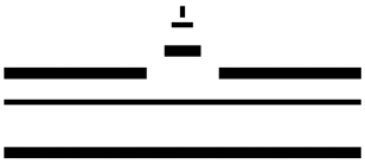

Westfälische
Wilhelms-Universität
Münster

Sequence attacks on SCADA systems

Bachelor thesis

presented to the
Westfälische Wilhelms-Universität Münster
Faculty of mathematics and computer science
Department of computer science

by

Benedikt Ferling

Assessor
Prof. Dr. Anne Remke
Dr. Dietmar Lammers
Westfälische Wilhelms-Universität
Münster

Supervisor
Justyna Chromik
University of Twente

Münster, July 2016

Declaration of Academic Integrity

I hereby confirm that this thesis on *Sequence attacks on SCADA systems* 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, den 12. Juli 2016 _____

signature of student

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, den 12. Juli 2016 _____

signature of student

Acknowledgement

I express my gratitude to all those who gave me the possibility to complete this thesis. Thanks to Prof. Dr. Remke without whom this work and the cooperation with the University of Twente didn't come about. Special thanks to Justyna Chromik for her assistance especially in the technical parts and who always had answers for me when I got stuck. Furthermore I would like to thank Marco Caselli for his time for giving me a deeper look into his work. I also thank Maria Berens who looked closely at the final version of the thesis for English style and grammar. Thanks to my parents Agnes and Norbert Ferling and my wife Katharina Ferling for supporting me in all other areas while writing the thesis. And last but not least thanks to Franziska, who always had a smile for me.

Contents

1 Introduction

Supervisory Control and Data Acquisition (SCADA) systems are deeply ingrained in the fabric of critical infrastructure sectors [1]. These systems are commonly deployed to aid the operation of these infrastructures by providing automated processes that ensure the correct functioning of these infrastructures.
Due to their standardization and their connectivity to other networks, they are increasingly a subject to serious damage and disruption [1]. The US grid hack [2] and the Ukrainian grid hack [3] are two examples that show the consequences of poor security. As a result of cyber-attacks on networks controlling mentioned grids, many people were left without electricity for hours. In general, these systems are barely protected from the escalating cyber threats [1]. Even if the security of the SCADA system is addressed, the attacker can use more sophisticated approaches to perform the attack. Stuxnet has demonstrated that skilled attackers can strike critical infrastructures by leveraging knowledge about the controlled process, especially by sending misplaced legal messages [4]. Common intrusion detection systems are only able to identify messages with malicious content within the network. One way to exploit those systems using the knowledge about the processes are the so-called sequence attacks. They subvert infrastructure operations by sending misplaced industrial control system messages [4]. Industrial systems show regular and consistent communication patterns, thus analyzing the communication using a stochastic approach in order to identify sequence attacks is an appropriate choice to investigate [5][6].

M. Caselli et al. showed, that it is possible to detect sequence attacks within SCADA networks [4]. However, the analysis of the structure they used may be time consuming, depending on the complexity of the communication within the network. In this thesis we rebuild the approach of M. Caselli and show that it is possible to successfully reduce the complexity of the structure in order to identify most types of sequence attacks.

2 SCADA networks and their security

This section introduces the basics of SCADA. It provides an overview of the hard- and software components used in SCADA networks. Furthermore the IEC 60870-5-104 protocol - one of the important protocols used in combination with these networks - is introduced. Besides, a brief summary of the protocol, a detailed explanation of the structure is provided. In addition an introduction to intrusion detection systems is given and a description of what "sequence attacks" are.

Chapter 2.1 describes the SCADA system with its components and communication models. In Chapter 2.2 the IEC-104 protocol is described with all the relevant parts needed for this thesis. Chapter 2.3 introduces the different kinds of intrusion detection systems. In Chapter 2.4 the sequence attacks are introduced.

2.1 SCADA

Modern industrial processes and industries have systems and equipment which are often separated by large distances. The management of these systems can be very time-consuming. Collect data from devices, evaluate the data and execute actions depending on the data are regular jobs in the process of managing industrial systems. The devices within the system are often wide spread, which makes the useage of modern IT indispensable [7].

A **Supervisory Control and Data Acquisition** (SCADA) system is a computer-based system for monitoring and controlling industrial processes [4] [1]. The system refers to a combination of telemetry and data acquisition which it does by interconnecting *field networks* and a *control network*. The field network hosts devices that are directly connected to the physical process. The control network hosts the servers that supervise the control of the physical process. More and more SCADA systems are connected to the business management which usually refers to the *enterprise network* [8]. Furthermore it is possible to transmit data from the field network to the internet. An example of a SCADA system is shown in Fig. 1. This example uses multiple firewalls to connect the networks.

The field network hosts devices like sensors, actuators, RTUs and PLCs. *Sensors* transmit the values of the monitored process over serial wires to the PLCs. *Actuators* are system components which are responsible for controlling a systems behaviour. They control a physical process, i.e., open or closing a valve, changing the pressure inside a pipe or tank or setting the temperature of a systems component. *RTUs* and *PLCs* which collect the field data and send it back to the data acquisition server via the communication system. They also receive commands, usually from the control network, which are then executed on the appropriate actuator, e.g., closing a valve. Today the difference between RTUs and PLCs is

infinitesimal, although PLCs are more often used in areas where output arrangements and multiple inputs are needed. This is due to their nature of having electrical noise immunity and vibration and impact resistance [9]. Besides their main purpose to collect data and send this data to a central station, PLCs may also communicate on a peer-to-peer basis with other PLCs. Furthermore, they may act like a relay station, i.e., they collect data from other PLCs and forward them to the central station. Today the RTUs are mostly replaced by PLCs as PLCs can easily be set up for a variety of different functions [7]. In this thesis both RTUs and PLCs will be referred to as PLCs.

The **process control network**, also refered to as *control network*, is responsible for controlling the process by evaluating the data of the field network and triggering appropriate actions. The *data acquisition server* within the control network is responsible for collecting, storing and providing the data of the field devices such as RTUs and PLCs. *HMIs* are responsible to enable human operators to interact with the physical process, e.g, the HMI presents the data of the data acquisition server and the human operator can initiate commands that affect the process.

SCADA software is used to interact with the components of the SCADA system, e.g. monitoring data, watching trends, sending commands, etc. On the one hand it can be divided into open source and proprietary software for interaction with the operators. On the other hand it can be divided into the different communication protocols used for the communication between the devices. An example of proprietary protocol is the S7 protocol of the Siemens company. Open source protocols are for example Modbus or IEC 60870-5-104. In the early days of industrial systems most companies developed proprietary protocols. The main problem with proprietary protocols is the reliance on the supplier of the system. Open protocols have nowadays gained popularity because of their interoperability - hardware of different manufacturers can be installed on the same system. This thesis focuses on the IEC 60870-5-104 protocol [7].

The **communication channels** are responsible for transferring the data between the devices. They can be wired, fiber optic, radio, telephone line, microwave and possibly even satellite. They connect the components of the SCADA system, such that they can communicate [7].

The **SCADA communication model** consists of two approaches in which devices can communicate. The first one is called *master-slave*. This approach uses the idea that the master device is in total control of the communication system. The device makes regular request for data to be transferred to/ from each of the slave devices in the network. This request could be for instance a software update request, or a restart command. An example for the download

of data is the sensors data the PLC is connected to. The slave only answers to request in this configuration. There are some advantages of this approach. First the software is simple and reliable due to the simplicity of the communication. Second, link failures between the master and a slave node are detected quickly and third, no collisions can occur on the network. However there are also some disadvantages. The slave is unable to communicate to the master for urgent actions. Even if this function is enabled, the master may be processing another slave at that moment. Another disadvantage is, that slaves that need to communicate with each other have to do this through the master. This adds complexity to the design of the master station.

The other approach to communicate is called *peer-to-peer*. In this approach each device within the network can initialize a connection, therefore collisions of requests and responses are unavoidable and need to be handled appropriately [7]. One method handling those collisions and controlling the communication is known as "carrier sense with multiple access/ collision detection" (CSMA/CD). This and other methods are handled by the underlying protocols.
The important communication part, within a *peer-to-peer* network, is the PLC-to-PLC communication. In this situation a request from a PLC is first sent to the master station. However the destination of the request is not the master station itself, but another PLC. The master station needs to forward the message and when it receives the reply it needs to forward it appropriate to the requesting PLC. Additional logic is used to handle this process without losing requests or responses [7]. The main advantage of this system is the low cost of the installation. What can be an advantage on the one hand - no central station is in control of the data - can be an disadvantage on the other hand. In industrial networks the central organisation of the data leads to an improvement of the precise statements that often can only be made by comparing data of different field devices. Another disadvantage is the software. Peer-to-peer networks needs a more complex software in order to guarantee a collision free communication. The disadvantages however overweight the advantages in terms of industrial control systems.

SCADA encompasses the collecting of the sensor information via a PLC, transferring it back to the data aquisition server, carrying out any necessary analysis and displaying the information on the HMIs. The required control actions are then conveyed back to the process [7]. The accurate and timely data allows for optimization of the systems operation and process. Further benefits are more efficiency, reliability and most importantly, safer operations. This all results in a lower cost of operation compared to earlier non-automated systems. A disadvantages of SCADA systems is, that the system is more complicated. Furthermore it is only possible to see as far as the PLCs. This is an important part as the PLC might transmit wrong data. Another disadvantage are the additional skills that are needed to manage the

system [7]. Nevertheless, the advantages overweight the disadvantages for modern industrial processes and industries.

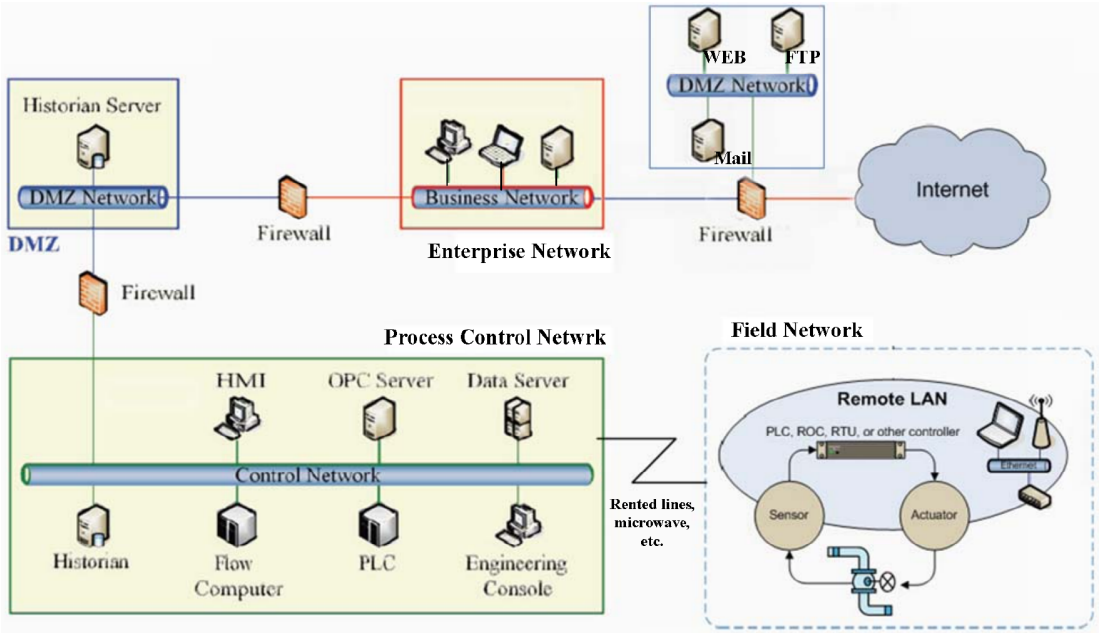

Figure 1: Architecture of a SCADA system [8]

Today SCADA systems use special communication protocols like Modbus, MMS, IEC 60870-5-104 or S7 for efficient and optimum data transfer. Since we analyze the communication between two devices only, the difference of the approaches "master-slave" and "peer-to-peer" are small. The difference is that both stations may request data or respond to requests. This thesis focuses on the IEC 60870-5-104 protocol.

2.2 IEC 60870-5-104 Protocol

The standard IEC 60870-5 refers to a collection of standards produced by the International Electronical Commission (IEC), to provide an open standard for the transmission of SCADA telemetry control and information [7]. It provides a detailed functional description for telecontrol equipment and systems for controlling geographically widespread processes. The standard is intended but not limited to applications in the electrical industries, since its data objects are applicable to general SCADA applications in any industry [7].
The standard was initially completed in 1995 with its profile IEC 60870-5-101(IEC-101) and is titled "Companion standard for basic telecontrol tasks". At this point it covered only transmission over relatively low bandwidth bit-serial communication circuits. With the increasingly widespread use of network communications technology, the standard needed to

be extended. With the protocol IEC 60870-5-104(IEC-104) - titled "Network Access using Standard Transport Profiles"- the standard now provides the possibility to communicate over networks using TCP/IP. To be precise the changes compared to IEC-101 lay in the transport, network, link and physical layer services to suit the complete network access. The protocol uses TCP/IP to establish connections. This enables the communication over networks, e.g., local area networks or wide area networks. Thereby, known technology like routers or virtual private networks can be used to connect different facilities.

