Developing a robust communication system for a distributed IDS

Jan Speckamp

August 16, 2023
Version: 1.0

Westfälische Wilhelms-Universität Münster

Department of Mathematics and Computer Science
Group of Safety-Critical Systems

Master Thesis

Developing a robust communication system for a distributed IDS

Jan Speckamp

1. Reviewer Prof. Dr. Anne Remke
Department of Mathematics and Computer Science
Westfälische Wilhelms-Universität Münster

2. Reviewer Dr. Dietmar Lammers
Department of Mathematics and Computer Science
Westfälische Wilhelms-Universität Münster

Supervisors Prof. Dr. Anne Remke and MSc. Verena Menzel

August 16, 2023

Jan Speckamp
Developing a robust communication system for a distributed IDS
Master Thesis, August 16, 2023
Reviewers: Prof. Dr. Anne Remke and Dr. Dietmar Lammers
Supervisors: Prof. Dr. Anne Remke and MSc. Verena Menzel

Westfälische Wilhelms-Universität Münster
Group of Safety-Critical Systems
Department of Mathematics and Computer Science
Schlossplatz 2
48149 Münster

Contents

Nomenclature

API	Application Programming Interface
C2	Command & Control Server
FLOSS	Free/Libre Open Source Software
GM	Global Monitor
GSM-R	Global System for Mobile Communications - Railway
HLR	Home Location Register
IDS	Intrusion Detection System
LM	Local Monitor
MAC	Message Authentication Code
MSC	Mobile Switching Centre
NM	Neighborhood Monitor
OCSP	Online Certificate Status Protocol
OPC-UA	Open Platform Communications Unified Architecture
PEP	Python Enhancement Proposal
PKI	Public Key Infrastructure
RTU	Remote Terminal Unit

List of Figures

List of Tables

1 Introduction

With the ongoing change from centralized hierarchical energy grids towards more decentralized and smart energy grids, strategies for monitoring and securing such grids pose a great challenge and form a large base for research. For this reason, this master thesis extends the distributed process-aware Intrusion Detection System (IDS) developed in previous work by Menzel [1] and Chromik et al. [2, 3].

1.1 Motivation

The power grid is one of the most prominent critical infrastructures in use today, and its importance for every day life cannot be understated. Consequently, securing the grid against malfunctions and attacks by malicious actors is of high importance. This necessitates research into improving the security of the grid. For this, Chromik et al. [2, 3] and Menzel [1] researched and iteratively developed a distributed process-aware IDS, designed to monitor and protect the power distribution grid. Chapter 3 will define and analyze the IDS in detail. As the IDS is designed for integration into critical infrastructures, it needs to be able to handle real-world conditions, such as malfunctions of components or attacks against the IDS itself [4]. While previous research has shown promising results for monitoring of the grid given idealized test scenarios [1, 2, 5], the resilience of the IDS itself has been left out of scope until now. Especially the communication system, which constitutes a core part of the distributed IDS, has not been analyzed in detail.

The need for a thorough approach when designing the communication system shows when recent attacks on critical infrastructure such as the attack on the German train network on the 8th of October 2022 are examined. Unidentified attackers halted all train traffic in northern Germany by simply severing two fibre connections, which cut off connection to two centralized components that are required for operation. The communication network was not resilient enough for the operating conditions, compromising its operation. Section 2.4 will elaborate further on this incident.

1.2 Objective and Research Questions

This thesis aims to achieve the objective:

Enhance the IDS to strengthen its resilience against internal malfunctions and external threats by developing a robust communication system.

This will be achieved by answering three different research questions. They aim to first formalize the system with special focus on communication aspects, expand the IDS to handle internal malfunctions in a second step, and finally evaluate and prototype defensive strategies to protect against external threats.

Research Question 1

How should communication inside a hierarchically-organized distributed IDS be facilitated?

The distributed nature of the IDS naturally leads to the need for communication between different components. A formalized model (referenced as *base model*) will be developed that describes the IDS. It specifies what data needs to be exchanged to allow for operation of the IDS and specifies workflows that should be used to facilitate this communication. The three main factors that are evaluated are data minimization, data integrity and efficiency. The base model describes the IDS given perfect conditions, explicitly neglecting any operational issues that may arise in a real-world scenario. Based on this model, the current prototype is analyzed and a set of improvements compiled. At least all changes that are integral to the fulfillment of this thesis will be implemented into the prototype.

Research Question 2

How can the IDS organize itself without the need for centralized management?

The IDS incorporates several different components, whose interaction with each other forms the basis for the monitoring. These components are distributed across the grid, forming a large and dynamic network. This poses challenges, as the components need to be orchestrated, i.e. components need to find each other to connect and exchange data. While this could be achieved using a central management entity, using this approach negates many of the benefits. It inherently introduces a single point of failure, compromising the resilience of the whole IDS. The IDS should

therefore be able to organize and manage itself without the need for such an entity, while still supporting common workflows and dynamic reconfiguration of the system during operation.

Research Question 3

How can the IDS be resilient against malfunctions and outside attacks?

While perfect conditions prevail in a test scenario, it is important to consider malfunctions of components in an environment representing the real world. To handle this, several core assumptions of the IDS need to be revised. Specifically assumptions about availability, synchronization and topology shall be updated to reflect the real operating environment more appropriately. As the IDS is designed to run on a physically distributed system, attack scenarios that partially compromise the network are immanent. Such scenarios can for example constitute a Byzantine Generals Problem [6], where some nodes inside a network act treacherous. Active defense of networks against such conditions, e.g. by detecting and excluding such nodes from the active system, could help to minimize the impact of such attacks [4].

Evaluation

A practical evaluation of changes implemented in the IDS will be done using a physical testing environment based on the Raspberry Pi Platform. This extends previous work by Großhanten [5] who showed the feasibility of running the IDS on this distributed physical hardware. As the testbed currently only includes two monitored subnets, the testbed is further expanded to allow for testing resilience in more detail.

1.3 Outline

This thesis is structured into eight chapters. The first two chapters give an introduction to the topic, state the problem, and identify research questions to be answered.

The third chapter then formally defines the existing IDS on a conceptual level. Based on this, requirements that need to be met by the communication network, i.e. necessary workflows and data that needs to be exchanged, are identified and described. Additionally, all data that is sent through the system is analyzed and

recommendations on the methodologies that should be used for transmitting the different types of data are derived. To ensure this analysis is not influenced by the already implemented prototype, and may have shortcomings and use suboptimal methodologies, it is performed on the formal model only.

In chapter 4 the existing prototype is evaluated against the previously defined requirements. This analysis focuses on areas that show potential for improvement, specifically highlighting issues concerning some of the design choices at a fundamental level.

The fifth chapter remediates the identified weaknesses and problems of the existing prototype by redeveloping the communication system used by the IDS. Rumor-spreading algorithms are introduced and identified as a promising methodology for implementing the communication system.

To show the feasibility of the newly developed system, the theoretical developments are then prototypically implemented. Chapter 6 details the newly developed prototype and describes organizational aspects and the implementation.

The seventh chapter validates whether the previously developed prototype functions properly and tests if the problems identified in chapter 4 are successfully remediated. For this, the testbed introduced in previous research is expanded and different scenarios are tested.

Chapter 8 gives a conclusion on the thesis and reflect on the research questions and the answers that were given. Based on the results future steps are outlined and the research results reviewed.

2 Background & Related Work

To establish a common understanding of the environment this thesis is based on, as well as showing the relevancy of research into the topic, this chapter describes and analyzes the most important factors that motivate this thesis. Additionally, it establishes terminology and showcases the operating environment the system faces. Section 2.1 first gives a short introduction to electrical power grids, specifically so called *smart grids*. The focus lies on the historical development and the threats critical infrastructures like smart grids face.

Afterwards, *Supervisory Control and Data Acquisition (SCADA) Systems* which are very commonly used in electrical grids are introduced. As their name suggests these systems are used for monitoring and controlling the grid and are therefore very safety-critical. Previous attacks on such systems are sketched out, emphasising the need for researching methods to secure them.

Section 2.3 then introduces *Intrusion Detection Systems* (IDS) which are the de-facto standard for identifying and defending against attacks on digital systems. The different methodologies that can be used as well as common approaches to implement them are outlined.

The concept of *resilient communication*, whose implementation is the main goal of this thesis, is introduced in the last section. Specifically, the necessity of a robust communication network is showcased by analyzing attacks against critical infrastructures that exploited issues caused by the lack of resilience in the communication network.

2.1 Smart Grid

The electricity grid is certainly one of the best-known examples of a critical infrastructure that is important across national borders and has been in daily operation since its beginnings more than a century ago. As technology progressed over the last century the demand for electricity steadily grew and the grid had to adapt accordingly. This did not manifest in an abrupt change, as the grid is far too large

and complex to allow for this, but as a still ongoing gradual development. As summarized by Palensky et al. [7] there are two main factors that characterize the development of the grid:

- The energy generation and consumption got a lot more diverse. Especially with the rise of renewable energy, e.g. solar energy or wind energy, the amount of producers grew and they are now geographically distributed over a larger area. Solar energy and singular onshore-windmills are very decentralized, while offshore windparks are very centralized but quite far away from the consumers. This starkly contrast the historical structure, which was characterized by centralized power generation, e.g. by big nuclear or coal power plants, as the sole producers of electric energy. These centralized power plants were mostly located very near to areas with high energy consumption, only requiring transport over small distances. This development makes the grid much more complex to control and monitor, as it now forms a very heterogeneous landscape instead of the classical small hierarchical structure.
- Information technology has developed immensely. Computing capabilities increased manyfold and almost every device now has networking capabilities. This made distributed computing affordable and provides a new paradigm of controlling the grid, as it opens up many possibilities for monitoring components and remote control capabilities can be vastly expanded.

These developments can be summarized as a development from a *traditional* power grid towards a *smart* grid. It must be noted that those terms are very generalizing and broad and do not constitute a clear distinction criteria. In reality, most systems are hybrid, incorporating smart components as well as components that do not offer any smartness [7, 8].

As those terms are very vague, different definitions exist. For the purpose of this thesis, the definition by the the European Smart Grid Task Force is used, which defines the *smart* grid as: “electricity networks that can efficiently integrate the behaviour and actions of all users connected to it — generators, consumers and those that do both — in order to ensure an economically efficient, sustainable power system with low losses and high quality and security of supply and safety” [9].

Allowing for better control of the electrical grid is one of the main features that can be achieved with increasing *smartness* of the grid. An example where this control is needed is the requirement to keep the operation of the electrical grid stable. To achieve this many control measures must be taken, depending on the concrete situation. In addition to warranting an equilibrium between generation

and consumption of electricity at any point in time to the millisecond, it must also be ensured that all components, e.g. power lines, circuit breakers, switches and transformers operate within their specifications and are not overloaded. This is relevant with regards to renewable energies, as especially wind energy can only be efficiently harvested in specific geographic locations. In Germany, this manifests in the prevalence of wind energy in northern Germany including the north sea. Unfortunately, northern Germany is not as densely populated as southern Germany which therefore has higher energy demands, resulting in a big geographical distance between producers and consumers. Whilst it would be ideal to simply transport the energy to the regions that demand it, there is only a finite amount of transport capacity available as the physical infrastructure has a maximum capacity [10]. Increasing this capacity requires construction of new high-voltage transport networks, which is a long and expensive process.

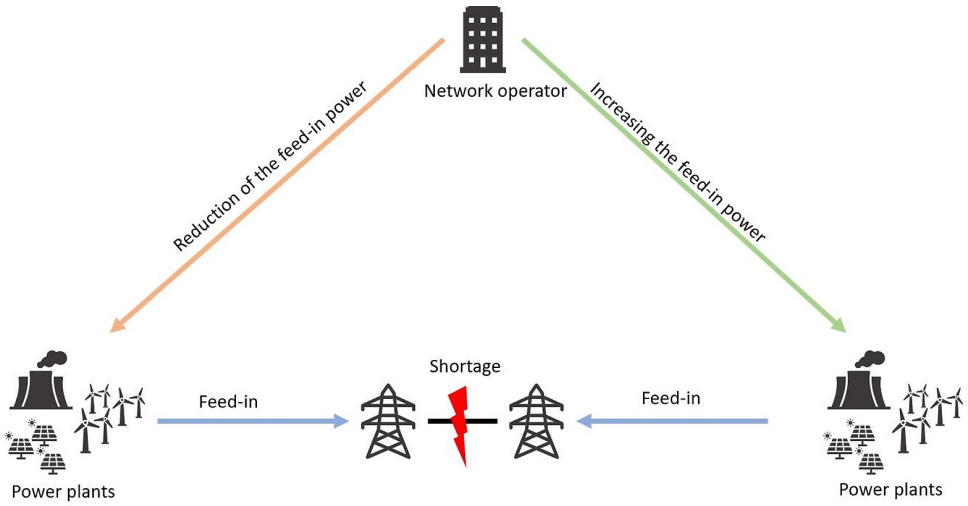

Fig. 2.1.: Abstract view on *Redispatch* process [@11].

