Bachelor Thesis

Comparison of SCADA protocols and implementation of IEC 104 and MQTT in MOSAIK

Thomas Teodorowicz

Matr. 421489

Supervisor: Prof. Dr. Anne Remke

University of Münster, Germany

August 15, 2017

Declaration of Academic Integrity

I hereby confirm that this thesis on *Comparison of SCADA protocols and implementation of IEC 104 and MQTT in MOSAIK* is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

Münster, August 13, 2017 _____

(Thomas Teodorowicz)

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

Münster, August 13, 2017 _____

(Thomas Teodorowicz)

Abstract

Abstract

Smart grid security is an important area of research, as security problems in the software controlling smart grids can have severe consequences on the real world. Supervisory control and data acquisition (SCADA) protocols are an integral part of the SCADA systems that are used to control smart grids and they are also potential targets for attackers. Therefore, it is important to know which SCADA protocols are most up to date concerning security risks, while still providing the necessary functionality to run a SCADA system. Hence, this thesis will give an overview of SCADA, introduce several protocols for SCADA systems, and shortly evaluate how well the protocols perform in terms of interoperability, flexibility, reliability, security, ease of use, and peer-to-peer capability. The state-of-the-art SCADA protocol IEC 60870-5-104 and the promising protocol Message Queuing Telemetry Transport (MQTT) will be compared with each other in more detail. Additionally, experimental implementations of MQTT and IEC 60870-5-104 within a simulation created with the co-simulation framework Mosaik will be presented. Mosaik will also be introduced in more detail. Missing features in these implementations and their importance will be introduced and evaluated. MQTT will be found as a potential candidate for SCADA systems of smart grids that have to interact with the Internet of things. It will also be found that MQTT has to be extended to be useful within SCADA systems.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Stability of power grids is important for modern life. Blackouts can lead to inconvenience (e.g. hallway lights become inoperative), loss of money, or even life-threatening situations (e.g. traffic accidents, due to traffic lights being deactivated). Hence, hacks on power grids that aim to create power outages are a real danger and should be prevented as effectively as possible. A famous recent example is the attack on the Ukrainian power grid [1]. Supervisory Control and Data Acquisition (SCADA) systems are typically used as a control and security measure for power grids. For this purposes, so-called remote terminal units (RTUs) are used by the SCADA system to collect data from field devices.

With the advent of photovoltaics (PVs) and similar devices, power can be generated by the consumers. Power grids with consumer generated power are called smart grids. The flow of current in smart grids may change based on the power generated by the consumers. To manage smart grids more effectively, more control is given to RTUs. However, the additional control comes at the cost of security as was shown in theory by [2] and also in a real world scenario by the Ukrainian power grid hack [3]. To increase security in smart grids, [4] proposes a local intrusion detection system (IDS). Additionally, [5] proposes a behavior-rule based model that analyses the behavior of devices for anomalies.

Another important aspect of SCADA systems are communication protocols. Protocols used in SCADA systems are normally specifically designed to be used in SCADA systems. They ensure effective communication within the SCADA system, but they are also possible attack points for hackers. Examples of possible attacks on SCADA protocols are man-in-the-middle attacks [6] and denial-of-service attacks [7]. Therefore, it is important that the protocols provide security features. Moreover, they should not be too complex, because more complexity increases the risk of mistakes being made. When considering more decentralized control, it might also be important to let remote devices communicate with each other. Hence, protocols should also enable peer-to-peer communication. It has to be noted that [8] found that the system state may not be consistent when using peer-to-peer communication which has to be considered when designing a protocol that possesses peer-to-peer capabilities. Additionally, protocols might need to be able to deal with the Internet of things (IoT) in the near future [9].

Even among the most commonly used SCADA protocols there is no protocol that is clearly superior to the others. Additionally, these protocols may not even be suited for the work with smart grids or with the IoT as concluded by [9] where the protocol Message Queuing Telemetry Transport (MQTT) is proposed as a possible alternative. The legacy protocol Modbus, the commonly used DNP3 protocol and IEC 61850 were examined in [10] and it was concluded that an extended version of IEC 61850 might be more useful than traditional SCADA protocols. [8] found in experiments that neither UPD nor TCP based DNP3 is fast

enough to meet the delay requirements in smart grids controlled by decentralized systems. The problem concerning system consistency when using peer-to-peer communication that was mentioned earlier, is that asynchronous messaging which is used for peer-to-peer communication, can lead to different behavior of physical devices under the same circumstances [8].

This thesis aims to find the most optimal protocol to be used in SCADA systems, because a SCADA protocol is needed for a project that executes a smart grid simulation which is created with the co-simulation framework Mosaik. The project and Mosaik will be discussed in more detail in this thesis. In addition, a look is taken at various SCADA protocols and a short review of their advantages and disadvantages is given considering the characteristics interoperability, reliability, ease of use, flexibility, security, and peer-to-peer capability. Furthermore, the protocols MQTT, a protocol that is potentially useful for future SCADA systems and IEC 60870-5-104, a standard SCADA protocol, will be compared directly in more detail. Moreover, both of them will be implemented into the smart grid simulation mentioned above, to compare the ease of implementation, and to have a working implementation for future work with the project. Then missing features of the implementations will be shown and their importance will be evaluated. Lastly, the conclusion is reached that MQTT has great potential as a SCADA protocol in the age of smart grids, especially if SCADA and the IoT are required to interact. However, MQTT is not able to replace the standard protocols as it is defined, due to a lack of SCADA specific definitions that are required for interoperability between field devices by different vendors.

Chapter 1 provides a brief introduction into the topics of smart grids, SCADA and SCADA protocols. Chapter 2 gives an overview of important concepts regarding SCADA, RTUs, and the IoT. In chapter 3 an overview of MQTT and various important SCADA protocols is given. The specifications and other aspects of the protocols MQTT and IEC 60870-5-104 are compared in chapter 4. Chapter 5 discusses basic concepts of Mosaik. Chapter 6 gives an overview of a smart grid simulation project and shows the implementations of MQTT and IEC 60870-5-104 within this project. The comparison and implementations will be evaluated in chapter 7. Lastly, a conclusion and possible future research ideas will be given in chapter 8.

CHAPTER 2

Background of SCADA

This chapter explains what SCADA is 2.1 and introduces the concept of RTUs 2.2 and the IoT 2.3 in the context of SCADA systems.

2.1 SCADA

SCADA (Supervisory Control and Data Acquisition) systems are systems that use RTUs to collect field data and send it via a communications system to a master station [11, 12]. SCADA systems are widely used to control industrial processes (e.g. power grids, gas pipelines, and so on), because they are ideal to connect and manage systems and devices that are distributed over a large area [12, 13].

As the name suggests, SCADA systems are used for supervision and control of the industrial processes as well as data acquisition from field devices [13]. These tasks are separated on the field network layer and the process control network layer [14]. The field network includes physical components like sensors, actuators, RTUs, and more. Actuators are used to directly control the industrial process (e.g. turning a valve on/off). Sensors record data (e.g. current value of a power line) that can be procured by RTUs or other controllers. The data is then transmitted to the process control network and commands are potentially received depending on the results of the analysis done by the SCADA system. The transmission can be done via a variety of communication systems like fiber, optic, radio, and so on [12, 27]. The process control network itself enables communication between data servers, RTUs, human machine interfaces (HMI), historical databases, potential enterprise networks, and other components [14]. Data servers are used as a focal point for communication with users via HMIs, with devices in the field via process controllers and with other data servers [13]. Enterprise networks can be connected via TCP/IP protocols to allow remote access to the process control network [14] which is especially interesting considering the IoT. A typical SCADA architecture can be seen in 2.1.

SCADA software is used to enable communication between hardware components and the SCADA system [12, 27]. This software can either be proprietary or open. The disadvantages of proprietary software are the reliance on hardware vendors and the limited interoperability which is the ability to use hardware from different vendors together [12, 27]. Therefore, open software became popular and is widely used in SCADA systems [14]. The SCADA communication uses communication protocols that are typically based on the master/slave principle, but depending on the protocol some kind of peer-to-peer communication may also be integrated. Another possibility is to use peer-to-peer communication exclusively.

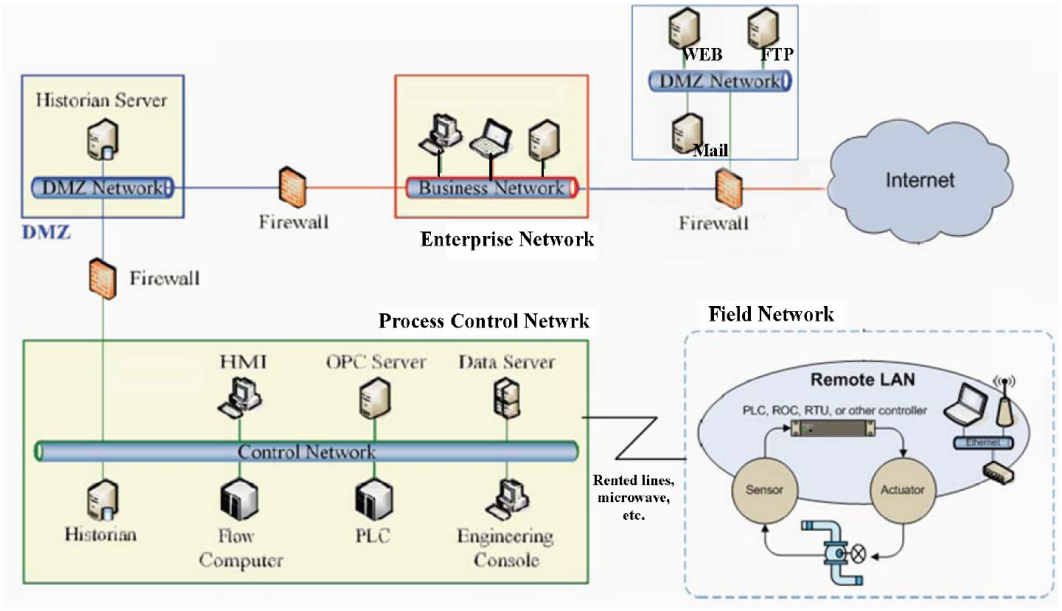

Figure 2.1: Typical SCADA System Architecture [14].

In a master/slave relationship, the slave is typically not allowed to send messages to the master, unless the master requested messages or if the right to freely send messages was granted to the slave. Peer-to-peer communication allows every client to communicate in an even relationship with every other client. The advantage of peer-to-peer communication is a reduced complexity, because the master is not required to handle the communication between slaves. A disadvantage of pure peer-to-peer communication are new attack opportunities. For example, a compromised client may not be recognized as such by a client that is communicating with the compromised client, although it was already recognized by another client, whereas in a master/slave communication, only the master has to know that a client is compromised to react accordingly (e.g. preventing communication of that client with other clients). Server/server and server/client communication often use TCP/IP protocols, while server and process controllers can be connected directly, via networks, or through a fieldbus system [13].

By providing data (e.g. a power supply current) polled from the process controllers and the data's change over time, SCADA systems allow operators to supervise the processes from which the data is acquired [13]. Another important feature is the automation possibility which means that the system can react to events with predefined actions (e.g. send E-Mails in case of an alarm), instead of having to wait for the command of an operator [13]. Furthermore, SCADA systems use user groups to manage read/write access on process parameters in the system as well as other functionalities [13] which provides a basic layer of security. As mentioned in 1 further security can be provided by using a local IDS [4] as extension of a centralized IDS. Other services SCADA systems provide are alarm handling, logging, archiving, and report generation [13]. To summarize, the advantages of SCADA systems are the efficient and automatic management possibilities provided by them.

2.2 RTU

RTUs (remote terminal units) are autonomous units within the SCADA system that control and collect data from remote devices for the central station [11, 17]. RTUs generally use microprocessors that can be configured from the central station [11, 17]. In addition, to communication between the central station and RTUs, it is also possible to utilize peer-to-peer communication between RTUs [11, 17]. In the case that the central station cannot communicate directly with an RTU, other RTUs can be used as a relay station. Programmable logic controllers (PLCs), distributed control systems (DCSs) and intelligent electronic devices (IEDs) are typically used with RTUs [11, 3 f.]. PLCs and DCSs serve as a bridge between sensors and a fieldbus, while IEDs are a combination of sensors, and PLCs and DCSs that are directly connected to a fieldbus [11, 3 f.]. Both techniques have advantages and disadvantages which have to be considered. Most notable advantages of IEDs as opposed to PLCs/DCSs are that an operator can see the sensor level and that only minimal wiring is needed [11, 3 f.]. For PLCs/DCSs the most notable advantages are that large amounts of data can be recorded and stored, and that thousands of sensors can be connected to them [11, 3 f.].

2.3 IoT

The IoT (Internet of things) consists of all kinds of physical objects (e.g. cars) that possess sensors and processing capabilities that can be accessed via networks [9]. A mundane example of these capabilities is using a smartphone application to start a washing machine remotely. These capabilities can potentially be used in SCADA networks, but traditional SCADA protocols (e.g. Modbus) do not support connections to IoT software [9]. It has to be mentioned that the combination of SCADA and the IoT leads to new security threats, some of which are explored in detail in [9]. Furthermore, a real attack on a car wash that abuses the IoT was already successfully made as described in [15].

CHAPTER 3

Background of SCADA communication protocols

This chapter explains the basics of SCADA communications in section 3.1. The chapter also discusses the legacy SCADA protocol Modbus 3.1.1, some protocols that are typically used for communication within a SCADA system today 3.1.2 and the protocol MQTT 3.1.3.

3.1 SCADA protocols

Protocols are an integral part of SCADA systems, as they not only ensure a correct and efficient communication between the central station and RTUs, but also between RTUs. Since the inception of SCADA several hundred protocols were created [2] to optimize the communication. As mentioned in 2.1 mostly TCP/IP based protocols are used, because TCP is designed to reliably send messages. This is important, as the SCADA system could react in a wrong manner, if the system does not work with the newest information or uses old information instead of new information, just because the information was lost or received in the wrong order. Additionally, to this, a general reliability is needed to prevent errors which is important to keep the system consistent and efficient. Besides using TCP, reliability can be achieved by checking for connection loss, using error correction (e.g. checksums), and more. Reliability can be measured by checking the amount of errors that happen in a clean implementation of the protocol. It has to be noted that not only the amount of errors has to be measured, but also their severity.

Considering other protocol traits, it is important for protocols to be interoperable. Interoperability means that devices made by different vendors can use the same protocol to communicate with each other without any need to map the protocols to each other. The complexity of the SCADA system would rise if devices from different vendors were not interoperable, because the implementation would have to deal with all the various, possibly proprietary, protocols, instead of a single well-defined standard protocol. Hence, interoperability reduces the complexity of creating SCADA systems which in turn reduces the amount of mistakes that can be made. The interoperability capabilities of a protocol can be measured by looking at the amount of vendors that support the protocol.

Furthermore, protocols should provide flexibility to be as useful as possible in as many use-cases as possible. Predefined flexibility allows freedoms in the implementation without breaking interoperability. Flexibility cannot be measured, but it can be evaluated if the flexibility provided by a feature has valid use-cases. For example, the quality of service level of MQTT 3.1.3 provides flexibility by enabling messages to be sent either with more

reliability (e.g. important messages) or better performance (e.g. frequent messages of no significant importance).

