specifies IEC 60870-5-104, which is an ICS protocol that is not secure by design.

Chapter 3

Representing protocols

This chapter starts in section 3.1 with an explanation regarding what is meant with the implementation of a protocol. Next, methods to formally describe protocols are given in section 3.2, after which in section 3.3 an elaboration is given on how automata, i.e. these formal descriptions, are learnt. In section 3.4 the algorithm to learn these automata is given. Section 3.5 gives a description on how to compare these automata with each other, after which section 3.6 this chapter concludes.

3.1 Protocol implementation

In short, an implementation is the interpretation of a specification that is presented in a standard for a particular protocol. When a specification is unclear, e.g., it consists only of a long ambiguous textual description [34], it might be interpreted differently then intended during the implementation of a protocol. This causes differences in interpretation of the specification which results in differences in the implementations of this specification.

For example, suppose a specification defines a protocol structure as shown in Figure 3.1 and specifies that: "If flag F is set in the packet, field X contains the value of parameter x." This specification insufficiently defines what should happen if flag F is not set. Figure 3.2 shows three different possible implementations that follow this specification. The implementation in Figure 3.2a sets X to zero when flag F is not set. In the implementation in Figure 3.2b it is undefined what should happen with X. In the implementation in Figure 3.2c the entire field X is skipped and the next field starts where X would start if the flag is set. All these three cases comply to the given specification although their behaviour is rather different. Therefore, this is a case of underspecification, which could cause problems if two devices are implemented differently.

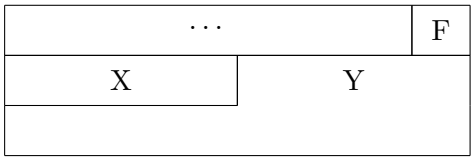
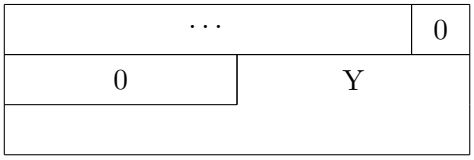
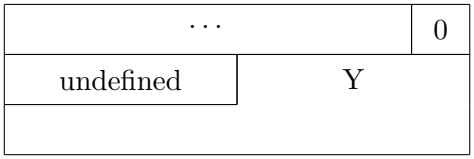


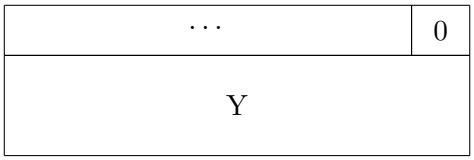
FIGURE 3.1: Example protocol



(A) Field X is zero when F=0



(B) Field X is undefined when F=0



(C) Field X is skipped when F=0

FIGURE 3.2: Possible implementations

The implementation of a protocol should be able to handle every input in every state. If there are states in a protocol where some input cannot be handled, this could potentially result either in incompatibilities between different implementations or even worse, in security flaws [34]. In such a case, it might be possible to let an implementation crash by sending a specially crafted input when the implementation is in a certain state of the protocol. This could, e.g., be used in a denial of service attack. Van den Broek et al. show that this was possible in GSM [45]. By sending specially crafted messages, they were able to crash a mobile phone.

3.2 Formal protocol description

As described in the previous section, protocols describe which messages can be exchanged and which rules have to be followed. An implementation should be able to handle all of these messages in each state. Therefore, the messages of these protocols can be regarded as symbols in the alphabet of all possible messages.

Gunawan et al. describe three types of models to formally describe protocols: the State Transition Model, the Programming Language Model and the Hybrid Model [16]. State Transition Models describe protocols as a finite graph where the protocol's events are described as inputs and outputs between nodes. Programming Language Models describe protocols in a high level programming language notation. Hybrid Models combine both State Transition Models and Programming Language Models to describe protocols.

State Transition Models are the simplest models and are best understandable for humans. Therefore, this study elaborates on the most widely accepted State Transition Model: the finite state machine (FSM), which is also known as the finite automaton. Henceforward, this study refers to these FSMs using the term automaton.

Automata provide a method to systematically model the implementation of protocols. An automaton can be inferred from an implementation using a learning algorithm as described in section 3.3. These automata can be used to compare the implementations to explore if there are differences between them. These differences could, e.g., indicate incompatibilities or possibly even a security flaw in one of the implementations. For example, in the research of Fiterau-Brostean et al. [13] differences are found between the automata of TCP implementations of different operating systems. As a result of these differences in implementation, irregular situations exist in which the behaviour of TCP differs per operating system, making it possible to fingerprint the operating system.

Similarly, Verleg [46] discovered differences between different implementations of the SSH protocol. He discovered that in a situation where the standard does not specify what to do, different implementations act differently.

Generally, there are two different types of automata in the form of FSMs: i) those that either accept or reject an input and ii) those that return an output depending on the input. From the first type, deterministic finite automata are described in section 3.2.1. For the second type, Mealy machines are described in section 3.2.2.

3.2.1 Deterministic finite automaton

A simple automaton is the deterministic finite automaton (DFA). A DFA consists of states and actions [31]. By performing an action in a certain state, you are able to reach a new state. Furthermore, it has one starting state and one or more accepting states. In Figure 3.3 an example DFA is shown. The starting state of this DFA is s_0 and the only accepting state also is s_0.

DFAs can be used to describe which sequences of actions are accepted and which are rejected. When a specific sequence of actions starting from the starting state transitions the DFA into an accepting state, that means that that sequence of actions is accepted. Otherwise, if the DFA is not in an accepting state after the sequence, that means that the sequence is rejected by the DFA.

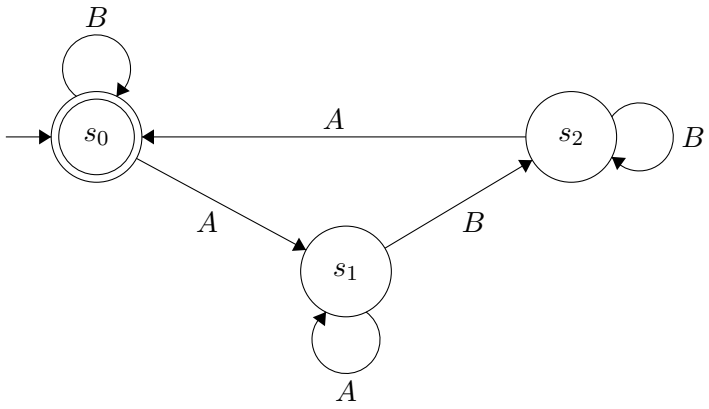


FIGURE 3.3: Deterministic finite automaton

3.2.2 Mealy machines

To represent the implementations, the research by Verleg [46] is used as example on how to represent the automata in the form of Mealy machines. A Mealy machine is a comprehensible automaton consisting of states and transitions. A combination of both the current state and the input determine the next state. Except for that, this combination also determines the output. This results in each transition containing both an input and an output.

In contrast to DFAs, Mealy machines do not only describe whether a sequence of inputs is accepted or rejected, but is also able to describe more of the behaviour of the system in terms of output.

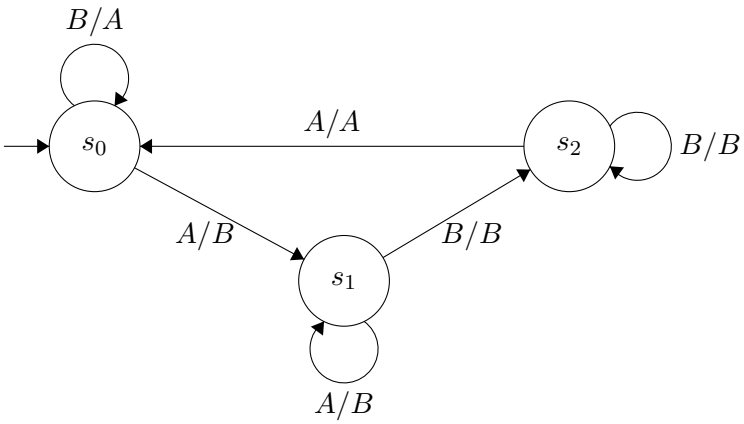


FIGURE 3.4: Mealy machine

In Figure 3.4 a Mealy machine is graphically represented. The circles represent the states and the arrows represent transitions. As explained each transition represents both an input and an output, therefore also the arrow represents these two in the form of *input/output*. For example, if in Figure 3.4 in state s_0 the input A is received, the output B is returned and the state becomes s_1.

3.2.3 Protocol inputs (alphabet)

As the messages in a protocol contain many different values per field, the total number of possible messages is rather large. Therefore, it is important to specify the granularity/extensiveness of the alphabet to be able to contain the time complexity for learning

within feasible boundaries.

The complexity of this alphabet is determined by the level of detail of the alphabet. Generally, the more extensive the alphabet is, the more complex the learnt automaton becomes.

3.3 Automata learning

The method used to infer automata is similar to the approaches used by Caldwall et al. [6] and Verleg [46]. In these approaches the automaton learning process requires three components, as seen in Figure 3.5. These components are a learner, a teacher and an intermediate layer (or mapper). The explanation of these terms can be found below.

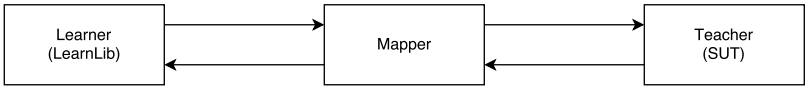


FIGURE 3.5: Automata learning setup

Learner tool that implements a learning algorithm.

Teacher subject under test (SUT), e.g., the implementation of a protocol, of which the automaton is learnt.

Mapper abstraction layer used to abstract from the specific content and reduce the so-called alphabet (explained in section 3.4) of all possible requests. This layer determines the level of abstraction and therefore of detail of the alphabet. The mapper is constructed using extra information from informal specifications or observing the behaviour of the SUT [2].

The learner learns the SUT's automaton by querying the teacher and parsing its responses. Figure 3.5 also shows that these queries and responses pass through the mapper. This mapper transforms the abstract queries from the learner to concrete queries that the teacher can understand. Vice versa, the mapper also transforms the concrete responses from the teacher to abstract responses, so that these can be used by the mapper. Section 3.4 introduces the algorithm that is used for learning the automata. Subsequently, Chapter

4 (Methodology) describes the tool that actually applies this algorithm and produces these automata.