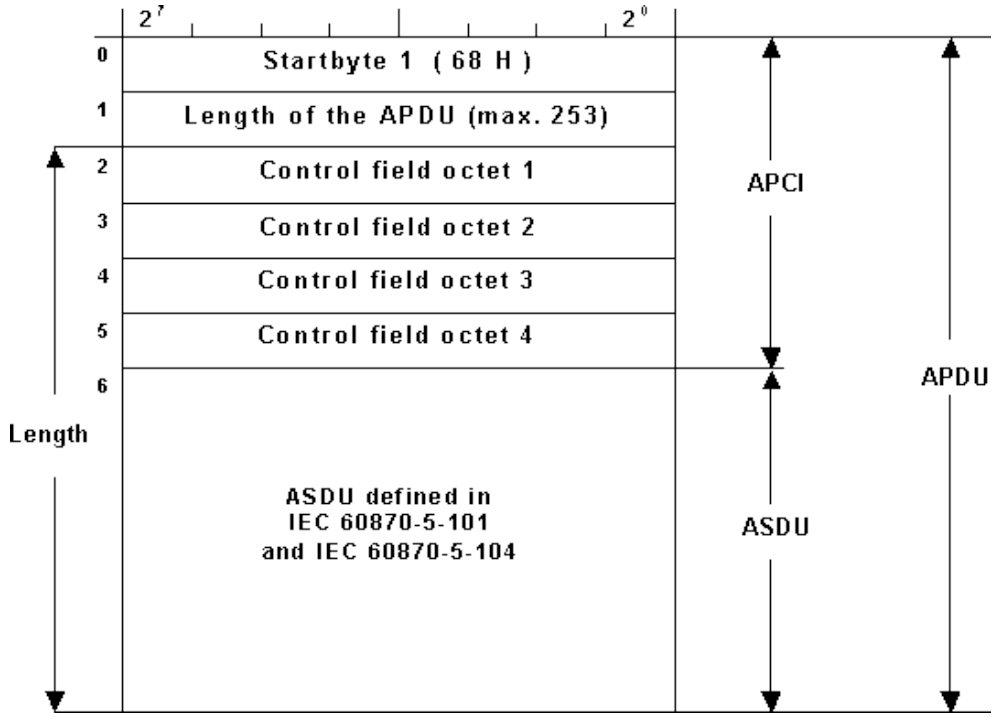

Figure 2: Telegram format with variable length (I-format) [10]

The overall **message structure** of IEC-104 is rather complex. This section introduces - apart from some basic information - those parts of the protocol that are relevant for this thesis. The overall structure of an IEC-104 packet, that we consider in this thesis, is shown in Fig. 2. It consists of the *application protocol control information* (APCI) and the *application service data units* (ASDU) which both together are called the *application protocol data unit* (APDU). There are three types of control field formats. The I-format, the S-format and the U-format. Since this thesis focuses on the ASDU, only packets with the I-format are considered, which are the only packets containing an ASDU. The ASDU is the part of the packet where all the relevant information for this thesis is found. Furthermore, only fields considered in this thesis are explained. Since third party libraries are used in this thesis for the parsing of the packets, the exact bytes are not covered here. For further information about the protocol and the exact structure we refer the reader to [7] or [10].

The **application protocol control information** (APCI) is the application header. It contains general information about the structure found in the packet. Since this thesis focuses on the ASDU within the APDU, the important bytes are the first two bytes of the first control field octet inside the APCI. This is where the type (I, S or U) is defined. If the first byte is 0, the control field format is from the type I and the packet has an ASDU. This is then used for further analysis. All other fields inside the APCI are not covered in this thesis.

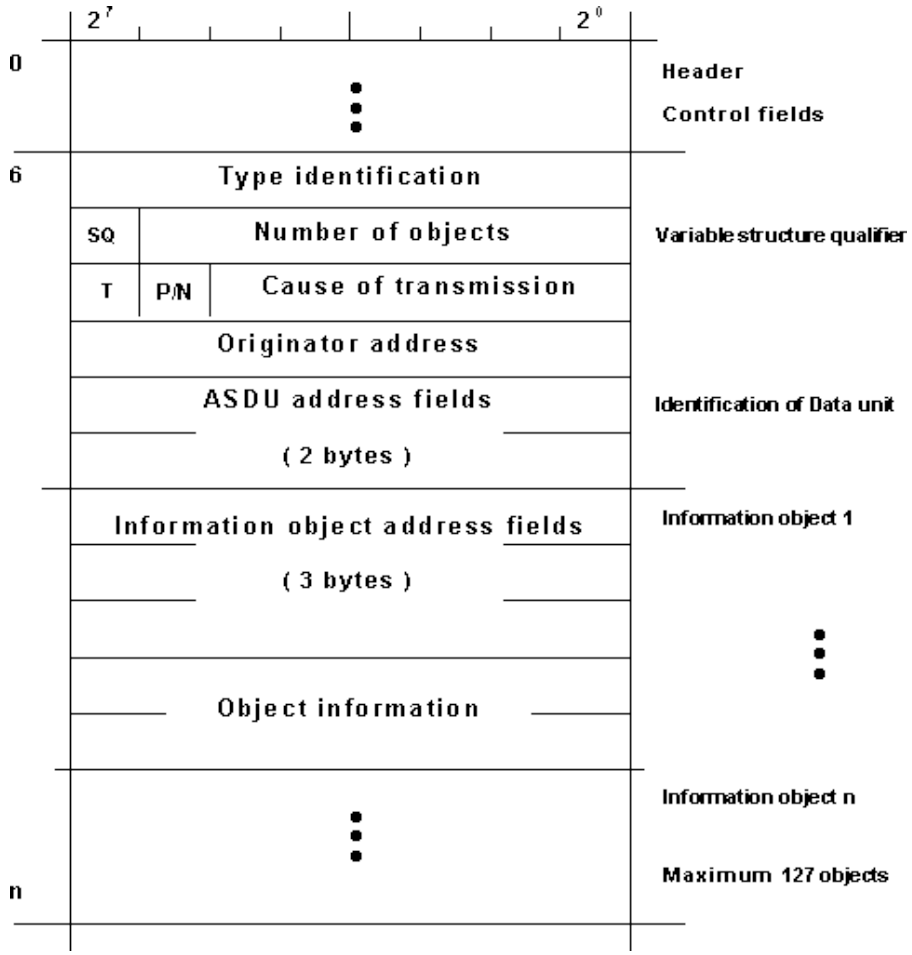

Figure 3: Structure of the ASDU [10]

The **application service data unit** (ASDU) structure is slightly more complicated. It is shown in Fig. 3. The important fields for this thesis are: "Type identification", "Number of objects", "ASDU address fields" and the "Information object address fields". These fields are important, because they identify a special operation within the network. This will be addressed later in the thesis in Sec. 3.4. In addition the network address of the communicating devices would be needed, but this thesis concentrates on the analysis of the communication between two devices only. Therefore the communicating devices are always known. The *Type*

identification (TypeId) refers to a list of known values in the range from 1 to 127, although not all numbers are in use. Each number is mapped to a function that the device should execute, e.g., 103 → *clock synchronization command* or 102 → *read command*. *Number of objects* defines the number of *information objects* found in the ASDU. This can be more than one. The TypeId is assigned to all these information objects inside this packet. The *ASDU address field* (ASDU-Address) contains the address which all objects of the ASDU refer to. The *Information object address* (IOA) refers to an specific information object. A packet may have up to 127 information objects, with the depending object information. An information object can be an actuator or a sensor attached to the device. The device in this case is, for example, a PLC. Attached to the PLC are various sensors and actuators over serial lines. Each sensor or actuator is connected to a port, which is then refered to by an IOA. It is to note that each vendor has its own specification for their devices, although the general structure found in [7] and [10] applies.

As shown in Fig. 1, PLCs are part of the field network and sometimes also of the control network. This depends on the architecture of each particular network. Sometimes the field network and the control network are ingrained into the same physical network. The system of Fig. 1 is connected to the internet. Although the networks are protected by firewalls, an attacker might still be able to infiltrate the system. Various ways of bypassing firewalls and other system protections exist. As soon as an attacker has access to the industrial network, the field and control devices are no longer safe.
The IEC-104 protocol is based on TCP/IP - a reliable network protocol. Furthermore the communication between the master station and a PLC can be encrypted. Thus the system is able to exchange messages in a reliable and secure way. But these two facts do not lead to a secure system. The IEC-104 protocol is only responsible for the exchange of information between devices within an industrial network. It has no built in security, which guarantees the communication with a special host or device. Furthermore no mechanism for intrusion detection, intergrity or confidentiality is implemented. Most of the SCADA-systems today are not encrypted and the TCP/IP-protocol also does not control what is sent within a communication. This leads to the problem that an attacker can compromise the field or the control network. This thesis focuses on sequence attacks, an attack scenario which is not detected by most of the IDS-systems today because they focus on packets with malicious content [4]. Before the sequence attacks are discussed an introduction into the terminology IDS is given.

2.3 Intrusion detection systems

When it comes to security there are several approaches to protect a system. They all target on a specific type of security, e.g., removing malicious software, blocking content that is not

allowed to enter or exit a system or detecting suspicious activities. All these systems try to be as precise as possible. However not all systems are equally precise and some alert activities are malicious although these activities are absolute legit to the system. These alerts can be classified using a *confusion matrix*.

A **confusion matrix** is a specific table layout that allows one to see if an algorithm, used for the classification of something, works as expected. Within the matrix each column represents the instances of a predicted class and each row represents the instances of the actual class. In this work we classify malicious activities and valid activities. However in terms of security an activity might also be a network packet or a file. An algorithm that works 100% correctly will classify every valid activity or file as valid and every malicious activity or file as malicious. However this does not apply to many algorithms and leads us to four classifications in our thesis:
A **true positive** occurs when something that is recognized as malicious and is in reality malicious. This can be a virus for example. A **true negative** occurs when something that is recognized as harmless and is in reality harmless. This can be a normal picture or a simple text document. A **false positive** occurs when something that is recognized as malicious although it is not. This can be a Microsoft Word document with macros, whereby the macro is just used for the forms inside the document but is recognized as harmful to the system. A **false negative** occurs when something that is recognized as harmless is in reality harmful and malicious. An example is a program that hides inside a file. The file is scanned and recognized as harmless. But by opening the file the malicious program inside the file executes and compromises the system. This is the worst of all scenarios since no alarm is triggered although the system is compromised.

This thesis focuses on intrusion detection systems. The other approaches were just examples of other ways to secure a system. An intrusion detection system (IDS) is a security system that monitors computer systems or network traffic and analyzes the behavior of traffic for possible hostile attacks [11]. Computer systems and networks can be secured in multiple ways e.g., firewalls, anti-virus-software or physical access limitation etc. All these systems have a specific aim - none of them can handle them all. An IDS aims to detect hostile attacks on a system or network. IDSs can be divided into two dimensions. The first dimension is the target the IDS is monitoring. It consists of two types, host and network based IDSs. The second dimension can be distinguished between the method they identify attacks - static versus heuristic versus anomaly detection method [11] [12].

The first dimension is separated into *host based* and *network based* IDSs. The question this dimension answers is, what the IDS should monitor. If it is a special host and the activities inside the host, then the host based approach is to select. If the network is the

target to monitor, then the network based approach is to select. Below the two types are explained.

A host based IDS must be installed on every monitored computer. The advantages of such a system are that they can make very specific statements of an attack on the system and the system is comprehensively monitored. What is an advantage on the one hand is a disadvantage on the other hand – the IDS is specialized for that operating system and can barely be used on other operating systems. Another disadvantage is that the system can be victim itself by a denial of service attack – i.e., a technique where a system is paralyzed due to overloading [12].

Network based IDSs record the network traffic, analyze it and report malicious activities. Since the network traffic can be tapped at a central point of the network only one system is needed to monitor the whole traffic for a network segment – this applies to switched networks only. However the bandwidth of networks may be greater than the bandwidth a system can analyze on the fly. To be able to continuously monitor the network, some packets need to be dropped if the throughput is greater than the system can analyze in the same time. This results in a loss of precision of statements. Another disadvantage is that the failure of the network IDS leads to a complete unmonitored network. There are several places where to implement the network based IDS. Fig. 4 shows three different options [11].

1. The IDS is connected to "SWITCH A" (red line). In this scenario the IDS recognizes all hostile activity leaving or entering the network. However it will not recognize hostile activity within the network itself. An attacker inside the network can still harm the system.
2. The IDS is connected to "SWITCH B" (purple line). This is nearly the same scenario as in option one. The difference is, that the IDS will not "see" hostile activity that is blocked by the router. It might be of interest to see what is happening outside and who is trying to get into the network.
3. The IDS is placed at "SWITCH C" (blue line). The IDS now recognizes hostile activity within the network as well as malicious activity that comes through the firewall. It still does not recognize hostile activity from outside that is blocked by the firewall or the router [11] [12].

The second dimension follows up with the question how an intrusion is detected. There are three main approaches how a malicious activity can be detected – *static analysis*, *heuristic analysis* and *anomaly based detection*.