One of the measures deployed to ensure that power lines are not overloaded is called *Redispatch* [@12, 13]. It is designed to ensure grid stability by allowing grid operators to remotely control energy producers in the network, even if they are owned by other parties. For example, it might be necessary to limit generation of wind energy in northern Germany while starting up coal-fired power plants in southern Germany. This ensures that power lines transporting the energy are not overloaded, as the production is shifted closer to the consumers requiring less transport capacity. All participants in the network that can produce more than 100 kW of power are legally required to participate in this process and allow for remote control by the grid operator. Especially with renewable energies, whose production

can only be controlled and forecasted to a certain degree, this process is carried out continuously throughout the day.

It is expected that this trend of increasing complexity of the electrical grid, mainly caused by the shift towards renewable energy, continues in the future. This necessitates innovation and research into more measures to control and monitor the grid to ensure stability despite the growing complexity.

2.2 SCADA Systems

Supervisory Control and Data Acquisition (SCADA) Systems are the de-facto standards for control and monitoring of critical infrastructures such as electrical grids. They have been around since the 80s and have evolved and rapidly developed with the changing industrial landscape and the increasing availability of computing capacity. Initially, they were used in isolated networks, e.g. inside control centers of (electrical) transmission system operators, where the attack surface was very limited thus ideally serving the need for highest security and intrusion prevention. While this might have sufficed in the past, nowadays an increasing amount of SCADA systems is connected to the public internet. This increases the attack surface manyfold, so security - which remains as the top priority for stable system operation - must be the primary focus for protecting communication channels and preventing attacks [14, 15, 16].

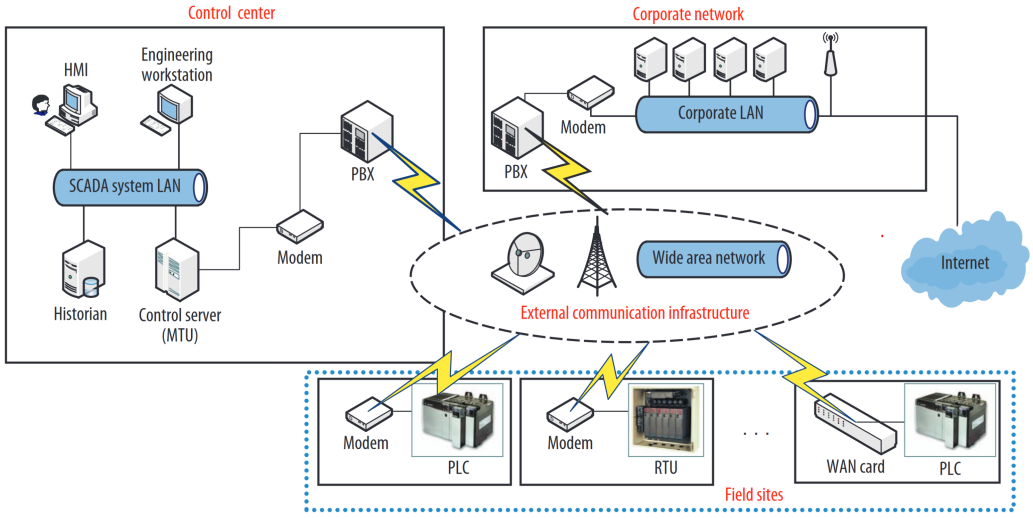

Fig. 2.2.: Simplified overview of a SCADA system [14]

Due to SCADA systems being used in critical infrastructures, they are prime targets for attacks by malicious actors, whether politically motivated, during warfare or motivated by financial gain through extortion [17, 18]. The latest examples for successful attacks include the attack on the Ukrainian power grid in 2015 and Operation Ghoul which infiltrated several systems in the middle east in 2016. The history of attacks goes back as far as 1982, and as it is expected that SCADA systems will stay around and evolve even further, research into the security of these systems is needed [19, 20, 21, 22].

2.3 Intrusion Detection Systems

Intrusion Detection Systems (*IDS*) are extensively used in modern infrastructures to provide continuous monitoring for protecting systems against threats. Different methodologies exist for facilitating the detection of attacks with the following three categories being the most prominent [23]:

- *Anomaly-based:* The *anomaly-based* approach is an iterative approach that tries to automatically classify normal system behaviour and subsequently detect any deviation from said behaviour, referred to as an *anomaly*, and mark it as malicious. As there is no strict definition of what *usual behaviour* is, any previously not observed behaviour may be classified as an anomaly. This approach is therefore susceptible to false positives, as distinguishing benign behaviour from malicious behaviour is difficult.
- *Signature-based:* The *signature-based* approach relies on identifying attacks and threats to the system based on known *signatures*. The signature is a fingerprint of the malicious behaviour, that uniquely identifies it. This can for example be done by scanning program files for specific byte sequences or detecting patterns that are commonly used by a malicious actor such as specific network requests. This approach is well suited for defending many systems against already known attacks, as signatures can easily be shared between systems. As a fingerprint of the threat is required for detection, this method cannot defend against novel attacks that have not been observed and classified yet. The most common application of this are antivirus programs, which compare system files and program behaviour against a curated list of known threats.
- *Specification-based:* Similar to the previous approach the *specification-based* approach relies on information about the operating environment. Contrary to *signatures* which fingerprint malicious behaviour, the *specification* fingerprints

the system behaviour. Any deviation from this behaviour can then be detected. As this requires a complete and very precise specification of the system, the applicability is limited to systems that can be specified in such a precise manner with reasonable effort.

As each methodology has individual strengths and weaknesses, IDS often combine methodologies. Additionally, novel approaches exist that do not clearly fit in any of these categories but form their own categories. Furthermore, each methodology can be implemented in different ways and with various different technologies, which results in a very wide landscape of tools existing.

The IDS developed by Chromik [3], which will be evaluated further in this thesis, fits into the category of *specification-based*, but uses a novel model-based approach to alleviate certain problems of this methodology. Chapter 3.3 elaborates further on the properties of this IDS and the different components used.

2.4 Resilient Communication

With a distributed system the need for communication between different system components naturally arises. In systems that are physically close to each other, such as systems running distributed inside a single datacenter, the components are usually linked to each other via very reliable connections. Due to the physical proximity, only a few middleware components (routers/switches) are needed and physical connections are not exposed to any hazardous environments.

Since smart grid components are distributed physically, as energy producers and consumers are geographically separated, this assumption of a reliable network does not hold true anymore. Any communication between components needs to traverse large distances and many traversed physical links are exposed to hazardous environments. Hazards include cable cuts, natural disasters, power outages, accidents or malicious attacks against the physical infrastructure [24]. As many of these hazards cannot be prevented, communication networks must be designed to handle such failures e.g. by implementing fault-tolerance, self-healing and redundancy. In the context of this thesis, the sum of all measures taken to prevent failures will be referred to as *resilience*.

The necessity of designing the system with this resilience in mind becomes apparent when recent incidents and attacks on critical infrastructure are examined. A good example is the attack on the train network of Germany on Saturday the 8th of October

2022. Unidentified attackers managed to halt all train traffic in the northern part of Germany for three hours by severing two glass-fibre connections that were used for the *Global System for Mobile Communications Railway* (GSM-R) [@25]. The system is used for communication and coordination in the train network and is mandatory for safe operation. Specifically the mobile switching centre (MSC) Hannover, which is responsible for handling all GSM-R phone connections in northern Germany, was physically cut off from the home location register (HLR). This register is required for any phone call as it provides the uplink to the actual telecommunication network. Whilst the HLR systems were deployed redundantly in physically separated locations, Berlin and Frankfurt respectively, no backups were in place for when both are inaccessible [@26]. The incident showed that this was not a sufficient measure and the system was not resilient enough for the condition it operates in.

While in an ideal world all possible failures would be prevented, this is often not possible in a real scenario. Usually only limited resources are available and any measure employed needs to be closely evaluated to ensure its cost-effectiveness. The maintainability of the system also needs to be ensured, i.e. the system should not become too complex.

3 Definition of the IDS

This chapter defines the current IDS and analyzes workflows and communication models that are needed for system operation. Based on this definition it evaluates different approaches for implementing them and answers the research question:

How should communication inside a hierarchically-organized distributed IDS be facilitated?

3.1 System Overview

The IDS which will further be evaluated was initially designed by Chromik et al. [2]. The core idea behind the IDS is similar to the concept of the *digital twin*, which became popular in recent years. The digital twin is based on the concept of "a system that couples physical entities to virtual counterparts" [27] by creating a digital mirror image of the physical world. One exemplary use case for this approach is to try out processes, e.g. optimizations, that cannot be easily tested in the physical world, in the virtual world first before actually deploying them. Another big use case is the monitoring of physical processes. By mapping all physical measurements into the virtual model, rules can be defined in a very intuitive way, e.g. defining the rule "Tank A can hold a maximum of $5m^3$", and can simply be checked against the virtual twin of the system. Defining the rules in such a way is very natural and therefore much easier than defining a rule that pertains to specific sensors and their measurements.

The IDS uses a digital twin of the monitored smart grid and checks this model against physical and safety properties that should always hold. These properties are referred to as *requirements*. The *physical* requirements check whether the underlying physical rules, e.g. the conservation of energy, hold. *Safety* requirements check that the grid components are not overloaded, i.e. that every power line operates within its specifications [28, 29].

The sole input to the IDS are measurements taken by physical sensors at various points in the electrical grid. Using these measurements, the system evaluates the

digital model and may output a *requirement violation* if any of the requirement checks fail. This requirement violation could then be used in various ways. Besides acting as an alert for human operators that may intervene, it could serve as an input to other automated systems that take appropriate control measures to restore a safe system state. Chromik showed that this can for example be used to intercept control commands sent out by a SCADA system, evaluate their effects in the virtual model, and then possibly drop the commands before they are executed if they would lead to an unsafe system state[3].

Contrary to a conventional IDS that is deployed as a central entity, e.g. inside a network-wide firewall, the IDS is distributed across the network. This is not a completely novel approach in computing, similar approaches are used in the field of *edge computing*. The main goal is to reduce bandwidth usage by physically moving computing resources close to the data producers. In the IDS this distribution additionally serves multiple other purposes. Firstly, the division of the grid into several independently monitored subgrids makes the computation of requirements much more manageable, as less data required for each individual computation. Secondly it makes the the system more resilient to attacks, as each distributed component acts independently and no central target for attacks exists [1].

3.2 System Components

Figure 3.1 shows an overview of the individual components that form the distributed IDS. Each components will be explained further in the following sections.

As illustrated in Figure 3.1, the distributed IDS consists of several components, most importantly several Local Monitors and Neighborhood Monitors. The exact purpose of these components and the flow of information between them will be explained in detail in the following sections.

Local Monitor

The *Local Monitor* (LM) is the core component introduced by Chromik et al. [2] for supervision of the grid. It is connected to a remote terminal unit (RTU) which is directly integrated into the grid and provides sensor data from sensors connected to the grid. Using this data the LM is then responsible for evaluating several requirements, all requirements that are checked are shown in Table 3.1. As each LM is connected to one RTU, only the grid that is accessible from the RTU can be

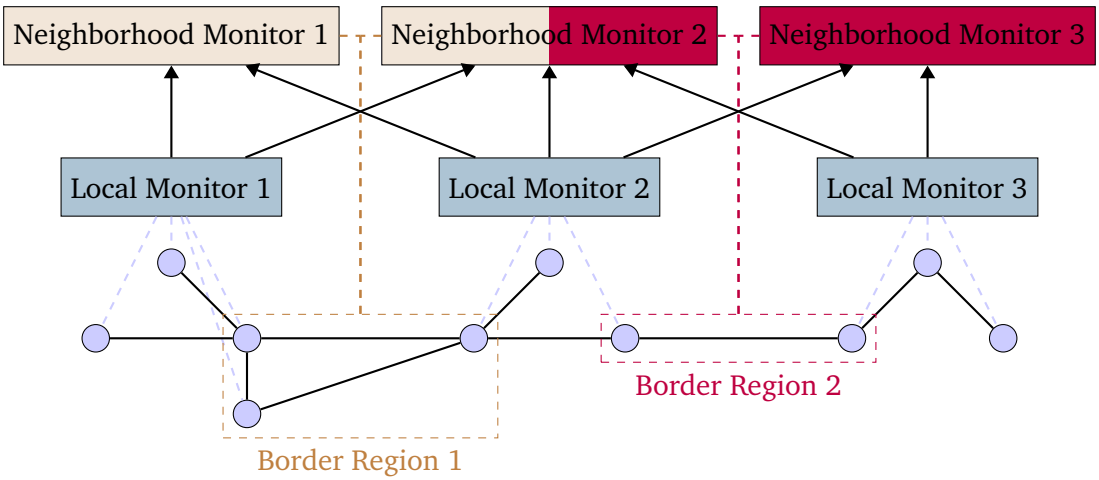

Fig. 3.1.: Schematic overview of IDS components. Dotted lines show which grid components a monitor is monitoring. Arrows show the flow of sensor data through the system.

monitored. If the grid deploys multiple RTUs, each RTU is paired with an LM who is responsible for the individual subgrid.