3.4 Algorithm

As a learning algorithm to learn automata, Angluin's $L^\star$ algorithm [3] can be used. The original version of this $L^\star$ algorithm produces DFAs by randomly sending different sequences of inputs from the alphabet and then observing if these are accepted. If a specific input, that causes all subsequent inputs to change their accepting behaviour, is found, a state change is inferred. By repeating this procedure, a DFA is inferred.

However, this original $L^\star$ algorithm only produce DFAs that describe whether inputs are accepted or not. To be able to produce Mealy machines that can represent the more complex behaviour of input/output systems, the algorithm needs some adjustments. Shahbaz and Groz describe these adjustments and show how the $L^\star$ algorithm can be altered to produce Mealy machines [37]. The result of this is the $L^\star_M$ algorithm (Algorithm 1).

Instead of observing whether an input is accepted, this altered $L^\star_M$ algorithm observes the output that is returned for each input [8]. If, then, an input is found that causes all subsequent inputs to produce different outputs, the algorithm infers that a state change occurred. By repeating this procedure, not a hypothesis for a DFA, but a hypothesis for a Mealy machine is inferred.

Before looking in greater detail to the learning process in the $L^\star_M$ algorithm, some terminology is defined:

distinguishing sequence input sequence that produces different outputs for different states

access sequence input sequence to reach a specific state from the initial state

counterexample input sequence that results in different outputs for the teacher and the hypothesis

When looking in greater detail to the $L^\star_M$ algorithm, the learning consists of first creating a hypothesis and then iteratively refining this hypothesis in a process that consists of two steps: the membership step and the equivalence step.

In the membership step the learner chooses an input sequence and sends it to the teacher to obtain outputs. The learner tries to discover different states by finding distinguishing sequences. If a new state is found, its access sequence is stored. When every input in every

known state results in a transition to another known state, the model is closed. When every state has exactly one transition for every input, the model is consistent. When the model is both closed and consistent, a hypothesis is generated and this hypothesis is passed to the equivalence step.

In the equivalence step, the learner sends equivalence queries to the teacher to check if the the hypothesis is equal to the automaton of the teacher. If these are not equal it means that a counterexample is found. Then, the membership step is repeated with the counterexample used as an additional distinguishing sequence. When no more counterexamples are found in the equivalence step, the hypothesis is returned as final and the learning has finished.

Algorithm 1 shows the pseudo-code of the complete $L_M^\star$ algorithm. Here, the Input Alphabet A_I is the alphabet as described earlier. The Output Alphabet A_O consists of all possible outputs. The hypothesis is structured as the so-called Observation Table ($\mathcal{OT}$). This $\mathcal{OT}$ can be represented as a 3-tuple (S, E, T), that consists of a set of prefixes (S), a set of suffixes (E) and a finite function (T) that maps combinations of prefixes and suffixes to an output.

The $\mathcal{OT}$ is called *closed* when adding any symbol of the alphabet has an equivalent output to one of the rows that already exists. The $\mathcal{OT}$ is called *consistent* if for all prefixes that have the same output row, the outputs of that prefixes followed by the same letter of the alphabet and suffix are also the same.

If the Observation Table is both closed and consistent, the membership step ends and the equivalence step starts to investigate if a counterexample can be found.

Algorithm 1 $L^\star_M$: learning algorithm $L^\star$ for Mealy machines [3, 31, 37, 17]

Input Alphabet A_I
Output Alphabet A_O
Observation Table $\mathcal{OT} = (S, E, T)$, where initially $S = \{\epsilon\}$ and $E = A_I$

repeat
  $\mathcal{OT} \leftarrow$ UPDATE($\mathcal{OT}$)
  while $(\neg isClosed(\mathcal{OT}) \vee \neg isConsistent(\mathcal{OT}))$ **do**
    if $\neg isClosed(\mathcal{OT})$ **then** ▷ $\exists s_1 \in S, a \in A_I . \forall s \in S . row(s_1 \cdot a) \neq row(s)$
      $S \leftarrow S \cup \{s_1 \cdot a\}$
      $\mathcal{OT} \leftarrow$ UPDATE($\mathcal{OT}$)
    end if
    if $\neg isConsistent(\mathcal{OT})$ **then** ▷ $\exists s_1, s_2 \in S, a \in A_I, e \in E . row(s_1) = row(s_2) \wedge T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$
      $E \leftarrow E \cup \{a \cdot e\}$
      $\mathcal{OT} \leftarrow$ UPDATE($\mathcal{OT}$)
    end if
  end while
  $M_{conj} \leftarrow M(\mathcal{OT})$ ▷ conjecture Mealy machine after the first phase of this round
  $\sigma_{conj} \leftarrow EO(M_{conj})$ ▷ EO denotes a call to the Equivalence Oracle
  if $\sigma_{conj} \neq \perp$ **then**
    $S \leftarrow S \cup Prefix(\sigma_{conj})$ ▷ $Prefix(\sigma)$ generates the set of all possible prefixes of σ including σ itself
  end if
until $\sigma_{conj} = \perp$ ▷ until no counterexample is found

function UPDATE($\mathcal{OT}$)
  for $(s \in (S \cup S \cdot A_I), e \in E)$ **do**
    $T(s \cdot e) \leftarrow MO(s \cdot e)$ ▷ MO denotes a call to the Membership Oracle
  end for
  return (S, E, T) ▷ updated $\mathcal{OT}$
end function

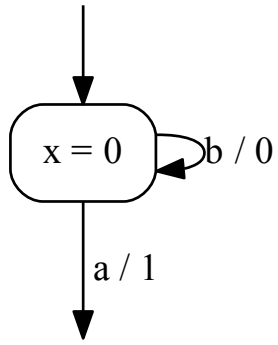
function M($\mathcal{OT}$)
  $Q \leftarrow \{row(s) | s \in S\}$
  for $(s \in S, a \in A_I)$ **do**
    $\delta(row(s), a) \leftarrow row(s \cdot a)$
    $\lambda(row(s), a) \leftarrow T(s \cdot a)$
  end for
  $q_0 = row(\epsilon)$
  return $(Q, A_I, A_O, \delta, \lambda, q_0)$ ▷ Mealy machine
end function

3.4.1 Example

As an example of how the algorithm runs, this section describes a scenario that shows how Mealy machines are generated. Given a SUT that holds a variable x and has an input alphabet a, b and an output alphabet $0, 1$. Initially this value $x = 0$. When the SUT receives an input a the value of x is XORed with 1 and the result is returned as output. When the SUT receives a b the current value of x is returned as output. For example, when first an a is sent, the SUT returns a 1. If the a is followed by a b this results in the SUT again returning a 1. When subsequently another a is sent, this now causes a 0 to be returned.

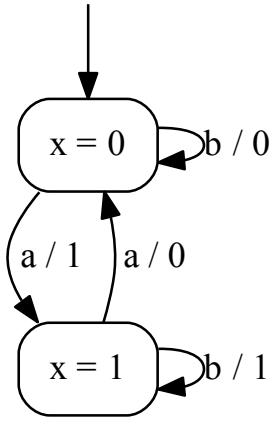
Figure 3.6 shows the $\mathcal{OT}$s and their corresponding automata during the learning process. The left side of the $\mathcal{OT}$ shows the prefixes, i.e. the messages that are sent first, and the top side of the $\mathcal{OT}$ shows the suffixes, i.e. the last messages that are sent. Figure 3.6a shows the initial $\mathcal{OT}$ after the initial update. This $\mathcal{OT}$ is not closed (as can also be seen in the automaton), because there is a row in the bottom part of the table that does not occur in the top part of the table (the row with prefix a). Therefore, the algorithm adds this row to the top part of the table (so that $S = \{\epsilon, a\}$) and updates the table again. This update makes the table both consistent and closed, and because the equivalence oracle does not find a counterexample, this is the final $\mathcal{OT}$ as shown in Figure 3.6b.

		E	
		a	b
S	ϵ	1	0
$S \cdot A_I$	a	0	1
	b	1	0



(A) First $\mathcal{OT}$ and Mealy machine

		E	
		a	b
S	ϵ	1	0
	a	0	1
$S \cdot A_I$	b	1	0
	aa	1	0
	ab	0	1



(B) Final $\mathcal{OT}$ and Mealy machine

FIGURE 3.6: Observation Tables and automata learnt from Example

3.5 Automata comparison

When this methodology is used to formally describe multiple implementations as automata, it is possible to compare these automata. Moreover, it could be valuable to compare a derived automaton directly to the specification. To be able to compare an automaton to an automaton that is produced using $L^\star$ or derived from the specification, a method of comparison is needed. Possible methods of comparison are described next.

To compare automata the notion of automaton equivalence can be used. Two automata are called equivalent when they recognise the same language [36]. For DFAs, this can

easily be understood, as they are equivalent when the same sets of inputs lead to either an accepting state or not. For other automata, like Mealy machines, this means, that two automata with the same input and output alphabet, are equivalent if they produce the same output for each input, when starting from the same initial state. Therefore, it is possible to check if two automata are equivalent by traversing over both automata in parallel, while checking if the same input produces the same output. In order to show that two machines are not equivalent, a sequence of inputs needs to be found, such that this produces different outputs for the two machines. If such an output exists in some state, the automata are not equivalent. Otherwise, if such a difference cannot be found after traversing over the entire automaton, this means that the automata are equivalent.

Aarts et al. [1] describe an alternative method which uses conformance learning model-based testing. This is a method to compare two models that were learnt using automata learning. In this method they use the CADP (Construction and Analysis of Distributed Processes) toolset, which is a toolset that can check the equivalence of two Mealy machines.

Another possibility for the comparison of an automaton to a specification, is by using Holzmann's SPIN [18]. SPIN is an automatic model checker that is able to automatically check an automaton against a specification when that specification is modelled using SPIN. Also, the inferred automata can be compared with existing automata that originate from the specification. For example, F. Qin-cui et al. describe how to model the protocol specification of IEC-104 as an automaton [15].

3.6 Conclusion

In this chapter, methods to formally describe protocols as automata were described. From these methods, the $L^\star_M$ algorithm for learning Mealy machines from implementations was examined more in depth. In the end of this chapter, methods of comparing these produced automata were described.

Chapter 4

Methodology

The following chapter describes the design of the set-up. Beside the earlier described teacher, learner and mapper that are part of the set-up, this chapter introduces a fourth component to the set-up: the checker. The set-up with these four components is graphically shown in Figure 4.1 and is described in section 4.1, after which the separate components are described in depth in sections 4.2 to 4.5.