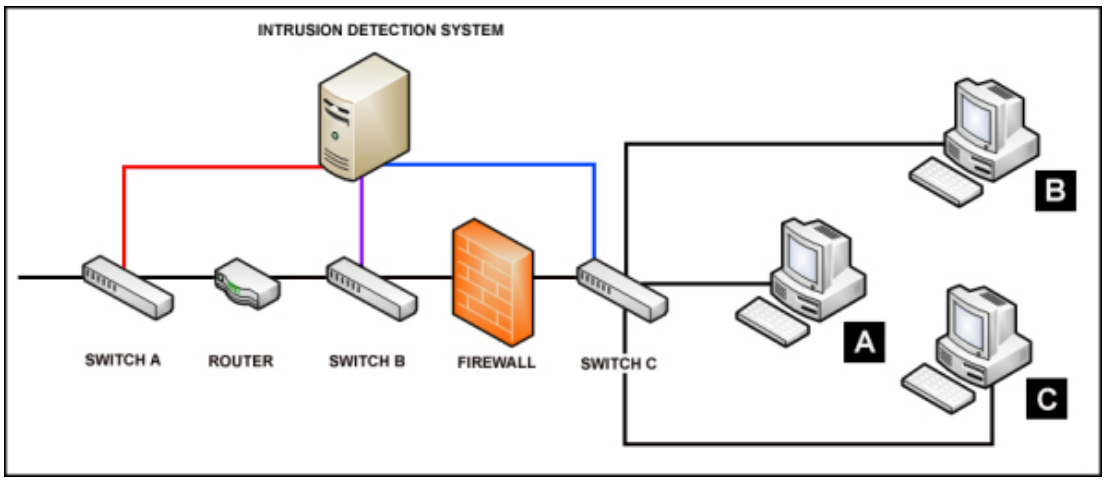

Figure 4: Integration of an network based IDS [13]

Static analysis is based on signatures. A signature is a unique sequence of bytes used to identify a specific type of data, e.g., a file, a virus or a network packet. The IDS has a database of different signatures which are known to be malicious. They are often classified by the potential of their danger [11].
IDSs with the static analysis passes four main stages. In the first stage the system must be trained with the signatures it should identify. In the second stage the IDS perceives the traffic. This is an essential part of the IDS because if the system is placed on the wrong part of the system it may not be able to recognize anything. The third stage is the analysis of the traffic. Within this stage the database with the signatures, which was set up in the first stage, is essential, since the IDS identifies the malicious activities based on the databases knowledge. Stage four is the action that is triggered if a malicious activity is recognized. This can be an email, a pop up on the operators screen, a text message, etc. The operator may manually initiate an action if necessary [11].

Heuristic methods take place everywhere, where unknown attacks or attacks with a variation shall be detected. Based on little information they can produce relatively good statements that are close to reality. However due to their nature they also produce more false positive alerts - i.e., an alert that is in reality no alert [11].
The process of this method is similar to the static analysis. In stage one the needed information is stored in a database. There may be a lack of information so that the system can not fully guarantee the identification of a hostile attack. This may be intended if the algorithm works significantly faster. Otherwise it may be caused if the defenders have not collected enough information so far. Stage two applies equally. In stage three the analysis of the traffic is different. It will no longer create a signature of the packet comparing it with

signatures in the database but rather tries to get an answer from a complex algorithm. There are several approaches how to search for unknown attacks or those with a little variation. Since this thesis does not focus on heuristic methods, these approaches are not discussed here. Stage four is the same as in the static analysis [14].

Anomaly based detection is a method of identifying a hostile attack by considering every activity as hostile if it is not known to be normal to the system. This presupposes that the IDS is trained with activities that are allowed in the system. The system is therefore divided into four stages as well. In the first stage the system is trained to recognize normal activities. Stage two is the same as in the static analysis. In the third stage the recognized activities are tested against the known activities. If the tested activity is not known to the system, it is potentially hostile. Stage four then applies and a message is sent to the operator [14] [11].

This work will focus on the network based IDSs. Furthermore we use the anomaly based detection approach. The reason is that the build system can be interpreted as a database of known and good activities. All activities that are unknown to the system are potentially malicious. Therefore one is able to detect nearly every attack that may harm the system no matter if known or unknown. However the system still underlies a dependency. The dependency of the training data itself. The training data is the data set that is used to tell the IDS which activities are good and legit. This data set should include all activities that do not harm the system and that will occur. Activities that do not harm the system and that do not occur in normal traffic will inflate the training system. This results in a long analysis, thus the traffic is not analyzed in real time anymore.

2.4 Sequence attacks

Sequence attacks are a class of attack, which are specific to industrial control systems. They can harm a system by sending misplaced messages or commands although each of the sent messages or commands is valid to the system [4].

Long time industrial systems have been considered safe because of the nature of their existence. They were often proprietary and separated from other networks. With the development of open protocols and the need to be able to remotely control multiple networks, e.g., connecting multiple systems over the internet, the attack vector increased. Moreover, most of the systems are not encrypted, because they are considered safe, e.g., behind a firewall. Thus an attacker can easily gain control over the system once he is inside the network. The attacker can then attempt to take control of the industrial processes by leveraging the lack of integrity and authentication mechanisms. To take control of the process, an attacker can

either try to reprogram the logic of a PLC or directly control the process from the network using the same control messages used by legitimate operators. When having control over the process, an attacker may send commands in such order or timing that is not intended by the process. The sequence of these commands, that may harm the physical process, is interpreted as an attack - the sequence attack [4].

Anomaly based IDSs may recognize those attacks. But this depends on the approach, i.e., how they detect anomalies. There are several approaches for anomaly based IDSs, since there are several ways on how to harm a system with legit commands.
One way to harm a system is to use commands that are valid within the whole system. However, not every legit command is valid for every device in the system. If an IDS is just looking for valid commands within the system, it may not recognize commands that are invalid for a specific device. The device may not handle the command appropriately and may crash. An IDS, that looks for system wide valid commands only, will not recognize this kind of attack. Furthermore, an IDS, that only looks for commands that are valid within the whole system, will not recognize sequences that lead to a compromised system. They only detect single commands that are not normal for the system.
Another way to harm a system is to use valid commands in such an order, that a device misbehaves. Experiments showed that it is possible to successfully attack some PLCs merely by sending messages in an inconsistent order. For example sending a "start program" while the program is running causes an error that is not properly handled by the controller. This leads to a crash of the PLC. This kind of attack may impede the process of communicating with the device. Furthermore the physical process may be compromised as well, e.g., if an actuator for opening and closing a pipe is not reachable anymore [4].
A third way is to interfere directly with the underlying physical process. In a report by the U.S. President's Commission on Critical Infrastructure Protection there is an attack scenario described, that involves a water distribution facility. The scenario describes that major control valves on a water pipeline can be rapidly opened and closed to cause water hammer. Water hammer can cause up to serious damage to a physical system, e.g., noise and vibration or a pipe collapse [15].
Stuxnet is one of the best known malware that used sequence attacks to disturb physical processes and damage plants. Stuxnet focuses on SCADA systems of the Siemens company. It interfered with the control of the frequency converter, that are inter alia used to control the speed of engines. By constantly changing the speed, slowing down and accelerating, the centrifuges were damaged. Since these commands were valid and no known anomaly was detected, Stuxnet was able to leave behind a lot of damage.
In order to be able detect attacks to initiate countermeasures and protect our infrastructure, an IDS is required.

3 Discrete Time Markov Chains and sequences

In this section the DTMC is introduced. Besides the formal definition and attributes used in this thesis, the minimization is discussed. Two approaches of minimizing the DTMC using *bisimulation minimization* are provided. This is followed by the definition of sequences. The definition involves the mapping of the IEC-104 protocol to a sequence. In addition the representation of such a sequence within a DTMC is explained. An algorithm for building a DTMC of a given IEC-104 communication is provided which is used throuout the thesis.

Chapter Chapter 3.1 introduces the Discrete Time Markov Chain and the used terminology in this thesis. In Chapter 3.2 the minimization of Discrete Time Markov Chains is explained. Chapter 3.3 defines a sequence in context of the communication within SCADA networks and the IEC-104 protocol. Chapter 3.4 describes the process a sequence is represented by a Discrete Time Markov Chain. Furthermore an algorithm is provided.

3.1 Discrete Time Markov Chain

A **Discrete Time Markov Chain** (DTMC) is a stochastic process. What distinguishes a DTMC from other stochastic processes is a specific kind of "memorylessness" which is known as the "Markov property". This property defines that within a stochastic process a future state depends maximally on its first predecessor [16].
DTMCs have many applications as statistical models of real world processes. They are suitable for modeling random state changes of a system, in which changes have few or no influence on the future of the system. A simple example is the coin toss. The previous toss does not give you a higher probability in predicting the outcome of the next toss. If the coin shows tails in the current toss the outcome of the next toss is either tail or head, both with a probability of 50%. This applies vice versa as well [16].
A more complex example, where the current state has influence on the process, is a person that stands in one corner of a room [17]. The corners of the room are labeled s_1, s_2, s_3 and s_4 and the person stands in the corner s_1 as initially shown in Fig. 5. The person throws a coin to decide whether he moves clock-wise or counterclock-wise through the room. The person repeats this any number of times. One question may be: Where does the person after n steps stand? A possible stochastic process is $(X_n, X_{n+1}, X_{n+2}, ...)$, whereby X is a stochastic variable and $n \epsilon \mathbb{N}$ an index from a discrete state space. Possible values X_n are $\{s_1, s_2, s_3, s_4\}$. Let

$$\mathbb{P}(X_0 = s_1) = 1$$

be the starting condition, i.e., the person starts in corner s_1: Then the probability of being in state s_2 is

$$\mathbb{P}(X_1 = s_2) = 1/2,$$

and for being in state s_4

$$\mathbb{P}(X_1 = s_4) = 1/2.$$

To calculate the distribution for X_n with $n \geq 2$ we need conditional probabilities. For example the probability of being in state s_3 in the next step, given that one is currently in s_4:

$$\mathbb{P}(X_{n+1} = s_3 \mid X_n = s_4) = 1/2.$$

Based on the decision rules moving around in the room the probability of X_{n+1} with $n\epsilon\{1, 2, 3, 4\}$ depends only on X_n [17].

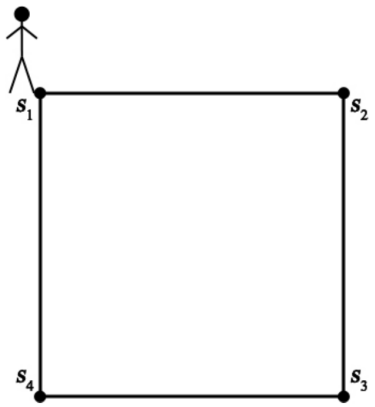

Figure 5: Initial situation of the person in a room [17]

3.1.1 Definition

Let S be a discrete set of states and P be a function that assigns a probability to every transition between two states such that the sum of all outgoing transitions of each specific state is 1. A **DTMC** is described by Eq. (1) [18] [16] [16]

$$DTMC = (S, P, s_{init}, AP, L), \tag{1}$$

whereby

- S is a set of states
- P is the probability function described by (2) and (3)

- s_{init} is the initial state
- AP is a set of atomic propositions
- L is a labeling function - described by (4) - that assigns a (possible empty) set of atomic propositions L(s) to a state $s \epsilon S$

The probability function is defined by:

$$P : S \times S \rightarrow [0, 1], \tag{2}$$

such that the sum of the probability for all outgoing transitions of s equals 1.

$$\sum_{s' \epsilon S} P(s, s') = 1 \ \forall \ s \ \epsilon \ S \tag{3}$$

The labeling function assigns a set of atomic propositions, from the atomic proposition set AP, to each state. It is defined by:

$$L : S \rightarrow 2^{AP} \tag{4}$$

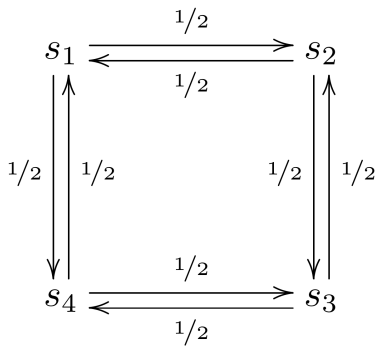

Figure 6: DTMC representing the person in a corner example [17]

Fig. 6 represents the resulting DTMC of the person in a corner example. From each corner the probability is 0.5 moving clock- or counterclock-wise.

A **path** σ in a DTMC is an infinite sequence

$$\sigma = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_i \rightarrow \cdots \tag{5}$$

with $s_i \ \epsilon \ S$. It is a concrete walk along transitions of the DTMC with a probability greater 0 - a possible realization of the stochastic process [19]. Sometimes it is also called an execution of the DTMC [20]. For example $(s_1, s_2, s_3, s_2, s_3)$ is a valid path whereas (s_1, s_3)

is not a valid path since there does not exist a transition between s_1 *and* s_3.

Properties are characteristics that a DTMC or state have. Each concrete DTMC or state may have different properties [20] [21]. The properties that are important for this thesis are:

Reachability: The reachability of a state is described as:
Let S be a set of states, P be the probability function and (S,P) is the DTMC. Let $s \in S$ and $S' \subseteq S$ be a set of states ($s_1, s_2, ...$), then

$$P_{DTMC}(s, S'), \tag{6}$$

is the probability of reaching a state of S' from the state s. For the reachability of S' it is required, that the probability is greater 0. However the exact value of the probability plays no role in this thesis, thus the calculation of these probabilities is not discussed here. The important part is that a state can be reached from another one, e.g., there exists a path from s to a state of S'. A counterexample is shown in Fig. 7. In this example and all following we use the following formalism: $s\_i \hat{=} s_i \ \forall \ i \ \mathbb{N}$. Let S' be $\{s_1\}$. Then

$$P_{DTMC}(s_3, S') = 0,$$

because there does not exist a path from s_3 to a state of S'.

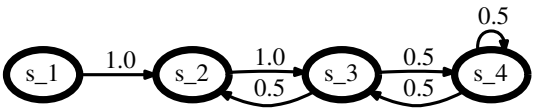

Figure 7: DTMC with non-reachable states

Irreducibility: This property plays a role for the size of a DTMC. It answers the question whether all states of a DTMC are reachable [21].
A DTMC is *irreducible* if from each state, one can reach any other state in a finite number of steps. Otherwise it is called *reducible*. In Fig. 7 state s_1 cannot be reached from state s_2, thus the DTMC is reducible. In Fig. 6 the DTMC is irreducible because every state can be reached from anywhere in the DTMC.