Additionally, protocols should be easy to use to reduce complexity and therefore the amount of human errors. Easier understanding of the protocol can be achieved by providing additional guides and reference implementations. Another option is to distribute the specification for free which might lead to the development of an open-source community. The open-source community not only provides free testing of the protocol capabilities, but also a hub for new users that are interested in learning how to use the protocol.

Protocols should also provide security as they are potential points of attack as explained in chapter 1. Security is difficult to measure and even if a protocol is used successfully in real systems for years, it might still not be secure enough in the future.

Lastly, protocols should support peer-to-peer communication, thereby being able to support more decentralized systems like the one introduced in 1 as well as systems that interact with the IoT [9]. Peer-to-peer communication can either be fully supported or be the extension of a master/slave system. The latter may not be enough for the use in decentralized systems nor in systems that interact with the IoT.

3.1.1 Modbus

Modbus was developed in 1979 by Gould Modicon (now Schneider) [12, 45] as a protocol for client/server interactions (e.g. monitoring) between intelligent devices and it became a de facto standard as it is widely used for industrial processes with hundreds of vendors supporting it [16]. Modbus is based on the master/slave principle that allows up to 247 slaves [12, 46], but it does not support peer-to-peer communication. As security was not the main concern of the Modbus design, several possible attacks exist (e.g. reading Modbus messages) [17]. Moreover, it is relatively slow compared to other protocols [12, 45ff].

Transactions are initiated by the master and they can either be addressed to a single slave or broadcasted to all slave addresses [12, 46]. A broadcast does not expect a response while a request expects an answer. Furthermore, a message can be sent in either the "ASCII" or the "RTU" mode [12, 46]. The "ASCII" mode is useful for testing, because the message contents can be read. However, as this is slower and because the messages are significantly longer than the "RTU mode" messages, it is not used in normal operation. Instead the "RTU" mode is used which represents a message in hex. Another notable thing is that new requests can only be sent after a response was received for the last request or if a timeout was recognized [12, 46]. Additionally, messages are ignored if they were changed during the transmission which provides basic security [12, 47]. Such changes are recognized by analysing error detection codes that are sent with each message.

Messages are sent in frames that are a series of bytes that can be separated into four sections [12, 46]. Figure 3.1 provides a visualization of the message frame format.

- The first section contains the address of the slave that the message will be sent to or the address of the slave, if the message is an answer to a request. Although 247 slaves can be addressed in theory, the maximum number is limited in practice with the normal case being only two or three slaves [12, 47].
- The second section contains the code of a function that is requested of the slave or the

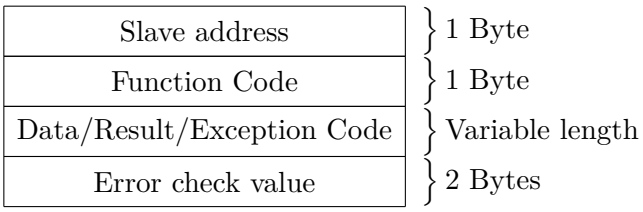

Figure 3.1: Visualization of the Modbus message frame format.

code of a function that was requested in case the message is an answer to the master. The functions are predefined (e.g. polling control functions) and they are designed as control commands for devices typically found in SCADA systems (e.g. actuators) [12, 46]. If an exception occurred, it is signaled by setting the most-significant bit of this section to one [12, 47].

- The third section contains data needed for the function, if sent as a request and the result of the function, if sent as an answer. This section will contain an exception code (e.g. 1 = requested function not supported), if an exception occurred [12, 56 f.].
- For both, requests and answers, a number that is needed to check the message for errors is contained in the fourth section. The number is calculated via a cyclic redundancy check (CRC-16) of the message frame [12, 47].

The four data types coils, discrete inputs, input registers, and holding registers are provided by Modbus [12, 48 f.]. The registers are two bytes each and the other two types are one byte each. Each of the types possesses function codes that are designed for them (e.g. 1 = read coil status).

As a measure of modernization Modbus was extended to be usable with TCP/IP, which is meant to reduce complexity and to make use of the established Ethernet [16]. To further increase usability, the Modbus protocol specifications and a guide are offered for free by the Modbus organization [18]. In conclusion the advantages are high interoperability and simplicity while the downsides are a lack of peer-to-peer support and security problems.

3.1.2 Standard Protocols

Currently the standard protocols that are typically used in SCADA systems are IEC 60870-5-101 and its sibling IEC 60870-5-104, DNP3, and IEC 61850. Although these protocols themselves may become legacy protocols in the near future, they are still widely used in many industrial processes (e.g. electrical) [10]. A short overview will be provided for these protocols in the following subsections.

3.1.2.1 DNP3

DNP3 was published in 1993 by the DNP3 User Group as a standard for SCADA systems that defines communication between control stations and RTUs [10]. DNP3 is widely used for industrial processes (e.g. electrical) around the world with the exception of the European

electrical distribution industry, where IEC 60870-5-101 is favored, and it is supported by a large amount of vendors in various industries (e.g. electrical) [12, 66]. DNP3 was designed for SCADA systems used in the electric distribution, but it also found use in the oil and gas, water/waste water, and security industries [12, 66]. Additionally, it was designed with interoperability in mind and to reliably transmit small packets of data, which is advantageous for efficient SCADA communication. DNP3, like IEC 60870-5-101, is based on early IEC 870-5 documents, which is why they share some similarities in lower level functionality [12, 65].

DNP3 provides several features that are useful for SCADA systems (e.g. time stamped messages, time synchronization, time-stamped events, user definable objects, and message broadcasting) [12, 69]. One of the features, the so-called quiescent operation, is the option to not only let a master poll for information, but to also let a slave report by exception or event [12, 70]. This is useful to reduce transmission frequency, as instead of regularly polling for information, new information (e.g. power line malfunctions or significant current changes) can be reported by the slave as soon as it is observed. To prevent undetected loss of connection, a periodic background poll is used to check if the connection to the slave stations still works as intended. It also has to be noted that the typical master/slave relationship is extended by allowing a slave to initiate communication for quiescent operations [12, 71].

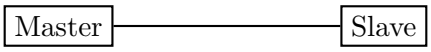

Figure 3.2: This diagram exhibits the master to slave topology.

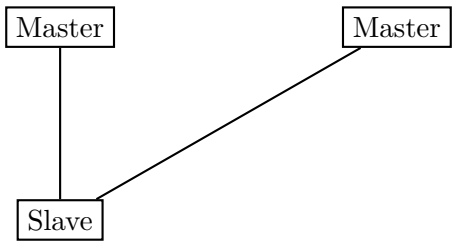

Figure 3.3: This diagram shows a topology with multiple masters.

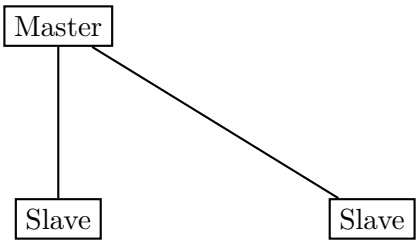

Figure 3.4: This diagram exhibits the multiple slave topology.

Another strong point of DNP3 is that it supports peer-to-peer besides the traditional master/slave relationship [10]. The master/slave principle (see figure 3.2) is also extended by allowing multiple masters (see figure 3.3), multiple slaves per master (see figure 3.4), and

the possibility of a slave station being the master of other slave stations (see figure 3.5) [12, 70]. Although peer-to-peer communication is provided by allowing slave stations to initiate communication, one of the station is always designated as a master station [12, 71]. Moreover, only the master station can request data or send commands.

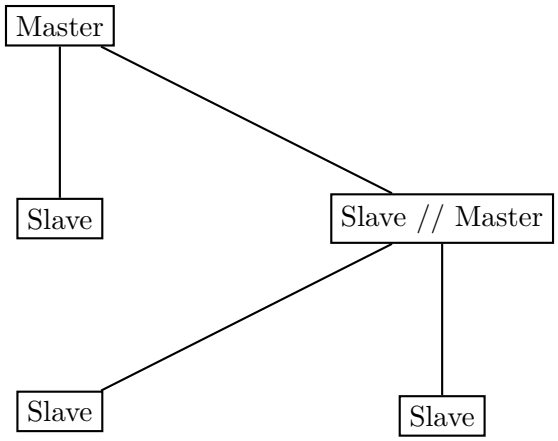

Figure 3.5: This diagram shows a hierarchical topology.

The DNP Users Group also provides some guides to help with the relative complexity of DNP3 [19]. Furthermore, an open-source implementation [20] is provided by Automatak LLC. As mentioned in 1, [8] found DNP3 to be in need of extension to be usable for decentralized systems. As a conclusion, the advantages are high interoperability, native TCP/IP compatibility, reliable communication optimized for SCADA systems, and seemingly no major security concerns. Although peer-to-peer communication is supported, it might not be sufficient for the IoT, because DNP3 was not designed with the IoT in mind and because the peer-to-peer communication is not the main focus of the protocol (e.g. one of the stations has to be a master station).

3.1.2.2 IEC 61850

IEC 61850 is a standard designed for communication within a substation automation system [10], which is useful for RTUs. IEC 61850 was also designed with interoperability, support for security, high-speed IED to IED communication, use in electrical systems, TCP/IP, and more in mind [21].

To achieve interoperability, IEC 61850 provides abstract data objects which can be mapped to any protocol that meets the requirements that IEC 61850 imposes concerning data and services [21]. Additionally, a substation configuration language (SCL) is provided that is used to describe the relationship between substations and the system [21]. If a device provides an SCL file, it can be used to configure the device automatically, which prevents human errors and reduces the amount of configuration effort. The SCL as well as object orientation are a unique approach compared to standard SCADA protocols, as these typically define how data is transmitted, but not how it is handled within a substation [21]. Another interesting feature of IEC 61850 are Generic Object Oriented Substation Events (GOOSEs) that allow fast inter-substation peer-to-peer communication [10].

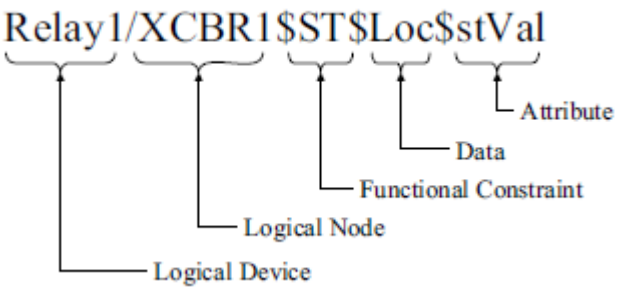

Figure 3.6: IEC 61850 object name anatomy [21]

In the IEC 61850 device model, devices are separated into physical devices that are connected to the network, and logical devices that belong to a physical device [21]. A physical device can act as a data concentrator by being used as a proxy for multiple devices. A logical device can contain several logical nodes, which are named groups of data elements and services related to the data (e.g. the logical nodes beginning with the letter C are used for supervisory control) [21]. If two logical nodes are used for the same purpose they are differentiated by adding a number as a suffix. The data elements also possess a name, which is defined by the standard and is related to their functionality (e.g. Loc = indicates if the operation is either remote or local) [21]. Furthermore, the data elements correspond to the common data classes (CDC), which are used to describe the type and structure of data elements (e.g. CDC for measured information). A CDC consists of a defined name and attributes which themselves have a defined name, a specific purpose, and a defined type [21]. The attributes are further grouped into sets of functional constraints (e.g. status attributes). Lastly, an abstract view of a device is mapped to a specific protocol stack, based on MMS (ISO9506), TCP/IP, and Ethernet [21]. To achieve this, the information of an abstract device is mapped to unique object names for every data element (see figure 3.6). It is also possible to map IEC 61850 to other protocols. Profiles that are designed based on the service that is to be mapped, are provided for the mapping process to other protocols.

Although IEC 61850 is only defined for use within a substation, [10] suggests to map IEC 61850 on other protocols such as DNP3 or IEC 60870-5 to make it possible to use its special features in a SCADA system without taking much of a performance hit. [10] also suggests to make use of GOOSE on a bigger scope than substations. The advantage of this would be to be able to quickly transfer important information. In total, IEC 61850 could potentially be used as an useful extension for SCADA protocols or it could be extended to be usable as a standalone SCADA protocol, because it provides interoperability (e.g. object orientation), security, ease of use (e.g. SCL), peer-to-peer communication, the option to use TCP/IP, and unique features (e.g. GOOSE).

3.1.2.3 IEC 60870-5-101 and IEC 60870-5-104

IEC 60870-5-101 provides a standard for a SCADA communication protocol under IEC 60870-5 [12, 177]. Furthermore, it defines data types, commands, and other communication

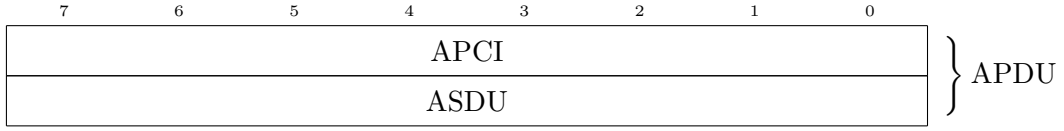

Figure 3.7: IEC 60870-5-104 APDU consisting of the APCI and an ASDU.

details that are needed for communication with RTUs in electrical systems [12, 177]. IEC 60870-5-101 itself is expanded by IEC 60870-5-104, which extends IEC 60870-5-101 by using TCP/IP as the underlying transport protocol, by setting a specific value for some parameters (e.g. information object address length) that were variable in IEC 60870-5-101, and more [22]. Although IEC 60870-5-104 allows peer-to-peer communication [22], it is only minimally specified (e.g. peers switch the role of master and slave to conduct peer-to-peer communication), which could be problematic considering interoperability. As mentioned in 3.1.2.1 IEC 60870-5-101 and IEC 60870-5-104 are mainly used in European electrical distribution systems, and they are pretty similar to DNP3 on a low level, because they are based on early IEC 870-5 documents.

In IEC 60870-5-104 packets are sent as application protocol data units (APDUs), which are up to 255 bytes long and include not only data, but also meta information [22]. Each APDU consists of the application protocol control information (APCI) and an application service data unit (ASDU) (see figure 3.7). The APCI contains the start byte, the length of the APDU, and 4 control field octets (see figure 3.8). The start byte is set to 68 in hex. It has to be noted that the length can only be up to 253, because the start byte and the header are not counted explicitly. The control fields contain one of the three types of frame formats that are explained in the following list:

- The first format is the i-frame format, which is used for information transfer. Hence, APDUs using this format always contain the ACPI and an ASDU. The i-frame itself includes 2 bits to check the frame type, the 15 bit long send sequence number (SSN), and the 15 bit long receive sequence number (RSN). Both, the masters and slaves, can check if the amount of sent and received messages is equal by comparing their SSN/RSN with the RSN/SSN received. It has to be noted that as long as an APDU was not received, it is stored in a buffer for further actions (e.g. resending).
- The second format is the s-frame format, which is used for numbered supervisory functions. APDUs using this format only consist of the APCI. The s-frame itself includes 2 bits to check the frame type and the 15 bit long receive sequence number (RSN).
- The last format is the u-frame format, which is used for unnumbered control functions. APDUs using this format only consist of the APCI. The u-frame itself includes 2 bits to check the frame type. The other 6 bits of the first control field are used to determine the control function type and whether the function is used as a command or as a confirmation of a command of the respective type (e.g. if the fourth bit is set the function to start data transfer is confirmed). The three functions that are available are used to indicate a test, to start data transfer, and to stop data transfer respectively.