4.1 Set-up

The learner implements the $L^\star_M$ algorithm as described in section 3.4. As shown in Figure 4.1, the learner consists of several components: (i) the experiment, (ii) the membership oracle + cache and (iii) the equivalence oracle. The experiment uses the membership oracle to learn a hypothesis (which is structured as an Observation Table). During learning the membership oracle sends abstract membership queries to the mapper (as presented in Figure 4.1 as steps 2 and 3), from which it in return receives abstract membership answers (steps 8 and 9). These queries and answers are called abstract because they are human readable and they make up the input and output alphabet of the Mealy machine. There is a cache between the membership oracle and the mapper, which stores each membership query and its corresponding answer. When a query is passed to this cache it will first attempt to look it up. Only if the answer is not found in the cache, it will be passed to the mapper. The membership oracle will continue sending queries until it has a closed and consistent Observation Table. Then a hypothesis Mealy machine is produced from the Observation Table.

When the experiment finds a hypothesis, this hypothesis is checked using the equivalence oracle. This oracle will attempt to find a counterexample to the hypothesis. In order to do that, the oracle first checks if there are not any inconsistencies with the cache

```
1       :     Learn / Refine Hypothesis
2, 3    :     Abstract Membership Query
4, 5    :     Concrete Membership Query
6, 7    :     Concrete Membership Answer
8, 9    :     Abstract Membership Answer
10,11   :     Hypothesis
12      :     Abstract Equivalence Query
13,14   :     Concrete Equivalence Query
15,16   :     Concrete Equivalence Answer
17      :     Abstract Equivalence Answer
18      :     Confirmation / Counterexample
19      :     Final Hypothesis
```

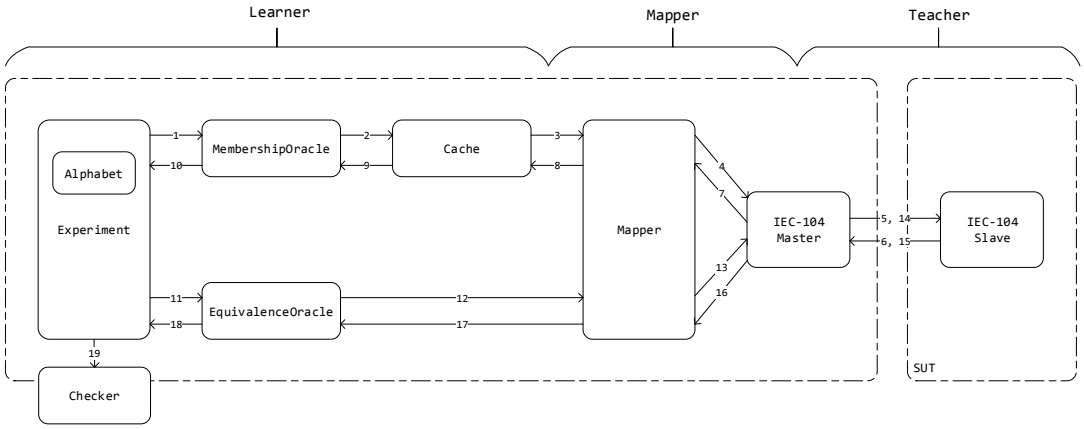


FIGURE 4.1: Test Set-up

and then starts to send 1000 random queries to the mapper, while checking if it matches the hypothesis. While sending these queries, there is a 1% change that the SUT and the hypothesis are reset to the initial starting position. When one of these queries contradicts with the hypothesis, this query is called a counterexample. If no counterexample is found after the 1000 random queries, the hypothesis is confirmed.

As described, the learner sends queries to the mapper. The main goal of the mapper is to transform the human readable abstract queries to concrete queries that comply to the protocol. Conversely, it also transforms the concrete answers back into more simple and human understandable abstract answers that can be used in the Mealy machine.

The teacher is the SUT, which is an implementation of an IEC-104 controlled station (slave). To manage the connection with this slave and to really send the messages to it, the mapper sends its messages to an IEC-104 controlling station (master) that was implemented. This master then takes these messages, sends them to the SUT and waits for the SUT to respond. If the SUT does not respond within a specified interval, the master returns to the mapper that there was no answer. This interval can be configured.

Finally, the final hypothesis is compared to the standard by the checker. In order to do this, the set-up contains a automaton for the standard, with which the produced automaton is compared.

4.2 Learner

To learn the automata of the IEC-104 implementations a framework named LearnLib [27] was used. This framework was used to implement both the learner and the mapper as described in the previous section. In the following section this framework is described in more depth.

4.2.1 LearnLib

LearnLib is a Java library that contains a framework to implement both the classical $L^\star$ and the $L^\star_M$ algorithm, as explained in section 3.4. It can be used to determine which input from the alphabet to send to the SUT. Then it can use the resulting output to learn the states of the automaton and determine which query to send next [46]. Then it will produce a Mealy machine that can be compared with the Mealy machine that was deduced from the specification.

4.2.2 Design of the Alphabet

The alphabet is chosen so that it implements all the different message format types from the IEC-104 standard as described in section 2.4. Of these, all three different U-format types are part of the alphabet and for all of the different categories of I-format messages there is one in the alphabet. The complete alphabet is shown in Table 4.1.

Table 4.1: Alphabet

Word	**Representation**
U[STARTDT] U[STOPDT] U[TESTFR]	U-format message for starting or stopping data transfer or for testing the connection, as described in section 2.4
S	S-format message as described in section 2.4
I[0]	Artificial I-format message with undefined TypeID 0
I[C_SC_NA] I[C_DC_NA] ⋮ I[F_DR_TA] I[F_SC_NB]	All specified I-format ASDU command types are in the alphabet. A complete list of these can be found in Appendix A
L_{max}	Artificial message that only contains the START byte (104) and the length field set to the maximum om 253 without actually containing that length

Because of the limitations for learning within feasible boundaries as described in section 3.2.3, not the entire alphabet is used in every run, but a sub-alphabet is constructed out of this alphabet. Automata are created with these sub-alphabets.

Every I-format messages uses a Cause of Transmission that is allowed according to the definition in the standard. When it is allowed, this Cause of Transmission is Activation. Otherwise, it is the most common Cause of Transmission for that TypeID.

4.3 Mapper

As described in section 3.3, the mapper is an abstraction layer that translates between abstract messages on the learner side and concrete messages on teacher side and vice versa. For every abstract message in the alphabet, the mapper contains an implementation of a

concrete message. These concrete messages are structured according to the formats as defined in the IEC-104 specification (except for the abstract messages I[0] and L_{max}). For example, the abstract message I[C_SC_NA] is translated to a concrete message that is shown in Figure 4.2. In this Figure the I-format APCI header from Figure 2.5c and the ASDU from Figure 2.6 are combined to form the I[C_SC_NA] APDU. To implement these concrete messages OpenMUC j60870 [32] is used.

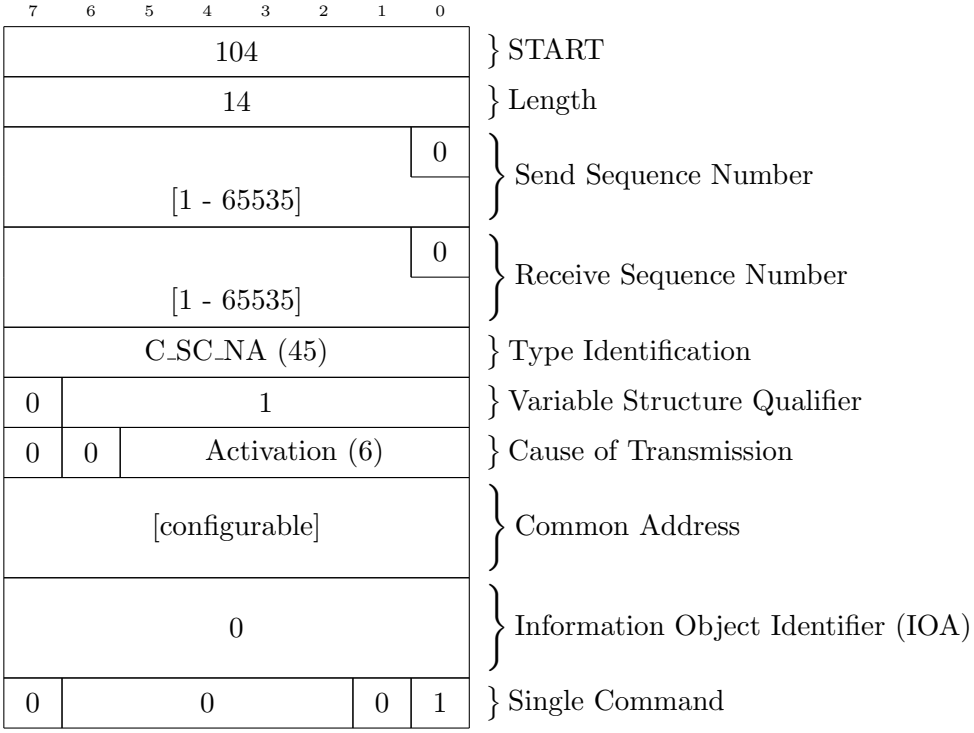


FIGURE 4.2: Example of concrete mapping from I[C_SC_NA]

For every concrete message that the mapper receives from the teacher, the mapper contains a translation to an abstract message. Such an abstract message contains the message format (U, S or I); when it is U-format, it also contains whether it is STARTDT, STOPDT or TESTFR; when it is I-format it also contains the ASDU type of the message. When an error occurs, e.g., the connection is lost, the mapper translates this to the abstract message *ERROR*.

4.4 Teacher

To communicate with the SUTs, which are IEC-104 controlled stations (slaves), an IEC-104 controlling station (master) was implemented using OpenMUC j60870 [32]. However, before j60870 could be used for this research, several additions and modifications had to be made. OpenMUC j60870 already implements large parts of the protocol, however, as it was necessary to be able to control all fields inside the APDUs, several parts needed to be rewritten. The connection handler needed to be adapted to be able to have control over the APDUs. Furthermore, j60870 did not support sending and receiving of STOPDT and TESTFR messages, so these had to be implemented as well.

Also, all the different types of ASDUs needed to be implemented, as well as the custom ASDU with TypeId 0 (I[0]) and the custom APDU with the length set to the maximum of 253 (L_{max}). For the implementation of these, parts of j60870 were used.

In the implementation of this tool the default parameters for IEC-104 were used as described in section 2.4.