3.2 Minimization

Ensuring the correctness of the identification of sequence attacks is important. If the IDS does not work correctly, it may see attacks that are not there, thus binding too many re-

sources trying to find the non-existent source of the false alarm. An even wors scenario is if it does not even recognize the attacks. In addition the software needs to work without delays in real time to detect current attacks. To test traffic in real time against the training DTMC, it needs to be at a reasonable size. Thus the size of the training DTMC plays an important role identifying sequence attacks. The DTMCs, built using the approach in this thesis, represent the communication between one PLC and its master station. In power stations or hydroelectric power station the installation of the SCADA network typically involves more than one PLC. An IDS needs to keep track of the communication of all components in the network segment as discussed in Sec. 2.3. Thus the size of the DTMC plays an even bigger role in terms of the complexity of the IDS [4] [22].

Using a small amount of training data will held the resulting DTMC small. This implies that the identification may not be precise, as states may be missing or relations between the states may be inaccurate. A well trained DTMC is needed for precise statements. This leads to the question how the DTMC can be held at a manageable size without losing the specific properties used to identify sequence attacks. One existing approach is the bisimulation minimization [22] [16]. Let $DTMC = (S, P, s_{init}, AP, L)$ be a DTMC as described in Sec. 3.1. A relation $R \subseteq SxS$ is a strong bisimulation if for any $(s_1, s_2)\ \epsilon\ R$:

$$L(s_1) = L(s_2) \tag{7}$$

both states hold the same atomic propositions

$$P(s_1, C) = P(s_2, C), \tag{8}$$

for all equivalence classes C of S under R.

In our representation of a sequence in a DTMC, states with the same atomic propositions are merged together - which will be covered later in Sec. 3.4. Since we merge the events into one state from scratch, the resulting state holds the same atomic propositions for all events. Hence there is just one equivalence class for each state (multiple events), thus the reaching probabilities are equal. Although the algorithm implements the required properties from (7) and (8) the number of states within the resulting DTMC obtains more than 50 states. The number of states for longer time capture will only increase. Moreover, the presented DTMC is only created for one pair of devices. Since one DTMC is needed for each pair of devices communicating, a further minimization is needed. A first approach is to delete all non reachable states. Since we are interested in reachable states within the DTMC the non reachable states can be removed. Looking at how we build the DTMC this can be done with a small algorithm:

The input of Alg. 1 is a DTMC and the output is the corresponding irreducible DTMC that we can use in this thesis. Lines 1 till 9 define a loop that repeats itself until no state is removed from the DTMC anymore. In line 2 the variable "statesRemoved" is defined and set to the value "False". Line 3 till 8 define a loop that iterates over all states in the DTMC. Line 4 and 7 define a conditional statement. It the state is not accessible, then line 5 and 6 are executed. In these lines the state is removed from the DTMC and the variable "statesRemoved" is set to the value "True". After the loop of line 3 and 8 has passed through all states of the DTMC, the condition of the outer loop is tested. If the value of the variable "statesRemoved" is "False", there is no further state to remove in the DTMC and the algorithm returns the DTMC. Otherwise the algorithm jumps to line 2 and repeats the steps from there. This approach makes an irreducible DTMC of the original DTMC, i.e., a DTMC where every state can be reached from every other state. The effect of this approach is very small, since there are only a few states at the beginning that can be deleted.

Data: DTMC
Result: irreducible DTMC

```
1 repeat
2     statesRemoved ← False;
3     foreach state ϵ DTMC do
4         if state not accessible then
5             remove(state);
6             statesRemoved ← True;
7         end
8     end
9 until statesRemoved == False;
```

Algorithm 1: Creating a irreducible DTMC

A further possible approach is to merge properties that are not needed for the identification of a specific type of sequence attack, but this might influence the identification of other sequence attacks. The introduction of possible approaches for minimizing the DTMC will be part of Sec. 4. Beforehand the sequences themselves, the representation within a DTMC and the detection of sequence violations need to be discussed.

3.3 Sequences

In order to detect sequence attacks one needs to be able to extract a sequence of messages from the network traffic and identify information that is needed to construct an IDS. The easiest way to define an event in the context of network communications is to consider all the traffic frames one by one. However, all traffic frames are not equally important and not

all frames need to be included in a sequence. Therefore, it is necessary to group traffic frames. This thesis considers the communication between two devices only and therefore the following definition of an event applies [4]:
Let t_n be the time, $n \in \mathbb{N}$ such that $t_n < t_{n+1}$. ***An event*** *s_{t_n} is a six tupel <Direction, Service, Data, FO, LO, Count> whereby:*

- **Direction** contains the information :"request" or "response"
- **Address** contains the "ASDU-address" and the "IOAs"
- **Service** is the "Type-Id"
- **FO** is the first occurrence of the event
- **LO** is the last occurrence of the event
- **Count** represents how often the state is visited

Each event is identified by "Direction", "Address" and "Service". "FO", "LO" and "Count" are used to calculate the probabilities of the outgoing transitions and the median time for a jump to another state.

A sequence (l), is a time ordered list of events s_{t_n} in a system.

3.4 Representation of sequences in a DTMC

In Sec. 3.3 the communication was transformed into a time-ordered list. In order to identify sequence attacks this list needs to be modeled. A DTMC as described in Sec. 3.1 is used to model the communication patterns and protocol behavior. This is done for two reasons [4]:

1. A flexible definition of event is needed that does not necessarily consider all the attributes used to build the sequence. We show in Sec. 4 that it might be useful to merge parts of the event's six tupel that are used for identification.
2. It is necessary to identify temporal consequent events. But only the predecessor is necessary, which is exactly what the "Markov property" demands. The transitions are used to
   (a) indicate the strength of the relationship between an event and its predecessor.
   (b) understand the relationship changes over time.

The algorithm for creating a DTMC from a sequence is presented in Alg. 2. This algorithm is used whether we use the minimization or not, since the minimization consists of the

attribute abstraction. This abstraction is handled by the algorithm in step two, which will be now explained. The algorithm in total consists of 5 steps. *The input data* is the sequence of events. The sequence is time ordered, oldest event first and newest last. *The outcome* or return value of the algorithm is the DTMC which represents the sequence [4].
The first step is a loop over all events in the sequence. It is defined in line 1 and closed in line 14. The loop processes the events in the order of time, the oldest event first and the newest last.
Step two is the extraction of the attributes from an event. The extracted attributes are stored in the variable "$State_{DTMC}$". These attributes are those needed for identification of the corresponding state of the event: Request, Response, ASDU-Address, IOSs, TypeId and the "control field format". This happens in line 2.
Step three handles the new created state. The lines 3, 5 and 7 define a conditional statement that handle the created state and execute the appropriate action. If the new created state is element of the DTMC, line four is executed. In this line the state in the DTMC that equals the new created state from line 2 is updated. The update involves a counter that counts how often this event occurred in the sequence. If the State does not equal any existing state in the DTMC, the new created state is added to the DTMC. This happens in line 6.
Step four handles the transitions. It is a conditional statement which is defined in lines 8, 10 and 12. Line 8 tests whether the transition from "previousState" to "$State_{DTMC}$" is element of the DTMC. If so line 9 is executed. In this line the appropriate transition is updated. The update involves the time of the jump and a counter that counts how often this special jump happens in the sequence. If the transition is not already part of the DTMC, then line 11 is executed. In this line a transition from "previousState" to "$State_{DTMC}$" is added to the DTMC.
In *Step five*, which consists only of line 13, the variable "previousState" is updated. The new value of the variable is the created state of line 2.

Data: Sequence of Events
Result: DTMC representing the sequence of events

for all e_{t_n} ϵ *sequence* **do**
  $State_{DTMC}$ $\leftarrow$ extractAttributes(e_{t_n});
  if $State_{DTMC}$ ϵ $DTMC$ **then**
    update($State_{DTMC}$, DTMC);
  else
    add($State_{DTMC}$, DTMC);
  end
  if $Transition_{previousState,State_{DTMC}}$ ϵ $DTMC$ **then**
    update($Transition_{previousState,State_{DTMC}}$);
  else
    add($Transition_{previousState,State_{DTMC}}$, DTMC);
  end
  $previousState$ $\leftarrow$ $State_{DTMC}$
end

Algorithm 2: DTMC modeling of sequences found in [4]

4 Sequence attacks

Traditional IDSs focus on malicious packets rather than finding sequences of legal commands that can harm the system. This thesis focuses on legal commands that are either sent in the wrong order or with incorrect timing. This chapter shows what kind of sequence attacks exist and how to detect them using a DTMC. Furthermore possible limitations are discussed. In addition the sequence attacks in combination with possible minimizations are discussed.

Chapter 4.1 describes the different kinds of sequence attacks and how they can be detected using DTMCs. In Chapter 4.2 the possible minimizations and their limitations are discussed.

4.1 Detecting sequence violations

Before the system can detect malicious activities a system description is needed. This description needs to be clean and violation free. Of course this can be done by hand, but this would include possible mistakes that occur while entering the data. Furthermore it takes a lot resources and is all in all not practicable. Another yet naive approach is to consider all valid commands as violation free. This section shows that this approach is insecure because there exist ways how a system can be damaged by using only valid commands. Furthermore the resulting DTMC would be far too big because of the state explosion problem. This applies here because one needs all possible transitions between all states which is exponential. A third approach is giving the IDS a description of the system in form of a network trace. This trace needs to be clean and free of violations. The training itself was discussed in Sec. 2. The disadvantage of this approach is that there might be information missing if the trace with a too short period of time was used. If the period is too large the resulting system might be too big so that the analysis in real time may fail. However this seems to be the only approach that has a good trade-off between used resources and possible mistakes. Furthermore the DTMC can be held at an appropriate size while still being able to detect sequence attacks. The resulting DTMC is called the "training DTMC".

A sequence is a succession of events in a system. The succession can be described by a DTMC whereby the transitions are jumps to other states. For example consider a system consisting of two pipes A and B which should never be open at the same time. Furthermore is it not usual to rapidly open and close the same pipe again and again. Legal commands for the pipes are 'open' and 'close'. Let the initial state of the system be s_1 with both pipes closed. In terminology of a DTMC one have 3 states: $S = \{s_1, s_2, s_3\}$ whereby

- s_1 is the initial state with both pipes closed

- in s_2 pipe A is open and pipe B is closed
- in s_3 pipe A is closed and pipe B is open

P is defined such that every pipe is opened with probability 0.5 and closed with a probability of 1:

- $P(s_1, s_2) = P(s_1, s_3) = 0.5$
- $P(s_2, s_1) = P(s_3, s_1) = 1$

Fig. 8 describes the system. The black state with the label 'ENTRY' is the entry point to the system. The blue states describe the safe states of the system.

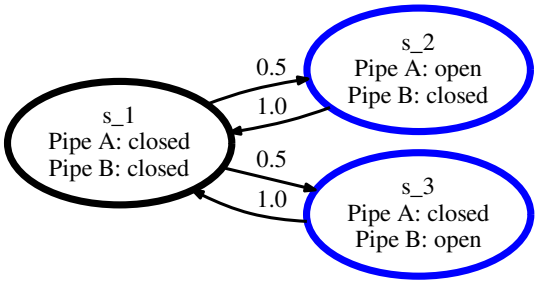

Figure 8: DTMC representing the simple pipe example

To open a closed pipe one has to make sure to be in state s_1 first, e.g closing pipe B before opening pipe A. There are three possible violations that can occur to the system, which are discussed using a simple pipe example.

1. **State exists and transition does not exist**
   A transition-violation occurs when a command or request is sent to the controller at a point the controller wasn't expecting this command. This violates the sequence of commands which the controller expects and may crash it [4]. Consider the system is in state s_2 of the simple pipe example. Pipe A might get the command to open although it is already open. The controller may try to open again, resulting in a force that is unexpected to the physical system. Fig. 9 shows the violation in red.

2. **State and transition do not exist**
   A state-violation occurs when either a command or request is sent to the controller at a point the controller wasn't expecting it or the physical process wasn't prepared for it. In terminology of the simple pipe example the controller for pipe A may receive the

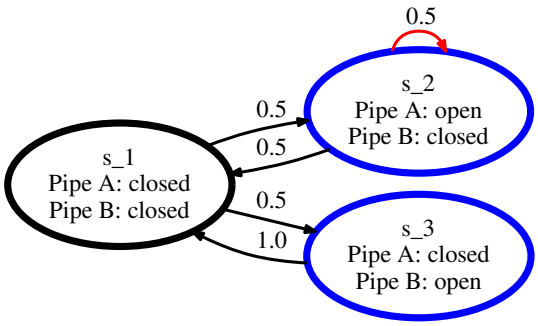

Figure 9: DTMC with a violation compared to Fig. 8 by using a non-existing transition

command to open although pipe B is still open. The controller Controller A reacts to the command by opening the pipe. The system is now in a state where both pipes are open. This may result in physical damage of the system e.g., too much pressure in one of the pipes. Fig. 10 shows the violation in red.