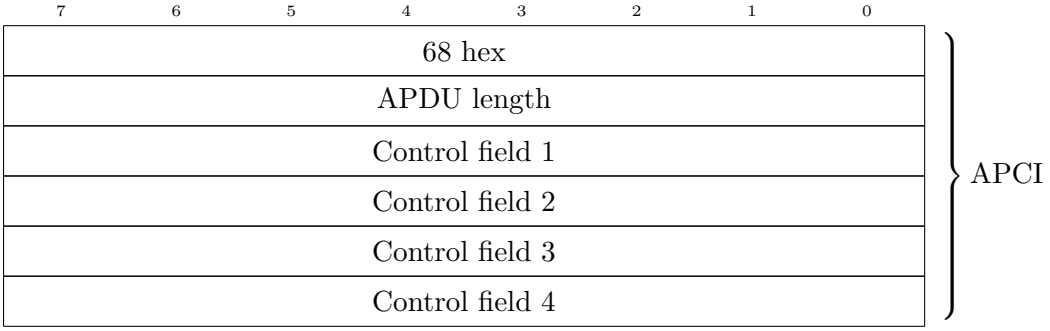

Figure 3.8: IEC 60870-5-104 APCI consisting of the start number, APDU length, and control fields.

The ASDU consists of the type identification, the variable structure qualifier, a cause of transmission, a common address, and information objects (see figure 3.9), which are explained in more detail in the following list:

- The type identification is one byte long and indicates the type of the ASDU. Each type is mapped to a number (e.g. 45 = single command), and every type is designed to be used by either masters or by slaves. Additionally, many types are available with and without time-tags.
- The variable structure qualifier is one byte long and contains the amount of information objects that are being sent via the APDU. One bit is reserved for the sequence bit, which indicates if a sequence of information objects is sent or not. If the bit is set, all information objects of the APDU are addressed as one object. However, if the bit is not set, all information objects can be addressed individually.
- As the name indicates the cause of transmission includes the cause of transmission. However, it also includes a bit to indicate if the APDU is only a test APDU and a bit that is used to confirm if a command was executed successfully. The causes of transmission are mapped to the numbers that can be represented with 6 bits (e.g. 5 = request or requested). Additionally, the cause of transmission includes a byte that is used as an originator address. The originator address is used to identify the origin of commands, interrogations, and so on. If the originator address is not needed, it can be ignored by setting it to 0.
- The common address can be used to address single stations, every station at once, or even substations. The common address is a number represented by 2 bytes. A broadcast is indicated by setting all bits to 1.
- Information objects consist of an information object address (IOA) and depending on the ASDU type they may also contain data (e.g. a float value) and information elements. Information elements are defined for various purposes and in various formats. An example of an information element would be the quality descriptor, which has a bit that is set if an overflow occurred. If the sequence bit is set, only one IOA is used for every information object, and if the sequence bit is not set an IOA is used for every information object. As an IOA is of 3 bytes length, the use of a sequence approximately doubles the amount of data that can be sent with a single APDU, at the cost of losing direct addressing.

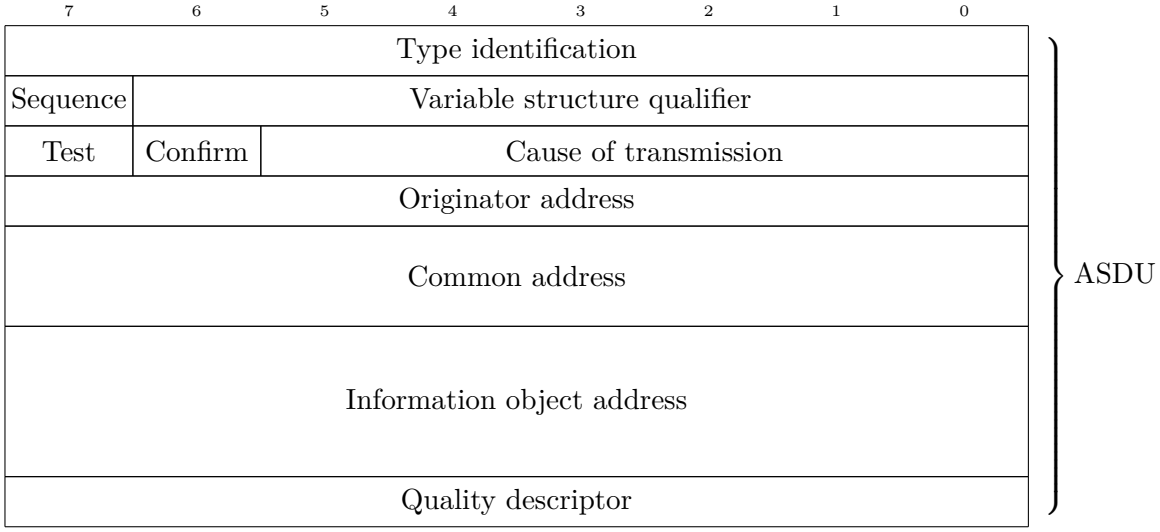

Figure 3.9: Exemplaric IEC 60870-5-104 ASDU consisting of the type identification, variable structure qualifier, cause of transmission, common address, and a single information object that only contains a quality descriptor.

All in all, IEC 60870-5-104 provides good interoperability, basic peer-to-peer communication, TCP/IP usage, and there are seemingly no major security concerns. However, similar to DNP3, peer-to-peer capabilities might not be sufficient for the IoT, because neither IEC 60870-5-101 nor IEC 60870-5-104 were designed with the IoT in mind.

3.1.3 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight extension of TCP/IP that uses a publish-subscribe model [9]. This means that MQTT clients can publish a message to a topic on a MQTT server which will then be forwarded by the server to all MQTT clients that are subscribed to the topic [23]. MQTT topics look like this: "rtu/data/voltage angle". "/" is used as a separator of topic levels, which is needed for the wildcard characters "#" and "+" [23]. "#" has to be the last character of a topic and is used to represent the parent topic level, as well as any child topic level that could be created. For example, the topic name "rtu/data/#" would include the topics "rtu/data/current", "rtu/data/substation1/current" and "rtu/data". It is also possible to only have "#" as a topic name, which would mean that every message will be received. The other wildcard "+" can be used on any topic level, more than once in a topic, and together with the "#" wildcard [23]. The "+" has to occupy the entire level when it is used and it represents the topics that can be made when replacing the "+" with a normal identifier. For example, the topic name "rtu/data/+/+" would include the topic "rtu/data/substation1/current", but not "rtu/data/substation1/current/branch1".

MQTT was designed with security in mind and the specification provides suggestions for this. Transport layer security (TLS) is suggested to be used for secure data transmissions and to verify the integrity of clients and servers [23]. Additionally, hash algorithms provided by TLS can be used to verify the integrity of data. It is also suggested that clients should

be authenticated by the server [23]. For this purpose the CONNECT packet, which is used to establish a connection to the server, includes a field for a username and password. Furthermore, it is suggested to restrict the access right of clients to a server's resources, by using the information provided for the authentication of a client. The specification also mentions several other possible security aspects that are left to the implementation (e.g. the detection of abnormal client behavior).

Another important feature of MQTT is the Quality of Service (QoS) [23]. The QoS is a variable for publishing and subscribing that can be set to either 0, 1, or 2 and determines how a message is sent and how the receiver will respond to the message. The following list explains the levels in more detail:

- A level of 0 means that a message is sent only once and that no response will be sent [23]. Hence, it is not guaranteed that a message will be received, but the performance is optimal.
- A level of 1 means that a message is guaranteed to reach the receiver at least once [23]. To achieve this, the receiver has to send a PUBACK message to acknowledge the receiving of the message. After acknowledging the message, the receiver has to treat incoming messages with the same identifier as the acknowledged message as new messages. This can lead to a case where the same message is sent more than once and being accepted every time. Therefore, at least twice as many messages will be sent compared to level 0, which decreases the performance.
- Finally, a level of 2 means that every message is guaranteed to be sent and received exactly once [23]. This is achieved by sending three extra messages. First, the receiver acknowledges that the message was received by sending a PUBREC message. The sender acknowledges this by sending a PUBREL message as answer. Lastly, a PUBCOMP message is sent by the receiver that acknowledges the sender's 2nd message. The performance suffers due to the additional overhead that is introduced by sending three extra messages for every normal message.

MQTT is a useful protocol in the context of the IoT [9] and it is already being used by companies like Amazon [24], where MQTT is used as a message server that links cloud applications and devices that are part of the IoT (e.g. sensors) via responsive long-lived connections. This is a use case that is pretty similar to those SCADA systems in which data is constantly requested from other stations. Furthermore, legacy devices can be used with MQTT via edge-of-network devices and in combination with MQTT-enabled-message-oriented middleware. With these, the control stations and the remote units can be decoupled and instead communicate via MQTT [25]. The use of MQTT-enabled-message-oriented middleware simplifies peer-to-peer communication, lifts middleware duties from control stations, and allows easier integration of enterprise software [25].

In an effort to increase interoperability MQTT was standardized [26], but as the specification is not targeted at SCADA, no special data types or the like are supported [23]. In addition, open-source implementations of MQTT clients and servers are supported by the Eclipse Foundation [27], which provides a simple way to use MQTT. To conclude, MQTT provides an easy to use TCP/IP based communication protocol that allows peer-to-peer communication, communication with enterprise software, and inclusion of legacy devices. Additionally, MQTT has seemingly no major security concerns and is even designed with

security in mind, and it is suited to be used in the age of the IoT. The downside of using MQTT in SCADA is the lacking native support of SCADA.

CHAPTER 4

Comparison: IEC 60870-5-104 vs. MQTT

This chapter compares the protocols IEC 60870-5-104 and MQTT in terms of interoperability, ease of use, security, flexibility, reliability, peer-to-peer capabilities, and other advantages or disadvantages. These characteristics and their importance were discussed in 3.1. IEC 60870-5-104 was chosen as a representative of the more traditional SCADA protocols, while MQTT seems to be a promising SCADA protocol for the future. In section 4.1 the specifications will be compared and in section 4.2 other notable characteristics will be considered.

4.1 Specification Comparison

First of all, a look at interoperability is taken. IEC 60870-5-104 was specifically designed for SCADA systems, therefore providing three types of frames and many types of ASDUs (see 3.1.2.3) that are specified for special purposes within the SCADA system (e.g. requesting data as a float, sending unnumbered control functions, and more). On the other hand MQTT was not designed for SCADA, hence it does not provide special message types for SCADA systems.

Another interoperability advantage of IEC 60870-5-104 is the originator address that is used to identify the origin of commands, interrogations, etc., while MQTT does not natively support something like this. A workaround for this would be to have a message's origin as part of the topic, but as this is not specified in a standard, it may be bad for interoperability. It is also important to mention that the originator address only supports up to 255 originator addresses which may be a disadvantage if more are needed (e.g a system with many master stations that are needed for a SCADA system with a decentralized IDS).

Although IEC 60870-5-104 is focused on interoperability, there is still some flexibility provided by offering hundreds of APDU types that can be send for a variety of causes. This is not quite as flexible as MQTT which, due to a lack of interoperability efforts for SCADA systems, has no such restrictions. However, this is a worthy tradeoff considering the amount of interoperability this provides for IEC 60870-5-104.

In IEC 60870-5-104, the controlling station has to send the APDU types containing unnumbered control functions, to either start or stop the data transfer with a controlled station. The controlled station has to acknowledge this by sending a APDU with a start or stop confirmation respectively (see figure 4.1). This allows the controlling station to have more nuanced power, as a connection does not have to be stopped to prevent data transfer. In MQTT messages can be published freely after the connection was established. However, the receiving of data can be stopped by unsubscribing from the relevant topics. It has to be

noted that the server would still receive the data which should be prevented by notifying the publishing client, if its data is not needed anymore. However, it is not specified how to handle this case, hence MQTT offers less interoperability in this regard.

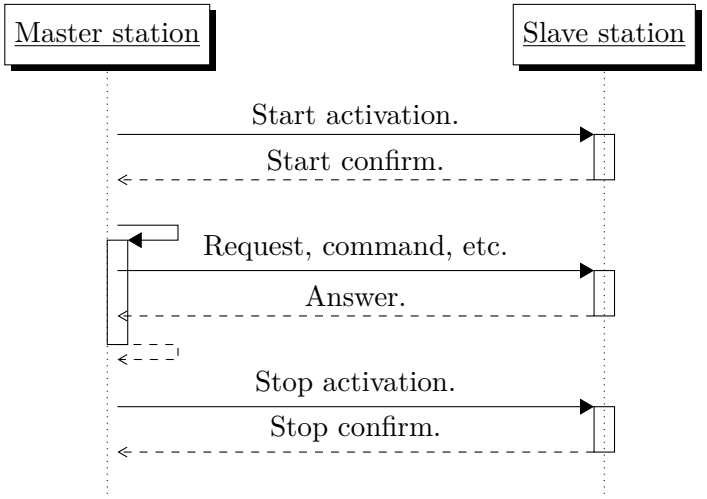

Figure 4.1: Typical data transfer workflow between two stations via IEC 60870-5-104. The same workflow is true for peer-to-peer communication, where one of the stations has to be set as master.

Next up, security will be compared. Because IEC 60870-5-104 uses TCP, and because MQTT is based on TCP, it is possible to use TLS as a security measure. TLS is used to encrypt the packets that are sent via TCP which is a counter measure to the interception of packets. For MQTT this is even suggested in the specification. MQTT also specifies the utilization of usernames and passwords, gives security suggestions, supports client authentication, and supports access control lists (see 3.1.3), while IEC 60870-5-104 provides nothing comparable to this.

Another feature of MQTT is the QoS as described in section 3.1.3. The QoS feature provides flexibility by being able to easily trade performance for reliability, depending on the specific use case. For example, values (e.g. current values of a power line) that have to be exchanged often should be sent with a QoS of 0, provided an occasional loss is acceptable. Messages that could lead to serious damage if lost (e.g. a power line malfunctioned) should be sent with a QoS of 1 or 2. However, the QoS of 1 can only be used if the multiplication of a message can be handled which might be complicated depending on the use-case.

It has to be mentioned that while QoS 2 guarantees that the message is received, that only means that a published message was received by the server or that a message that was sent by the server was received by the subscriber. Thus the publishing client does not know if its message was received by a subscriber. For example, if a subscribing client misses several messages for a certain topic due to a timeout, it can at maximum receive the last message that was published to the topic with the retained flag set to 1. For context, the retained flag ensures that the server saves the last message that was sent on a certain topic. IEC 60870-5-104 handles this more reliably by sending a SSN and RSN with every APDU as described in 3.1.2.3.