4.4.1 Subjects under Test

The SUTs are the slave implementations of IEC-104. All of these implementations used the default parameters for IEC-104 as defined in Table 2.4. For this research, it was possible to acquire access to five different implementations, of which three are simulators and two are real devices. These implementations are the following:

Axon Test Simulation tool from Axon Group for both masters and slaves of DNP3, IEC-101, IEC-104 and Modbus [4]. From this tool only the IEC-104 slave was used. This simulation tool was obtained from `http://www.axongroup.com.co/axongroupen/axon_productos_int.php?i=36`.

IEC 870-5-104 Simulator Simulation tool from Mitra Software that simulates an IEC-104 slave [29]. This simulation tool was obtained from `http://mitraware.com/apps/iec104sim.php`.

IEC-Test Free simulation tool from Siemens. This simulation tool is not publicly available.

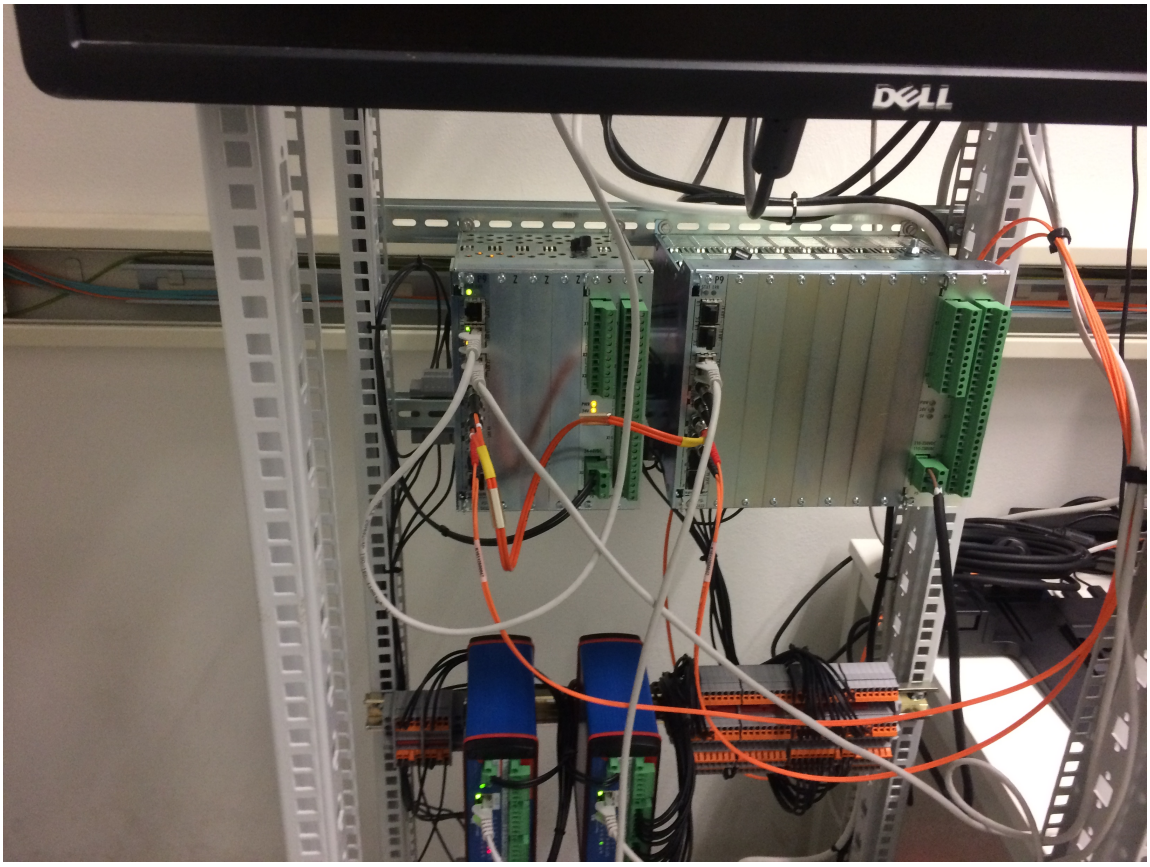


FIGURE 4.3: Test Set-up at Stedin

Sprecher Sprecon-E-C-92 Implementation of an RTU in a real device. Functions as a gateway to translate IEC-104 to IEC 61850. Figure 4.3 shows the set-up of this device. Access to this device was provided by Stedin.

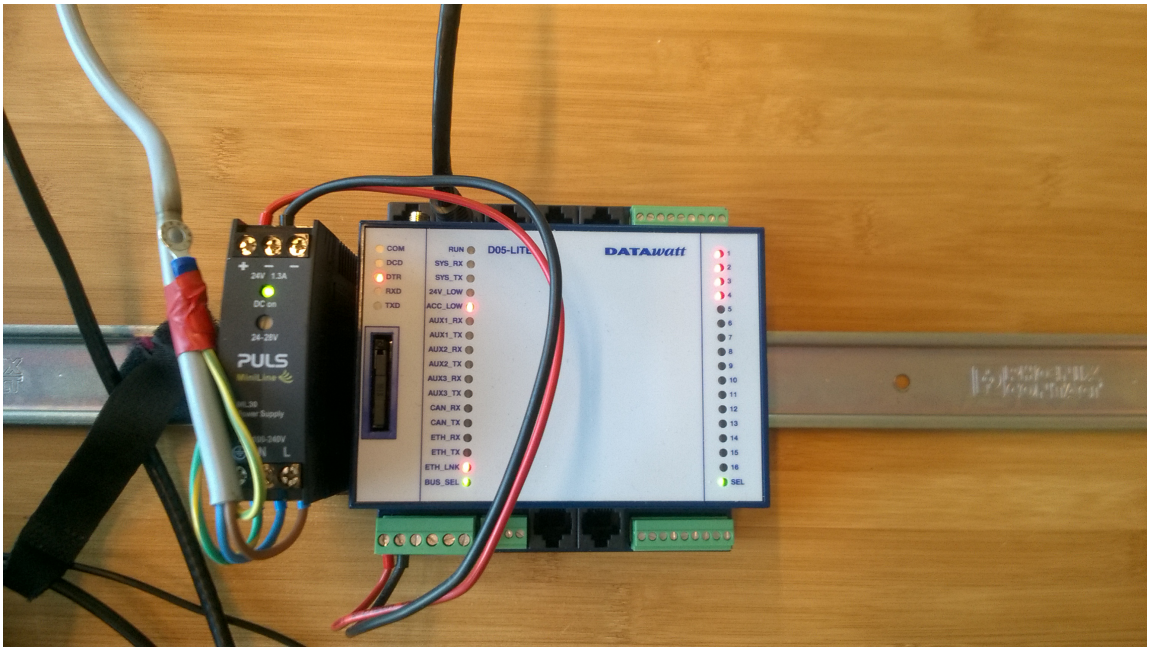


FIGURE 4.4: Test Set-up Datawatt

Datawatt D05-Lite Implementation of an RTU in a real device. Can be used as a fully functional PLC with support for several ICS protocols, e.g., Modbus and IEC 60870-5-101, -103 and -104. This device was used as an IEC-104 controlled station. Figure 4.4 shows the test set-up of this device. Access to this device was provided by Cogas.

4.5 Checker

To check if the implementation of a device complies to the standard, its automaton is compared to an automaton that is deduced from the State Transition Diagram in Figure 2.7. In this deduced automaton, that is shown in Figure 4.5, the transitions that are caused by a timeout are omitted. These transitions can be omitted as the tool is built so that timeouts do not occur during automata learning. The STARTED state is split into two states, as that actually also is the case in the transition diagram, because a STOPDT message can cause different transitions depending on if there are UNCONFIRMED I-frames or not. Therefore, the STARTED state is the state in which there are no unconfirmed I-frames and

the UNCONFIRMED STARTED state is the state in which there are unconfirmed I-frames. The other states are the same as in the state transition diagram, where TERMINATED is the state in which the connection is terminated.

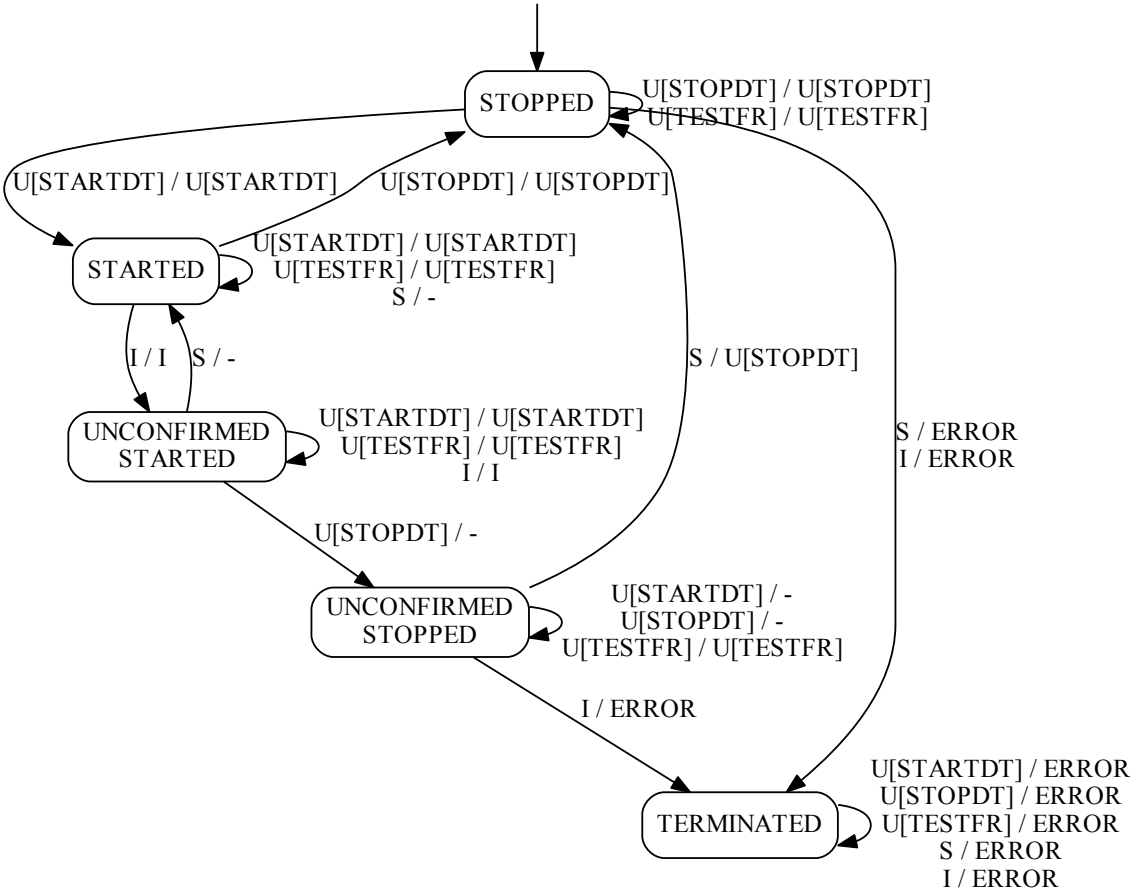


FIGURE 4.5: Automaton deduced from IEC-104 State Transition Diagram (Figure 2.7)

The implementation of a device complies to the standard if its produced automaton is equivalent to this deduced automaton. To check this, a checker was built using AutomataLib [27], which checks whether both automata are equivalent by traversing both

automata as described in section 3.5. The algorithm that is implemented in the checker is shown in Figure 2.