The IDS does not have strict requirements on how the connection to the RTU is actually implemented, but it is very beneficial to implement this connection through a physical link. This removes many sources for malfunctions and reduces the attack surface, as a minimal amount of intermediary components is used. If this is not possible and the RTU can only be remotely accessed via a network link, physically locating the LM close to the RTU is beneficial as it reduces bandwidth, the risk for any malfunctions of any intermediary networking components and the overall attack surface.

Neighborhood Monitor

The original IDS by Chromik et al. [2] was extended by Menzel [1] who introduced a second level of monitors that fill the gap between two subgrids that are monitored by two LMs. Specifically, they monitor *border regions* between two subgrids. As the name suggests this region includes all components that are at the border of a subgrid and shared with the adjacent subgrid, e.g. a power line that connects the different subgrids. Previously, as any LM only has access to the measurements in its own subgrid, not all requirements could be checked for such power lines since this requires data from all points along the power line.

This additional component is called Neighborhood Monitor (NM) and each LM is assigned one associated NM that monitors all border regions of the subgrid. Figure

3.1 shows an example setup with three subgrids that have two border regions. Each border region is independently monitored by two NMs, while the subgrids themselves are only monitored by one LM.

The NM is not directly connected to any RTU to access sensor data, but receives all data from the connected LMs. As it does not need access to any physical grid components, it is entirely virtual and could be deployed anywhere. Despite this, it may be beneficial to locate it physically close to the associated LM, as this reduces network latency and removes potential sources of problems as less network components are involved.

Global Monitor

The Global Monitor (GM) introduced by Menzel [1] handles safety requirements that require global knowledge to be evaluated. An example of such a requirement is a total power equation where the power consumed by all consumers must equal the power generated by all producers. In a real scenario, this computation must also include the electrical losses inside the system and account for inaccuracies of measurements, further increasing the complexity. As the amount of knowledge increases linearly with the size of the monitored grid, solving the calculation in reasonable time quickly becomes impractical. The second problem is data availability, since evaluating such a model would require up-to-date measurements at every point in the system. This also becomes impractical with a large system [1]. The GM, while formally specified and part of the system, will therefore not be evaluated further in this thesis and left out of scope in the prototypical implementation.

Supplementary Components

In addition to the monitors themselves, several additional components are needed to utilize the system in any meaningful manner. Detecting a requirement violation inside a monitor is only the first step, actually utilizing this information is equally important. For this, supplementary components are needed that form the *data consumers* of the system, the destinations the generated information flows to. These components can be categorized into several classes, each serving a different purpose. For the scope of this thesis the analysis of supplementary components is limited to the following categories, while components that perform additional tasks may be added in further expansions of the IDS.

ID	Description
1	For every bus: The sum of incoming currents equals the sum of outgoing currents
2	For every bus: All voltages measured at the bus are equal
3	For every powerline: If the line contains an open switch, no current must flow
4	For every powerline: Current and voltage are the same at all points
5a	For every power generator: Measured power value equals current * voltage
5b	For every power consumer: Measured power value equals (-1 * current * voltage)
6a	For every transformer: Outgoing voltage equals incoming voltage / transformation ratio
6b	For every transformer: Outgoing current equals incoming current * transformation ratio
7	For every powerline: Current does not exceed defined threshold
8	For every powerline: Voltage is inside defined reference threshold
9	For every fuse + relay: Component is functional
10	For every fuse + relay: Trigger current is not exceeded
11a	For every transformer: Outgoing reference voltage is inside threshold
11b	For every transformer: Outgoing measured voltage is inside reference threshold
12	For every power consumer: Consumer measures positive voltage
14a	For every meter: Current is inside threshold
14b	For every meter: Current is inside threshold
15a	For every interlock: Appropriate number of switches are connected
15b	For every interlock: Maximum capacity of powerlines matches required current

Tab. 3.1.: Requirements checked by the LM [1]. Grey marks physical properties of the system that should always be fulfilled. Blue marks safety properties that must hold for safe operation.

- **Logging:**
  Logging systems are tasked with periodically capturing all system parameters. They are usually not actively used during normal operation, but logging information is used as an analytical tool after unexpected circumstances happened. It can also be used for measuring and tuning system performance, e.g. by correlating measurements with generated alerts to validate that all alerts were warranted or analyzing the system to identify bottlenecks and other issues. The granularity of the logging can vary heavily from logging each individual sensor measurement to only logging fatal exceptions, mainly depending on the capabilities provided by the implementation.

- **Monitoring:**
  Monitoring systems are tasked with monitoring the system in near-real-time. They primarily collect information about the physical host system itself such as system load, disk usage or uptime and do not interact with the business logic of the system itself.

- **Output:**
  Output components receive requirement violations or other detected events in order to further process them in various ways. This can include visualizations, alerting systems that send alerts to operators or autonomous systems that may then issue control commands to stabilize the grid, e.g. by shutting off systems.

As these components have very specialized requirements, dedicated technology stacks should be used for implementing such components. Examples for open-source software that is commonly used for such use cases is the *Elastic Stack*[1] for log analysis, or *Apache Kafka*[2] for stream processing of dataflows to various different consumers. How these established software software-stacks can be integrated into the IDS depends on the capabilities of the specific software. It therefore needs to be assessed individually and is out of scope of this thesis.

3.3 System Model

The following chapters will define the formal IDS system. Using the formalized grid model by Menzel [1], the IDS can be interpreted as a tuple:

$$IDS = (\omega, \mathcal{L}, \mathcal{N}, \mathcal{G}, \mathcal{C})$$

[1] https://www.elastic.co/elastic-stack/
[2] https://kafka.apache.org/

with:
$\omega = \{o \mid o \epsilon \Omega^3\}$ set of electrical subgrids
$\mathcal{L}$ = set of Local Monitors (LM)
$\mathcal{N}$ = set of Neighborhood Monitors (NM)
$\mathcal{G}$ = Global Monitor (GM)
$\mathcal{C}$ = set of connections between components

While the structure of any specific IDS is dependent on the underlying grid itself, the following properties always hold:

- $|\mathcal{L}| = |\omega| \rightarrow$ the number of LMs equals the number of subgrids
- $|\mathcal{N}| = |\mathcal{L}| \rightarrow$ the number of NMs equals the number of LMs
- $|\mathcal{C}| >= |\mathcal{L}| \rightarrow$ the number of connections is at least as large as the number of LMs (as each LM needs to connect to at least one NM)

Based on this definition of the IDS, the data that needs to be exchanged between components can be identified and categorized in the following section.

3.4 Information Archetypes

As it is common in distributed systems, various points of data must be exchanged between the components on a regular basis in order to function properly. This data can be categorized into several classes, referred to as *archetypes*, where each has different requirements. Three generally applicable criteria to be fulfilled by each data exchange can be established: *authenticity*, *time criticality* and *minimization*.

Authenticity summarizes the security requirements of the data. While data should always be transmitted over secure channels, additional measures can and may need to be employed to prevent attacks not covered by a secure channel alone. Especially when assuring data integrity it is necessary to augment the original data with additional metadata, such as message authentication codes (MAC). Validating the origin of the data by checking whether it is trustworthy might also be required. While fully securing every message and ensuring full integrity and authenticity is technically possible, the associated cost, such as the time required for performing the validation, needs to be critically evaluated. Especially when querying of external

[3] Ω = electrical grid. For the full definition see [1]

services is required, e.g. verifying certificate information, the time required quickly exceeds reasonable bounds.

Time criticality of the data refers to the maximum delay before messages are received. Especially when data is used as a base for decisions, e.g. automatically triggering workflows in autonomous systems, near-real time transmission is important and should be prioritized. This may come at the cost of more messages being transmitted, as messages cannot be bundled together, or reduced security, as verification steps might need to be skipped if they cannot be performed fast enough.

Minimization of data refers to the need for efficient encoding of the information to be transmitted. Especially if the transfer uses a network with limited resources such as a very low bandwidth, efficient encoding is required as to not overload the network. There is always a tradeoff between the expressiveness and flexibility, e.g. the capability to include additional information and the size of the message. Section 3.5 elaborates further on the benefits and drawbacks of different encoding schemes.

The three archetypes that are present in the IDS can be categorized as follows:

- **Configuration data**:
  Configuration data includes all data that is used for managing the system. This includes certificates, addresses of components, configuration for parametrized requirements (threshold values) and information about the monitored subgrid. This data does not change often and therefore only needs to be transmitted infrequently. As configuration data has severe impact on the system, the authenticity of the data needs to be assured.

- **Sensor data**:
  Sensor data describes all data that is used as an input to the monitoring system during operation. This data is mostly made up of sensor measurements that are provided by an RTU. Since these measurements serve as the sole input to the entire IDS, transmission is very time-critical. Any delay in transmission will subsequently delay system operation and therefore the detection of requirement violations. Additionally, this data is assumed to always be present and needs to be transferred very frequently. Therefore, minimization of this data, reducing it to the minimum size possible e.g. by filtering out data not needed by the other party, is required to minimize the amount of data that needs to be transferred via the network.

- **Event data**:
  Event data is data that is actively produced by the system. This includes

monitoring data about the system state, requirement violations and other events detected by the system such as detected abnormalities or attacks. As the IDS is supposed to monitor the network in real-time and this data is the main output of the system, it is time-critical. Minimization of the data is less important, because these events are presumed to be a deviation from normal operations and only generated infrequently.

Table 3.2 summarizes the requirements by each data archetype.

	authentic	time-critical	minimized
Configuration Data	yes	no	no
Sensor Data	partial	yes	yes
Event Data	yes	yes	no

Tab. 3.2.: Requirements of different data archetypes.

3.5 Communication Models

Since the IDS is a reactive and data-driven system, where all outputs are directly caused by a given input, no action can be taken without first receiving input data. This inherently creates a problem, because data is generated at a single point but required as input at multiple points in the system. Specifically, the LM communicating with an RTU is the sole producer, and all neighboring NMs require this data. It therefore needs to be distributed inside the system in a secure manner, while preserving accuracy and minimizing delay [1, Section 3.3]. Communication between components for the purpose of data sharing is therefore a central task that needs to be handled by the system. To perform this communication, defining a *communication strategy* is crucial. The following section will elaborate further on the different strategies that could be used.

Communication strategy: Push vs. Pull

One of the core requirements is the time-criticality of communication. This is mostly influenced by the *communication strategy*, which governs the way how data is transferred. Communication strategies can generally be classified into *push-based*

and *pull-based* strategies. Both have strengths and weaknesses and are suited for different kinds of data and different environments.

In *pull-based* strategies, the communication is initiated by the party that currently does not have the desired information. This is done by sending a request for data to the other party, which then responds with the data if it is available. This process can be described as *pulling* the data, as the uninformed party performs the active part, and the informed party does not perform any action other than responding to requests. This mode is very common, the most prominent example is surfing the internet - when instructed to access a website the browser actively requests the relevant data from a remote system.

This strategy is well suited for when the communicating parties do not know each other beforehand, because they require any communication other than the request for data itself. It is also well suited when always-present data is requested, as the request can always be fulfilled immediately. Conversely, the efficiency of this approach decreases if the data is not available when the request is received, as in this case the request does not return the queried information and the requesting party needs to repeat the request at a later time. Especially if the data is only available irregularly, e.g. because it is generated by an external system or generated in irregular intervals, many of the requests do not result in a response containing the desired information, but still require resources and bandwidth to handle.

In a *push-based* strategy this flow is reversed. Data is not requested by an uninformed party, but instead proactively pushed to it by the party providing the data. This is done by first establishing a permanent communication channel that is subsequently used for transferring the data. This mode is well suited when the two parties know each other and the data to be transferred is either frequently changing or not guaranteed to be available, e.g. because it depends on a process that takes a variable amount of time. In such cases this approach is much more efficient, as it avoids unnecessary requests. The *push-based* system is especially well suited when data transfer is triggered by an event on the side of the providing party, as it minimizes any delay.

Evaluation for the IDS

Table 3.2 shows the requirements regarding the time-criticality of the different archetypes present in the IDS. Transmission of configuration data is neither time-critical nor dependent on any dynamic processes and is therefore agnostic to the mode that is used for transmission. Sensor and event data on the other hand is

especially time-critical, as it forms the input and output of the whole IDS system. Sensor data is always generated by the LMs and flows to the NMs, while event data is generated at the monitors and flows to consumers. The generation of sensor data is dependent on external systems and only the LM requesting the data from the RTU knows when new data is available. Similarly event data, albeit originating inside the system itself, is only generated sporadically with variable time intervals. These factors heavily favor transmission via a push-based model, as only the data-providing side knows when data is available that warrants a transfer.