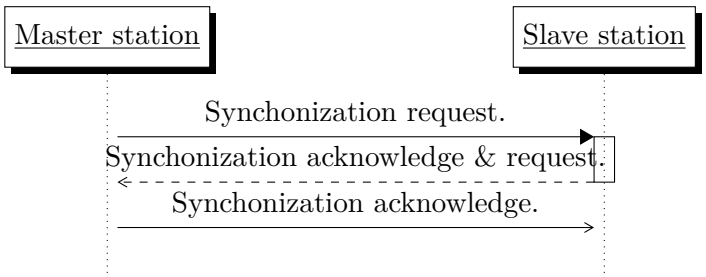

Figure 4.2: IEC 60870-5-104 connection establishment workflow. The same workflow is true for peer-to-peer communication, where one of the stations has to be set as master.

Figure 4.3: MQTT connection establishment workflow.

Both, IEC 60870-5-104 and MQTT, provide rules for connection establishment. In IEC 60870-5-104, first either the controlling station establishes the connection to a controlled station or in case of peer-to-peer communication one of the stations that is determined via a fixed selection parameter establishes the connection. In IEC 60870-5-104, the connection is established with a three-way-handshake (see figure 4.2). MQTT demands from the clients to send a connect message to the server which will be answered with a CONNACK message (see figure 4.3). The CONNACK message returns a number representing either success or various reasons why a connection was refused. To conclude, IEC 60870-5-104 has a more reliable connection establishment method, while MQTT has a more flexible one, because the reason for connection refusal can be analyzed to determine an appropriate reaction.

MQTT also defines a will feature. When a client connection was accepted, a will message can be stored on the server which will be published in case of an unintentional disconnect of the respective client. This provides another layer of reliability.

Additionally, subscriptions to a topic are answered with a SUBACK message by the receiver. If the subscription was successful, the SUBACK message will contain a return code that informs the subscriber which QoS level was maximally granted to the subscription. If the subscription failed, an error code will be returned instead. This feature also improves the reliability of MQTT.

On the other hand IEC 60870-5-104 also provides a feature improving reliability. Commands are required to be answered with a confirmation of whether the command was successfully executed or not.

Finally, peer-to-peer communication will be considered. As mentioned in 3.1.2.3, IEC 60870-

5-104 is not focused on peer-to-peer communication and therefore less capable in this regard compared to MQTT, where peer-to-peer communication can be easily achieved by using MQTT-enabled-message-oriented middleware as mentioned in 3.1.3.

4.2 Other Notable Things

MQTT's publish-subscribe model and the topic feature are intuitive and easy to use. In addition, as mentioned in 3.1.3 there are open-source implementations of the MQTT client and server as well as introductory guides [28], and an open specification. This is a big help for beginners, whereas IEC 60870-5-104 neither offers anything of comparable quality to the open-source implementations and the guides nor does it offer an open specification. On top of that it is required to at least read IEC 60870-5-101 before reading the IEC 60870-5-104 specification and even then, there are still parts that are only covered in detail in the remainder of the IEC 60870-5 specifications. Therefore, beginners will have an easier time with learning MQTT. Although MQTT has proven to be useful in real applications 3.1.3 and may be interesting for SCADA considering the IoT [9], IEC 60870-5-104 has already proven its value as a SCADA protocol 3.1.2.3.

CHAPTER 5

Background of Mosaik

This chapter introduces the basic characteristics and components of the Mosaik framework.

5.1 Mosaik

Mosaik [29] is an open source co-simulation framework written in Python 3 and published by OFFIS [30]. Mosaiks tries to synchronize multiple simulations and exchange data between them, to perform more advanced simulations. The main intention is to support the simulation of a smart grid scenario, of which an example schematic can be seen in figure 5.1. To achieve this, several features are provided by Mosaik [30]. First of all, an API to enable communication between Mosaik and simulators is provided. Furthermore, handlers for various simulation processes are provided. Additionally, tools are provided that are used to create simulation scenarios that involve several simulations. In addition, a stepwise execution schedule is created for the various simulations involved. Lastly, data can be exchanged between two simulators. It is also notable that the API is language agnostic which means that the simulators can be written in any desired language [30]. Another feature of Mosaik is the web visualization, where a representation of the scenario can be seen via a web browser. The representation is a graph with colored nodes, where each color represents one of the simulators and with edges that represent a connection between two simulators. An exemplary web visualization is also a part of figure 5.1.

Figure 5.1: Schematic of a typical Mosaik use case [30]. Blue represents household simulators, green PV simulators, gray a load flow analysis tool, and yellow a monitoring and analysis tool [30]. A coordinated simulation can be started after the four simulators were connected to Mosaik. A visualization of the simulation can be seen in the last step of the schematic which uses the same color scheme as before.

The aforementioned features are implemented in the four main components of Mosaik [30]. To facilitate communication between simulators and Mosaik, a communication protocol is provided by the Mosaik Sim API. As can be seen in figure 5.2 a low-level and a high-level

API are provided. The low-level API uses sockets to exchange JSON encoded messages between a simulation and Mosaik, while the high-level API provides an abstract base class that takes care of the socket communication (i.e. serialization of the messages and so on) [31].

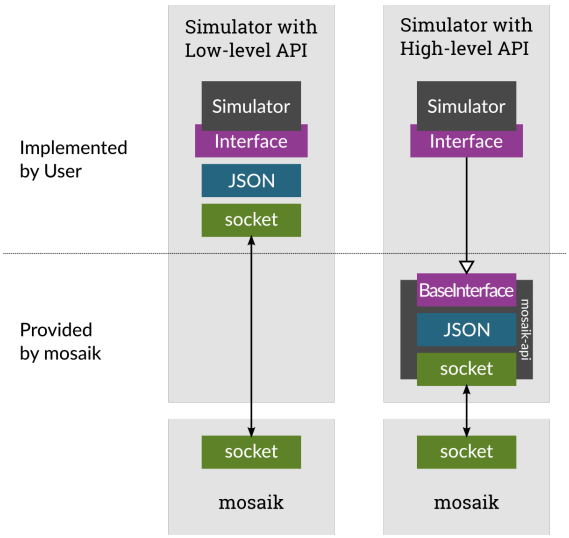

Figure 5.2: Overview of the low-level and the high-level Mosaik Sim API [31].

The scenario API is used to create simulation scenarios in Python [30]. Three steps that create the scenario can be distinguished [32]. First, all involved simulators are started. After that, models are instantiated within the simulators which are represented by entities that are connected in the third step to establish a data flow between the simulators.

Next up is the simulator manager (SimManager), which handles the simulation processes as well as the communication with them [33]. The SimManager has three different ways to interact with simulators that can be seen in figure 5.3. The SimManager can either start a simulator process itself, connect to a running simulator process, or import a simulator module that was written in Python 3. Importing a simulator module has the advantage that network communication is not needed anymore, because the simulators can be executed like a normal Python 3 module. This leads to an increased performance, because the computational overhead associated with network communication can be avoided. However, this method is only useful as long as the simulator is computationally inexpensive, because all simulator processes are run in a single thread [33]. Therefore, simulator processes that are computationally expensive should be started separately, to be able to run the processes in parallel.

Lastly, the Mosaik simulator is used to coordinate the scenario simulation [30]. The Mosaik simulator can handle simulators that use different and even changing step sizes. Furthermore, the Mosaik simulator can run simulation steps in parallel and it can also control simulators by only letting them perform a simulation step if needed (e.g. data is needed for another simulator). This is used to synchronize simulators by only allowing a simulation step, if all the necessary input data was calculated [34].

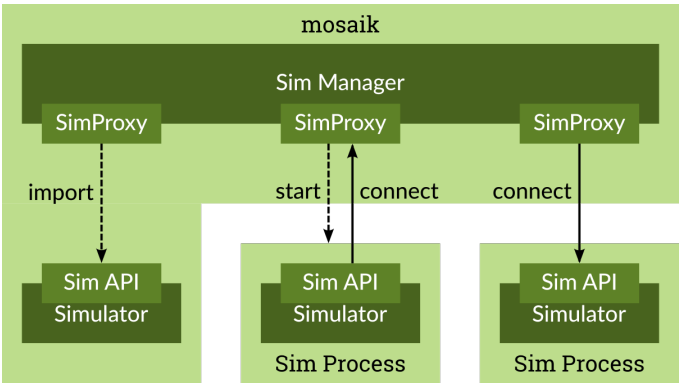

Figure 5.3: Overview of the three options that are provided by the simulator manager to use simulator processes [33]. Note: SimProxy objects mask the simulator process kind with the result that other Mosaik components do not know which of the three options is being used [33].

CHAPTER 6

Implementation of IEC 60870-5-104 and MQTT

First, this chapter will give a short overview in section 6.1 of the project that the protocols will be implemented in, and then interesting details of the implementations will be presented in section 6.2 and 6.3 for MQTT and IEC 60870-5-104 respectively.

6.1 Project Overview

The project in question is a simulation environment for smart grids implemented in Python 3.4 [35]. The smart grid co-simulation framework Mosaik that was discussed in section 5.1 is used to coordinate the various subsimulations by passing values between them in every simulation step. The simulation expects a smart grid that is simulated over a specific time period with adjustable time steps. A smart grid consists of branches that represent power lines, nodes which represent points of connection, houses that can either consume power from the grid or provide power to the grid, and PVs that can provide power to the grid. One of the nodes, the so-called RefBus, represents a connection to either the overarching grid or to any other kind of a power source (e.g. a power plant). The branches are connected with each other via nodes, while the houses and PVs are connected to the nodes. In addition, more than one branch can be between two nodes which represents alternative branches. Additionally, Mosaik provides a visual representation of the grid used in the simulation as shown in figure 6.1.

To simulate a SCADA system, RTUs can be placed on nodes which are used to interact with sensors and switches that belong to the nodes as well as sensors on branches that are connected to the respective nodes. The sensors can be used to read the current physical values (e.g. voltage angle) that the simulation provides for the branches and nodes. The switches are used to turn branches on and off. The central station is represented by a simple graphical user interface (GUI) (Fig. 6.2) for operators which gives status updates and can be used to interact with the RTUs. In addition, a local IDS based on [4] and extended by parts of [5] is provided for the RTUs. A simple TCP connection is used for interaction between the GUI and RTUs, while the RTU to RTU communication that is needed for some functions of the local IDS, is only partially based on TCP and instead abuses that the simulation has direct access to all the RTUs. It is also notable that RTUs are identified by an entity identification of the form "X-rtu", where X is a number that is unique for every RTU.

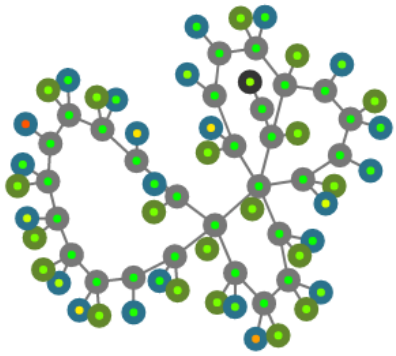

Figure 6.1: Grey lines are branches, green circles are PVs, blue circles are houses, the black circle is the connection to the overarching grid, and the gray circles are the nodes.

Figure 6.2: The reset button is used to reset an IDS internal value for every RTU. Above the reset button, messages are printed that are created during the simulation (e.g. attack reports).

The implementation of MQTT and IEC 60870-5-104 will therefore be used to simulate real communication between the GUI and RTUs as well as for inter-RTU communication. For this to work, certain functionalities have to be adapted. This includes the broadcasting of attack reports, resetting certain RTU values, determining if RTUs are adjacent, and retrieving voltage angle values read on nodes of other RTUs. It has to be noted that the RTUs' IDSs start the clients and servers for every RTU, while the operator GUI uses the so-called operator tools to start its client and server. It is also possible to easily swap between MQTT, IEC 60870-5-104, and the previous way of communication.

6.2 MQTT implementation

The implementation of MQTT uses the Eclipse Mosquitto™ server [36] and the Eclipse Paho™ MQTT Python Client [37] as a foundation. The server provided is a fully functioning MQTT server, while the client provides functions that implement functions detailed in the MQTT specification as well as some additional features (e.g. an event loop to run a client). Callback functions are executed by the client on certain events (e.g. when a message was published). The functionalities described in 6.1 are executed when the on_message call-

back is invoked. Although it is to be expected due to the server and client being based on the MQTT specification, it is still mentionable that both, server and client, are compatible with each other out of the box and that they are able to handle the communication details.

Within the project, a client is started for the operator GUI as well as a client for every RTU. The clients are run in the event loop which itself is run in an extra thread and they connect to the local address for unsecured MQTT servers by default. However, other addresses (e.g. iot.eclipse.org which is a public server provided by Eclipse) can be used as well. The server has to be started before the simulation to prevent the clients from being unable to connect. It is also important to mention that after sending the connect message and before starting the event loop, a short wait is introduced to make sure that the client is connected before anything is sent. Within the project 3 seconds wait time for each client were enough to safely establish the connection. The on_connect callback gives feedback on whether the connection was successful or not. If it was not successful, the reason, if known, will be returned. After successfully connecting to the server, the RTU clients subscribe to the topics used for commands by the operator GUI and the topic that is used to publish branch names. The topics are "operator/command" and "data/+/branch" respectively. The operator GUI client only subscribes to the attack broadcast topic which is "broadcast/+/attack". An overview of the communication components can be seen in diagram 6.3.

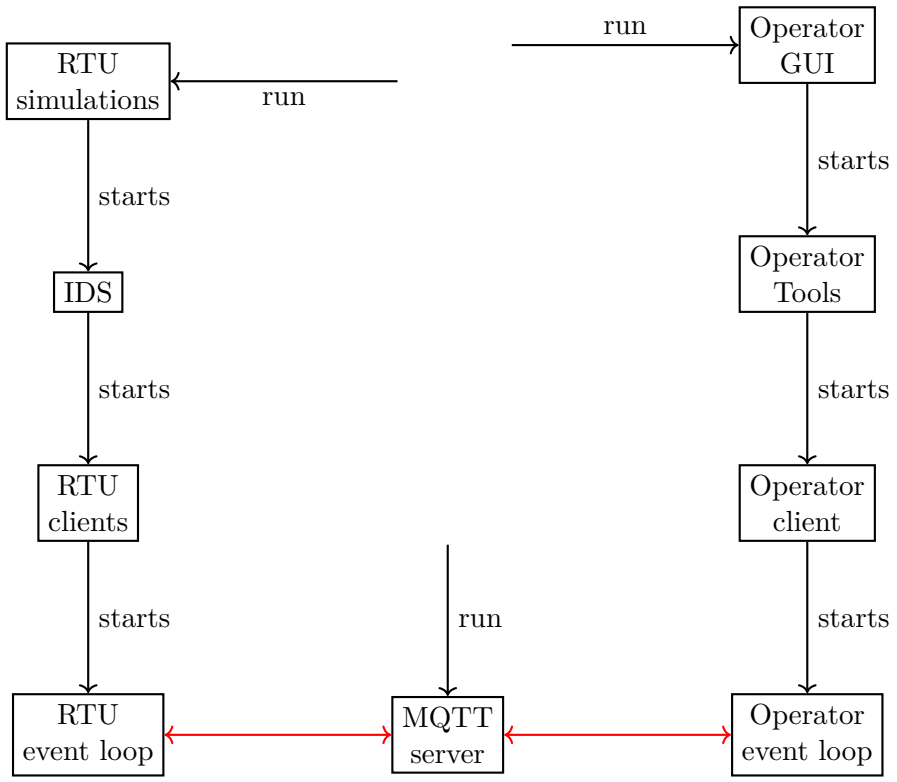

Figure 6.3: This diagram exhibits the components directly involved in the MQTT communication. Red arrows represent communication.