Algorithm 2 Checker algorithm for checking Mealy machine equivalence

Mealy machines $(Q, A_I, A_O, \delta, \lambda, q_0)$ and $(Q', A_I, A_O, \delta', \lambda', q'_0)$

$P \leftarrow \{(q_0, q'_0)\}$ ▷ queue of state tuples to process
repeat
  $(q, q') \leftarrow head(P)$ ▷ take first tuple from process queue
  for $a \in A_I$ **do** ▷ for every word in the alphabet
    if $\lambda(q, a) \neq \lambda(q', a)$ **then** ▷ if the outputs are different
      $d \leftarrow wordsToReach(q) \cup a$ ▷ separating word is found
    end if
    if $(\delta(q, a), \delta'(q', a)) \notin R$ **then** ▷ if successor state is not already processed
      $P \leftarrow P \cup (\delta(q, a), \delta'(q', a)))$ ▷ add successor states to process queue
    end if
  end for
  $P \leftarrow tail(P)$ ▷ remove processed tuple from process queue
  $R \leftarrow (q, q')$ ▷ add processed tuple to set of processed state tuples
until $P = \emptyset$ ▷ until all reachable combinations of states are processed
return $(d == \perp)$ ▷ return whether no separating word is found

4.6 Conclusion

In this chapter the used methodology was described. First, section 4.1 showed that the global set-up consists of a learner, mapper, teacher and checker. Next, the components of each of these were described and showed how this set-up produces automata.

Chapter 5

Results

The results are presented in the following chapter. First, in section 5.1, the results collected from the three simulators (Axon Test, Mitra Software IEC 870-5-104 Simulator and Siemens IEC-Test) are discussed. Next, in section 5.2, the results collected from real implementations (Datawatt and Sprecher) are examined.

As shown next, in most automata only one ASDU type is used for I-format messages. This is the case because differing the I-format ASDU type would not lead to a different automaton. Also, for certain ASDU types this would lead to differences that are only based on the amount of information objects returned per message. This would only lead to larger automata that actually are equivalent to those presented here.

5.1 Simulators

The simulators were used to build the learner and learn the first automata. For each of these simulators, the results and limitations are discussed here.

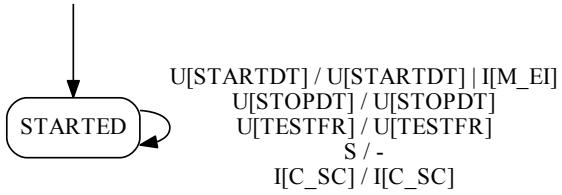


FIGURE 5.1: Automaton learnt from Axon Test simulator

5.1.1 Axon Test

The first tested simulator was the Axon Test simulator mentioned in section 4.4.1. As can be seen in Figure 5.1, Axon Test responds with a confirmation message when it receives any message. This shows that Axon Test only simulates responses on the IEC-101 layer. It only confirms the U-format messages of the IEC-104, but further ignores them. Therefore only a single state exists that is most similar to the STARTED state in the state transition diagram in Figure 2.7. Also, it can be seen that when Axon Test receives a STARTDT message it does not only reply with the STARTDT confirmation, but also with an *End of Initialization* I-format message. This message is sent because Axon Test was reset before every run. Although this simulator does not give an automaton that provides many new insights for comparison, it was useful to use it in the test phase in order to learn how the packages have to be structured, because it was crashed when it received a wrongly structured IEC-104 packet.

Limitations and challenges The IEC-104 standard was not implemented properly in Axon Test, as it did not respond to any messages after the connection was closed and a new connection was made. Because of this, the simulator had to be restarted before every run.

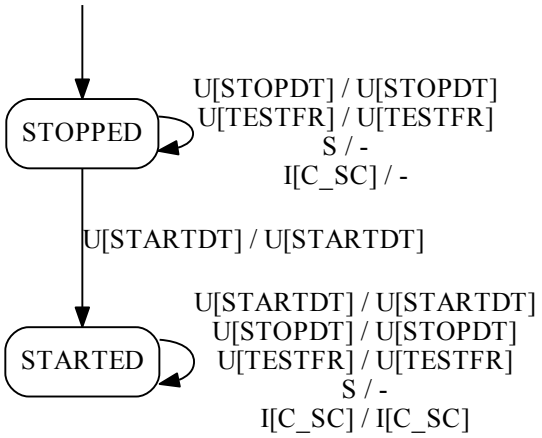


FIGURE 5.2: Automaton learnt from Mitra Software IEC 870-5-104 Simulator

5.1.2 Mitra Software IEC 870-5-104 Simulator

Mitra Software's IEC 870-5-104 Simulator is the only tested simulator that actually has more than one state. In its initial state the simulator only responds to U-format messages, but ignores I-format messages. Only when a STARTDT is sent, the simulator converts to a state in which it responds to I-format messages. However, once the simulator has reached this (STARTED) state, it cannot return to the initial (STOPPED) state when a STOPDT is sent. So after reaching the STARTED state, the simulator remains in this state as long is the connection is not reset. The learnt automata of this simulator can be seen in Figure 5.2.

Limitations and challenges The entire IEC-104 stopping process did not seem to be implemented in this simulator.

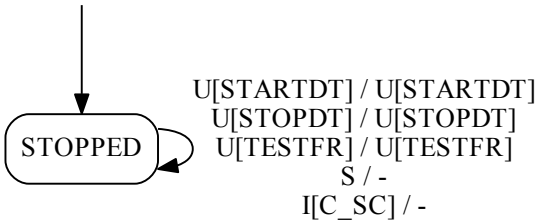


FIGURE 5.3: Automata learnt from IEC-Test Simulator

5.1.3 Siemens IEC-Test

The automaton that was learnt from IEC-Test (Figure 5.3) looks very similar to the automaton of Axon Test. In its initial state IEC-Test responds to U-format messages, however, it does not reply to any sent I-messages.

Limitations and challenges IEC-Test is a very manual tool that did not appear to have options to automatically reply to anything other than U-format messages.

5.2 Real devices

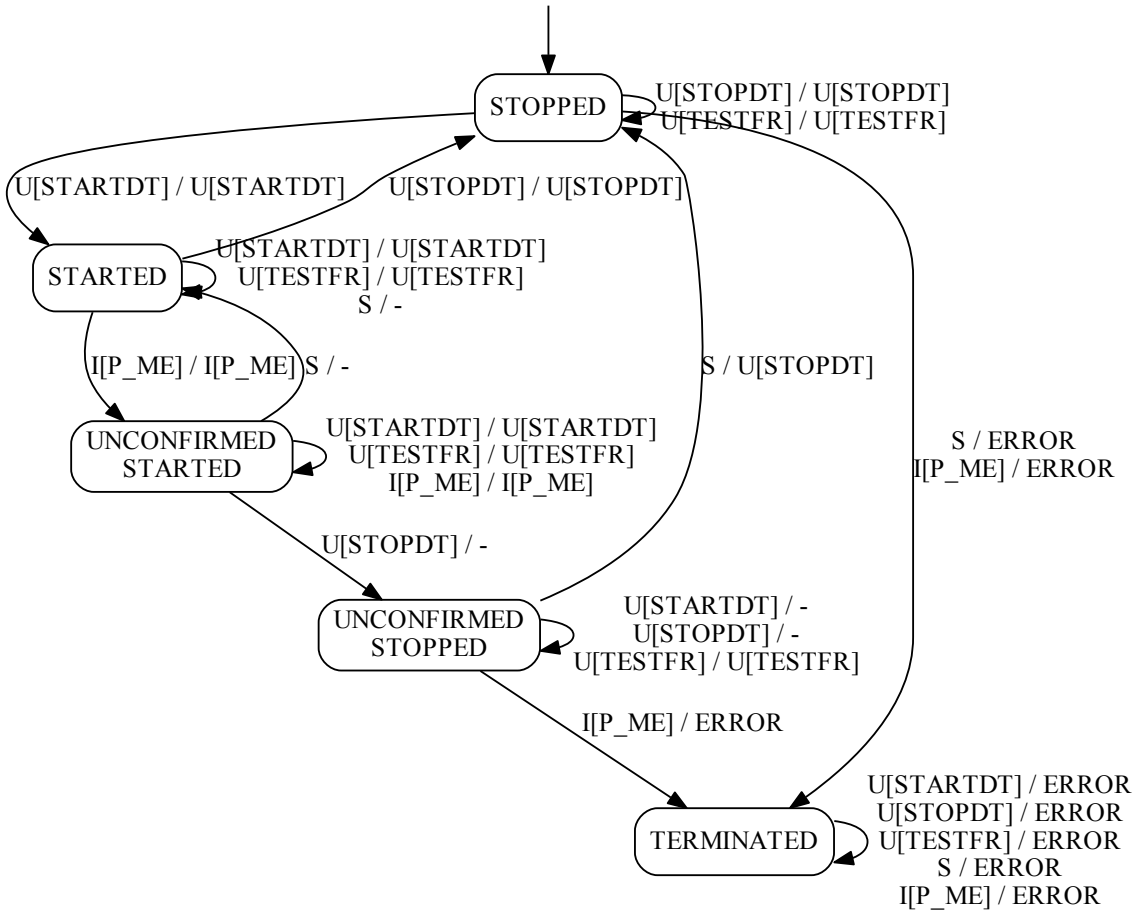


FIGURE 5.4: Automaton learnt from Datawatt implementation

5.2.1 Datawatt D05-Lite

Figure 5.4 shows that the automaton of Datawatt's RTU appears very similar to the deduced automaton in Figure 4.5. This is not only appearance, as the checker indeed indicates that this automaton is equivalent to the to the automaton that is deduced from the IEC-104 standard. What especially can be noticed is that in the UNCONFIRMED STOPPED

state, the only messages to which the RTU responds are the TESTFR and the S-format confirmation message. Also a TERMINATED state is reached when specific unexpected messages are sent in a state, which also follows from the standard.