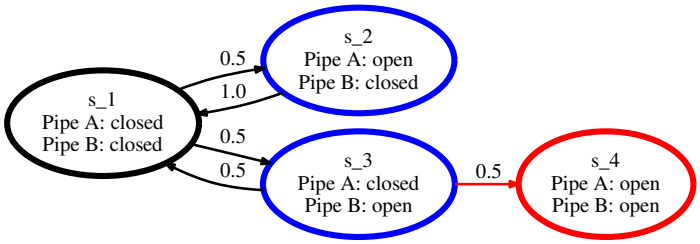

Figure 10: DTMC with a violation compared to Fig. 8 by accessing a non-existent state

3. **State and transition exists but a transition is used too frequently**
   A timing-violation occurs when a single transition is used unusually frequently. The commands are correctly executed at a point the system expects them but the physical process may be harmed. This happens if the physical laws apply that were not foreseen. For example the system is rapidly getting commands to open and close the same valve. This may cause "water hammer", resulting in a number of simultaneous water main breaks. Fig. 11 shows the violated transition-probability in red. It has a significant higher probability - $P(s_1, s_2) = 0.975$ - than the same transition in the original DTMC. Since the probability of all outgoing transitions of one state equals one, $P(s_1, s_3)$ equals 0.025.
   There is a limitation in this scenario. If the training data was collected from several days and pipe B is far less used than pipe A, then it is normal for the transition of

(s_1, s_2) to have a higher probability. A sequence attack cannot be identified in the moment of the command, since the transition exists in the system. If the transition is used frequently over a short period of time, the transition probability might still lay in an interval around the probability that is defined as "not harmful". Therefore a median time is needed to get statements on how frequently over time a transition can be used. The median time can be calculated using the first occurrence of an event and its last occurrence. This means that a single event cannot be tested against the training DTMC. A parallel DTMC need to be built and tested against the training DTMC.

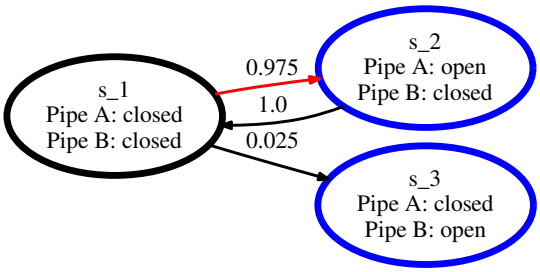

Figure 11: DTMC with a violation compared to Fig. 8 by using a transition too frequently

Apart from the limitation in scenario three which can be handled by looking at the frequency of the transition, there is a general limitation in using DTMCs for detecting sequence attacks. Given a PLC that has two groups of components. Each group behaves differently and consists of different components, but the groups depend on each other. This dependency cannot be mapped within a DTMC. There are always details lost and an attack might not be identified. Consider therefore the following example:
Given a PLC with three pipes – A, B and C. Pipe A always has to be closed when B or C are open. B and C can both be open or closed the same time. "o_X" be "open pipe X" and "c_X" be "close pipe X" with $X \in \{A, B, C\}$ and let all pipes be closed at the beginning. A valid path is:

$$o\_A \rightarrow c\_A \rightarrow o\_B \rightarrow c\_B \rightarrow o\_A \rightarrow c\_A, \tag{9}$$

since pipe B and C are closed. Other valid paths are:

$$o\_A \rightarrow c\_A \rightarrow o\_C \rightarrow c\_C \rightarrow o\_A \rightarrow c\_A, \tag{10}$$

$$o\_A \rightarrow c\_A \rightarrow o\_B \rightarrow o\_C \rightarrow c\_B \rightarrow c\_C \rightarrow o\_A \rightarrow c\_A, \tag{11}$$

$$o\_A \rightarrow c\_A \rightarrow o\_A \rightarrow c\_A. \tag{12}$$

Let these paths in the order (9), (10), (11) and (12) be the training data. Fig. 12 shows the corresponding DTMC, without probabilities for a clearer graph. The blue and the red transitions are those which might be critical. In this scenario however the blue transition is safe, since it can only be taken when all pipes are closed. The red transitions are those to take care of.

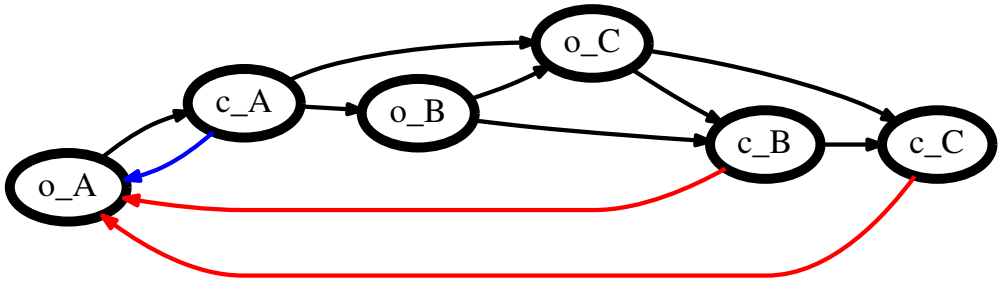

Figure 12: DTMC representing a general limitation

Consider the following path: $o\_A \rightarrow c\_A \rightarrow o\_B \rightarrow o\_C \rightarrow c\_C \rightarrow o\_A$. It is valid, looking at the transitions, but now pipe B and A are open which is a violation to the definition. The reason for this violation is the "Markov property". In this scenario a sequence depends on more than just its predecessor, resulting in possible violations of the system. This scenario shows that there is a limitation in identifying sequence attacks with DTMCs.

In two cases one can identify a violation of the sequence by detecting non-existing transitions in the system. The third needs a deeper look into the transition probability. A fourth scenario showed a limitation by using DTMCs for detecting sequence attacks. This thesis will now focus on the three violations that can be detected. These three violations show that it is possible to use unintended paths within the DTMC by using valid commands only. It leads to the question whether there exists paths with valid commands that can harm the system. In addition possible minimization are discussed with which the violations can still be detected.

4.2 Sequence attack scenarios and minimization

Order based device Compromise: A typical sequence is the upload of new programs to a PLC. The majority of application level protocols provide functionality for this process. This includes messages from unloading and deleting existing programs to transferring, storing and loading new programs. These functionalities are typically achieved by sending sequences of messages. The sequence of messages for uploading a logic program to a PLC looks typically

like Fig. 13. The states represent the events of the process:

- s_1: "unlock the PLC"
- s_2: "lock the PLC"
- s_3: "stop the running program"
- s_4: "delete the existing program"
- s_5: "transfer the new program code to the PLC"
- s_6: "create program"
- s_7: "start program"

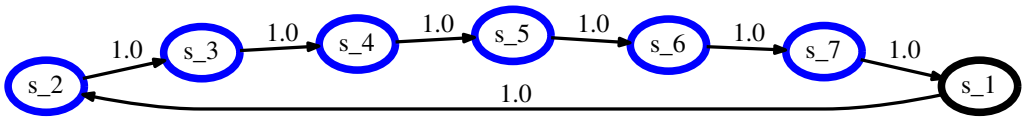

Figure 13: Order based device reprogramming

The process starts in state "s_1". Now the PLC is running. The transition from state "s_1" to state "s_2" is triggered by the command "lock the PLC". Then the command "stop the running program" is sent to the controller and the process is in the state "s_3". The process follows with the commands "delete the existing program", "transfer the new program code to the PLC", "create program", "start program" and "unlock the PLC" and ends in the state "s_7". Experiments in an industrial control laboratory environment have revealed that it is possible to attack some PLCs merely by sending some of these messages in an inconsistent order [4]. For example, sending a valid "start program" message when the program is still running causes an error that is not properly handled by the PLC controller firmware, which causes the PLC controller to crash. Fig. 14 shows the malicious transition in red which causes the PLC controller to crash [4]. The controller is in state "s_7", thus already running. If it now gets the command "start program" it crashes.

Now consider a more complex version of Fig. 13. The controller has 3 sensors attached to it. One can read those values by sending the command "read value" with the specified sensor address. They are addressed accordingly to Sec. 3.3. The IOAs shall be 1, 2 and 3. Fig. 15 is an example DTMC. The result is depending on the count of read commands and jumps to "s_2". In all the new states "s_8", "s_9" and "s_10" the PLC is running. It is valid

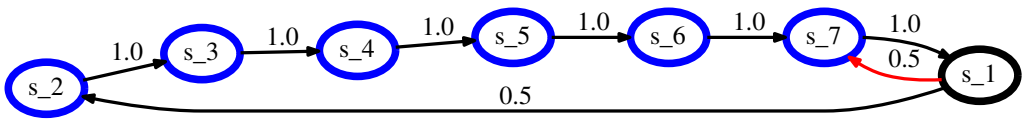

Figure 14: Compromised order based device reprogramming

reading multiple values and after some time executing the command "lock the PLC" since the controller is running.

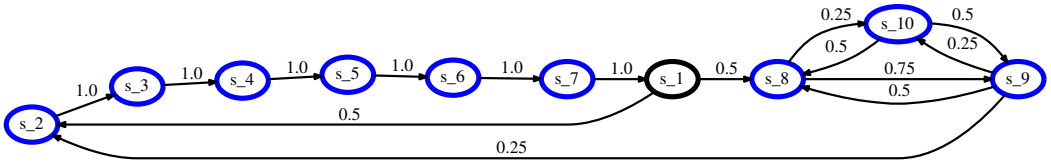

Figure 15: DTMC representing order based device reprogramming incl. other traffic

Now consider this DTMC, being in state "s_8" and executing the command "start program". A transition is used that is not present in the training DTMC. Figure Fig. 16 shows the violation in red.

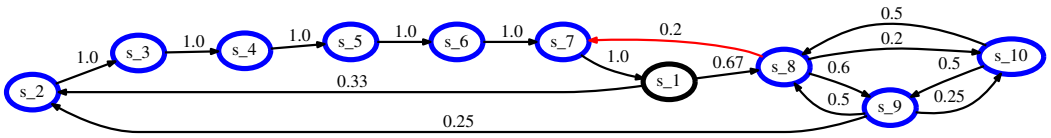

Figure 16: DTMC representing order based device reprogramming incl. other traffic and a violation

Now that a sequence attack can be detected, one wants to try to minimize the DTMC for a faster analysis. This can be done by merging the IOAs, i.e., a state is now identified only by its "Direction", "Service" and "ASDU address". The states "s_8", "s_9" and "s_10" result in one state. They use the same "direction", "ASDU type", "ASDU address" and the IOAs are merged. The "ASDU type" of the commands, needed for the PLC update, differ for all states. Accordingly these states are not merged although they share the same IOA. This is shown in Fig. 18. There is no transition from state "s_8" to "s_7". Thus the sequence attack

of using the command "start program" while being in state "s_8" can still be identified - see Fig. 17. This means the IOAs can be merged while still being able to identify this type of sequence attack. But the following example shows, that this minimization needn't apply to all scenarios.

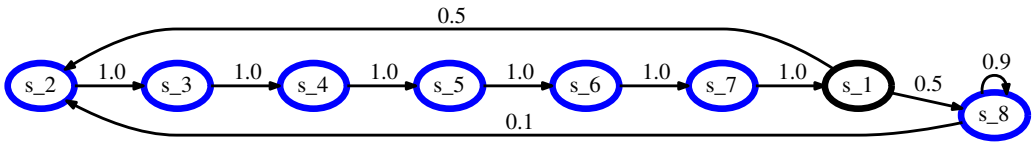

Figure 17: DTMC representing order based device reprogramming incl. other traffic - merged IOAs

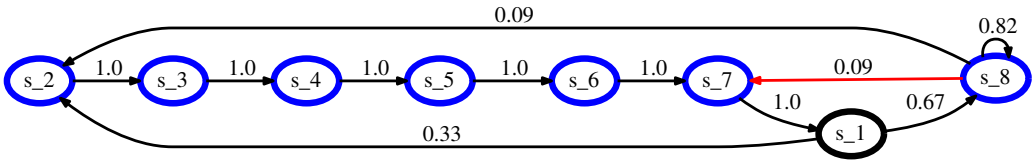

Figure 18: DTMC representing order based device reprogramming incl. other traffic and a violation - merged IOAs

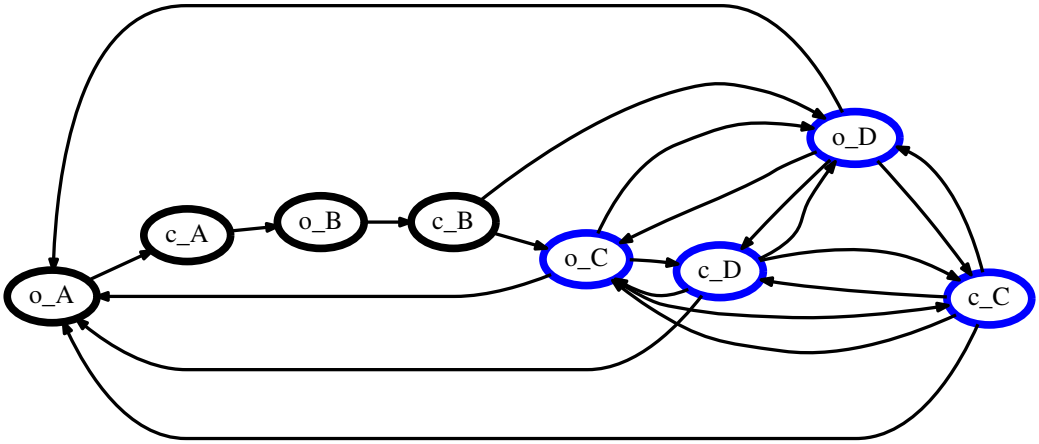

Figure 19: DTMC representing the system of four pipes

Let a system consist of four pipes – A,B,C and D – that can be opened and closed. Initially they are all closed. Pipes C and D can be opened and closed at any time, while A and B always follow the sequence: $open\ A \rightarrow close\ A \rightarrow open\ B \rightarrow close\ B$. The resulting DTMC is shown in Fig. 19. It has 8 states in total and 17 transitions. The black states indicate the sequence that the system must apply. o_X stands for "open pipe X" and c_X means "close pipe X", whereby X ϵ $\{A, B, C, D\}$. A sequence attack by opening pipe B without closing pipe A first would be detected because of the missing transition. If we follow the minimization from above and merge all IOAs, the system would consist of two states. All opening commands are merged together and all closing commands are merged together. Fig. 20 shows the resulting DTMC. o_ABCD stands for "open pipe A or B or C or D" and c_ABCD means "close pipe A or B or C or D". Now the system can only distinguish between opening and closing any pipe. Since it was possible to open pipe C and directly open pipe D a self loop is attached to the open command. The same applies to the close command. Now it is not possible to detect the sequence violation anymore since there exists a transition from open pipe A to open pipe B. This means a new minimization is needed in order to be able to detect this kind of attack. A slightly different minimization would be to merge only specific states. In this example the states $o\_C\ and\ o\_D$ are mereged as well as $c\_C\ and\ c\_D$. The corresponding DTMC is shown in Fig. 21. o_CD stands for "open pipe C or D" and c_CD means "close pipe C or D". The resulting DTMC now has only 6 states and 10 transitions, but the attack scenario will still be detected by the DTMC.