Data Encoding

Whilst the communication strategy elaborated on earlier defines how data is transferred, the data encoding governs what is actually transferred. While the goal of communication is to transfer some *information* between two parties, the network layers work on a simple stream of bytes without any connotation as to what information these bytes hold. Transferring information between two parties therefore needs encoding and serializing to bytes by the providing party, and then deserialization and decoding by the receiver. To successfully do this, both parties need to agree on an interpretation of the raw bytes being transferred. Different formats exist for facilitating this encoding and decoding.

```
{
  "id": "ea4ab234-254b-498e-b0e4-848903666df7",
  "value": 123.5,
  "parameters": [
    {
      "type": "depth",
      "value": 13,
      "unit" : "metres"
    },
    {
      "type": "accuracy",
      "value": 23,
      "unit": "not defined"
    }
  ]
}
```

Fig. 3.2.: Example data encoded in JSON format. Using control characters such as double quotes, curly brackets and square brackets, the schema of the data is directly embedded. Additionally, key-value pairs are used to organize the individual values.

The formats can mainly be distinguished into *text-based* and *binary-based* formats. Among the *text-based* formats, the most prominent and commonly used formats

are JSON and XML. The main feature is that in such formats the payload, the data actually transferred between two parties, contains both the information itself as well as its structure. This structure is also referred to as *schema*. Embedding it allows for flexibility because as the schema can be different in each data package. No previous agreement between the two communicating parties is needed. Especially for use cases where the transferred data is very dynamic, e.g. because it depends on user input, or in cases where the two communicating parties do not know each other, these properties are very helpful. The drawback is, that much more data needs to be transferred, as the schema is sent with every transfer even if it never changes. The specific overhead this incurs depends on the specific schema used, it is generally preferred to have small schemas with large sections of data.

An alternative to text-based formats are *binary-based* formats. The most commonly used formats among them are *Protobuf* and *Avro*. Additionally various different specialized formats exist, such as IEC 60870-5-104[4], that is commonly used for SCADA/RTU communication in power grids, or framework-specific formats, such as the *OPC-UA Binary Format*[5] that is used in the OPC-UA framework. The main difference to text-based formats is the strict separation of schema from data. Instead of sending the schema with each transfer, both parties agree on a schema beforehand and then only send the raw information. Figure 3.3 shows an examplary schema with encoded data. The advantage of this approach is that it allows for very efficient encoding, as no superfluous data will be transmitted. The drawback is that, due to the static nature of predefined schemas, changing them, i.e. adding or removing properties during runtime, is not easily possible. Additionally, agreeing on a schema beforehand requires either knowing the partner or communicating with them before actually transferring data. This exchange only needs to happen once and the resulting overhead amortizes with the subsequent transfers. The specific overhead therefore is dependent on the complexity of the schema and the actual number of transfers that utilize the schema.

Evaluation for the IDS

As pointed out in section 3.4, the IDS has three different archetypes of data which are all internal to the IDS and exchanged between components that know each other. These circumstances favor the use of a binary-based format. For event data it could be argued that flexible data should be included, e.g. information about the specific values that triggered a requirement violation if they constitute a heavy

[4] https://www.ipcomm.de/protocol/IEC104/en/sheet.html
[5] OPC-UA file format description

```
Schema:
syntax = "proto3";

message Request {
  message Parameter {
    string type = 1;
    uint32 value = 2;
    string unit = 3;
  }

  string id = 1;
  float value = 2;
  repeated Parameter parameters = 3;
}


Data (base64 encoded for readability as it contains unprintable characters):
CiRlYTRhYjIzNC0yNTRiLTQ5OGUtYjBlNC04NDg5MDM2NjZkZjcVAAD3QhoRCgVkZXB0
     aBANGgZtZXRyZXMaGQoIYWNjdXJhY3kQFxoLbm90IGRlZmluZWQ=
```

Fig. 3.3.: Example data encoded in Protobuf format. The schema of the data is not coupled with the data but seperate. The data itself only contains values, decoding it requires knowledge of the schema.

deviation from the expected values. As currently none of the checked properties warrants such a handling, it will be omitted for simplicity. If at a later point this is required, embedding a text-based detailed description of the violations inside the binary format is possible.

4 Analysis of the existing IDS Prototype

The following chapter analyzes the existing prototype of the IDS. It mainly focuses on the differences between the actual implementation and the theoretical model as defined in the previous chapter. The prototype was developed as part of a project seminar at University of Münster and is based on the work by Menzel [1].

4.1 System Design

Since the IDS is designed as a distributed system, the existing prototype is also organized as a multi-component infrastructure. Figure 4.1 shows all components currently used by the prototype [cf. 5, 30].

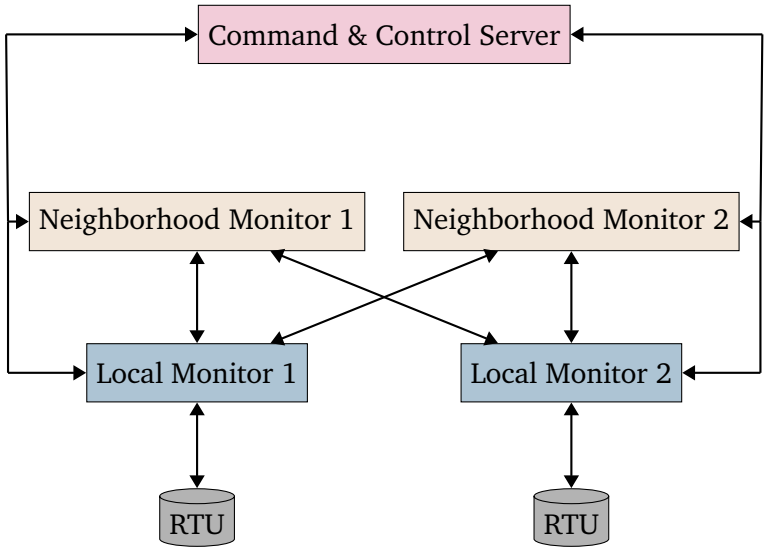

Fig. 4.1.: Schematic overview of components used by the current IDS prototype as used by [5, 30]

.

The monitors are implemented close to the conceptual model (cf. section 3.3). They communicate directly via dedicated connections. There is one additional component present, the Cmmand & Control server (C2). It is connected to all IDS components

and provides supplementary services (cf. section 3.2) for the IDS. The individual components are analyzed further in the following sections.

Local Monitor & Neighborhood Monitor

The monitors are implemented very close to the conceptual models as defined in Section 3.2. The connection to the grid is realized by using a pull-based approach to read out sensor values from the RTU based on a timer. The *Modbus 3* protocol is used for communication with the RTU Simulators used in the testbed, as this is currently the only supported protocol. To initialize and run an LM, various configuration parameters need to be supplied. These are shown in Table 4.1.

Key	Description
IDS_C2_ADDRESS	OPC-UA address of the C2 server
IDS_OPC_DOMAIN	OPC-UA domain
IDS_LM_OPC_ADDRESS	OPC-UA adress of this component
IDS_NM_CERT	Certificate of associated NM
IDS_C2_CERT	Certificate of C2 server
IDS_CERT	Certificate of LM (own certificate)
IDS_PRIVATE_KEY	Private key for certificate
IDS_PRIVATE_KEY_PASSWORD	Password for private key
IDS_RTU_MODBUS_HOST	Hostname of RTU
IDS_RTU_MODBUS_PORT	Port of RTU
IDS_RTU_CONFIG_FILE	JSON configuration of RTU

Tab. 4.1.: Configuration parameters of LM.

Command & Control Server

The Command & Control (C2) server is a component serving as a central management service for the IDS. It is responsible for management and organization of components, capturing alert data, providing data to the visualization and monitoring the system as a whole. It thereby bundles all supplementary components (cf. section

3.2) into a monolithic application and constitutes the sole data consumer for all generated information in the entire system.

Communication

The prototype uses the *OPC-UA* framework for facilitating the communication between components. Components connect directly with each other without the presence of a server, forming a peer-to-peer network. Figure 4.2 shows all the different messages that are exchanged and how the exchange is triggered. Three different methodologies are used for facilitating data exchange: traditional pull-based reading, push-based exchange via events and remote procedure calls (RPC). These are triggered either based on a timer or by external events. It can be noted that most of the communication takes place between the C2 server and the monitors, while the communication between monitors themselves is limited to exchanging sensor data.

4.2 Problems & Weaknesses

The existing system has several weaknesses that are elaborated on in this section. The two main factors, the reliance on centralized components and the assumptions the IDS makes about the operating environment, will be analyzed further in this section.

Centralized components

The biggest weakness with regards to resilience of the system is the centralization introduced by the C2 server. While the monitoring itself is performed in a distributed fashion, all management of the system as well as output handling is done on a centralized server. This includes the identification of border regions, assigning of monitors to regions, collecting detected events and collecting monitoring data. This inherently introduces a single point of failure as an active connection to this single server is required at all times. Any problems with this component or the infrastructure around it will have detrimental effects on the system, up to the point of a total system failure. While various redundancy strategies could be developed to guarantee high availability, they all heavily increase the complexity of the system and rather serve as a temporary workaround than as a permanent solution.

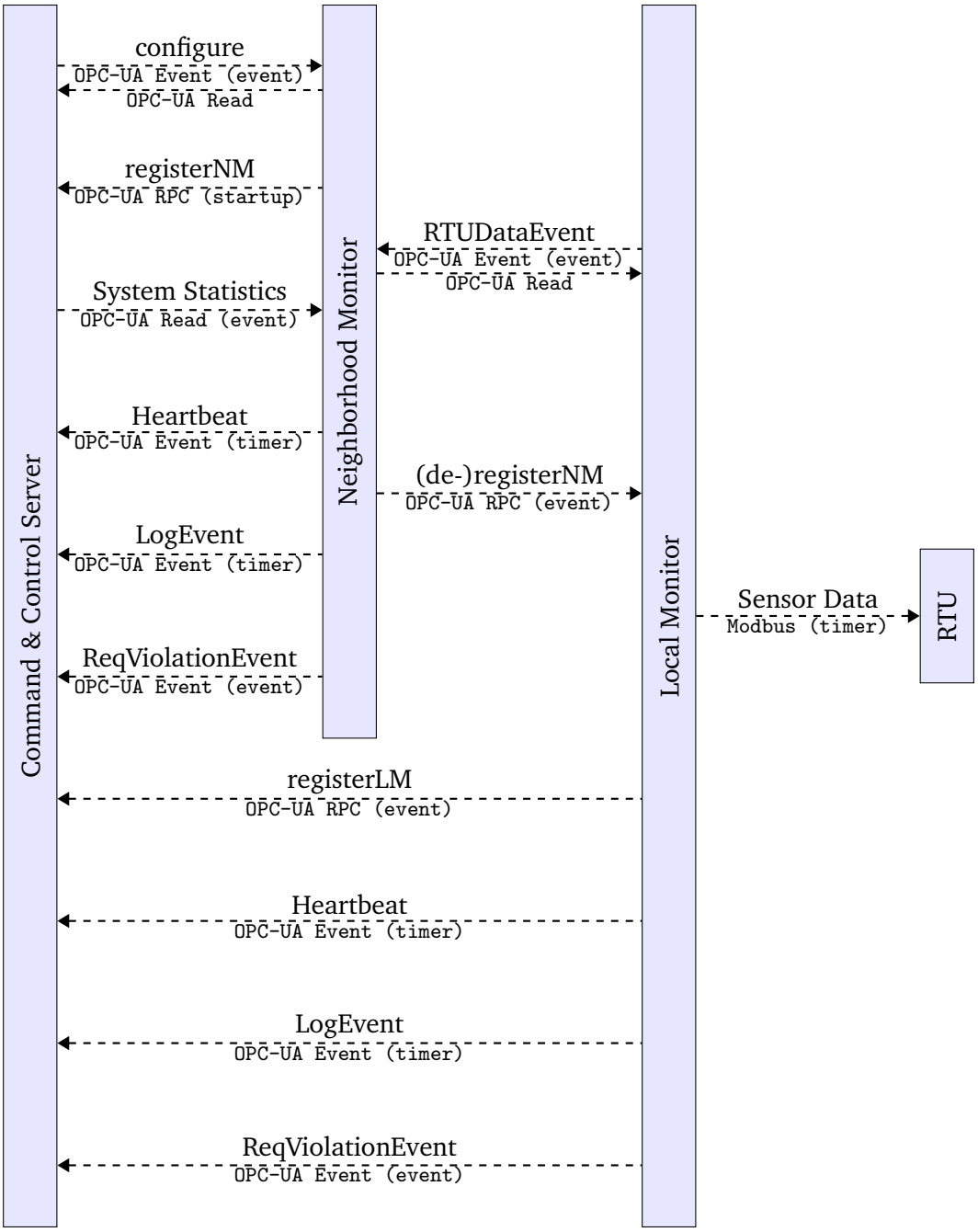

Fig. 4.2.: Overview of all communication workflows currently used in the existing prototype. Arrow directions indicate the initiator of the request.