The first functionality that is executed by the RTU clients is the determining of ad-

jacent RTUs. To achieve this, every RTU publishes its branches' names to the topic "data/X/branch", where X represents the entity identification of the RTU. Then the other RTU clients can check if they share a branch with the clients of any other RTU, in which case they let the IDS save these RTUs as adjacent RTUs. It has to be noted that every RTU client waits for the other clients to be started to make sure that every RTU receives all branches. After every client was started the publishing is executed. The message is sent and received with a QoS 1, as this is a one time cost and should be received at least once, because some of the logic relies on the knowledge of adjacent RTUs. Additionally, the case of receiving a message more than once is no problem, as it is only important to know if a branch is shared by two RTUs to know if they are adjacent or not. It has to be mentioned that this also triggers the subscribing to the topics used to receive voltage angles and attack broadcasts from adjacent RTUs. The topics are "data/X/voltageAngle" and "broadcast/X/attack", where X is the entity identifier of an adjacent RTU. See the message sequence chart 6.4 for a visual representation.

The voltage angle will be published to the topic "data/X/voltageAngle", where X represents the entity identification of the RTU. This happens in every simulation step before the IDS logic is executed. Hence, the IDSs always have the most up to date voltage angle values of adjacent RTUs to work with. As this message has to be sent often, it is sent and received with a QoS of 0, because it keeps the performance cost low and the loss of a single value is acceptable for the IDS. QoS 1 would also be interesting for this, but it would be hard to deal with a message being received more than once. The message sequence chart 6.5 provides a visual representation.

Attack reports are normally broadcasted to every RTU and the operator GUI, but only adjacent RTUs and the operator GUI react to the report. Hence, only the adjacent RTUs and the operator GUI are subscribed to the topic "broadcast/X/attack", where X represents the entity identification of the RTU that sends the report. However, this can easily be changed to a real broadcast if necessary. The RTU client will react to the attack reports by changing certain internal values of the IDS that are out of scope of this thesis and the GUI will print the attack reports for the operator. The attack reports are sent and received with a QoS of 2, as these reports are important for the work of the IDS and they are rarely sent which keeps the performance cost negligible. 6.6 shows this process in a message sequence chart.

Lastly, a reset command for every RTU is published by the operator GUI client to the topic "operator/command", if the user requests it via a button in the GUI. The command is forwarded from the GUI to the operator client via the operator tools. Then the reset will be executed by the IDS of an RTU as soon as the RTU client receives the command. This is a command by the central station and should therefore be guaranteed to be received. Hence, the QoS level used is 2. The message sequence chart 6.7 shows the workflow of this functionality.

Although the server can be used as preconfigured, there are several interesting features (e.g. using TLS) that can be used with some further configuration. As a proof-of-concept a simple smart grid was created, in which the username/password and access control list

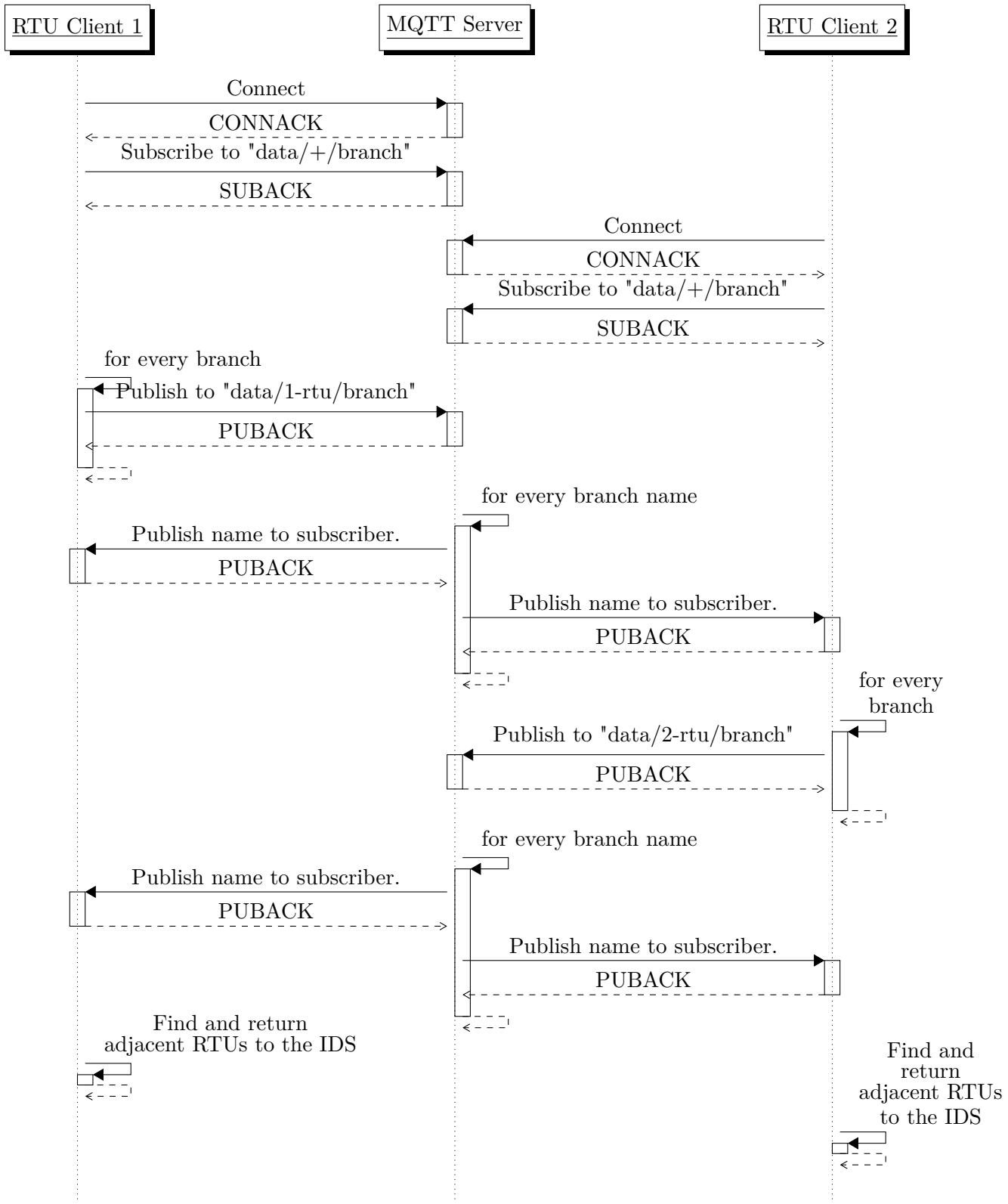

Figure 6.4: Workflow to determine adjacent RTUs in the MQTT implementation. Note: MQTT servers send a CONNACK message to confirm connection establishment and a SUBACK message to confirm subscriptions. A PUBACK message is used to confirm publishing of messages published with a QoS level of 1.

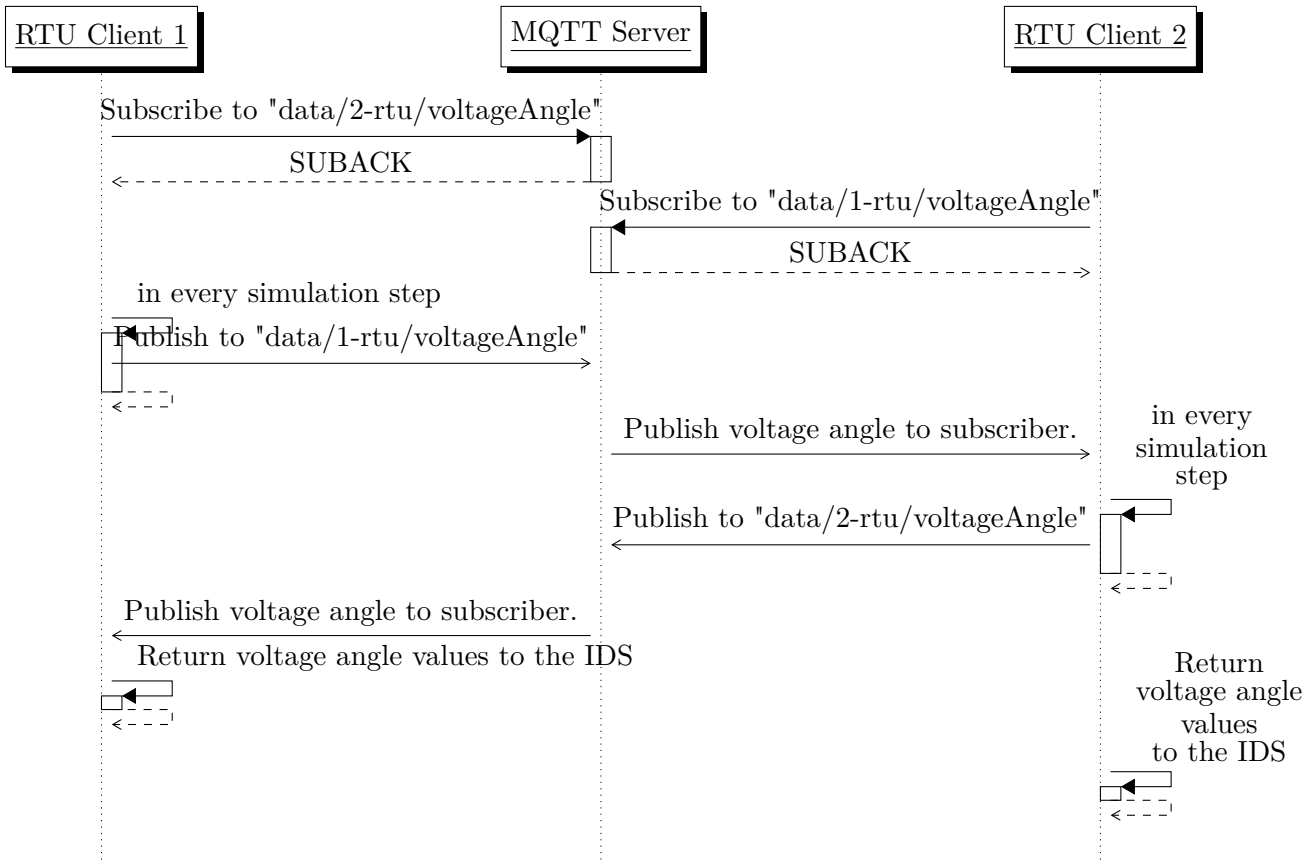

Figure 6.5: Workflow of exchanging voltage angle values in the MQTT implementation. Client 1 and client 2 are clients of RTUs that are adjacent to each other. Note: MQTT servers send a SUBACK message to confirm subscriptions.

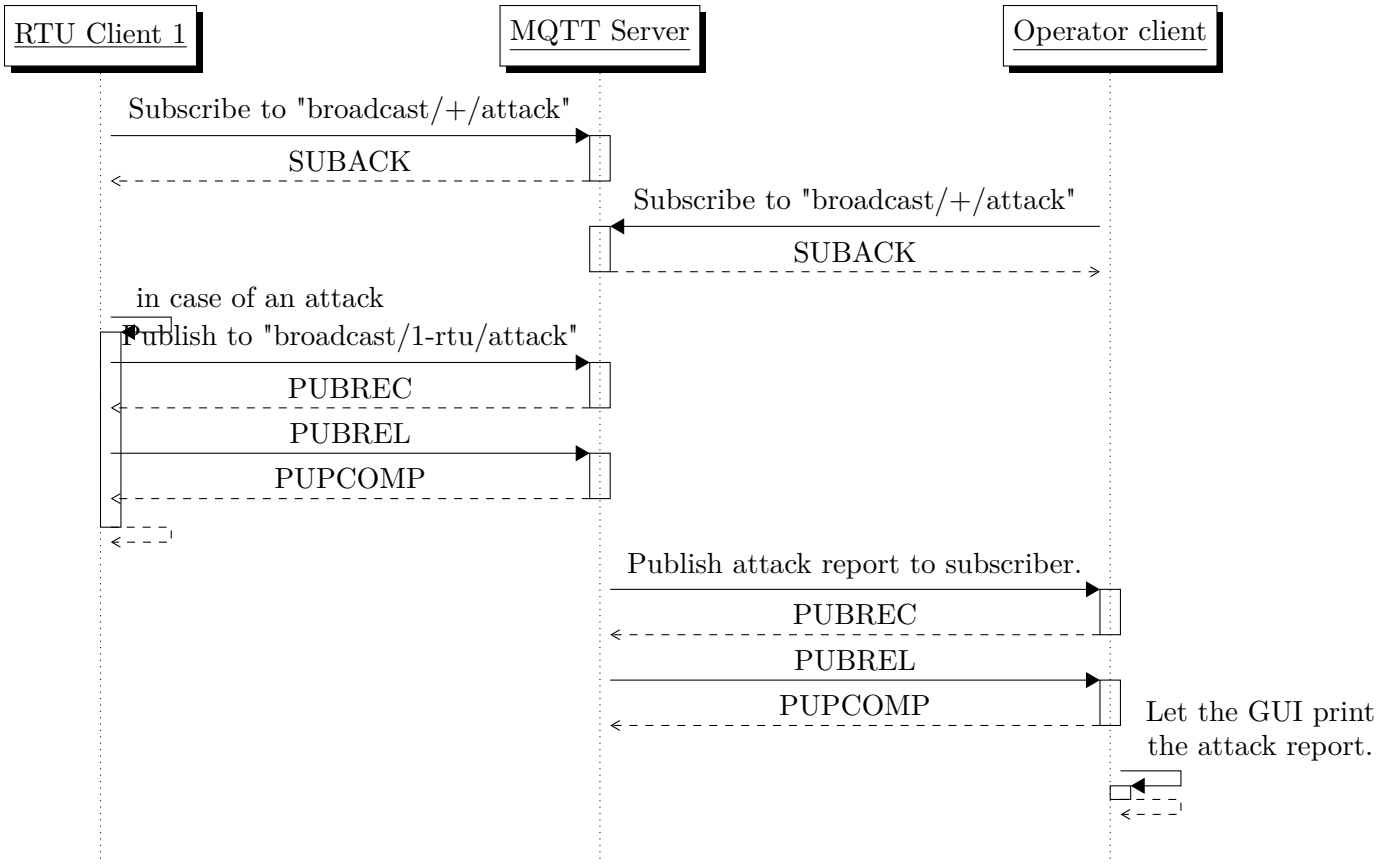

Figure 6.6: Workflow of attack report communication in the MQTT implementation. The same workflow is true for adjacent RTU to RTU communication besides the last self call. Instead of printing the attack report, certain IDS values would be changed by the receiving client. Note: MQTT servers send a SUBACK message to confirm subscriptions. PUBREC, PUBREL and PUBCOMP messages are sent to ensure the guarantees of messages published with a QoS of 2.