This minimization can be applied to the "order based device programming" example. The resulting DTMC might look like Fig. 18, depending on which read commands would be merged. That minimization would not effect the detection of the sequence attack and still lead to a

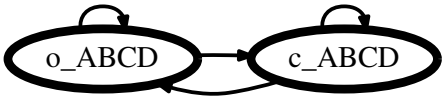

Figure 20: DTMC representing the system of four pipes by merging all reading commands and all closing commands

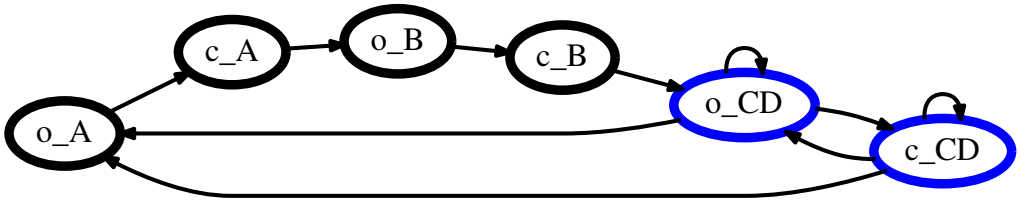

Figure 21: DTMC representing the system of four pipes by merging some commands

smaller DTMC. This means, that the minimization depends strongly on the sequence attacks that need to be detected.

5 Experiments and results

This section discusses how the analysis of the data was realized and shows the results of the experiments. A brief instruction into the used tools is given and the software written for the analysis is briefly explained. In addition, possible implementations of minimizations are discussed. At last the tool is run against data of a company which provided real data for this thesis. The differences between the original DTMC and its corresponding minimized DTMC are shown. The traces the company provided didn't include sequence attacks. They were clean traces with which one can train their IDS. The traces were not customized to include sequence attacks. Thus, the detection of sequence attacks using this approach and the minimization is not discussed in this thesis.

Chapter 5.1 briefly describes how the tool developed for the experiments was written. It also explains how the graphs are to be interpreted. In Chapter 5.2 two approaches of how the minimization can be implemented are described. Chapter 5.3 finally presents the results of the experiments. Non-minimized and minimized graphs are compared and discussed.

5.1 Implementation and representation of the graphs

The tool written for this thesis is based on the programming language python version 3.5. It is a high level language and known to be good for rapid application developing. Furthermore, it is open source and has a large library of additional functionality that can be used within a project. The tool integrates two libraries: "pyshark" [23] for reading the trace of packets and performing a deep packet inspection and "digraph" of the package "graphviz" [24] for generating the graphs.

Alg. 2 was implemented straightforward. There is no additional logic in creating the graphs or generating the DTMC other than structuring the code to keep it readable. For easy identification of the entry to DTMC there is an "entry" state at each graph, which is labeled "entry" and marked with the color **magenta**. All other states are marked with colors as well. States with same colors indicate that they are bisimilar if the minimization - merging all IOAs - applies. Each state has multiple attributes listed. *Direction* indicates whether the packet was a response or a request. The *I* indicates that the events represented by the state were ASDU-Type I events. The *Type-ID* is the type-id from the packet. The *ASDU-Addr* shows the ASDU-address. *IOAs* shows the addressed IOAs. *First Occurance* is the timestamp of the first occurrence of the event in the trace. *Last occurance* is the last occurence of such an event within the trace. *Count* is the number of events merged into this state.
The transitions were labeled as well. There is a fraction and a number $p \in]0, 1]$. The first

number of the fraction indicates how often this transition was used. The second number of the fraction indicates how many jumps in total were made from that state. The number p is the probability, using that transition. The transition lines have different levels of thickness. The thicker a line is, the more likely it is to use that transition within the whole graph. It is not leaned to the transition probability. Instead it is leaned to the number of jumps compared to all jumps inside the graph. This way one can indicate the main path within the DTMC.

5.2 Implementation of the minimization

The minimization was done in two ways. In case (i), the DTMC was built as described in Alg. 2 and afterwards be minimized step by step, which is described by Alg. 3. In case (ii), the DTMC was built using Alg. 2 with another abstraction of the definition state. Thereby a state was identified by only using its "Direction", "Service" and "ASDU address". Both ways lead to the same result. In the following the two approaches are discussed in detail and beforehand Alg. 3 - used by case (i) - is explained.

```
   Data: DTMC
   Result: minimized DTMC
 1 repeat
 2     statesMerged ← False;
 3     foreach state_p ϵ DTMC do
 4         foreach state_o ϵ DTMC do
 5             if state_p == state_o then
 6                 merge(state_p, state_o);
 7                 remove(state_o);
 8                 statesMerged ← True;
 9                 break;
10             end
11         end
12         if statesMerged == True; then
13             break;
14         end
15     end
16 until statesMerged == False;
```

Algorithm 3: Minimization of a DTMC step by step

Alg. 3 expects a DTMC as input and its output is a minimized DTMC. Line 1 till 16 form a loop which repeats until the variable "statesMerged" equals "False". In line 2 the variable

"statesMerged" is set to "False". Line 3 till 14 form a loop over all states of the DTMC. A state inside the loop is referenced as $state_p$. The $_p$ stands for "persistent". Line 4 till 11 form another loop over all states of the DTMC. This time a state is referenced as $state_o$, whereby the $_o$ stands for "obsolete". Inside both loops lines 5 till 10 form a conditional statement. If $state_p$ equals $state_o$ in all atomic propositions, line 6, 7, 8 and 9 are executed. The atomic propositions in this algorithm are: Request, Response, ASDU-Address, Type-Id and the control field format. The IOAs are not part of the atomic propositions anymore. In line 6 $state_o$ is merged into $state_p$. This means the counter of how often this state occurred needs to be set, the first and last occurrence for the median time calculation needs to be updated. This also implies the transitions. The source of every outgoing transition from $state_o$ needs to be set to $state_p$. If the transition already exists in $state_p$ it is updated, otherwise it is added to $state_p$. Also all incoming transition into $state_o$ need to be redirected to $state_p$. In this thesis the IOAs were merged as well, as shown in the graphs. In line 7 the obsolete state $state_o$ is deleted from the DTMC. Line 8 sets the variable "statesMerged" to "True". Line 9 consists of the simple statement to jump out of the inner loop. Line 12 and 14 form a conditional statement: If the variable "statesMerged" equals "True", then line 13 is executed. Line 13 jumps out of the second loop into the first loop. This means as long as one state can be merged into another one, both inner loops are executed. Only one state is merged into another one at a time.

In case (i), Fig. 22 and Fig. 23 show how states can be merged together. Fig. 22 shows the originating DTMC and Fig. 23 is the resulting DTMC after the minimization using Alg. 3. In the figures the colors show bisimular states that are equal. Both yellow states can be merged together, since they equal in the atomic propositions of the minimization algorithm. During the merge, the outgoing transition of the upper yellow state into the other yellow state results in a self-loop. The source of the outgoing transition of the lower yellow state is set to the upper yellow state. Furthermore the first and last occurrences are adjusted and the count of events in the lower state is added to the count of events of the upper state. This is one possible way of minimizing the DTMC.

In case (ii), generating a minimized DTMC of the data, Alg. 2 is used for building the DTMC. Instead of identifying an event with the attributes Request, Response, ASDU-Address, IOAs, Type-Id and the control field format, it is identified using all the attributes but the IOAs. This is a variation of the identification of a state as explained in Alg. 2, where events with the same atomic propositions are merged together. Using the same data as in case (i), the resulting DTMC is identical to Fig. 23. Therefore both ways lead to the same result but using the method in case (i) may lead to unnecessary mistakes because of the extended logic. Thus, a minimization should be done from scratch using Alg. 2 and an appropriate abstraction of the atomic propositions used for the identification.

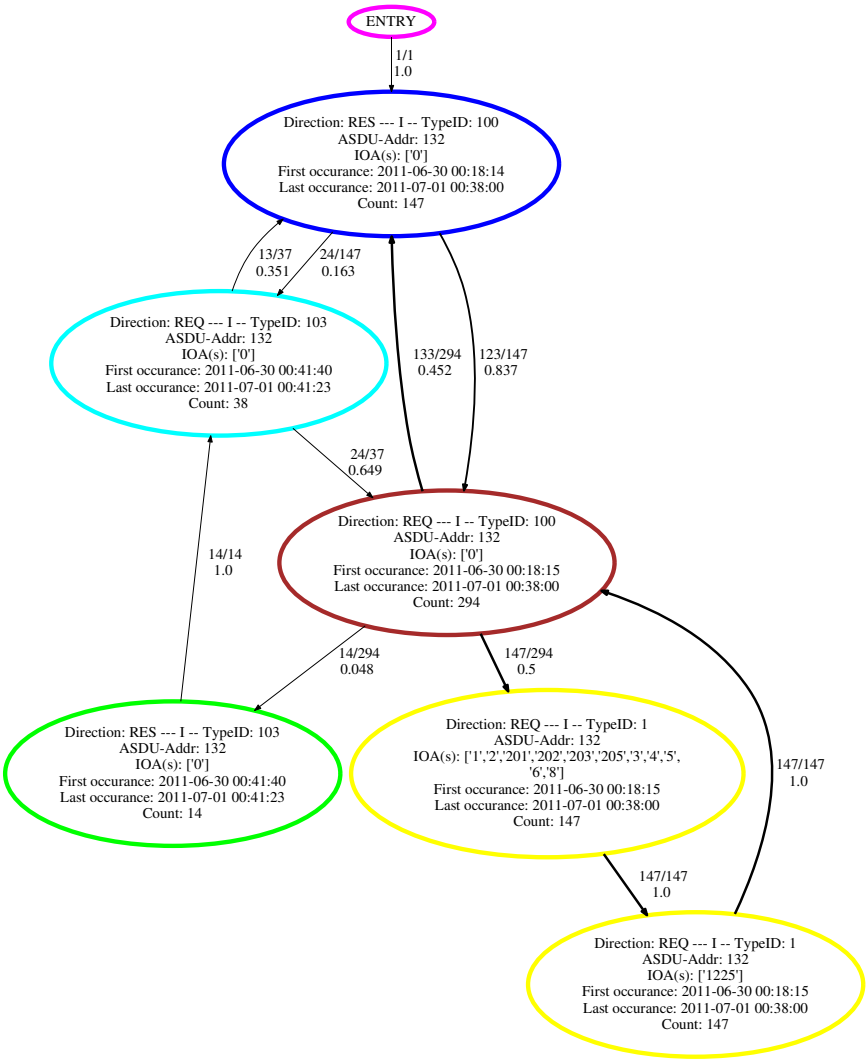

Figure 22: DTMC representing legit activities of device B over the period of one day