The existence of a centralized component also needs to be critically examined when the scalability of the system is evaluated. With an increasing amount of distributed components, centralized components need to scale accordingly. This inherently forms a bottleneck in the system, as the vertical scalability of any system is limited due to technical constraints. As shown in Section 2.4, reliance on a central component also introduces a very lucrative target for potential attacks.

Communication channels

As described earlier the OPC-UA Framework is used for facilitating all communication. The suitability of the framework for the task has been elaborated on by Deuschle et al. [30]. But as Figure 4.2 illustrates, there is no general policy on which communication methodology is used for which data. A multitude of different workflows are used, which results in a very complex system. This leads to suboptimal results, e.g. when new data is available the NM receives an event from the LM and then subsequently needs to manually pull the data. This delays data receival as additional requests need to be made, without providing any benefits.

Assumptions

The system makes several assumptions regarding the operating circumstances of the IDS that are unlikely to hold up in a real-world scenario:

- **Temporal Synchronization**
  As the components of the IDS are designed to run distributed on physical hardware, there is no common source for real-time timestamps. With the presence of clock drift this can cause desynchronization between clocks and cause errors. In the context of the IDS, synchronization between components is especially critical as the hierarchical structure of the IDS mandates that measurements from different physical systems are compared during evaluation of requirements. Großhanten [5] has already observed such effects. During the test scenarios, desynchronization between components occured. Due to random delays in data transmission, amplified by the lack of synchronization, sensor values from different timestamps were compared. This caused false positive requirement violations. This is a common issue for distributed systems and different strategies exist to mitigate the impact on actual systems, e.g. Google uses a clock that includes uncertainty in accuracy when synchronizing

datacenters. Another mitigation is the usage of logical clocks such as Lamport clocks that are independent of real time [31, 6].

- **Static & Predefined Topology**
  Currently the IDS is based on pre-calculated configuration files, which requires full knowledge of the grid and all IDS components prior to starting the IDS. As already identified by Menzel: "[The system assumes] that the static data was safely exchanged before the start of the monitoring system and during operation only the dynamic data, the data generated from measurements, needs to be communicated" [1, Section 3.3]. While this may be sufficient for simple proof of concept scenarios, it idealizes the real environment. Changes in the topology, e.g. removal or addition of components when the network is restructured, are expected and must be handled without requiring a complete shutdown of the IDS.

Certificate handling

All communication between components is encrypted using common public key cryptography. To do this the current implementation requires the certificates of all communication partners to be known before communicating. For example, every LM requires the certificate of the C2 server as well as the certificates of all relevant NMs (see Table 4.1). While this might not be a problem for the limited scope the IDS has been tested on, this approach does not work well when expanding the IDS to more monitors.

Furthermore, this approach does not work when the network topology is not static. As certificates of all connected components need to be known when deploying a monitor, supplementary components that are added after a component has been deployed can never be connected. This is a conceptual problem and there is no remediation for this using the current implementation.

The implementation currently accepts all certificates without checking their expiry date or validating the authority signing the certificates. Additionally, only one NM certificate can be configured, forcing all NMs to use the same certificate. Both of these problems are not conceptual and could be remediated in the existing prototype.

5 Development of a new communication system

To remedy the weaknesses of the currently used communication system and enable further enhancements, the following chapters incrementally develop a new communication system for the IDS. The main focus of this new system will be its robustness against malfunctions and attacks while preserving all functionality the current communication system offers. The new system is designed to be expandable, allowing for future development of additional features, e.g. active defense against threats or specialized monitoring components.

5.1 Assumptions

Several assumptions need to be made to limit the scope for this thesis, mostly concerning the operating environment of the IDS. A detailed analysis checking whether all assumptions hold true in real-world environments is out of scope for this thesis, but it can reasonably be assumed that they do.

The first assumption is that each element in the network has a universally unique identifier (UUID) as specified by RFC 4122 [32]. This extends to components not actively managed by the system itself such as meters, substations or power lines. As identifiers are the base used for identification and configuration of components, duplicate identifiers would lead to non-determinism and may create invalid system configurations. This must also hold for all smart grid elements, as otherwise two IDS components could never decide if they are looking at the same component. Whether this identifier is predefined based on physical properties, e.g. by using a serial number of a device, or is created via an algorithm is not relevant. If programs are used to automatically generate the identifier it is necessary to assure that this is deterministic and the identifier stays the same for a given element.

The second assumption is that all deployed components of the IDS are networked and can reach to all relevant other components via this network. Since the IDS is designed as a distributed system to be deployed on physically separate hosts, having

network access is mandatory to allow for any coordination between components. The system is agnostic to how networking is implemented, it could be done via dedicated network cables, broadband, other wireless network technologies, e.g. LoRaWan[1] or other means as long as bidirectional point-to-point data transfer is supported.

The last two assumptions concern the implementation of cryptography in the IDS. To perform authorization and authentication various cryptographic methods are necessary. It is therefore assumed that each component has capabilities to store and handle required data, e.g. secure storage of private keys. The availability of dedicated cryptography modules, e.g. the capability to use full disk encryption, the existence of a trusted platform module (TPM) or secure sources of entropy for random number generation, is dependant on the specific hardware used [5, chapter 4.3.2]. Furthermore, the usage of certificates is dependent on the existence of a trusted public key infrastructure (PKI) which is also not directly part of the IDS and is assumed to exist outside of the system.

5.2 Workflows

The primary task of the IDS is the detection of requirement violations. Being designed as a self-organized system without a central management component, the IDS needs various workflows to facilitate the management of the network. The most basic example is the construction of the network itself, specifically finding other components and establishing connections to them.

While various workflows are possible, for a first step all workflows that are required for basic system establishment and operation, referred to as *core* workflows, are specified. These can later be expanded on, e.g. by workflows that help to secure the network or provide additional remote monitoring and controlling capabilities. The core workflows that are required are primarily the addition and removal of components.

5.3 Design Concepts

Based on the previous analysis (c.f. chapter 4.2) multiple core concepts shall be respected when developing and implementing a new communication system:

[1] https://lora-alliance.org/

Identifier	Description
AddLM	Adds a new LM
RemoveLM	Remove a connected LM
AddNM	Adds a new NM
RemoveNM	Removes a connected NM
AddSC	Adds a new supplementary component
RemoveSC	Removes a connected supplementary component

Tab. 5.1.: Workflows required for establishing the IDS network.

- **Fully distributed:**
  The system should not use centralized components. If certain components are mandatory, e.g. for providing configuration data, they must be distributed as well to assure that failure of a singular component does not pose a problem to the overall system.

- **Robust:**
  The system should anticipate component faults, network errors, commission and decommission of components and all other workflows that are expected in a real-world scenario. As described in section 5.2 such events are to be expected and may not hinder the system more than required.

- **Dynamic topology:**
  The system should not require a predefined static component topology but instead allow for dynamic reconfiguration of the system. This includes addition or removal of monitors and output components as well as redefinition of border regions between monitors.

- **Modular framework:**
  A multitude of components could be used in the system, e.g. various data consumers that implement analysis and logging of events. As they all require the same functionality to interface with the IDS, it should be provided in reusable libraries. This heavily simplifies the development of additional components, as no in-depth knowledge about the inner workings of the communication network is required for the development of these new components.

Since the system needs to work without the presence of a central management entity and all communication is done in a peer-to-peer paradigm, any coordination

between individual components, e.g. finding local monitors providing data, needs to be initiated by the components themselves. This poses multiple challenges that will be elaborated on further in this section.

The first problem is that connecting to other components requires knowledge of the address of the communication partner. With centralized structures this is not a problem as the addresses are fixed and known beforehand. Due to the decentralized and dynamic structure of the IDS there is no static list of addresses. Connecting to a component therefore first requires a discovery step to determine the network address of the communication partner.

In the context of the IDS a second problem arises that amplifies the first one. As monitors do not even know the name of their communication partner, but only that they want to communicate with their direct neighbors. These are only identified by the presence of shared power lines (cf. section 3.2). Since only the monitors themselves have knowledge about their individual subgrid and the power lines within, evaluating whether two monitors are neighbors can only be done by the individual monitors themselves.

In order to solve these two challenges two approaches will be taken: To allow for identifying communication partners, all changes to the system, e.g. registration or removal of components, will be encapsulated inside a dedicated structure referred to as a *motion*. Section 5.5 will elaborate on their structure and properties. These motions will then be distributed to all network components using a *rumor-spreading algorithm* which will be described in more detail in Section 5.4. This implements the aforementioned discovery step, as it lets each IDS component individually decide whether it needs to act to any system change.

5.4 Rumor-spreading Algorithms

While many ways to broadcast a message through a network exist, this thesis focuses on *rumor-spreading* (also referred to as *gossiping*) algorithms. These algorithms are well suited for the use case of the IDS and have been used for similar problems in smart grids before. The core idea is that a piece of information, referred to as *message*, propagates incrementally through the whole network similar to how gossip spreads in a human social group (e.g. a neighborhood or social circle). All participants in the network exchange messages with their respective neighbors, informing each other about any new messages they received since the last exchange. The communication is triggered by the participants themselves and is not synchronized, resulting in rather

sporadic communication. This closely mimics the human behaviour of gossiping, which is also not centrally organized but happens naturally during interaction. As the communication happens asynchronously and over many different individual connections, any message is duplicated manyfold within the network. This provides a very robust and fault-tolerant method to share information with the whole network because failed connections, loss of messages, or other faults can often be tolerated [33]. This fault-tolerance cannot be established without consequences, in this case a tradeoff between redundancy and an increased volume of transmitted data. Due to the fact that there is no clear and direct routing for any message, a single participant receives every message multiple times via different connections, increasing the overall volume of transmitted messages.

This rumor-spreading paradigm has been intensely researched since the 80s for various use cases [34, 35]. While the early developments mostly focused on database replication [36], later research showed many more use cases such as discovery of network components or distributed computations [37, 38]. The rise of the Internet of Things, which provides large networks of connected sensors, is another more recent use case that shows the advantages of this paradigm [33].

In the context of smart grids, such rumor-spreading algorithms can also be used effectively, as shown by Schindler [39], who implemented a distributed energy market, or Koukoula [40], who used such an algorithm to ensure the safety of a smart grid by detecting and automatically remediating overvoltage conditions in the grid without the presence of a central control unit. Croce [41] also showed the feasibility of using a rumor-spreading algorithm approach for managing a microgrid.

Several different approaches exist for implementing this paradigm of rumor-spreading. The main goal is to guarantee that all nodes in the network receive the message as fast as possible, while minimizing the amount of messages sent and communication rounds needed. Various methodologies can be used, e.g. sending to all neighbors, randomizing the neighbors, using additional information like geographic location and many more. The main difference between these approaches is the selection of neighbors to communicate with in each step. *Address-aware* neighbor selection calculates a perfect message exchange sequence, e.g. by removing all redundant exchanges, based on knowledge about the network topology. This kind of selection is very difficult to deploy in a dynamic network topology with unreliable connections and is therefore not suited for use in the IDS. Therefore, in the following only *address-oblivious* algorithms, selection algorithms that do not rely on knowledge about the network topology and the addresses of communication partners, will be evaluated.

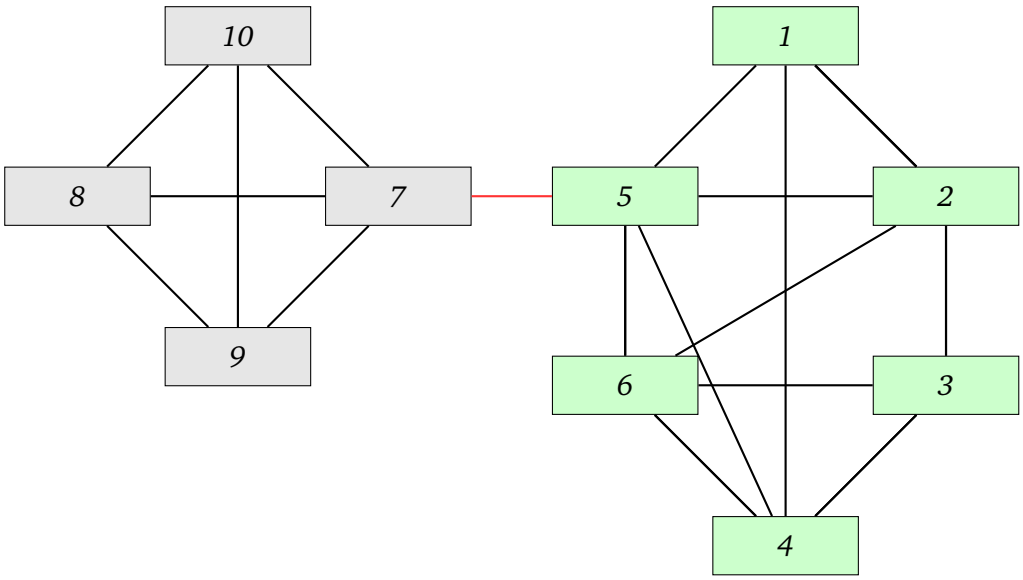

Fig. 5.1.: Network with bottleneck (marked red). Nodes that already received the message are marked green.