When looking at more different ASDU types, an automaton was found that does not comply to the standard when sending the, rarely used, I-format message for File Select. Then a new strange state appears, as can be seen in Figure 5.5. As the File Select message that is sent does not contain a valid address of a file, the RTU appears to be more strict than the standard by terminating the connection on the next received I-format message. However, according to the standard, the expected response would be a negative confirmation I-format message.

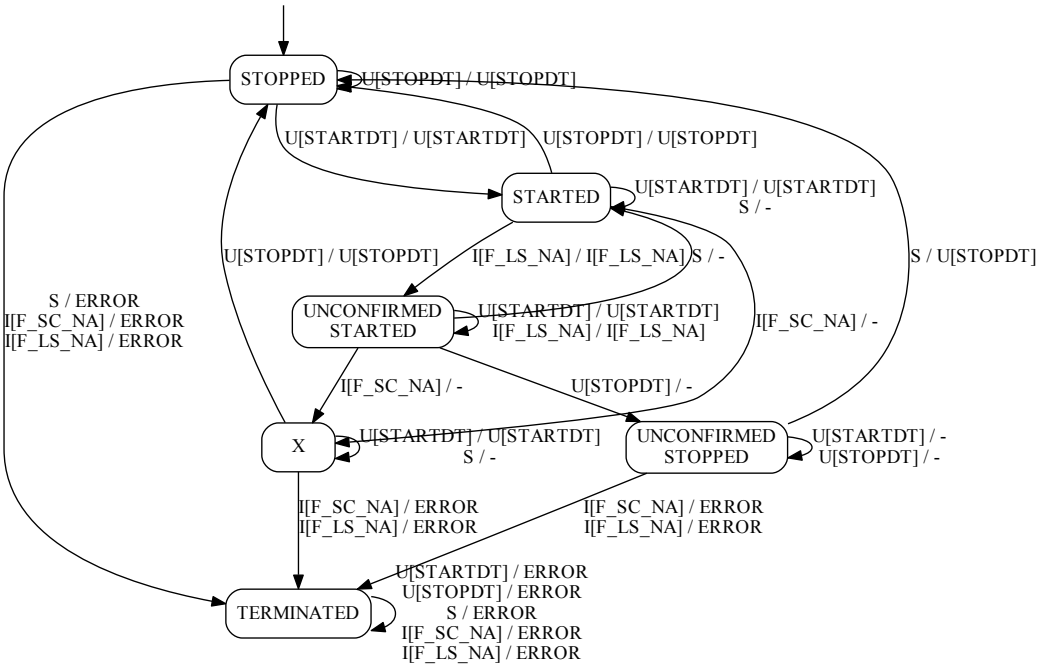


FIGURE 5.5: Automaton learnt from Datawatt implementation with strange behaviour

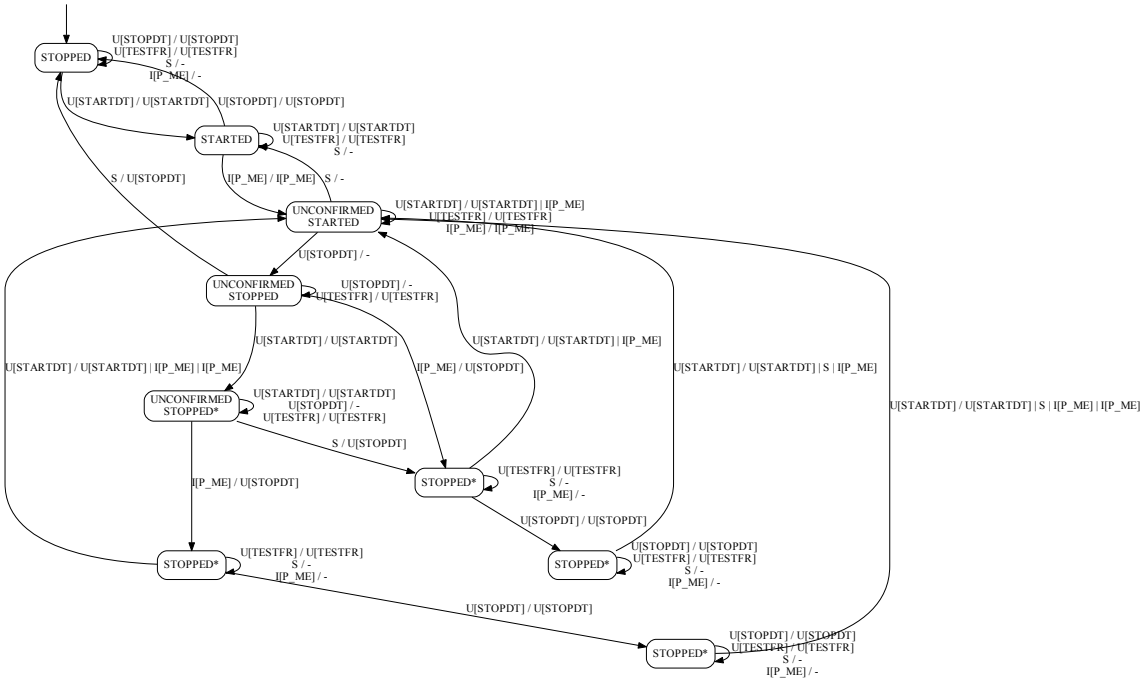


FIGURE 5.6: Automaton learnt from Sprecher implementation

5.2.2 Sprecher Sprecon-E-C-92

According to the checker, the automaton that was produced from the RTU from Sprecher is not equivalent to the automaton of the standard. Therefore, the RTU does not follow the protocol as specified by IEC-104 completely. This can also be seen in the automaton, as the connection is never actively closed when packages with a wrong type are received. This causes a couple of strange states to emerge in the automaton. When a new I-format message is sent from the UNCONFIRMED STOPPED state, this I-format message confirms everything up to the moment that the STOPDT message was sent, which should bring the state to STOPPED, but now that I-format message itself is not confirmed yet. This is an interesting difference with the RTU from Datawatt, as that RTU directly terminates the connection when a message is received that should not be received in that state. According to the standard, this termination also should happen. One of the reasons for

the Sprecher RTU to never terminate the connection when faulty packages are received, might be because it is used as a gateway to another IEC 61850 subnetwork.

Also, when the RTU is in the UNCONFIRMED STOPPED state, it is still possible to send a U-format STARTDT, which makes the RTU come in a sort of hybrid state between UNCONFIRMED STOPPED and STARTED.

5.3 Additional considerations

As described in section 4.2.2 two artificial messages, I[0] and L_{max}, were also implemented in the developed tool. The I[0] messages did not provide any behaviour that was different from the normal behaviour. However, the tests with the L_{max} message, consisting of only the START byte (104) and the maximum length (253), provided some interesting results that are discussed here separately. When an L_{max} message was sent in any state, the tested implementations did not respond until they had received the total number of bytes that were specified in the length field. This means that every message that is sent after an L_{max} message, is ignored, because the implementation thinks that message belongs to the initial L_{max} message. As showed in section 2.3.1, availability is of a great importance for ICS. When messages are ignored, this is a problem for the availability of the system. Therefore, an attacker would be able to disrupt the communication by sending an L_{max} message.

Furthermore, there are some rare occurrences where the algorithm gives some strange extra states during learning, while these in reality are not new states. These could be caused by interruption during learning. However, these automata are equal to each other. The checker properly handles these automata, because the comparison in the checker is based on the output sequences.

5.4 Conclusion

This chapter described the results of learning the automata of both simulators as real devices. Table 5.1 provides an overview of all implementations and i) their number of states, ii) if they comply to the standard, iii) if they have a started state, i.e. a separate state when STARTDT is sent and iv) if they have a terminated state, i.e. a state in which the connection is lost. Only the device from Datawatt seems to correctly implement the protocol as described in the specification for almost all messages. The other real device does not entirely comply with the specification, because it never terminates the connection and it has some strange behaviour when sending a STARTDT message to a connection

that is being closed. All simulators that were used, were very limited and therefore also do not comply to the specification.

When comparing only the real devices to each other and the standard, there are two main differences that should be noted. The first one is the difference in whether the device terminates the connection, when it receives a message that is not allowed in that state. According to the standard, this termination of the connection should happen. The Datawatt RTU correctly terminates in these cases, while the Sprecher RTU ignores the message. The second difference is the behaviour when a STARTDT is received in the UNCONFIRMED STOPPED state. The standard specifies that this STARTDT should not cause a state change. However, the standard does not unambiguously specify if the device should respond to this STARTDT message. The Datawatt RTU correctly ignores this STARTDT and also does not respond to this. However, the Sprecher RTU shows some strange behaviour at this point. Not only does the Sprecher RTU respond to the STARTDT with a STARTDT confirmation (which is underspecified), but also a change in state occurs, which results in strange behaviour in its subsequent responses.

Table 5.1: Results overview

Implementation	**#states**	**Standard compliance**	**Started state**	**Terminated state**
Standard	*5*	*True*	*True*	*True*
Axon Test	1	False	False	False
IEC 870-5-104 Simulator	2	False	*True*	False
IEC-Test	1	False	False	False
Datawatt D05-Lite	*5*	*True*	*True*	*True*
Sprecher Sprecon-E-C-92	9	False	*True*	False

Chapter 6

Conclusions

This chapter concludes this thesis by, first, reviewing the research questions that were posed in the Introduction. Then, in section 6.1 the strengths and limitations of this research are discussed and recommendations for future work are given in section 6.2. Finally, in section 6.3, concrete recommendations for vendors and operators are presented.

In this thesis, three research questions were posed. These three are concluded here separately:

RQ1: How can implementations of IEC 60870-5-104 be represented formally?

This question was divided into two sub-questions that were addressed first. After that, these two sub-questions were used to address this complete research question.

RQ1a: Which methods exist to generate formal representations?

This thesis described methods to generate formal representations in section 3.3. In order to generate these formal representations, these methods used the $L^\star$ algorithm and its deduced $L^\star_M$ algorithm. These are algorithms to generate a formal implementation in the form of an automaton for any sort of implementation.

RQ1b: What method is suitable for formally representing protocols?