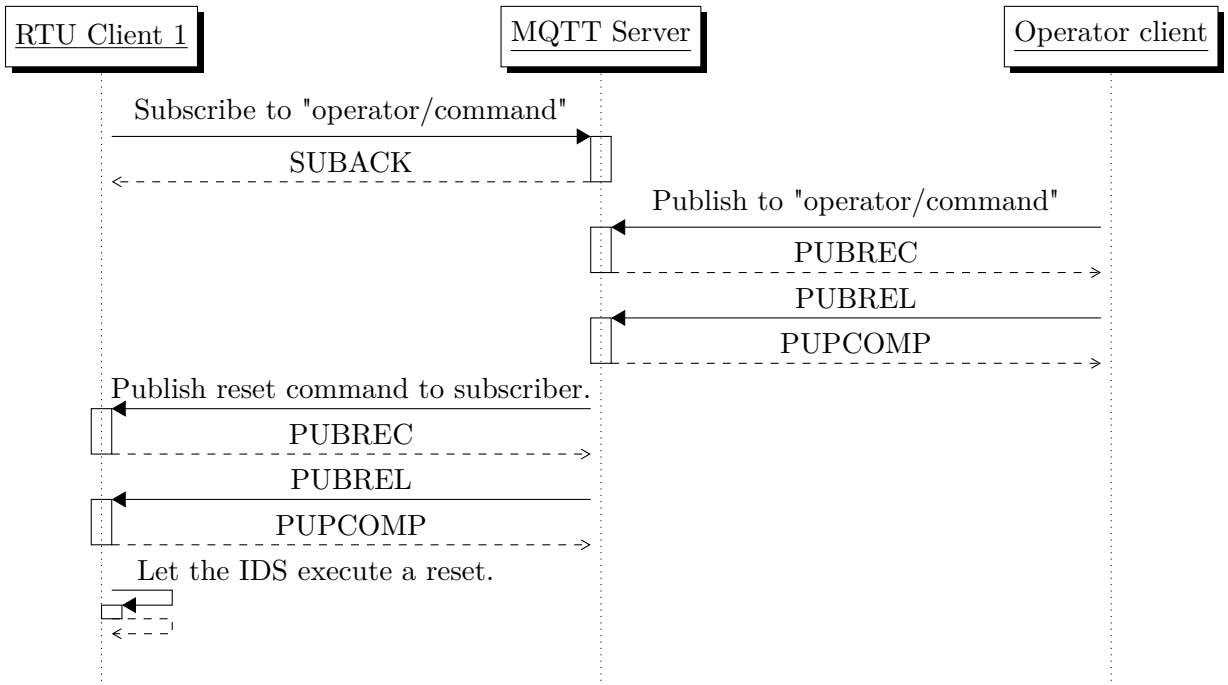

Figure 6.7: Workflow of a reset command in the MQTT implementation. Note: MQTT servers send a SUBACK message to confirm subscriptions. PUBREC, PUBREL and PUBCOMP messages are sent to ensure the guarantees of messages published with a QoS of 2.

features were tested. The username/password feature allows to authenticate the clients. This prevents unauthenticated clients from publishing messages. The access control list defines for each username which topics the client using the username is allowed to read from or write to. Additionally, the clients have to use the port used for secure MQTT connections. However, the features are not usable with grids that do not have exactly 4 RTUs and the provided access control list is only sensible for the functionalities described in this section.

Finally, when the simulation is done, all clients will be disconnected, and the event loops that ran the clients will be stopped automatically, whereas the server has to be closed manually. When the clients are disconnected the on_disconnect callback will print feedback on whether the disconnect was successful or not. It is also worth mentioning that the MQTT client as provided can easily be added to other simulations. To do this, the simulation has to start a client object and then order it to subscribe or publish messages to the topics of the features that are wished to be used. Of course, a server has to be provided separately.

6.3 IEC 60870-5-104 implementation

As mentioned in 4.2, there are no open-source projects that are comparable to the MQTT ones. Therefore, the server and client have to be built from scratch.

First of all, a wrapper and unwrapper class were created [38] based on the IEC 60870-5-104

specification. As indicated by the names, the wrapper is responsible for the serialization of an APDU, and the unwrapper is responsible for the deserialization of an APDU. The wrapper takes information needed to create APDUs, replaces the information by their numerical representations, then packs the numerical representations into a bytestring, and lastly concatenates the bytestring in the correct order to represent a single APDU (without header). The wrapper is then able to create a header based on the incomplete APDU. The unwrapper takes a single APDU (without header) as a bytestring and the APDUs length in bytes, does the reverse operations of the wrapper operations, and returns the information bits as a tuple. APDU headers can be unwrapped similarly to attain the APDU length.

It has to be noted, that only 5 of the APDU types and only the information elements needed for these 5 types are supported at the moment, because the other types and information elements are not needed for this thesis. Alike, only 6 of the causes of transmission can be used. All 3 frame types and all control functions are available. The variable structure qualifier, originator address, common address, IOA, and information objects are used for the APDU creation as explained in 3.1.2.3. Furthermore, the wrapper takes care of managing the IOA, while the common address, SSN, and RSN are expected to be given to the wrapper when creating an APDU. The next IOA value is stored by the wrapper and increased on APDU creation. In case the maximally allowed IOA value is exceeded, the value is wrapped back to 0.

Another feature of the unwrapper is that it can verify the contents of an APDU with a supplementary function. To do this, the function expects some of the anticipated parameters (e.g. APDU type) and checks if they are present in the given APDU. This is used by clients to check if the servers' answer is something expected. Furthermore, it is useful to easily handle all the different APDU types that a server might receive. Moreover, the wrapper provides a supplementary function that helps to split strings in the format expected by the APDU type that is used to sent bitstrings. Additionally, the wrapper and unwrapper provide error messages for every step of the APDU creation and unwrapping respectively.

Up next, the server and client which the IEC 60870-5-104 specification [22] calls slave and master respectively. In the project each substation, which means all RTUs and the operator GUI, possesses a slave and a master. Like in the original communication model, the RTU masters and slaves are started by the IDS, whereas the operator GUI master and slave are started by the operator tools. The diagram 6.8 provides an overview of the structure. The masters contain the logic to send messages, while the slaves contain the logic to receive messages. The names for masters and slaves were chosen this way, because masters mainly send messages, and slaves mainly receive messages. However, in actuality they are representing the same substation and may therefore take on the opposite role as their names would suggest. For example, the operator is always the master of every other RTU. Nevertheless, it possesses a slave to receive messages from RTU "masters". Although this convention seems to be unintuitive, it helps to clarify which substation acts as a master and which as a slave in case of peer-to-peer communication. Currently, the only exceptions where a message is sent by a slave substation are a little workaround which is not part of the normal IEC 60870-5-104 communication and the attack report feature. As mentioned, the common address, SSN, and RSN are expected for APDU creation, hence they are managed by the slaves and masters. That means the slave and master store the common addresses, SSNs and RSNs, take care of increasing their values, and lastly take care of the cases, in

which the values exceed the maximally allowed values, by wrapping the value back around to 0.

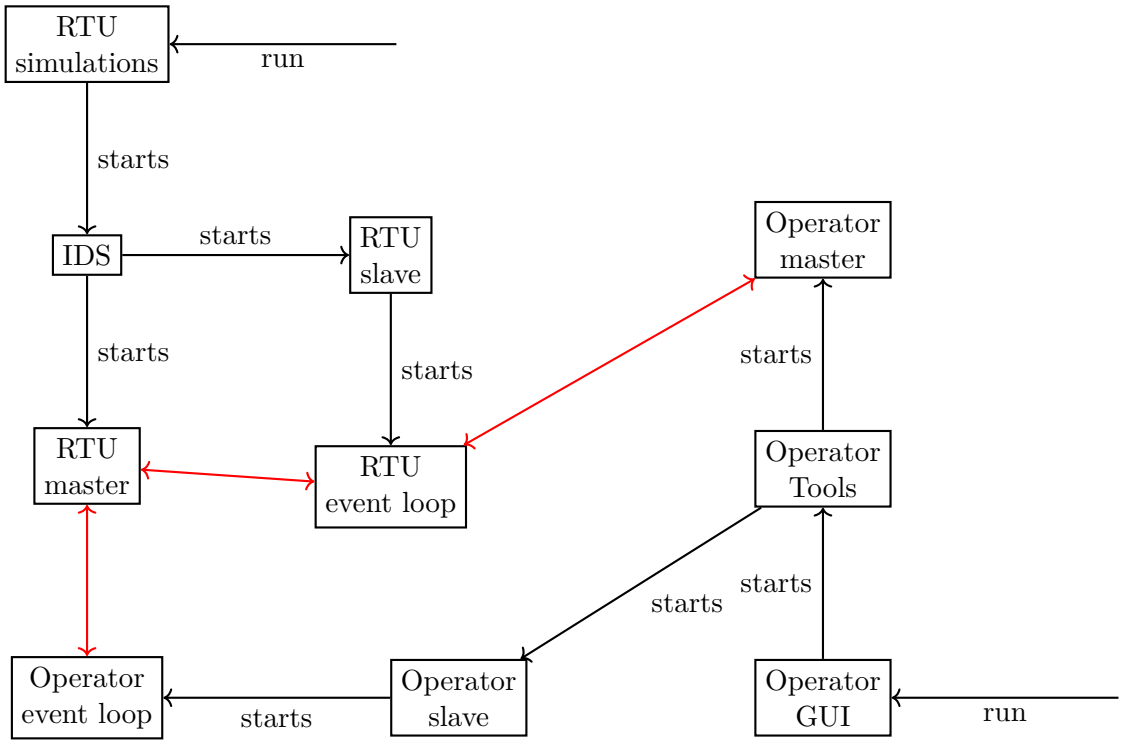

Figure 6.8: This diagram exhibits the components directly involved in the IEC 60870-5-104 communication. Red arrows represent communication.

Before getting to the functionalities, some things have to be mentioned concerning the communication. First of all, the standard Python 3 asynchronous network library asyncio [39] is used to create the underlying TCP/IP connection. Asyncio provides event loops that are used to execute asyncio coroutines in an extra thread. These asyncio coroutines contain the logic to send and receive messages.

Every slave uses a single coroutine that runs until the end to listen for APDU headers that have a fixed length of 2 bytes. By using the APDU length stated in the header, the coroutine is able to correctly listen for the rest of the APDU. If the length stated in the header and the real length of the APDU are different for some reason, an error will be printed and the APDU will be ignored. Otherwise, the APDU format will be checked by the verify function provided by the unwrapper. Then the message will be interpreted and an appropriate answer APDU created which is then send back to the master. If an error of some kind occurred, an error message will be sent to let the master know that an error occurred.

Meanwhile, masters have 3 coroutines at their disposal. The 3 coroutines are writing messages, writing messages and expecting a single answer, and writing messages and expecting several answers. When executed, each coroutine creates a connection to the targeted slave and writes the given messages to the slave. The coroutines expecting an answer, check if an error message was sent. Otherwise, they receive APDUs in the same manner as the slaves

receive them in their coroutine. The single answer coroutine forwards the received APDU to the caller. Whereas the first answer received by the multiple answer coroutine, indicates the amount of APDUs that will be sent. The APDU type to send float values is used to transmit the amount of answers. The coroutine then tries to receive the amount of APDUs stated by the float value given in the first APDU. An error message is printed, if not enough messages were received and if more are send, they are ignored.

Additionally, an APDU fabric is provided for both, the masters and slaves which helps with communication by wrapping the APDU and header creation into a single function. The sending of messages is handled by two functions available in the masters. The first function is used to designate the message targets, provide a hub to easily see which information is given to the APDU fabric for each functionality, task the second function to send a message to every target, and finally to collect the results of the second function. The second function sends a message to a single target, checks if the answer is of the expected format, processes the answer if needed, and returns the result to the first function. It has to be mentioned that the first function is accessed from the IDS to task the master to send requests to targets of the IDS's choosing. The targets are either specific entity identifications (e.g. operator), all RTUs, or a broadcast. The possible requests for RTU masters are to send attack reports, to ask for voltage angles, and to ask for branch names, while the operator master can send a reset command and ask for the number of RTU addresses.

Up next, some conventions that were made have to be talked about. First up, the first number of the qualifier of command private range is used for the reset command. Similarly, the first three numbers of the qualifier of interrogation private range are used to define 3 interrogation groups. The first group are the names of all branches of an RTU, the second group are the voltage angle values of an RTU, and lastly the third group are the addresses used by the RTU slaves. The other conventions concern the addresses used. First of all, all addresses are of the format "127.X.X.Y", where X is a number between 0 and 255 and Y is a number between 1 and 253. The addresses are based on the unique numbers that are part of the entity identifications. For example, the address of the first RTU is "127.0.0.1" while the address of the second RTU is "127.0.0.2". The operator uses the address "127.255.255.254". Therefore, up to 16777212 RTUs can theoretically be addressed. Lastly, the originator addresses are assigned from 1-254 to the RTUs 1-254 (i.e. the first RTU has the originator address 1 and the 254th RTU has the originator address 254) and the address 255 is assigned to the operator.

Now there is a little problem. As the RTU simulation and the operator GUI are started separately, there is no way for the operator substation to know how many RTU substations exist. Therefore, a workaround was created which is not part of the normal IEC 60870-5-104 communication. If RTUs are used, the first RTU master will send a message outside of the IEC 60870-5-104 communication to the operator slave, to let the operator know that at least a single RTU exists. As a reaction the operator master will be tasked by the operator tools to send an APDU of the type that is used to ask for information of an interrogation group. In this case group 3 will be asked, so the addresses of the RTU slaves. As this is a bonus feature only used by the operator once, and because the addresses are defined by special convention, the request is sent only to the first RTU, and the answer sent is a single APDU of the type to send a float value. The float value contains the amount of RTUs. From the amount each address can be unambiguously determined. Therefore, the operator

does not only know how many RTUs exist, but it can also connect to every RTU slave. A visual representation can be seen in 6.9.

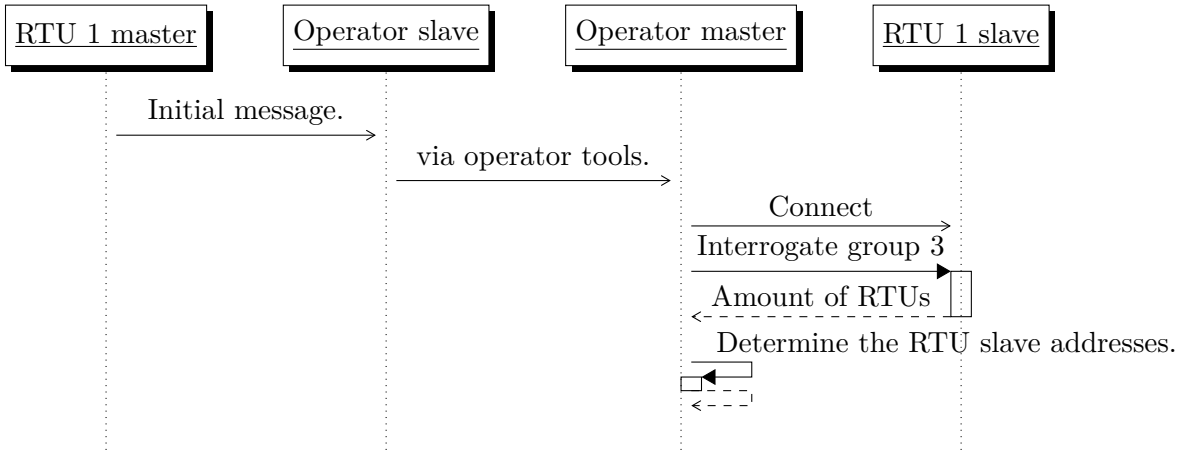

Figure 6.9: Workflow to initialize the operator to RTU connection in the IEC 60870-5-104 implementation.