One of the most promising approaches is the class of *fully random* rumor-spreading and *quasirandom* rumor-spreading algorithms. In these models the neighbor to communicate with is chosen randomly for each iteration. As there is no predefined path and messages are transmitted over many different connections these approaches have shown to be very resilient. Because there is no defined order of communication partners no bias towards or against a specific network topology exists [42, 43].

The drawback of this *fully random* selection is, an isolated with only a few connections to other nodes, is unlikely to be chosen as a communication partner, while nodes that are highly connected are chosen very frequently. This can have detrimental effects when the network contains bottlenecks, like as locations where a single node with very few connections acts as a connecting element between heavily interconnected networks. Figure 5.1 illustrates this problem. Due to the many connections of **node 5** into the right subgraph it is very likely for a node from this group to be chosen as the next communication partner despite this node already being informed about the message. With an increasing amount of connections to already informed nodes it becomes more unlikely to spread the message to uninformed nodes such as **node 7** and subsequently **nodes 8**, **9** and **10** despite them being well interconnected. Although this approach is guaranteed to converge given infinite time, as probability dictates that every element is chosen eventually, it is certainly not ideal for this use case.

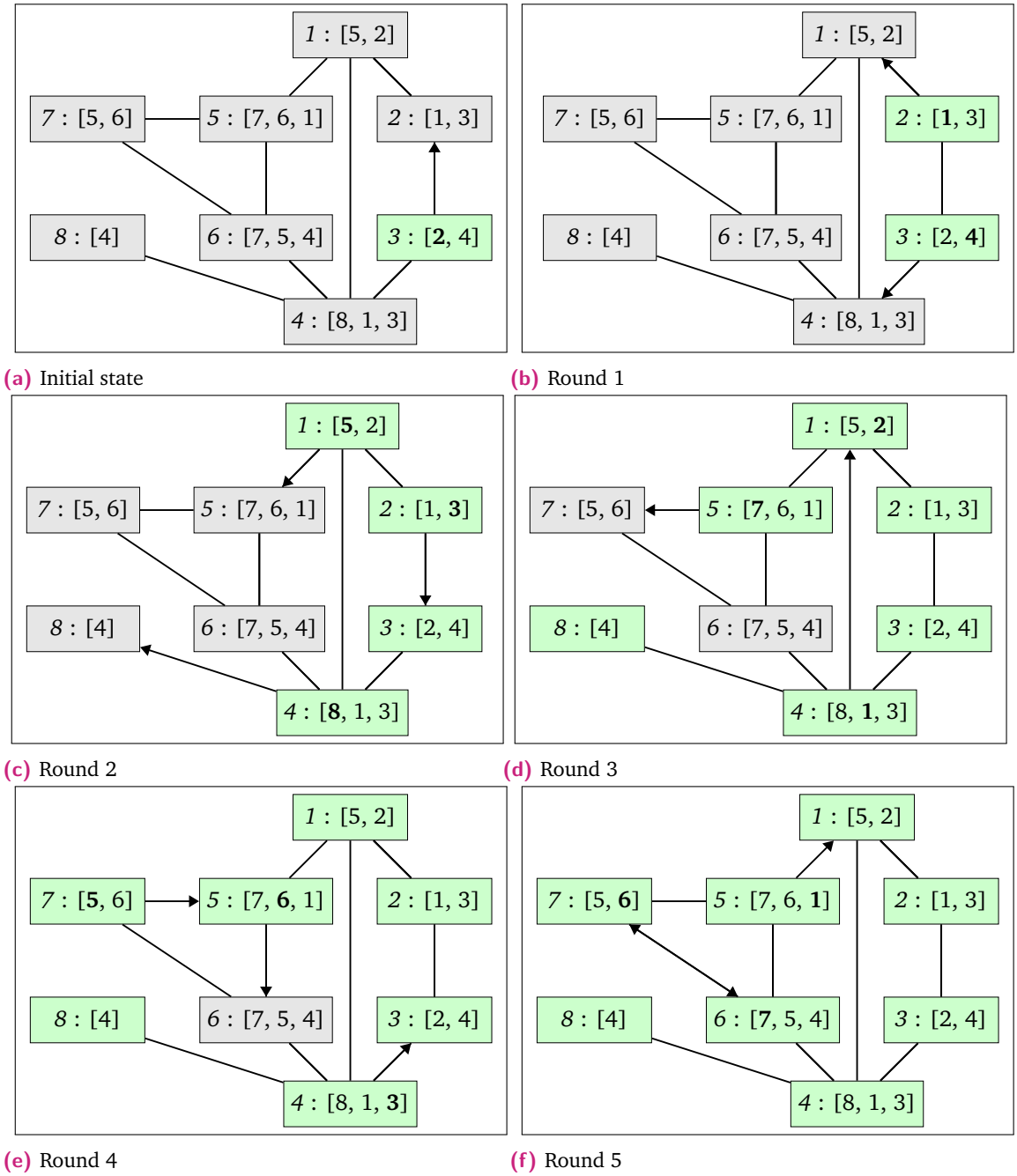

Fig. 5.2.: Example of quasirandom rumor-spreading algorithm. Nodes that are marked in green already received the message. Each node has a randomized list of all neighbours it sequentially gossips with.

Quasirandom algorithms can perform better in such situations [44]. They differ from fully-random algorithms in the selection of the next communication partner. Instead of choosing a random partner in each step, where every neighbor has the same likelihood to be selected, the order of communication partners is randomized once and then iterated on sequentially. Given *n* neighbors it is thereby guaranteed that after *n* steps each neighbor has been communicated with. Doerr [45] showed that this approach significantly improves the upper bounds of the algorithm compared to the fully random model.

Figure 5.2 shows an example propagation of a message through the network using a *quasirandom* approach. Each component holds a randomized list of its neighbors. This list is iterated upon in each step to choose a neighbor to gossip with. The message is then propagated to the chosen neighbor. The arrows show that the same message is received multiple times by a component, each time using a different connection. This provides redundancy, the loss of a connection does not impact the system at all. In an ideal scenario where every exchange succeeds, all components are informed after five steps. All steps thereafter are still performed, providing redundancy in case an exchange failed. After seven steps the message is fully handled by all components.

For many graphs, e.g. Hypercubes or Ramanujan graphs, an upper bound for the computational complexity of $\Theta(log(n))$ has been proven [45]. As the topology of the IDS network is not predefined, no tight bound for the complexity of any algorithm can be given, only the rough bounds of $\triangle diam(G)$ and $2n - 3$ that hold for any graph G containing n nodes can be guaranteed [45, Section 1.4]. But it can be noted that the algorithm performs better the lower the diameter of the graph is and the more edges it includes.

Using this approach, all components in the network eventually receive every message and can react to it. Duplicate receival of messages does not pose a problem, as these duplicates can easily be filtered out by the components themselves. As the algorithm requires multiple rounds before the information is fully propagated through the whole network, it implements a model of *eventual consistency*.

5.5 Motions

The rumor-spreading algorithms are oblivious to the actual data transmitted, as they only govern how data is transmitted through the system. The transmitted message should include all information that is necessary to allow the receiving components to

react to the message. To guarantee that each message has all necessary information, a message container called *motion* is defined.

All workflows of the system are encapsulated within such a dedicated motion. They can be summarized as a well defined request by a component to modify the system state. Motions are can be created by any system component (referred to as *caller*) and may address any number of other components (referred to as *callees*). A motion encloses all information that is needed by any other component to decide whether to take action upon receiving it. This encapsulation eliminates the need for any further communication and streamlines processes, as no additional requests for clarification need to be made. After creating and publishing a motion the component then takes a passive role and waits for action by other components.

To ensure integrity and authenticity, each motion is signed by the caller using a cryptographic signature. This guarantees integrity and authenticity, provided the used certificates are not compromised. Using this strict system, which forces all workflows to be encapsulated inside motions, assures that all workflows adhere to the same security requirements. Furthermore, this system allows for easy future extension, e.g. the addition of new workflows, without the need to modify the implementation of the communication.

In the following, the motions strictly required for the system to function, referred to as *core motions*, are described. These implement the core workflows as defined in section 5.2. These core motions are:

- RegisterMonitor
- RemoveMonitor
- RegisterConsumer
- RemoveConsumer

All motions share common control information that is required for validating and processing, shown in Figure 5.3. The system may later be expanded by adding motions that implement specific other functions, e.g. securing the network. In the following the individual core motions will be further examined.

Common Control Information

Description: Common control information that must be present in each motion

Parameters:

- `uuid` : Unique RFC 4211 identifier of this motion
- `version` : Version identifier of this motion
- `caller_uuid` : Unique identifier of caller
- `caller_address` : Address of caller
- `timestamp` : Timestamp of motion creation
- `signature` : Ed25519 signature

Fig. 5.3.: Common control information included in all motions.

Motion *RegisterMonitor*

The *RegisterMonitor* motion implements the *AddLM* and *AddNM* workflows (see section 5.2) using one shared container. The motion is created by any monitor not currently part of the system when it is initialized and ready to be integrated into the IDS. It includes two properties, the type of monitor that created the motion and a list of border component ids. These properties are used by the receivers to decide whether a reactionary workflow is triggered. Figure 5.4 illustrates the motion structure and the triggered workflows.

Motion *RemoveMonitor*

During operation it might be necessary to remove an already registered monitor from the system while the system is operational. As an example this is required during maintenance work on the subgrid, during a restructuring of the network, or when hardware is decommissioned. For this the monitor that wants to be removed publishes a *RemoveMonitor* motion. While the directly connected components could also be informed about the removal via the already established connections directly, this is not desirable as it is a very intransparent process. Modeling the removal as a motion allows for clean logging of system state changes and allows for other components to react upon the removal, even if they are not directly connected during

Motion *RegisterMonitor*

Description: Adds a new monitor to the system.
Caller: New monitor (not yet part of the system)
Callee: Bootstrapper
Properties:

- `monitor_type` : Type of Monitor (NM or LM)
- `data` : List of border components

Receiver Workflow: Local Monitor

1. Check if monitor_type is NM and data contains border components we are monitoring
2. If true: Connect to NM

Receiver Workflow: Neighborhood Monitor

1. Check if caller is assigned LM
2. If true: Connect to LM and subscribe to relevant events

Receiver Workflow: Consumer

1. Check if interested in events from monitor
2. If true: Connect to **caller** and subscribe to events

Motion *RemoveMonitor*

Description: Removes a monitor from the system.
Caller: Monitor
Callee: Any system component
Parameters:

- `reason` : Reason for removal.

Receiver Workflow

1. Check if monitor is connected to this component
2. If true: disconnect from component

Fig. 5.4.: Definition of *RegisterMonitor* and *RemoveMonitor* motions.

operation. This could for example be used to implement a failover mechanism which automatically deploys and starts replacement components.

The motion is *RemoveMonitor* has a single parameter, describing the reason for removal of the monitor. This reason is currently not used outside of debugging. In future work it could be used to multiplex between different reactionary workflows based on the reason for removal. For example the reason can be used to differentiate whether the component is only temporarily removed during expected maintenance or whether it encountered a unrecoverable error which should be investigated further by human operators.

Motion *RegisterConsumer*

To handle alert data, monitor the system or implement logging various data consumers can be created and linked to any amount of monitors. The registration of consumers works analogous to the registration of monitors but using the *RegisterConsumer* motion. The motion is parameterized with a list of identifiers, specifying which components it is interested in. Using this mechanism specialized consumers, e.g. a visualization that only handles a small subgrid, can be implemented. This list of identifiers needs to be provided when initially creating the motion. While this suffices for basic consumers that are interested in a very broad range of events, advanced filtering capabilities are needed when more complex consumers are integrated. Implementation of more in-depth filtering is out of scope of this thesis, but could easily be integrated into the existing motion structure.

Motion *RemoveConsumer*

Analogous to the *RemoveMonitor* motion, the removal of consumers is also handled by a dedicated motion called *RemoveConsumer*.

Motion *RegisterConsumer*

Description: Adds a new consumer to the system.

Caller: New consumer (not yet part of the system)

Callee: Bootstrap Service

Parameters:

- `id_filter[]` : List of identifiers (only components with identifiers that match this filter apply)

Receiver Workflow: Monitor

1. Check if consumer applies
2. If applies: Connect to caller

Motion *RemoveConsumer*

Description: Removes a registered consumer from the system.