The most suitable formal representation for protocols was a specific automaton, called a Mealy machine. Mealy machines are comprehensible due to their simplicity and they can clearly describe protocol behaviour, as described in section 3.2.

The method and formal representation that were found in the sub-questions above, were used to design a tool that is able to iteratively build automata from RTUs that implement IEC-104. This tool implements an IEC-104 controlling station and is able to send all messages that are defined in the IEC-104 standard, as described in section 2.4. Furthermore, it uses the $L^{\star}_{M}$ algorithm, that was described in section 3.4, to learn automata. The details of the design and functionality of this tool were described in Chapter 4. Also, the source code of the mentioned tool can be found on GitHub [24].

RQ2: To what extent do implementations of IEC 60870-5-104 comply to their standard?

To be able to address this question, four sub-question had to be addressed first:

RQ2a: What implementations of IEC 60870-5-104 are available?

Obtaining implementations of IEC 60870-5-104 in the form of real devices was a challenge, because vendors were reserved in sharing these. Eventually, it was possible to acquire access to two real devices. Next to that, three implementations in the form of simulators were available. These simulators were Axon Test, Mitra Software IEC 870-5-104 Simulator and Siemens IEC-Test. The real devices that were available were the Datawatt D05-Lite RTU and the Sprecher Sprecon-E-C-92.

RQ2b: How can the IEC 60870-5-104 standard be formally represented?

The IEC 60870-5-104 standard contains a state transition diagram (shown in Figure 2.7). Section 4.5 described how this state transition diagram can be represented as a Mealy machine. This Mealy machine, that represents the IEC 60870-5-104 standard and is compared with the Mealy machines inferred from the implementations, is shown in Figure 4.5.

RQ2c: How to compare these formal representations?

To be able to compare the inferred automata, a method for comparison was described in section 3.5. This method was used in an algorithm to compare the produced Mealy machines to the Mealy machine that was deduced from the standard in the previous sub-question. The component of the tool that contains this comparison algorithm together with the Mealy machine from the standard, was called the checker.

RQ2d: What do the findings from these questions tell about the IEC 60870-5-104 standard?

By analysing the produced automata, it can be concluded that vendor implementations differ greatly. This means that potentially systems for different vendors are not completely compatible. When the origin of these differences was investigated in the standard, there are some interesting findings.

One of these findings, described in section 5.3, showed that it was not specified in the standard whether the entire length of the packet had to be received before parsing the message format. Therefore, the vendor that implements the standard can either decide to parse the format when three bytes are received or when the entire length is received.

Another finding, described in 5.4, was that the standard does not exactly specify whether an implementation should respond when a message to start a connection is received, while the former connection is being closed, but still is waiting for the confirmation of messages. In this case the standard does not specify whether the device should ignore this message or respond. In one of the investigated devices this led to strange behaviour, as it seemed to react on this message as it did respond to it and after this deviated from protocol. Therefore, ignoring this message in such a state appears to be a better option and would prevent this from happening. However, the Sprecher device still should not show this behaviour according to the standard, as the state transition diagram in the standard shows that this should not lead to a new state.

Overall, none of the tested implementations comply to the specification for all possible IEC-104 messages. However, the real devices follow specification for the greater part, especially the RTU from Datawatt. This Datawatt RTU is the only tested implementation that correctly terminates the connection on the reception of incorrect messages in specific states. The Sprecher RTU ignores these messages, which means that the vendor either overlooked that the connection should be terminated or chose not to terminate the connection deliberately. If this was by choice, this might relate to the tested Sprecher RTU acting as a gateway, translating between IEC-104 and IEC 61850.

RQ3: What information security attributes are violated in the implementations that were researched in the second research question and how?

In order to address this question, the attributes needed to be defined by addressing the following sub-question first:

RQ3a: What information security attributes are relevant in ICS?

The information security attributes (confidentiality, integrity and availability) that are used for regular IT systems, were also applicable to ICSs. However, as described in section 2.3.1, the order of the importance for each of these attributes was reversed, so these are, in order of importance (starting from most important): availability, integrity and confidentiality.

The evaluation of this research question was divided into the three information security attributes. Each of these attributes was examined separately:

Availability In section 5.3, the results when sending the artificial L_{max} message were described. These results showed that it is possible to disrupt the communication. By sending a single (2-byte) L_{max} message, all subsequently received messages were ignored up to 251 bytes. If the first four bytes would be a correct U-format message, the implementations would not even notice a disturbance. This test was not run on all implementations, but this behaviour was noticed on all implementations on which it was tested. As availability and timely responses are considered to be the most important in ICS, this behaviour when an L_{max} messages is sent, is considered as a violation of availability, as it potentially could provide attackers basis for, e.g., denial of service attacks, which could result in a power outage for example.

Integrity As described in section 2.3.2, most ICS protocols lack authentication. As IEC-104 also lacks any form of authentication, it is possible to perform a man-in-the-middle attack (or other attacks described in section 2.3.2) and therefore influence the integrity of IEC-104.

Confidentiality All analysed implementations produced different automata, therefore it is possible for an attacker to fingerprint the used devices, i.e. identify which device is used, via the network. Confidentiality is considered to be of least importance to ICS, because most messages are predictable. However, recognising which devices are used without having physical access, is considered to be violating confidentiality, as it helps attackers to gain knowledge about the system. Attackers could, e.g., use this knowledge to find other vulnerabilities in the system.

6.1 Discussion

Reviewing this research, strengths and limitations can be identified. As a first limitation, there was no access to a simulators that simulated IEC-104 completely. There might be other simulators, that are not publicly available, that contain a better implementation of IEC-104.

One of the strengths of this research is that the tested devices are devices that are both actually used in the power distribution sector in the Netherlands. Unfortunately, for this research only two of these devices were tested, as only access to two devices could be acquired. Therefore, this is a limitation, as with more devices a more complete comparison could have been made.

Another strength of this research is the open-source mealy104 tool [24] that was built for this research. This tool was designed in a modular and generic manner, which makes it easy for other parties to extend the tool or to adapt it for other (ICS) protocols.

6.2 Future Work

A general first recommendation is that more research should be performed on the security of the (application layer) protocols that are used in ICS. More specifically, the method presented in this thesis should be applied to other ICS protocols. Even though currently IEC-104 devices are being produced that will be used for at least a decade, the power sector is gradually progressing towards the adaption of IEC 61850. Therefore, it would particularly be interesting to apply the proposed method to MMS, which is used in IEC 61850.

This research only focused on testing the controlled station side of IEC-104. Further research should be conducted regarding testing the implementations in the opposite direction, i.e. testing the controlling station.

To extend the methodology of this research itself, more variation could be brought in the alphabet. The messages that are currently in the alphabet only use correct Causes of Transmission. Therefore, it would be interesting to add messages with incorrect Causes of Transmission. Also, other messages with more variations in the ASDU layer might be interesting to investigate. Finally, beside the current I[0] and L_{max} messages, more custom defined messages could be added. Those could than even have more strange unspecified formats.

In this research, the automata that are built are based on the responses of the RTU. In future research this could be extended by examining the responses in the processes

or processes behind the RTU and what happens to those states. Then, it might also be interesting to extend the alphabet with more variations in the ASDU layer.

The presented tool itself could also be improved. By applying the work of Henrix [17], the learning process for learning automata can become more efficient. Also, it should be investigated if there are better alternatives for the current equivalence oracle that is used for the learning process. This equivalence is currently, as described in Chapter 4, based on a certain randomness in order to keep the time of learning within feasible boundaries.

6.3 Recommendations

Several recommendations for both IEC-104 vendors and operators can be derived from the research in this thesis. The first recommendation for both vendors and operators in general is to use the tool that was developed in this research to check if their IEC-104 devices comply to the standard. Furthermore, when the device does not comply completely, the behaviour of the device can be examined, because the tool provides an understandable automaton. In order to reduce the possibility of fingerprinting, it would be best if all devices would comply to the standard and thus have an automaton equivalent to the automaton in Figure 4.5.

A general recommendation for vendors is to examine the behaviour of their devices when length fields are used that do not match with the actual length. More specifically, when the first three bytes of an IEC-104 message are received, these bytes can already be parsed to identify which message format was received. If a U-format or S-format message is received, the length-field should always be 4. When, e.g., a message is received of which the third byte indicates that it has a U-format, but the length field is larger then 4, the device should already notice that this message is faulty. If the device recognises these messages as faulty, the device is less prone to denial of service attacks that utilise this.

A general recommendation for operators is to ensure that integrity is guaranteed, i.e. that authentication needs to take place before IEC-104 messages can be sent to the device. Operators can accomplish this by implementing standards like IEC 62351, which provides end-to-end encryption, as was mentioned in section 2.4.

A recommendation specifically for Sprecher is to comply to the standard more strict. Therefore, two changes need to be made: i) when in a certain state packages are received that are not allowed according to the standard, the connection should be terminated and ii) when a message to stop the data transfer (STOPDT) is received, the device should ignore all messages except an S-format message.

Bibliography

[1] Fides Aarts, Harco Kuppens, Jan Tretmans, Frits Vaandrager, and Sicco Verwer. Improving active Mealy machine learning for protocol conformance testing. *Machine Learning*, 96(1-2):189–224, jul 2014.

[2] Fides Aarts, Julien Schmaltz, and Frits Vaandrager. Inference and Abstraction of the Biometric Passport. In *Lecture Notes in Computer Science*, pages 673–686. Springer Berlin Heidelberg, 2010.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, nov 1987.

[4] Axon Group. Axon Test, 2016. `http://www.axongroup.com.co/axongroupen/axon_productos-text.php`.

[5] Rafael R R Barbosa. *Anomaly Detection in SCADA Systems*. PhD thesis, University of Twente, 2014.

[6] Ben Caldwell, Rachel Cardell-Oliver, and Tim French. Learning Time Delay Mealy Machines From Programmable Logic Controllers. *IEEE Transactions on Automation Science and Engineering*, 13(2):1155–1164, apr 2016.

[7] Censys. Censys, 2017. `https://censys.io/`.

[8] Georg Chalupar, Stefan Peherstorfer, Erik Poll, and Joeri de Ruiter. Automated Reverse Engineering using Lego. *Proceedings of the 8th USENIX Workshop on Offensive Technologies - WOOT '14*, page 9, 2014.

[9] Gordon Clarke and Reynders Deon. *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*. Newnes, 2004.

[10] Frances Cleveland. IEC 62351 Security Standards for the Power System Information Infrastructure. *IEC TC57 WG15 Security Standards ver 14*, 2012.