Finally, the functionalities will be addressed. First of all, the request for branch names to determine adjacent RTUs. For this, the multiple answer coroutine is used. Initially, an APDU is sent to interrogate group 1, the branch names, of every RTU. The slaves then create an APDU of the type to send bitstrings for each branch name with the help of the function provided by the wrapper. After that the APDUs are sent as an answer to the interrogation. Finally, the branch names of each RTU are given to the IDS that determines the adjacent RTUs as described in 6.2. See 6.10 for a visualization.

To attain the voltage angle values, the single answer coroutine is used. First of all, an APDU is sent to interrogate group 2, the voltage angle values, of every adjacent RTU. The slaves answer this by sending an APDU with the voltage angle value stored as a float value. The voltage angle values are given to the IDS for further usage. This process is visualized in 6.11.

In contrast to MQTT, the attack reports are broadcasted to every RTU and the operator GUI. To only send the report to the GUI and adjacent RTUs, more effort would be needed than in the MQTT implementation. As this is not too much of a performance difference, it was left out of the IEC 60870-5-104 implementation. As mentioned before, the attack report is an exception, because this is a slave substation sending a message to a master substation, instead of answering the messages of a master substation. The attack report is sent as a bitstring by using the corresponding APDU type and the helper function provided by the wrapper. The write only coroutine is used, because no answer is expected. When the attack reports are received by the master substations' slaves, they are forwarded to the GUI and IDS respectively for the purposes explained in 6.2. 6.12 shows the visualization of this.

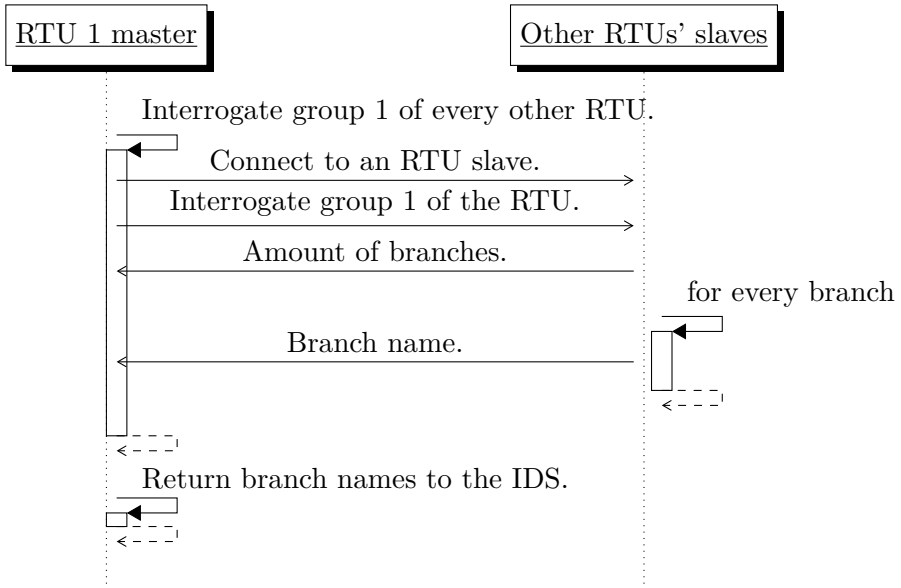

Figure 6.10: Workflow to determine adjacent RTUs in the IEC 60870-5-104 implementation. The other RTU slaves are shown in a single line for clarity, but they are actually not a single entity.

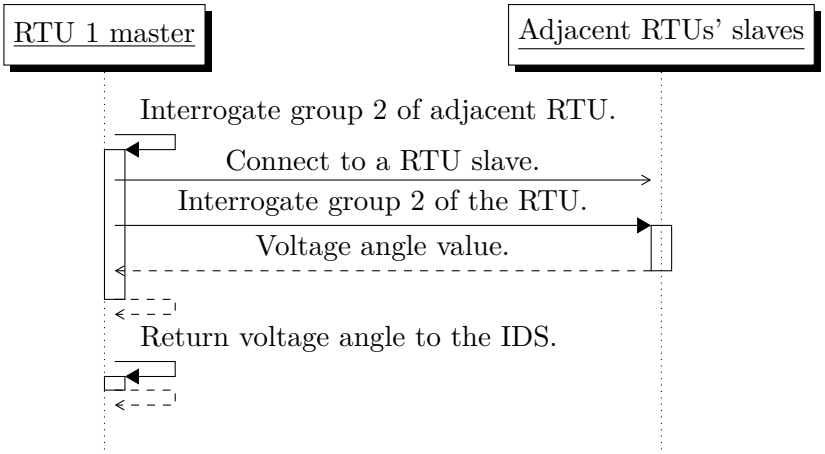

Figure 6.11: Workflow to exchange voltage angle values in the IEC 60870-5-104 implementation. The adjacent RTU slaves are shown in a single line for clarity, but they are actually not a single entity.

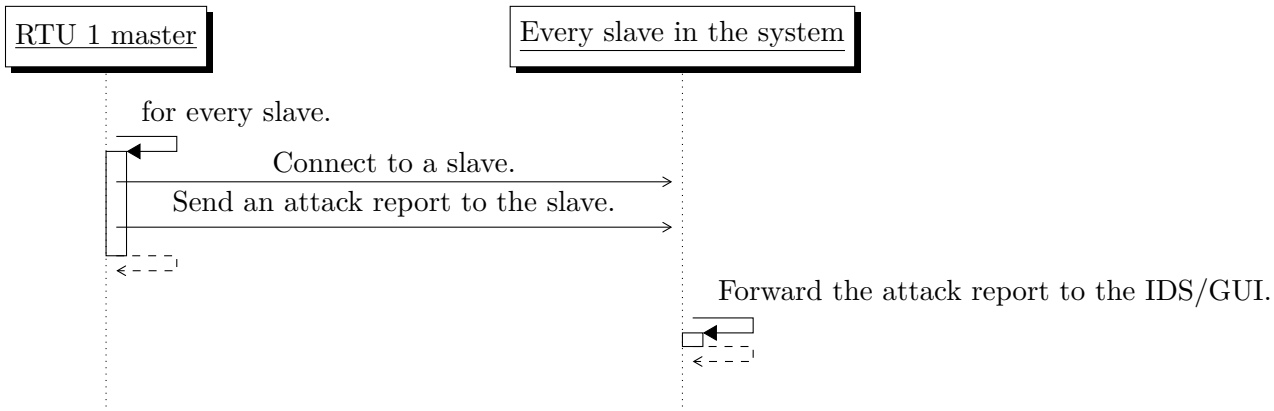

Figure 6.12: Workflow to broadcast attacks in the IEC 60870-5-104 implementation. The other slaves are shown in a single line for clarity, but they are actually not a single entity.

At last, the user can send a reset command similarly to the process described in 6.2. The command is sent using the APDU that is used for a single command. The private range number assigned to the reset command is given as the command description. After the reset was executed in the IDS, a confirmation APDU will be sent which means that it has to be of the same type as the command APDU that was originally sent. Confirmation APDUs also contain a bit to disclose if the command was successful or not as mentioned in 3.1.2.3. As the reset cannot fail at the moment, the bit is always set to 1, positive confirm. 6.13 shows this visually.

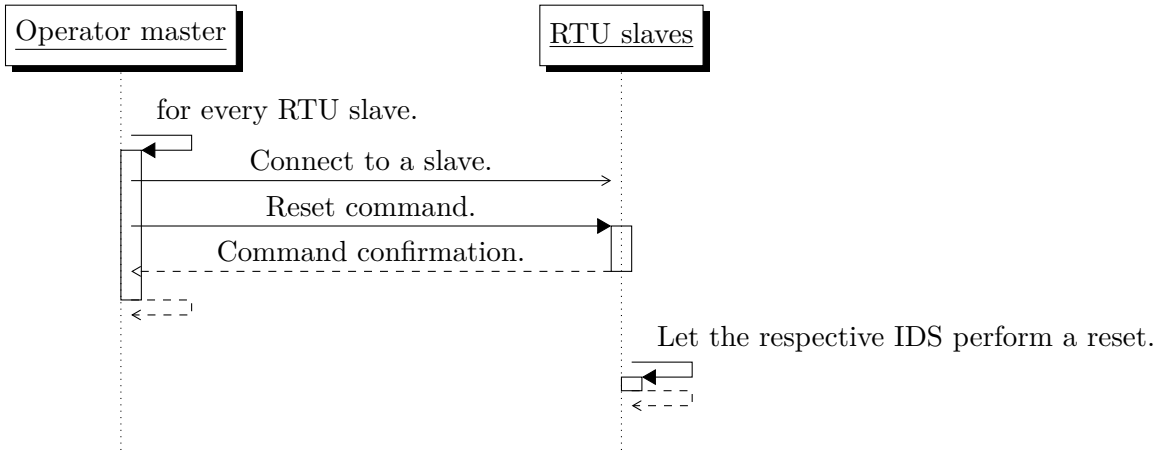

Figure 6.13: Workflow of a reset command in the IEC 60870-5-104 implementation. The RTU slaves are shown in a single line for clarity, but they are actually not a single entity.

Lastly, when the simulation is complete, the slave loops are stopped and closed. Similar to MQTT, the master and slave can easily be added to other simulations. To achieve this, the simulation has to start a master and slave object and use the inbuilt coroutines to send messages as needed. The caveat with this is the limited amount of implemented APDU

types and other missing IEC 60870-5-104 specifics and features that were either out of scope for this thesis or were left out due to a lack of time. It also has to be noted that some of the specifics and features are basic requirements and therefore this IEC 60870-5-104 implementation is not entirely completed (e.g. start and stop APDUs are not used for connection establishment (see 4.1)).

CHAPTER 7

Evaluation

In this chapter the comparison of MQTT and IEC 60870-5-104 will be evaluated and a conclusion be reached in section 7.1. Additionally, a look will be taken at each of the implementations in section 7.2 and 7.3 for MQTT and IEC 60870-5-104 respectively.

7.1 Comparison Conclusion

The table 7.1 provides a short summary of the characteristics presented in 4. As seen, none of the two protocols is clearly better than the other one. On one hand, MQTT provides more flexibility (e.g. QoS) and ease of use (e.g. open specification), but on the other hand, it is not designed as a SCADA protocol and therefore lacks in interoperability capabilities compared to IEC 60870-5-104. However, the lack of interoperability of MQTT within SCADA systems can be mitigated through edge-of-network devices as explained in [25].

While the practical implementations definitely proved MQTT to be easier to implement, albeit mainly due to the open-source support, it has to be noted that in reality a professional workforce would be hired to set up the protocol in a SCADA system which renders the ease of use a rather unimportant point to consider.

Although MQTT provides some interesting security features, IEC 60870-5-104 has already proven itself to be pretty secure within the SCADA environment. In terms of reliability both, MQTT and IEC 60870-5-104, offer different ways to achieve this. However, none of the two seems to be significantly superior in this regard.

IEC 60870-5-104 allows peer-to-peer communication and defines some aspects of it, whereas MQTT is not only ideal for peer-to-peer communication, but even able to easily include outside software. Moreover, MQTT is a protocol suited for the IoT.

However, MQTT as is would probably be in need of extension (e.g. data types), if it is to be used in SCADA systems, as the interoperability is of great importance. Additionally, it has to be carefully considered if MQTT is really secure enough for SCADA systems in the long term. Though, if in the future the IoT becomes important or useful for SCADA systems, MQTT would definitely be an interesting candidate to replace more common protocols that are not able to adapt to the IoT. If IEC 60870-5-104 would be able to adapt to such changes, remains to be seen.

All in all, MQTT as is, is not able to replace IEC 60870-5-104 or other standard SCADA protocols for that matter, due to security uncertainty and interoperability impediments which are of such importance that they outweigh the advantages that MQTT could bring.

Protocol name	IEC 60870-5-104	MQTT
Interoperability	The specification is an IEC standard that is specified for SCADA. It is also widely used for SCADA systems.	Neither specified for SCADA nor widely used for SCADA systems.
Ease Of Use	No support comparable to MQTT.	Open-source implementations, guides, and a free specification.
Security	Proven over a considerable amount of time in many SCADA systems.	Provides innate security and is proven in non-SCADA applications.
Flexibility	Some flexibility through hundreds of APDU types.	Great flexibility due to a free choice of topics and the QoS feature.
Reliability	Reliability through TCP usage, rules for connection establishment, and more.	Reliability through TCP usage, QoS level 2, will feature, and more.
Peer-to-peer	Rudimentary support.	Innate support.

Table 7.1: Overview of the comparison between IEC 60870-5-104 and MQTT.

However, it would definitely be worth to research how to effectively combine MQTT with SCADA.

7.2 MQTT implementation evaluation

For MQTT the basic features were already provided by the open-source Eclipse Mosquitto™ server [36] and the Eclipse Paho™ MQTT Python Client [37]. Nevertheless, many features that are provided are still unused either due to a lack of time or due to being out of scope of this thesis. The following list explains them and evaluates how important they are:

- The username/password feature is only hard coded for a specific scenario. It has to be made usable for an arbitrary amount of RTU clients, and the password file should be stored safely. This is an important feature that should definitely be included.
- The access control list is another important feature that is hard coded for a specific scenario. It too should be made usable for an arbitrary amount of RTU clients.
- The Last Will and Testament feature provided by the Eclipse Paho™ MQTT Python Client should be considered for future work, but in the scope of a small simulation it is not really needed. The loss of communication to an adjacent RTU will not hinder the local IDS that is currently used, hence this feature is not of utmost importance.
- Both, the on_subscribe and on_publish callbacks return mid-values that can be compared with the mid-values returned by the subscribe and publish functions, to make sure that subscribing and publishing was successfully executed. This is less important for small scale grids, hence it was not included yet.

- The on_subscribe callback also returns the QoS value which can be checked for validity similarly to the mid-value. As with the mid-value this is rather unimportant for small scale grids.
- The on_unsubscribe callback was not implemented yet, as there was no need for it. However, it might be interesting for new functionalities.
- Concerning security it would be beneficial to use security features like digital signatures or hash-based message authentication codes to verify data integrity. Definitely interesting in the context of the importance of security.
- The web visualization that is provided by Mosaik could be communicated with via MQTT. The inclusion of outside services is one of the advantages of MQTT, and this would be an interesting proof-of-concept.
- TLS is an important feature that should definitely be included.
- At the moment the MQTT server has to be started manually which is a slight inconvenience that could be improved. This is rather unimportant as it is just a quality of life feature.
- It would be easy to only publish voltage angle values, if they changed in the last simulation step, instead of publishing them every simulation step. This would not only lead to a better performance, but it would also be important to do before testing practical performance.