Caller: Registered consumer

Callee: Any component

Parameters:

- `reason` : Reason for removal

Receiver Workflow

1. Check if consumer is connected to this component
2. If true: disconnect

Fig. 5.5.: Definition of *RegisterConsumer* and *RemoveConsumer* motions.

5.6 Caller Validation

As motions actively change the system state, it is mandatory to cryptographically validate all motions before they are handled. This is done by validating the cryptographic signature included in the motion. If the signature is invalid an alert is triggered, as this immediately signifies the existence of either a malicious or erroneously configured component.

To assure cryptographic soundness and reduce the potential for bugs, the validation is done using an established PKI. This allows for the use of common cryptographic modules for implementing the checks themselves and allows for using off-the-shelf software for issuance and management of the certificates. Which infrastructure is used for these steps remains unspecified and depends on the actual use case and workflows in the deployment.

The workflow for validating a caller is composed of several steps:

1. Check if certificate is provided
2. Check if certificate is temporally valid (i.e. not expired)
3. Check if certificate issuer chain is valid
4. Check revocation status of certificate with PKI

If any of the steps fail, the verification as a whole will fail, the motion will be discarded, and an alert will be generated and send to the appropriate consumers.

6 Prototype

To demonstrate the feasibility of using this new communication network, it is prototypically implemented and tested. The development is inspired by the existing prototype by Deuschle et al. [30], but constitutes a nearly complete redevelopment. The following sections introduce the new prototype and illustrate the design decisions that influence it.

6.1 Requirements Specification

To allow for evaluating whether the prototypical implementation successfully implements the newly developed system a set of requirements is compiled. This includes functional as well as non functional requirements. The functional requirements mainly pertain to whether the prototype is functional and implements the system designed in the previous chapter. This is augmented by several non functional requirements, which concern code quality, documentation, and code organization. The complete list of requirements is attached in appendix A.1.

6.2 Organization

As the IDS is conceptually designed as a distributed system, the prototype is also split into multiple separate components. To organize these, multiple dedicated git repositories are used. The GitLab server provided by the University of Münster is used for hosting the git repositories[1]. Table 6.1 lists the repositories currently used.

To share common libraries the pipy package registry provided by GitLab is used[2]. As the registry is directly tied to the individual repository which are currently not public, using it requires additional configuration. Specifically, an access token needs to be provided to allow for the build process to access and download packages from

[1] https://zivgitlab.uni-muenster.de/ag-sks/distributed-grid-ids/
[2] https://zivgitlab.uni-muenster.de/ag-sks/distributed-grid-ids/ids-components/-/packages

Name	Description
ids-components	IDS core components (monitors, consumers, etc.)
ids-testbed	MOSAIK Testbed
ids-replay-tool	CSV Replay Tool [5]
documentation	Documentation
ids-attack-tool	Attack Tool [30]
pymodbus3	Modbus3 Library

Tab. 6.1.: Overview of git repositories.

the registry. Instructions for getting and providing the token are provided in the readme-files of the respective repositories.

6.3 Documentation & Code Style

A comprehensive documentation for the project is provided in HTML format. This documentation includes general information about the project, presentations and published research, contributor information as well as code documentation. The documentation is compiled using the *sphinx*[3] toolkit. To make the documentation available without requiring a dedicated server, it is directly hosted on the GitLab server [4].

To ensure readability of the code, it is automatically formatted according to PEP 8 guidelines. The code itself is documented using *docstrings*, as well as inline comments where appropriate. The docstrings use *reStructuredText* format, allowing automatic documentation generation by the sphinx toolkit.

6.4 Implementation

This subsection describes features and challenges of the actual implementation. As the communication network is a central part of the system, it heavily influences the overall design and constitutes the major part of the actual code. As section 4.2

[3] https://www.sphinx-doc.org/en/master/
[4] https://ag-sks.zivgitlabpages.uni-muenster.de/distributed-grid-ids/documentation/

pointed out this system has several weaknesses, this new prototype will therefore replace it with the new communication system designed in Chapter 5. This implementation is therefore a complete redevelopment of the IDS. Similar to the previous prototypes the OPC-UA Framework is used for implementing the IDS. It offers all required capabilities and has been identified as a future key technology in the field of industrial control applications [5, 46].

6.4.1 Bootstrapping

The lack of a central component creates a challenge during system initialization. Each component needs at least one communication partner to start gossiping with. In centralized systems, a central entity is always available using a static network address, but in the dynamic network of the IDS, no static addresses exist. To alleviate this problem a new component called *bootstrapper* is introduced.

The bootstrapper serves as an initial entrypoint to the system. When connecting to the system for the first time, components connect to the bootstrapper, which is available on a static address. Using this connection the initial motions of the new component can be spread via gossiping. The bootstrapper keeps a rotating cache of connections to the last components that connected to it, which are then used for further spreading the motion.

It has to be noted that this bootstrapper does not suffer from the problems the centralized C2 server had (cf. section 4.2). Specifically, the bootstrapper is not stateful and does not include any business logic. It can therefore be trivially deployed multiple times within the network. The bootstrapper does not participate in the logic of the IDS and is only used for initial onboarding of new components to the system. Theoretically, any component could serve as the bootstrapper for the system in parallel to the original functionality, but for simplicity a dedicated component was created.

6.4.2 Local Monitor

To initialize the LM various parameters need to be supplied. In the previous prototype each LM required a static JSON configuration that described the monitored subgrid. These configuration files were created manually by transcribing the grid definition used by the PyPower simulators used in the testbed. This approach is not scalable and manual transcription is very prone to errors, e.g. typos or copy-and-paste errors.

Key	Example	Description
uuid	3e5b9c46-5dcd-4c82-ade8-913947f00000	Unique ID of this monitor
identifier	lm00	Identifier of this monitor
opc_address	opc.tcp://0.0.0.0:10700/	Address of the local OPC server
opc_namespace	ids	*OPC-UA Namespace of the IDS*
root_cert_file	root_cert.der	Root Certificate
cert_file	lm00_cert.der	Own Certificate
private_key_file	lm00_key.pem	Private Key
bootstrap_uuid	3e5b0000-0000-0000-0000-000000000000	Identifier of the bootstrapper
bootstrap_address	opc.tcp://manul-master:10900/	Address of the bootstrapper
rtu_config	new_rtu_0.xml	RTU xml config file
retransmission_threshold	20	Step interval before retransmitting motions

Tab. 6.2.: Configuration of LM.

This configuration file has therefore been removed. Unfortunately, there is no way to scan an RTU and extract information about the subgrid it provides. A configuration file is therefore still necessary, but instead of using a custom configuration the LM uses the configuration file used by the RTU itself. While this is not the perfect solution, as RTU configuration files need to be shared with the LM, it eliminates one error source.

As described in Section 3.3, each LM has a designated NM that is tasked with monitoring the border region of the individual LM. This component is called the *associated* NM. To allow for flexibility, the components are only loosely coupled, i.e. they exist independently of one another. The coupling of the LM with the respective NM will be described further in Section 6.4.3.

During operation the LM can be in different states. Figure 6.1 shows the overall flow between those states. The LM starts in the *initializing* state. During this phase, all necessary structures, such as the internal server and RTU connection, are set up and the connection to the bootstrapper is established. Using this connection the RegisterMonitor motion of the LM is published and the LM transitions into the *waiting_for_nm* state. As soon as the connection with the RTU is established, the LM can start checking the safety and physical requirements of the grid (cf. Table 3.1). The LM stays in this state and passively waits for connections established by other components upon receiving the previously published RegisterMonitor motion. Specifically, it waits for the RegisterMonitor motion of the associated NM. Once this connection is established it changes into the *running* state.

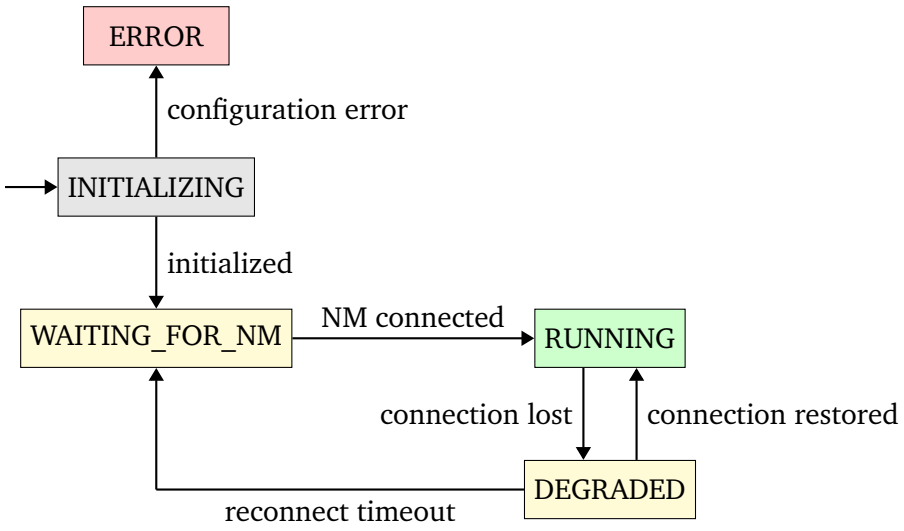

Fig. 6.1.: Status diagram of the local monitor states. During green and yellow states requirement checking is performed.

During operation the LM tracks all connected components, and keeps a list of components that are expected to be present. E.g. upon receiving a RegisterMonitor motion of an NM that monitors a border region, this NM is added to the tracker. If one of these components loses connection, the LM switches to a *degraded* state and reconnection is attempted. If the connection is reestablished, the LM returns to the *running* state. If the reconnection attempts are unsuccessful, the LM stays in *degraded* state. Due to technical limitations currently only the presence of a session is tracked and not what specific events these clients subscribed to. It is assumed that once a session is established the client then successfully subscribes to all relevant events. A special case exists when the connection to the associated NM is lost. If reconnection times out, the LM does not stay in *degraded* state but reverts back to *waiting_for_nm* state. Reconnection with the NM is then reestablished using the normal workflow used during system initialization. This allows for flexibility, as a different NM instance could take over responsibility if the original associated NM is not recoverable.

6.4.3 Neighborhood Monitor

Each LM has a associated NM, responsible for monitoring the border regions. As the NM is the flexible component, as it is fully virtual and does not require a connection to the physical grid (cf. section 3.2), the NM implements the mapping between LM and NM. For that the NM is is configured with the UUID of the LM whose border

region it should monitor. Analogous to the notion of the associated NM, this LM is referred to as *associated* LM.

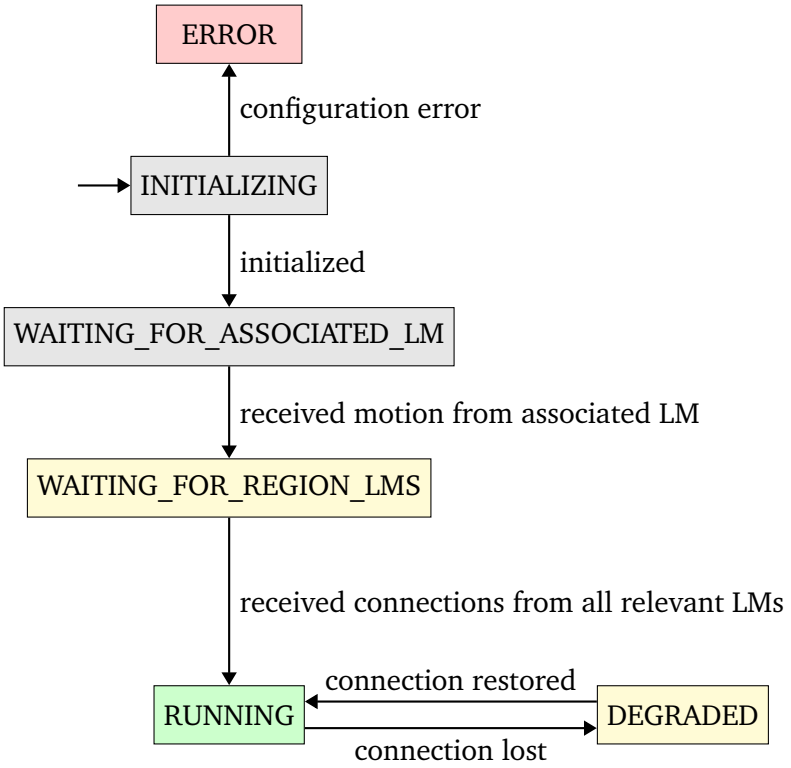

Fig. 6.2.: Status diagram of the NM states. In green states, physical and safety requirements are checked. In yellow states, partial or no requirements can be checked due to missing data.

Similar to the LM, the NM also has different states as shown in figure 6.2. It first starts in the *initialization* state. After the configuration has successfully been parsed and connection to the bootstrapper is established it transitions to *waiting_for_associated_lm* state. Contrary to the LM, which can start checking requirements immediately after initializing, the NM first needs to connect to the associated LM to retrieve the grid definition, as well as the adjacent LMs for retrieving sensor data. The first step is done passively by waiting for the RegisterMonitor motion of the associated LM. Once this motion is received, the NM can connect to the associated LM and retrieve the border region to be monitored. Based on this border region definition, the RegisterMonitor motion of the NM itself is constructed and published. The NM subsequently transitions into the *waiting_for_region_lms* state, where it passively waits for connections by LMs that react upon this motion. These LMs are here referred to as *region* LMs.