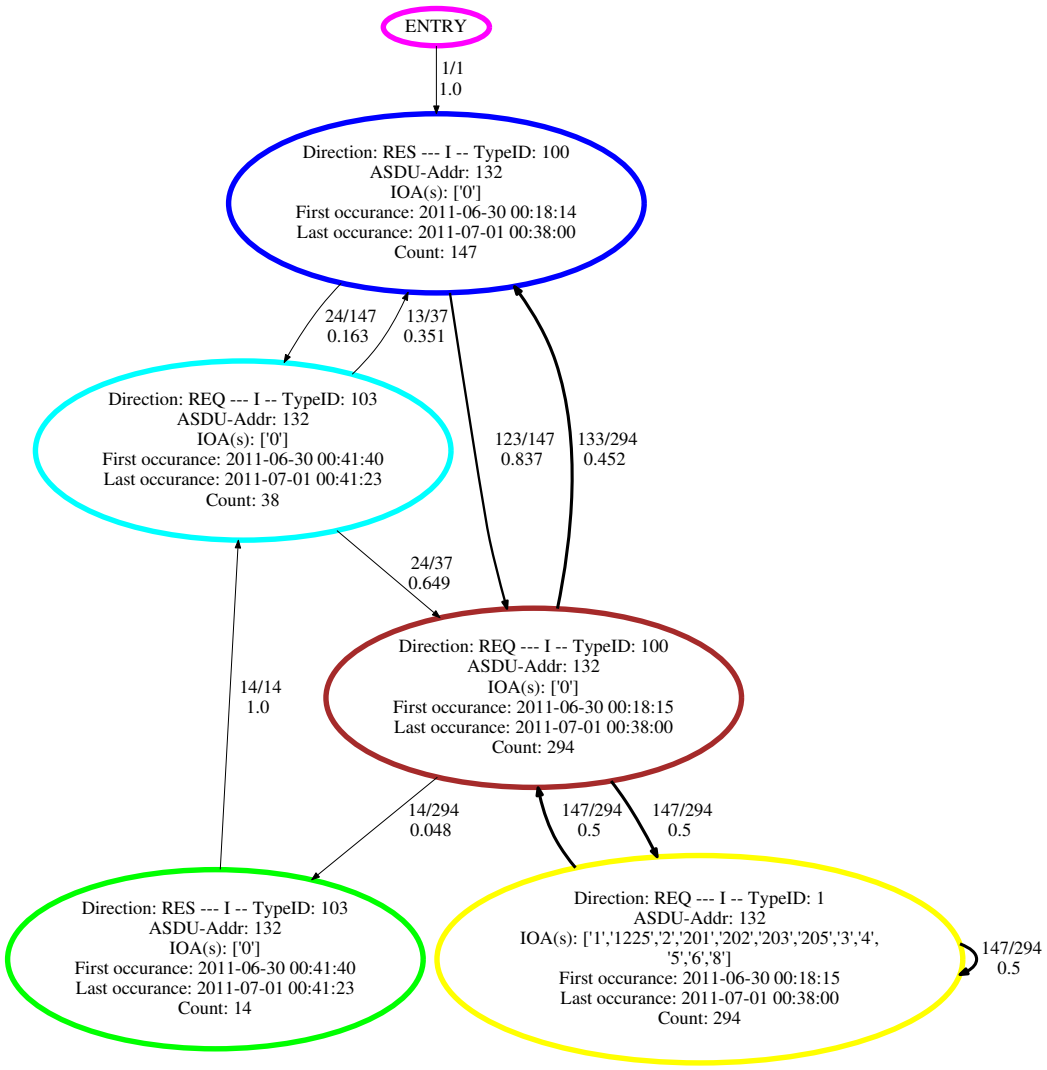

Figure 23: Minimized DTMC representing legit activities of device B over the period of one day

5.3 Experiments on traces

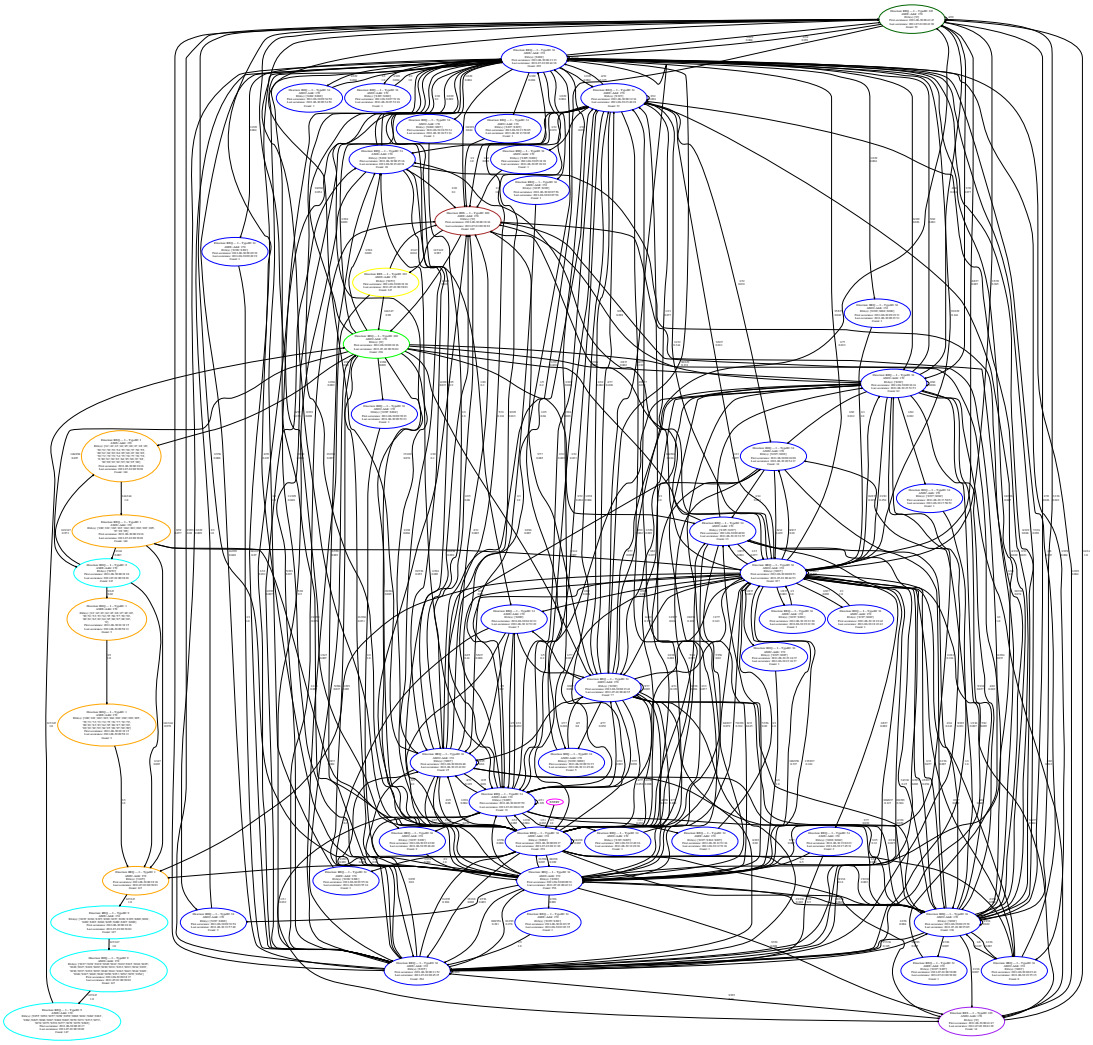

Figure 24: DTMC representing legit activities of device A over the period of one day

In order to demonstrate the advantages of the minimization, an experiment was created. In this experiment graphs with and without minimization were build, using the same data. The minimization used in this experiment was merging all IOAs together, thus identifying a state by all attributes but the IOAs. A possible application for this minimization was given in Sec. 4. The resulting DTMC for a given trace from a company over a period of one day for one particular device A looks like Fig. 24. Although the period of time is still small for a training DTMC, the resulting graph is complex. It already consists of 52 states and 282 transitions. Executing the algorithm on the same data by merging the IOAs results in the graph represented by Fig. 25. The graph now consists of 8 states and 23 transitions. For the states this is a reduction of approximately 84% and for the transitions of approximately

91%. This is a significant reduction of the state space.

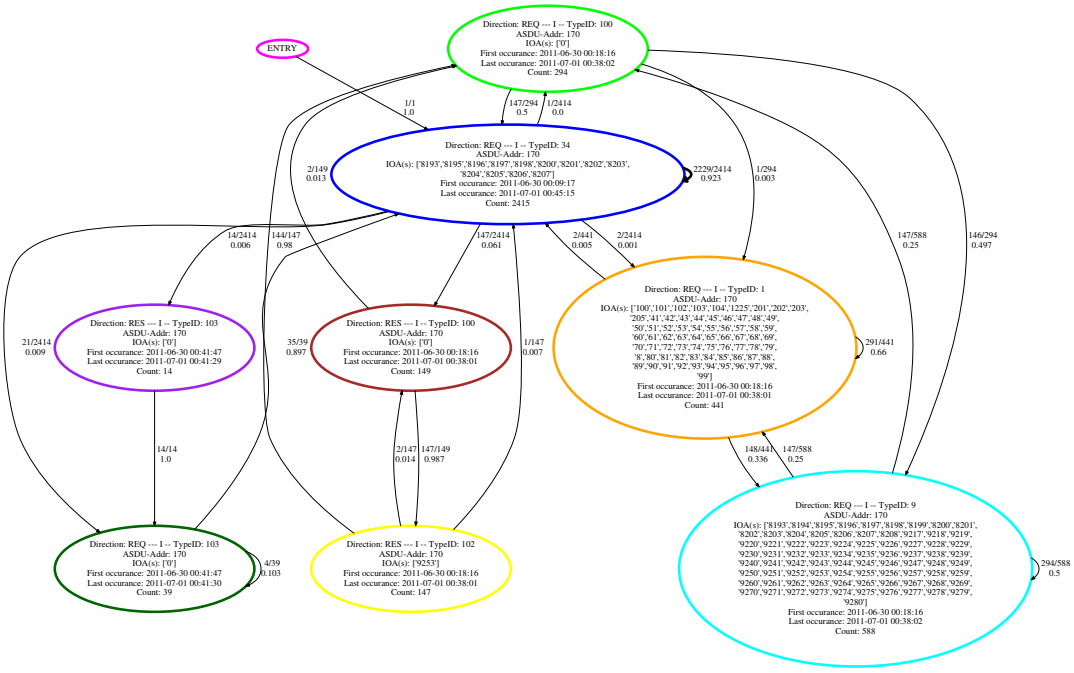

Figure 25: DTMC representing legit activities of device A over the period of one day using the minimization

Comparing the reduction on a trace of a complete week, the reduction is even more significant. Fig. 26 shows the graph representing the trace of one week. It consists of 81 states and 579 transitions. The related graph generated using the minimization is shown in Fig. 27. It only consists of 10 states and 36 transitions. This complies a reduction of approximately 88% for the states and approximately 94% for the transitions. This approach is promising regarding the possible reduction of the state space. However it may not always be possible to detect every sequence attack one wants to look at. Since the detection of sequence attacks is the essential part of the IDS, another approximation may be used.

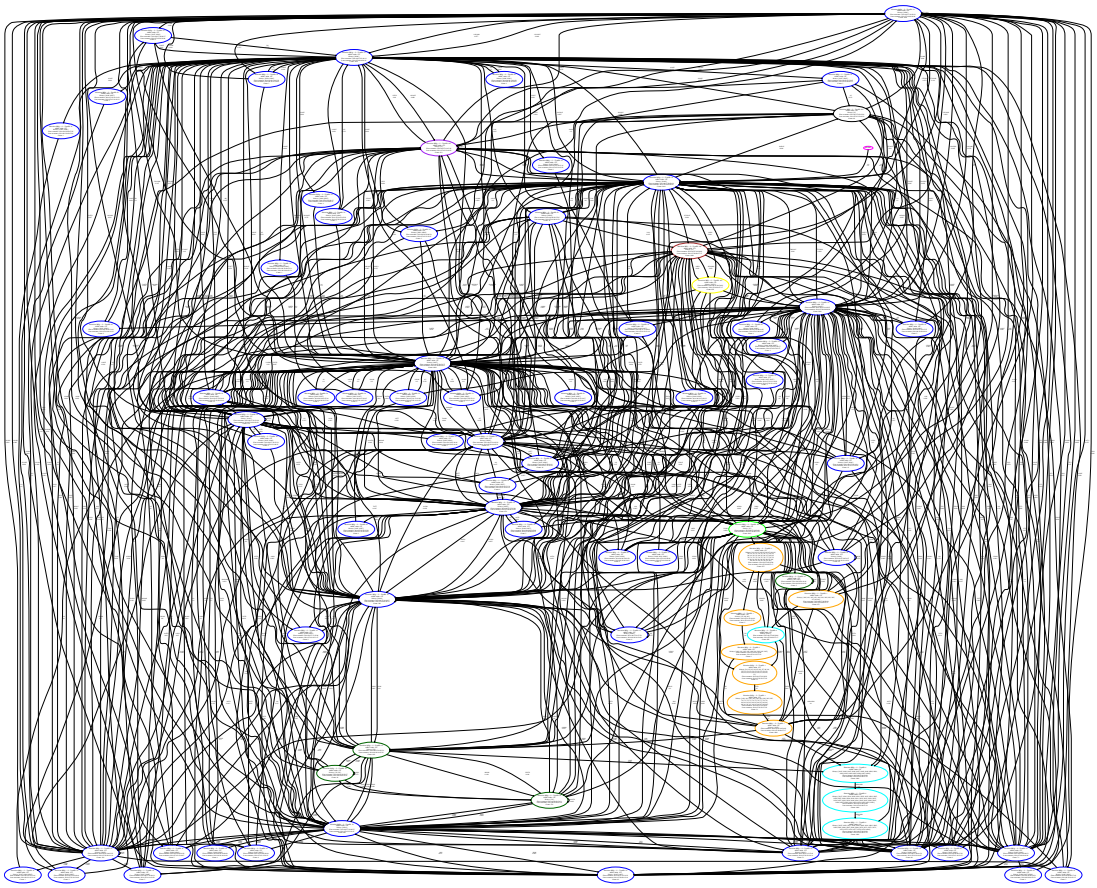

Figure 26: DTMC representing legit activities of device A over the period of one week