As a conclusion, it can be said that functions concerning security are of high priority, while the other functions are either useful to increase reliability, quality of life improvements, or only necessary for new use cases. Therefore, the implementation is complete in a technical sense, but some important improvements can still be made.

7.3 IEC 60870-5-104 implementation evaluation

It has to be noted for the IEC 60870-5-104 implementation that many features are incomplete or were not implemented either due to a lack of time or for being out of scope of this thesis. Many of these features are specified in the implementation of IEC 60870-5-104, hence the IEC 60870-5-104 implementation in this thesis, is only a partial implementation of IEC 60870-5-104. The features, of which most are defined in [22], had their importance evaluated and are being listed here:

- The IOA is currently handled by the IEC 60870-5-104 wrapper, but as each master and slave belonging to a substation have their own wrapper, the IOA is technically not correctly implemented. Therefore, the same wrapper should be used by masters and slaves belonging to the same substation. This however, could lead to race conditions as the masters and slaves run concurrently. Because the IOAs are currently not used, this is not to big of a problem. However, this is a required IEC 60870-5-104 feature that should be implemented correctly, especially if the IOA is to be used for future functionality.
- The same thing that is true for the IOA is also true for the common address. Hence, the common address should be moved into the wrapper and it should be depending on

the entity identification number, to be able to easily chose a unique common address for every station. As mentioned in 3.1.2.3, IEC 60870-5-104 also gives the option to allow the assigning of a common address to a part of a station. In this case some bits of the common address should be reserved to have free addresses for substations of every station. Alike to the IOA, this feature is not used at the moment, but as it is also in the specification it should be working as specified. Additionally, this has to be corrected in case the common address feature is needed for future functionality.

- As the operator station can issue commands and because every RTU station is used for interrogation, the number of originator addresses is exhausted with only 254 RTUs in use. This may be problematic in case the originator address will be used for future functionality that needs more addresses.
- Similar to the IOA and the originator address, the SSN, and RSN could be stored by the wrapper and unwrapper. However, the management of the SSN and RSN should be done by the masters and slaves, because the wrapper and unwrapper do not know if a message was sent or received successfully. This is not an important change, because the SSN and RSN can also easily be stored on the master and slave. However, doing this would tidy up the wrapper and unwrapper interface, because the current SSN and RSN values would not have to be given to the wrapper and unwrapper for every APDU creation made by the masters and slaves.
- To use the SSN and RSN feature to its full potential masters and slaves should have a buffer that stores a message, until it was confirmed via SSN and RSN that the message was received. As this is part of the specification and is used to directly increase reliability, it should definitely be included.
- When sending long data transmissions in one direction, it is required to send an acknowledging APDU type in the sending direction in order to acknowledge the APDUs, before a buffer overflow or a time out occurs. In case long data transmissions in one direction are needed for future functionalities, this should be included. Again, this is part of the specification and therefore needed for a complete version.
- The maximally allowed difference between SSN and RSN is defined in the specification, but currently not implemented. Another feature necessary for a complete implementation.
- Various time-out conditions and time frames for them are defined (e.g. time-out connection establishment), but have yet to be implemented. These should also be included for completion.
- The data transfer activation as described in 4.1 is currently not implemented, because the data transfer is mostly always active at the moment. However, it might already be beneficial for the connection between operator and RTUs, as these connections are not needed to permanently send data, but they should still be open in case of attack reports. Hence, this should also be added.
- It is also suggested to regularly test unused, but open connections. Although this is a suggestion, it seems to be a rather important feature that should be implemented.
- When floating point numbers are packed into a bytestring in the wrapper, floating point errors can occur. Because the physical data should be as accurate as possible

for the IDS to work correctly, it would be worth to look into preventing or at least mitigating this as much as possible.

- As mentioned in 6.3, many APDU types, information elements, and causes of transmission are not yet implemented. New types, information elements, and causes should at least be added, when they are needed.
- The time tag as defined in the specification should be added to the wrapper and unwrapper, as the time tag is used by many APDU types.
- The receiving of an APDU is suggested to be acknowledged by an acknowledging APDU type, to avoid retransmission of accepted APDUs. Avoiding retransmission is important for an efficient performance. Hence, this feature should be added, before testing practical performance.

As already mentioned and can be inferred from the list, many important features of the specification are still lacking or even missing. Most of these features are used to increase reliability and none are concerning security directly. However, to have more APDU types as well as the time tag would greatly increase the amount of use cases that can be covered. Therefore, a lot more work has to be done, before the implementation becomes complete.

CHAPTER 8

Conclusion

This thesis' aim was to find the most optimal protocol for use in SCADA systems. Therefore, various SCADA protocols were examined and their advantages and disadvantages carved out. The focus was placed on the traits interoperability, reliability, ease of use, security, flexibility, and peer-to-peer capability. IEC 60870-5-104 was shown to be a solid state-of-the-art SCADA protocol that supports peer-to-peer communication which seems to be a useful quality for advanced security measures (e.g. local IDS) in smart grids.

While MQTT is not designed for SCADA, it was found to be a potentially useful protocol in the context of smart grids and the IoT. Therefore, the differences between MQTT and IEC 60870-5-104 were looked into in more detail. Additionally, both MQTT and IEC 60870-5-104 were implemented into a smart grid simulation. This is a proof-of-concept of MQTT already being able to be used for communication within a simple SCADA system. It also turned out, that MQTT was vastly easier to implement, though this is mainly due to the open-source MQTT client and server, as well as beginner guides.

As a result of the comparison, MQTT showed itself to be superior in flexibility (e.g. QoS, ability to incorporate enterprise software), innate security features (e.g. client authentication), peer-to-peer communication, and ease of use, while it showed to be lacking in interoperability in the context of SCADA (e.g. no data types) and unproven to be secure enough for SCADA systems. Although by number there are more advantages than disadvantages, the interoperability is of such importance that MQTT as is, should most likely not be used to replace standard SCADA protocols just yet. However, an extended version that is geared towards SCADA or the use of MQTT via edge-of-network devices as mentioned in 3.1.3 could potentially be of great use for SCADA systems, especially if SCADA and the IoT become more entwined in the future.

For future research in this topic, it would be interesting to examine more traditional applications that use MQTT for security problems. While the IoT might become more important for SCADA, it should also be noted that the IoT also creates new attack opportunities [9]. These attack opportunities should be researched in SCADA systems that are connected to the IoT via MQTT. On this note, it would also be interesting to take a look at the Sparkplug specification by Cirrus Link [40] which tries to provide features needed for the use in SCADA systems (e.g. a topic namespace) that can be used with the previously mentioned edge-of-network devices or devices natively supporting MQTT. Sparkplug, like MQTT, is a well supported project with a guide [41] and reference implementations in various languages [42] provided by Cirrus Link. This MQTT extension would be an especially interesting topic to research, as the lack of SCADA support was the main complaint of this thesis. Although it is certainly interesting to use MQTT in SCADA systems, it might also be interesting to look into extending standard SCADA protocols to be usable with the IoT.

The implementations of MQTT and IEC 60870-5-104 also offer a possible opportunity for future work. In 7.2 and 7.3 missing features were listed and evaluated. After the realization of these features and maybe additional features, it would be interesting to test and compare the practical performance of the implementations. For example, it could be tested how much time is used to send a package containing a voltage angle value in each of the implementations. A useful tool for this endeavor would be the open-source project "Bro Network Security Monitor" [43] which can be used for network traffic analysis. When testing performance it has to be considered, that the MQTT implementation might have an unfair edge over the IEC 60870-5-104 implementation, due to the vast open-source support for MQTT.

Bibliography

[1] ICS-CERT. Alert (IR-ALERT-H-16-056-01) Cyber-Attack Against Ukrainian Critical Infrastructure. Available at: `https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01`, 2016. [Accessed: 2017-07-15].

[2] E. J. Byres, M. Franz, and D. Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of International Infrastructure Survivability Workshop (IISW)*. IEEE, 2004.

[3] R. M. Lee, M. J. Assante, T. Conway. Analysis of the Cyber Attack on the Ukrainian Power Grid. Defense Use Case. Electricity Information Sharing and Analysis Center (E-ISAC). Available at: `https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf`, 2016 [Accessed: 2017-07-15].

[4] J. J. Chromik, A. Remke, and B. R. Haverkort. Improving scada security of a local process with a power grid model. In *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research (ICS-CSR)*, Electronic Workshops in Computing, pages 114–123. BCS Learning & Development Ltd., 2016.

[5] H. Bao, R. Lu, B. Li, and R. Deng. Blithe: Behavior rule-based insider threat detection for smart grid. *IEEE Internet of Things Journal*, volume 3(no. 2):pages 190–205, 2016.

[6] P. Maynard, K. McLaughlin, and B. Haberler. Towards understanding man-in-the-middle attacks on iec 60870-5-104 scada networks. In *Proceedings of the 2nd International Symposium on ICS & SCADA Cyber Security Research (ICS-CSR)*, pages 30–42. BCS, 2014.

[7] Z. Lu, X. Lu, W. Wang, and C. Wang. Review and evaluation of security threats on the communication networks in the smart grid. In *Proceedings of Military Communications Conference (MILCOM)*, pages 1830–1835. IEEE, 2010.

[8] M. Wei and W. Wang. Toward distributed intelligent: A case study of peer to peer communication in smart grid. In *Proceedings of Global Communications Conference (GLOBECOM)*, pages 2210–2216. IEEE, 2013.

[9] C. Johnson. Securing the participation of safety-critical scada systems in the industrial internet of things. In *Proceedings of 11th International Conference on System Safety and Cyber Security (SSCS)*. IET, 2016.

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

[11] D. Bailey and E. Wright. *Practical SCADA for Industry*. Newnes, Oxford, 2003.

[12] G. Clarke, D. Reynders, and E. Wright. *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes, Oxford, 2004.

[13] A. Daneels and W. Salter. What is SCADA? In *Proceedings of 7th Biennial International Conference on Accelerator and Large Experimental Physics Control Systems*.

CERN, 1999.

[14] C. Wang, L. Fang, and Y. Dai. A simulation environment for scada security analysis and assessment. In *Proceedings of International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 342–347. IEEE, 2010.

[15] B. K. Rios and J. Butts. When iot attacks: Understanding the safety risks associated with connected devices. In *Proceedings of Black Hat USA*, 2017.

[16] Modbus Organization. Modbus faq: About the protocol. Available at: `http://www.modbus.org/faq.php`. [Accessed: 2017-07-08].

[17] F. Aloula, A. R. Alia, R. A. Dalkya, M. A. Mardinia, and W. E. Hajjb. Smart grid security: Threats, vulnerabilities and solutions. *International Journal of Smart Grid and Clean Energy*, volume 1(no. 1):pages 1–6, 2012.

[18] Modbus Organization. Modbus specifications and implementation guides. Available at: `http://www.modbus.org/specs.php`. [Accessed: 2017-07-08].

[19] DNP Users Group. Dnp3 guides. Available at: `https://www.dnp.org/DNP3Downloads/Forms/AllItems.aspx`. [Accessed: 2017-07-08].

[20] Automatak LLC. Dnp3 (ieee-1815) protocol. c++ with bindings for .net and java. Available at: `https://github.com/automatak/dnp3`. [Accessed: 2017-08-07].

[21] R. E. Mackiewicz. Overview of iec 61850 and benefits. In *Proceedings of Power Systems Conference and Exposition (PSCE)*, pages 623–630. IEEE/PES, 2006.

[22] IEC Technical Committee 57. Iec 60870-5-104: Telecontrol equipment and systems – part 5-104: Transmission protocols – network access for iec 60870-5-101 using standard transport profiles, 2006. 2nd Edition.

[23] MQTT Version 3.1.1. Edited by A. Banks and R. Gupta. 2014. OASIS Standard. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`. Latest version: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`.

[24] J. Barr. Aws iot – cloud services for connected devices. Available at: `https://aws.amazon.com/blogs/aws/aws-iot-cloud-services-for-connected-devices/`, 2015. [Accessed: 2017-07-09].

[25] A. Nipper. Scada is not a middleware. Available at: `http://www.plantengineering.com/single-article/scada-is-not-a-middleware/4a7623d2cd09cf1be6a21ba1d13838f2.html`, 2016. [Accessed: 2017-07-09].

[26] Mqtt version 3.1.1 becomes an oasis standard. Available at: `https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard`, 2014. [Accessed: 2017-07-09].

[27] Interoperability testing at eclipsecon 2014. Available at: `http://mqtt.org/2014/01/interoperability-testing-at-eclipsecon-2014`, 2014. [Accessed: 2017-07-09].

[28] Mqtt introductory guide. Available at: `http://www.hivemq.com/mqtt-essentials/`. [Accessed: 2017-07-09].

[29] W. Lee, M. P. Stromberg, A. Ward, C. Stewart, E. P. Garrison, G. T. Marth MOSAIK: A Hash-Based Algorithm for Accurate Next-Generation Sequencing Short-Read Mapping. PLoS ONE volume 9(no. 3), 2014.

[30] OFFIS. Mosaik documentation overview. Available at: http://mosaik.readthedocs.io/en/latest/overview.html [Accessed: 2017-07-29].

[31] OFFIS. Mosaik API documentation. Available at: http://mosaik.readthedocs.io/en/latest/mosaik-api/index.html [Accessed: 2017-07-29].

[32] OFFIS. Mosaik scenario definition documentation. Available at: http://mosaik.readthedocs.io/en/latest/scenario-definition.html [Accessed: 2017-07-29].

[33] OFFIS. Mosaik simulator manager documentation. Available at: http://mosaik.readthedocs.io/en/latest/simmanager.html [Accessed: 2017-07-29].

[34] OFFIS. Mosaik scheduler documentation. Available at: http://mosaik.readthedocs.io/en/latest/scheduler.html [Accessed: 2017-07-29].

[35] Python Software Foundation. Python Language Reference, version 3.4. Available at: http://www.python.org [Accessed: 2017-07-11].

[36] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," The Journal of Open Source Software, volume 2, no. 13, 2017.

[37] Eclipse Foundation. Eclipse paho^{TM} mqtt python client. Available at: https://github.com/eclipse/paho.mqtt.python. [Accessed: 2017-07-10].

[38] The wrapper and unwrapper are available at: https://github.com/ThomasTeodorowicz/iec104/tree/master/iec104/python3 [Accessed: 2017-08-12].

[39] Asyncio documentation available at: https://docs.python.org/3/library/asyncio.html [Accessed: 2017-07-12].

[40] Cirrus Link. Sparkplug Specification: Sparkplug MQTT Topic & Payload Definition. Version 2.1, 2016. Available at: https://s3.amazonaws.com/cirrus-link-com/Sparkplug+Topic+Namespace+and+State+ManagementV2.1+Apendix++Payload+B+format.pdf [Accessed: 2017-07-15].

[41] Cirrus Link. Sparkplug documentation. Available at: https://docs.chariot.io/ [Accessed: 2017-07-15].

[42] Cirrus Link. Sparkplug open-source reference implementations. Available at: https://github.com/Cirrus-Link/Sparkplug [Accessed: 2017-07-15].

[43] The Regents of the University of California through the Lawrence Berkeley National Laboratory and the International Computer Science Institute. Bro Network Security Monitor. Available at: https://www.bro.org/ [Accessed: 2017-07-15].