[11] CPNI and Red Tiger Security. Securing the move to IP-based SCADA/PLC networks. Technical Report November, CPNI, 2011.

[12] Edvard Csanyi. Three generations of SCADA system architectures, 2015. `http://electrical-engineering-portal.com/three-generations-of-scada-system-architectures`.

[13] Paul Fiterau-Brostean, Ramon Janssen, and Frits Vaandrager. Learning fragments of the TCP network protocol. *Lecture Notes in Computer Science*, 8718 LNCS:78–93, 2014.

[14] Igor Nai Fovino, Andrea Carcano, Marcelo Masera, and Alberto Trombetta. Design and Implementation of a Secure Modbus Protocol. In *IFIP Advances in Information and Communication Technology*, volume 311, pages 83–96. Springer Berlin Heidelberg, 2009.

[15] Fu Qin-cui, Liu Zi-ying, and Fu Ke-jia. Implementation of IEC60870-5-104 protocol based on finite state machines. In *2009 International Conference on Sustainable Power Generation and Supply*, pages 1–5. IEEE, apr 2009.

[16] E. Gunawan, Pek Tong Tan, and Nansi Shi. Selection of a formal description technique (FDT) for a FDT based protocol converter. In *[Proceedings] Singapore ICCS/ISITA '92*, pages 188–192. IEEE, 1992.

[17] M Henrix. Performance improvement in automata learning. Master's thesis, Radboud University Nijmegen, 2015.

[18] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, may 1997.

[19] ISO/TMBG. ISO/IEC Guide 2:2004. Technical report, International Organization for Standardization, 2004.

[20] O.J. Jacobsen and D.C. Lynch. A Glossary of Networking Terms. RFC, RFC Editor, 1991.

[21] Qi Jing, Athanasios V. Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. Security of the Internet of Things: perspectives and challenges. *Wireless Networks*, 20(8):2481–2501, nov 2014.

[22] Rao Kalapatapu. SCADA Protocols and Communication Trends. Technical report, ISA, 2004.

[23] BooJoong Kang, Peter Maynard, Kieran McLaughlin, Sakir Sezer, Filip Andren, Christian Seitl, Friederich Kupzog, and Thomas Strasser. Investigating cyber-physical attacks against IEC 61850 photovoltaic inverter installations. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, volume 2015-Octob, pages 1–8. IEEE, sep 2015.

[24] Max Kerkers. Mealy104, 2017. `https://github.com/mkerkers/mealy104`.

[25] Maryna Krotofil and Dieter Gollmann. Industrial control systems security: What is happening? In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 670–675. IEEE, jul 2013.

[26] Ralph Langner. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy Magazine*, 9(3):49–51, may 2011.

[27] LearnLib. LearnLib, 2016. `http://learnlib.de/`.

[28] Peter Maynard, Kieran McLaughlin, and Berthold Haberler. Towards Understanding Man-In-The-Middle Attacks on IEC 60870-5-104 SCADA Networks. In *2nd International Symposium for ICS & SCADA Cyber Security Research 2014*, pages 30–42. BCS Learning & Development, sep 2014.

[29] Mitra Software. IEC 870-5-104 Simulator, 2016. `http://mitraware.com/apps/iec104sim.php`.

[30] S. Mohagheghi, J. Stoupis, and Z. Wang. Communication protocols and networks for power systems-current status and future trends. In *2009 IEEE/PES Power Systems Conference and Exposition*, pages 1–9. IEEE, mar 2009.

[31] Oliver Niese. *An Integrated Approach to Testing Complex Systems*. PhD thesis, Universität Dortmund, 2003.

[32] OpenMUC. j60870, 2016. `https://www.openmuc.org/iec-60870-5-104/`.

[33] Al Sakib Khan Pathan. *The State of the Art in Intrusion Prevention and Detection*. Auerbach Publications, 2014.

[34] Erik Poll, Joeri De Ruiter, and Aleksy Schubert. Protocol State Machines and Session Languages: Specification, implementation, and Security Flaws. In *2015 IEEE Security and Privacy Workshops*, pages 125–133. IEEE, may 2015.

[35] Julian L. Rrushi. SCADA protocol vulnerabilities. In *Lecture Notes in Computer Science*, volume 7130, pages 150–176. Springer Berlin Heidelberg, 2012.

[36] John E. Savage. Models of computation, exploring the power of computing. *IEEE Design & Test of Computers*, page 672, 1998.

[37] Muzammil Shahbaz and Roland Groz. Inferring Mealy Machines. In *Lecture Notes in Computer Science*, volume 5850, pages 207–222. Springer Berlin Heidelberg, 2009.

[38] Shodan. Shodan, 2016. `https://www.shodan.io/`.

[39] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to Industrial Control Systems (ICS) Security. Technical report, National Institute of Standards and Technology, 2011.

[40] Michael Swearingen, Steven Brunasso, Joe Weiss, and Dennis Huber. What You Need to Know (and Don't) About the AURORA Vulnerability. *POWER*, 2013.

[41] TC 57 - Power systems management and associated information exchange. IEC 60870-5-101:2003. Standard, International Electrotechnical Commission, Geneva, 2003.

[42] TC 57 - Power systems management and associated information exchange. IEC 60870-5-104:2006. Standard, International Electrotechnical Commission, Geneva, 2006.

[43] TC 57 - Power systems management and associated information exchange. IEC TS 60870-5-7:2013. Technical specification, International Electrotechnical Commission, Geneva, 2013.

[44] Anshul Thakur. SCADA Systems, 2014. `http://www.engineersgarage.com/articles/scada-systems`.

[45] Fabian van den Broek, Brinio Hond, and Arturo Cedillo Torres. Security Testing of GSM Implementations. *Engineering Secure Software and Systems*, 8364:179–195, 2014.

[46] Patrick Verleg. *Inferring SSH state machines using protocol state fuzzing*. PhD thesis, Radboud University, 2016.

[47] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A Taxonomy of Cyber Attacks on SCADA Systems. *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pages 380–388, oct 2011.

Appendix A

IEC-104 ASDU command types

A.1 Process information in monitoring direction

1	M_SP_NA_1	Single point information
2	M_SP_TA_1	Single point information with time tag
3	M_DP_NA_1	Double point information
4	M_DP_TA_1	Double point information with time tag
5	M_ST_NA_1	Step position information
6	M_ST_TA_1	Step position information with time tag
7	M_BO_NA_1	Bit string of 32 bit
8	M_BO_TA_1	Bit string of 32 bit with time tag
9	M_ME_NA_1	Measured value, normalised value
10	M_ME_TA_1	Measured value, normalised value with time tag
11	M_ME_NB_1	Measured value, scaled value
12	M_ME_TB_1	Measured value, scaled value with time tag
13	M_ME_NC_1	Measured value, short floating point value
14	M_ME_TC_1	Measured value, short floating point value with time tag
15	M_IT_NA_1	Integrated totals
16	M_IT_TA_1	Integrated totals with time tag
17	M_EP_TA_1	Event of protection equipment with time tag
18	M_EP_TB_1	Packed start events of protection equipment with time tag
19	M_EP_TC_1	Packed output circuit information of protection equipment with time tag
20	M_PS_NA_1	Packed single-point information with status change detection
21	M_ME_ND_1	Measured value, normalised value without quality descriptor

A.2 Process telegrams with long time tag (7 octets)

30	M_SP_TB_1	Single point information with time tag CP56Time2a
31	M_DP_TB_1	Double point information with time tag CP56Time2a
32	M_ST_TB_1	Step position information with time tag CP56Time2a
33	M_BO_TB_1	Bit string of 32 bit with time tag CP56Time2a
34	M_ME_TD_1	Measured value, normalised value with time tag CP56Time2a
35	M_ME_TE_1	Measured value, scaled value with time tag CP56Time2a
36	M_ME_TF_1	Measured value, short floating point value with time tag CP56Time2a
37	M_IT_TB_1	Integrated totals with time tag CP56Time2a
38	M_EP_TD_1	Event of protection equipment with time tag CP56Time2a
39	M_EP_TE_1	Packed start events of protection equipment with time tag CP56time2a
40	M_EP_TF_1	Packed output circuit information of protection equipment with time tag CP56Time2a

A.3 Process information in control direction

45	C_SC_NA_1	Single command
46	C_DC_NA_1	Double command
47	C_RC_NA_1	Regulating step command
48	C_SE_NA_1	Setpoint command, normalised value
49	C_SE_NB_1	Setpoint command, scaled value
50	C_SE_NC_1	Setpoint command, short floating point value
51	C_BO_NA_1	Bit string 32 bit

A.4 Command telegrams with long time tag (7 octets)

58	C_SC_TA_1	Single command with time tag CP56Time2a
59	C_DC_TA_1	Double command with time tag CP56Time2a
60	C_RC_TA_1	Regulating step command with time tag CP56Time2a
61	C_SE_TA_1	Setpoint command, normalised value with time tag CP56Time2a
62	C_SE_TB_1	Setpoint command, scaled value with time tag CP56Time2a
63	C_SE_TC_1	Setpoint command, short floating point value with time tag CP56Time2a
64	C_BO_TA_1	Bit string 32 bit with time tag CP56Time2a

A.5 System information in monitoring direction

70	M_EI_NA_1	End of initialization

A.6 System information in control direction

100	C_IC_NA_1	(General-) Interrogation command
101	C_CI_NA_1	Counter interrogation command
102	C_RD_NA_1	Read command
103	C_CS_NA_1	Clock synchronisation command
~~104~~	~~C_TS_NB_1~~	~~Test command~~ [IEC-101]
105	C_RP_NC_1	Reset process command
~~106~~	~~C_CD_NA_1~~	~~Delay acquisition command~~ [IEC-101]
107	C_TS_TA_1	Test command with time tag CP56Time2a

A.7 Parameter in control direction

110	P_ME_NA_1	Parameter of measured value, normalised value
111	P_ME_NB_1	Parameter of measured value, scaled value
112	P_ME_NC_1	Parameter of measured value, short floating point value
113	P_AC_NA_1	Parameter activation

A.8 File transfer

120	F_FR_NA_1	File ready
121	F_SR_NA_1	Section ready
122	F_SC_NA_1	Call directory, select file, call file, call section
123	F_LS_NA_1	Last section, last segment
124	F_AF_NA_1	Ack file, Ack section
125	F_SG_NA_1	Segment
126	F_DR_TA_1	Directory
127	F_SC_NB_1	QueryLog - Request archive file