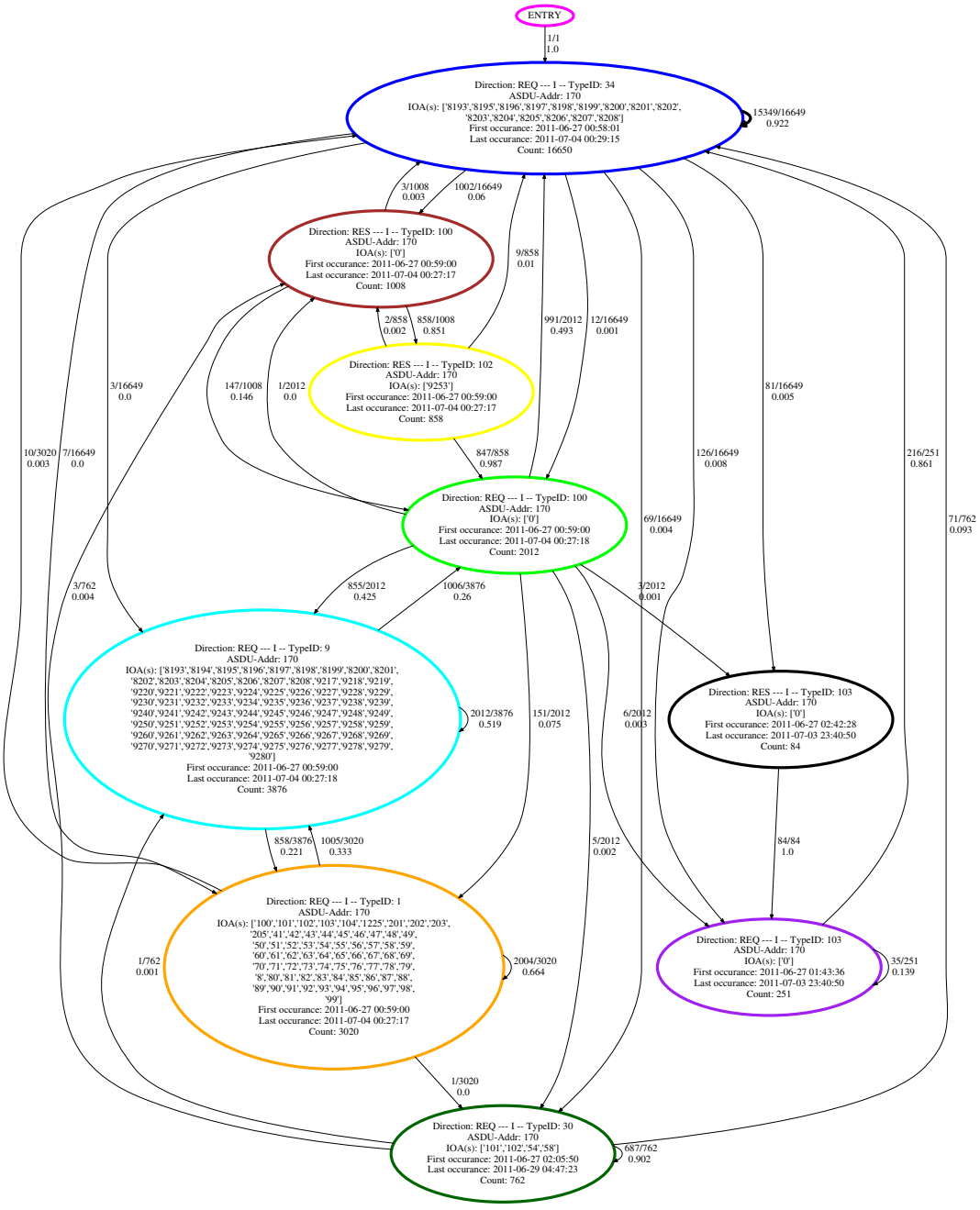

Figure 27: Minimized DTMC representing legit activities of device A over the period of one week

The results from above are experiments on two traces. In order to be able to give reasonable statements, more real data needs to be tested. This was done in the following. The tool was run to create a DTMC without and with minimization. In both cases the states and the transitions were counted and the results are shown in Tab. 1, Tab. 2 and Tab. 3. Seven devices were chosen by random. For each device the test was run using a timeinterval of one hour, one day and one week. The column *Label* is the label of the trace. Columns *States* and *Transitions* list the states and transitions for a given trace without minimization. The Columns $States_M$ and $Transitions_M$ list the states and the transitions for the corresponding minimized DTMC. σ represents the fraction $\frac{States_M}{States}$ and τ represents the fraction $\frac{Transitions_M}{Transitions}$. Thus, the last two columns show the level of minimization. The smaller the level of minimization, the more effective the algorithm worked.

It is to mention, that the time needed for the generation of a DTMC only depends on the count of packets within a given trace. The more packets to analyse, the more time the generation of the DTMC takes. The connection is linear. The tool written for this thesis was using a single core of one cpu only. On an *Intel Xeon E312xx (Sandy Bridge)* with a frequency of 2Ghz the analysis of a trace with 33166 packets needed approximately 141 sec. The number of packets does not contain any information about the time interval of the trace. There are devices with a lot of communication and devices with less communication. A trace containing all the communication within one facility in the company over the time interval of one hour had around 1.400.000 packets. The analysis of this trace took slightly more than one hour. The most time consuming part in this algorithm is the analysis of the packet itself. In addition, a not irrelevant part of constructing the DTMC needs to be taken into account. As the runtime optimization was not part of this thesis, further adjustments on the algorithm need to be considered for real time application. This includes speeding up the deep packet inspection as shown in [25] and possible optimization of the code.

Tab. 1, Tab. 2 and Tab. 3 show that the minimization can lead to significant improvement, but there still remain cases for which the chosen minimization is not that effective. As expected, more states and transitions give more possibilities for minimization. Additionaly, improvements can be seen when comparing the effectivity of one week to one hour.
In summary, the results presented show that the approach of minimization can lead to great improvements and should be considered when using DTMCs for the detection of sequence attacks.

Label	States	Transitions	$States_M$	$Transitions_M$	σ	τ
A	27	70	9	16	0.33	0.23
B	7	11	6	11	0.86	1.0
C	20	34	10	20	0.5	0.59
D	12	17	9	16	0.75	0.94
E	11	16	7	12	0.64	0.75
F	25	44	11	23	0.44	0.52
G	10	14	8	14	0.8	1.0

Table 1: Results using a timeinterval of one hour

Label	States	Transitions	$States_M$	$Transitions_M$	σ	τ
A	52	282	8	23	0.15	0.08
B	7	11	6	11	0.86	1.0
C	28	85	10	33	0.36	0.39
D	15	33	9	21	0.6	0.64
E	11	19	7	14	0.64	0.74
F	34	92	11	30	0.32	0.33
G	13	26	8	21	0.62	0.81

Table 2: Results using a timeinterval of one day

Label	States	Transitions	$States_M$	$Transitions_M$	σ	τ
A	81	579	10	36	0.12	0.06
B	7	12	6	12	0.86	1.0
C	67	185	12	44	0.18	0.24
D	36	84	12	37	0.33	0.44
E	11	24	7	16	0.64	0.67
F	73	238	11	48	0.15	0.2
G	13	31	8	24	0.62	0.77

Table 3: Results using a timeinterval of one week

6 Summary and conclusion

SCADA networks controlling often critical infrastructures are increasingly a subject to cyber attacks. One of the promising methods for improving the security of SCADA networks is the use of intrusion detection systems. This thesis has investigated the approach of [4] and tried to improve it. For this reason, this thesis first introduced SCADA system and the related hard- and software components. The IEC-104 protocol, one of the important protocols used within SCADA networks, was then discussed. Regarding the security of these networks, the term IDS was explained. Several approaches to intrusion detection were discussed and it was shown, that these approaches aim at a specific and not always the same goal. This was followed by the introduction of sequence attacks. Chapter three gave an introduction into the underlying structures of this thesis and its approach to build an IDS. The question why a minimization may be interesting and how a minimization in general works were explained. The representation of sequences in a DTMC is an essential step in this work and was discussed in Sec. 3.4. Furthermore, an algorithm was provided, that builds a DTMC out of a sequence. Sec. 3.3 addressed the sequence attacks and how to detect them using a DTMC. In addition, possible minimizations were shown as well as some problems regarding sequence attacks and DTMCs. Finally, Sec. 5 presented results of one possible minimization, using real data from a company.

Marco Caselli et al. showed that it is possible to detect sequence attacks within SCADA networks regarding the IEC-104 protocol [4]. They used a DTMC to model the communication between two devices and anlyzed the resulting structure. This thesis took up the approach and went one step further. The structure was minimized in order to handle the state space explosion problem. It was shown that a significant minimization of the DTMC is possible, while still be able to detect the appropriate sequence attack. We were able to reduce the state space of a DTMC describing sequences from a week by over 80% and the transitions by over 90%. This shows the potential of this approach and was the main aim of this work.

Nevertheless, it was also shown that the minimization strongly depends on the attack scenario that should be identified, thus each attack scenario needs its own minimization. In order to find a suitable minimization that detects all attack scenarios, one first needs to describe each minimization for each attack scenario on its own. Afterwards, these minimizations need to be merged into one. It might happen that the minimizations cannot be merged. In that case there does not exist a minimization for that configuration. Furthermore, this thesis showed that using a DTMC for the detection of sequence attacks lacks an identification problem. Not all kinds of sequence attacks are possible to describe within a DTMC, since some sequence attacks depend on more than just one time step. Furthermore, some

of the sequence attacks need precise physical description since some processes like "water hammer" strongly depend on the underlying physical process. For precise statements the frequence is essential and this may differ, depending if the target is a pipe, a tank or something else. Nevertheless, it is possible with this approach to detect some kinds of sequence attacks.

There is still space for further research, especially the structure which is used to model the sequences. It may not be practical to find a new minimization for each device and attack scenario for companies with large installations. Future research may therefore focus on finding other approaches, that allow faster analysis out of the box. This faces especially the problem of the expense used to find appropriate minimizations. Upon that, further research may also find other approaches that focus on detecting sequence attacks, which depend on more than one time step. Moreover this thesis didn't investigate in detecting real sequence attacks within network traces. Thus, further research may also focus on detecting attacks on real network traffic. In addition it is worth investigating research in finding out how much time can be saved using these minimizations in order to detect sequence attacks.

List of Figures

List of Tables

References

1. Zhu, B., Joseph, A. & Sastry, S. *A Taxonomy of Cyber Attacks on SCADA Systems* in *Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing* (IEEE Computer Society, 2011), 380–388.
2. Garance Burke, J. F. *AP Investigation: U.S. power grid vulnerable to foreign hacks* <`http://lasvegassun.com/news/2015/dec/21/ap-investigation-us-power-grid-vulnerable-to-forei/`> (viewed 06.06.2015).
3. Goodin, D. *First known hacker-caused power outage signals troubling escalation* <`http://arstechnica.com/security/2016/01/first-known-hacker-caused-power-outage-signals-troubling-escalation/`> (viewed 06.06.2015).
4. Caselli, M., Zambon, E., Petit, J. & Kargl, F. Modeling Message Sequences For Intrusion Detection in Industrial Control Systems. *IFIP* (2015).
5. Caselli, M., Zambon, E. & Kargl, F. *Sequence-aware Intrusion Detection in Industrial Control Systems* in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security* (ACM, 2015), 13–24.
6. Barbosa, R. R. R., Sadre, R. & Pras, A. *A first look into SCADA network traffic* in *Network Operations and Management Symposium (NOMS), 2012 IEEE* (2012), 518–521.
7. Clarke, G. & Reynders, D. *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems* (Newnes, 2004).
8. Wang, C., Fang, L. & Dai, Y. *A Simulation Environment for SCADA Security Analysis and Assessment* in *2010 International Conference on Measuring Technology and Mechatronics Automation* **1** (2010), 342–347.
9. Sherif, Y. S. & Zahir, S. On power line carrier communication (PLC). *Microelectronics Reliability* **24,** 781 –791 (1984).
10. Garance Burke, J. F. *LIAN 98(en) : Protocol IEC 60870-5-104, Telegram structure* <`http://www.mayor.de/lian98/doc.en/html/u_iec104_struct.htm`> (2016).
11. Mitchell, R. & Chen, I.-R. A Survey of Intrusion Detection Techniques for Cyber-physical Systems. *ACM Comput. Surv.* **46,** 55:1–55:29 (Mar. 2014).

12. Intrusion Detection Systems; Definition, Need And Challenges. *SANS Institute Reading Room* (2001).

13. Reiner, M. *Is your Network Intrusion Detecton System blind in one eye?* <`https://misterreiner.wordpress.com/2010/05/17/is-your-network-intrusion-detecton-system-blind-in-one-eye/`> (viewed 20.06.2015).

14. Shah, B. How to Choose Intrusion Detection Solution. *SANS Institute Reading Room* (2001).

15. Clark, R. M. & Deininger, R. A. Protecting the Nation's Critical Infrastructure: The Vulnerability of U.S. Water Supply Systems. *Journal of Contingencies and Crisis Management* **8,** 73–80 (2000).

16. Baier, C. & Katoen, J.-P. *Principles of Model Checking* (Massachusetts Institute of Technology, 2008).

17. Schomaker, J. *Einführung in die Theorie der Markov-Ketten* <`http://wwwmath.uni-muenster.de/statistik/lehre/SS12/SeminarAnwendungenWT/Vortraege/Schomaker.pdf`> (viewed 10.06.2015).

18. Dehnert, C., Katoen, J.-P. & Parker, D. in *Verification, Model Checking, and Abstract Interpretation: 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings* 28–47 (Springer Berlin Heidelberg, 2013).

19. Hansson, H. & Jonsson, B. A logic for reasoning about time and reliability. *Formal Aspects of Computing* **6,** 512–535 (1994).

20. Kwiatkowska, M., Norman, G. & Parker, D. *Stochastic Model Checking* in *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)* **4486** (Springer, 2007), 220–270.

21. Ross, S. M. *Introduction to Probability Models* 193–194 (Academic Press, 2006).

22. Katoen, J.-P., Kemna, T., Zapreev, I. & Jansen, D. N. in, 87–101 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007).

23. Combs, G. *Wireshark* <`https://www.wireshark.org/#aboutWS`> (2004).

24. Gansner, E. R. & North, S. C. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE* **30,** 1203–1233 (2000).

25. Smith, R., Estan, C., Jha, S. & Kong, S. Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata. *SIGCOMM Comput. Commun. Rev.* **38,** 207–218 (Aug. 2008).