The NM tracks the sessions of all connected LMs. If an LM disconnects the NM transitions to *degraded* state while waiting for reconnection.

6.4.4 Challenges

Several unexpected challenges arose during the development. The following sections will describe them and explain possible as well as actually implemented solutions.

Unsuccessful motions

While the gossiping algorithm guarantees that all components eventually receive every motion, this only refers to components which are actually connected to the system. This poses a problem, as due to the distributed nature it is not guaranteed that the designated receiver of a motion is already connected to the system and can receive the motion. For example, when an LM initializes and sends out a RegisterMonitor motion it expects the associated NM to connect back to it after receiving the motion. If this NM is not yet part of the system or temporarily disconnected it never receives the motion the connection never happens.

Two different approaches are possible for resolving this issue. Either motions that have already happened are stored or motions are repeated if they did not resolve successfully. The first approach requires a component that provides a log of motions, which allows components to synchronize themselves by accessing motions that have happened in the past. While off-the-shelf software products that explicitly support such usage, such as Apache Kafka[5], exist, they need to be adapted to the specific IDS use case. Additionally, mechanisms for synchronization need to be implemented in each component, which makes this approach relatively complex. The second approach, simply republishing motions if they are not successful, does not require any external components. The caller of the motion simply publishes the motion again with an incremented version number. As the gossip algorithm does not provide instantaneous delivery, this retransmission should only be triggered after a reasonable amount of time has passed. This approach is only possible if the caller can identify that the motion failed, e.g. when no components connect after a RegisterMonitor motion was send. The additional drawback of this approach is the increased amount of motions circulating in the network.

To keep the system as simple as possible, the second approach of retransmitting unsuccessful motions will be implemented in the prototype. If it later shows that the increased amount of motions creates a problem, the bootstrapper component could be extended to provide a history of motions.

[5] Apache Kafka Documentation

Handling of disconnects

Disconnects of components can and will happen, and should therefore be anticipated by the system. When a component detects that another component disconnect unexpectedly two different things could have happened. The first case is an intermittent loss of connectivity while both components are operational, where simply reconnecting solves all issues and normal operation can continue. The second case is the loss of connection because the component crashed fatally, e.g. caused by a power outage. Which of these cases is actually happening in a concrete scenario is not detectable from the viewpoint of a component.

The first case does not pose a problem - all components keep their configuration and eventually the connection is reestablished. In the second case one of the components lost its configuration, while the other components still have their respective configuration. This mismatch causes multiple operational issues, as the components have a differing understanding about the established network. When the crashed component recovers, e.g. power is reestablished, it receives unsolicited connection attempts by, from its perspective unknown, components. As the components are all independent the order of these connections is unpredictable, and connection attempts occur even before it is expected, e.g. still during the *initialization* phase (cf. Figure 6.2). This is especially a problem when the connection is not used to provide data for other components, but to actively subscribe to events generated by another component, as the handling required for these processes is not yet possible.

To solve this issue, unsolicited connections that occur while still initializing the component are cached and handling is deferred until initialization is completed.

Deployment

To allow for portability of the system, *Docker* is used as a containerization engine. This allows for a clean and well defined operating environment. Images are provided via the container registry integrated into GitLab[6].

While the system conceptually separates LM and NM as two standalone system components, it does not necessitate them to be on different physical hosts that are only connected via network. To distribute the monitors several strategies could be employed. As each LM always has an associated NM, colocating them on the same

[6] https://zivgitlab.uni-muenster.de/ag-sks/distributed-grid-ids/deployment/container_registry

host is beneficial, as this eliminates the need for network communication to transfer data to the associated NM. This cuts the amount of sensor data that needs to be transmitted via the network in half. This colocation of monitors has already been done before by Großhanten [5], and has not caused issues.

For deploying the containers, several different *docker compose* files are supplied. As these files need to be individualized for each deployment, as names and certificates differ, they are provided based on the testbed that is used for evaluation in chapter 7. The specific structure of these files is therefore discussed in Section 7.1.

Review: Fulfillment of requirements

This sectin reflects on the requirements to be met by the prototype as specified in Appendix A.1.

It can be seeen that not all functional requirements are met by the prototype. Specifically, the implementation omits cryptographically signing of motions and the respective counterpart of checking signatures. All other functional requirements are met as shown in Table 6.3. All non-functional requirements are met. General project documentation is provided via readme files project documentation using the sphinx toolkit, the code is additionally documented using common documentation practices. All software components are provided with build instructions in the form of dockerfiles, ensuring reproducibility.

ID	Importance Level	Fulfillment
F010	MUST	yes
F011	MUST	yes
F020	MUST	yes
F030	MUST	yes
F200	MUST	yes
F201	SHALL	yes
F202	SHALL	no
F203	SHALL	no

Tab. 6.3.: Fulfillment of requirements specification.

7 Evaluation

This chapter evaluates the implemented prototype, with special focus on the newly developed communication system. The first section describes the testing environment and infrastructure that is deployed to facilitate the test. Section 7.2 then introduces different scenarios that represent common circumstances encountered during operation of the IDS.

7.1 Testbed

To test the system, the existing testbed already used by Großhanten [5], Menzel [1] and Chromik et al. [47] is extended.

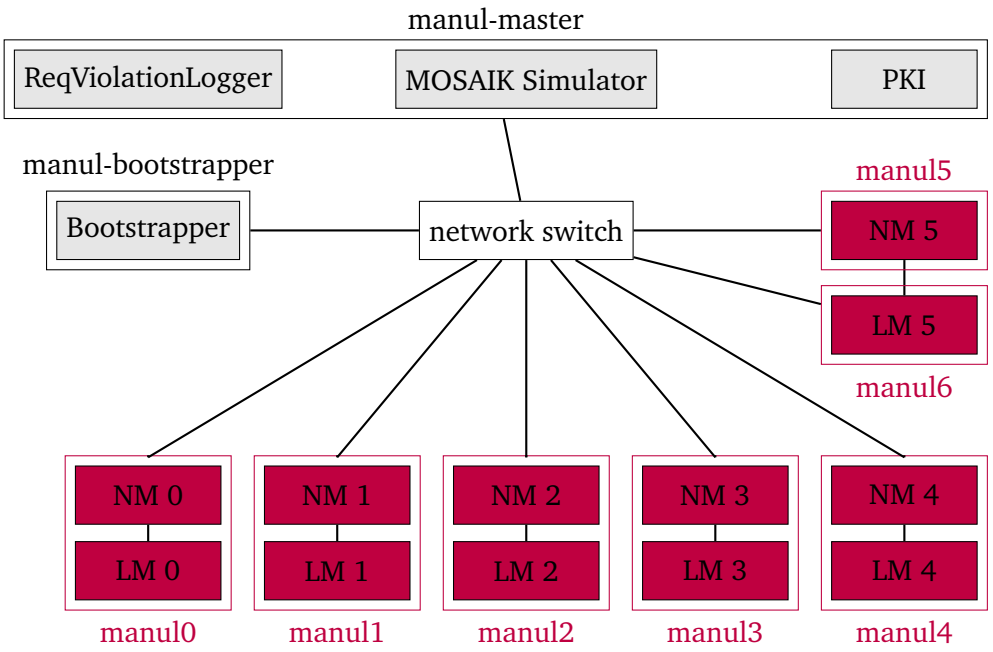

Fig. 7.1.: Structure of the physical testbed. Physical hosts start with the prefix *manul* and host different containers.

Figure 7.1 shows the physical infrastructure that is used for the testbed. Several physical host systems are used to isolate components, their names are prefixed with

Identifier	Hardware	Software
manul-master	8vCPU, 8Gb RAM	Ubuntu 22 VM on Proxmox CE
manul-bootstrapper	2vCPU, 2Gb RAM	Ubuntu 22 VM on Proxmox CE
network switch	TP-Link TL-SG1016D	OEM (by manufacturer)
manulX	Raspberry Pi 3 Model B v1.2	Raspberry Pi OS Lite (64-bit)

Tab. 7.1.: Overview on the hardware and software stacks used in the testbed.

manul. Different physical hosts are used to more accurately reflect a real world deployment. The suitability of the hardware was already examined extensively by Großhanten [5] and is therefore not evaluated further here. Table 7.1 shows the specific hardware and software stacks used for each component. While parts of the testbed have been kept, mainly the physical devices used and a subgrid, several notable adaptions have been made to the testbeds that were used previously.

To better test the capabilities of the new communication system that is designed to work with a large network, the testbed was expanded to monitor the whole simulated grid, previously only a limited subgrid was used. The division of the grid into different subgrids was done manually with the guideline of creating roughly equally sized grids, the existing subgrids were incorporated.

The second adaption is the separation of the MOSAIK simulator and other management software from the monitors. This is done to more accurately reflect the real-world scenario where these components are not running on the same hardware as the monitoring components themselves. This also helps to mitigate side effects, e.g. conflicts for memory and CPU resources, as most hosts have very limited hardware resources.

7.2 Evaluation scenarios

To allow for analyzing the performance of the IDS, scenarios are defined. They form a description of the operating environment and may include additional events such as the the malfunction of a component. The goal is to cover the different conditions that can occur in a real-world use case. Each scenario description consists of four different parameters: a short description of the scenario, the expected behaviour, instructions for the execution and the actual observed behaviour with an evaluation on the success of the scenario.

Scenario 1: Normal system operation

Description:
This scenario represents normal system operation. All external components are running normally and all network links are operational. The IDS components are started in no particular order.

Expectation:
It is expected that the IDS successfully establishes itself, and starts monitoring of the grid. Specifically, the following conditions are met:

1. All LMs successfully connect to the testbed and retrieve sensor data
2. All NMs find their associated LMs and establish connections with them
3. All monitors eventually transition into *running* state
4. During connection the certificate status is checked using OCSP.
5. No requirement violations are detected

Execution:

1. Setup non-IDS components (PKI + Testbed)
2. Start all IDS components

Observations:
As the monitors are started in parallel, a high amount of motions circulating through the system can be observed. After this initial flood of motions the system successfully establishes itself. During initialization, the monitors intermittently transition into *degraded* state. This is caused by a desynchronization between time their are added to the tracking mechanisms and the actual successful initialization of the connection and associated handlers. This behaviour is benign, but could be improved upon by better synchronizing the tracker with the actual connection. Figure 7.2 shows the output during initialization the LM and NM hosted on manul03. The logs from the other components are analogous and have therefore been omitted here. The observations match the expectations, no deviation is observed.

```
[lm03] 21:39:53.444:Changing status: LMStatus.INITIALIZING --> LMStatus.WAITING_FOR_NM
[lm03] 21:41:05.306:Changing status: LMStatus.WAITING_FOR_NM --> LMStatus.RUNNING
[lm03] 21:41:07.351:missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11114')]
[lm03] 21:41:07.351:Changing status: LMStatus.RUNNING --> LMStatus.DEGRADED
[lm03] 21:41:09.398:missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11114')]
[lm03] 21:41:11.446:missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11114')]
[lm03] 21:41:13.497:Changing status: LMStatus.DEGRADED --> LMStatus.RUNNING
```

(a) Output of LM 3. After 12 seconds the LM has found its associated NM and transitions to *running* state. Since the NM is still initializing the connection, the LM temporarily reports a *degraded* state.

```
[lm04] 21:31:16.500: Changing status: LMStatus.WAITING_FOR_NM --> LMStatus.RUNNING
[lm04] 21:31:18.604: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113'), UUID('3e5b9c46-5dcd-4c82-ade8-913947
    ↪ f11115'), UUID('3e5b9c46-5dcd-4c82-ade8-913947f11116')]
[lm04] 21:31:18.605: Changing status: LMStatus.RUNNING --> LMStatus.DEGRADED
[lm04] 21:31:20.690: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113'), UUID('3e5b9c46-5dcd-4c82-ade8-913947
    ↪ f11115'), UUID('3e5b9c46-5dcd-4c82-ade8-913947f11116')]
[lm04] 21:31:26.939: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11115'), UUID('3e5b9c46-5dcd-4c82-ade8-913947
    ↪ f11116')]
[lm04] 21:31:35.315: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11115'), UUID('3e5b9c46-5dcd-4c82-ade8-913947
    ↪ f11116')]
[lm04] 21:31:41.600: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11116')]
[lm04] 21:31:43.696: Changing status: LMStatus.DEGRADED --> LMStatus.RUNNING
```

(b) Output of LM 4. After initially missing connections to several NMs, they are subsequently established

```
[nm03] 21:39:44.514:connection_callback from bootrapper
[nm03] 21:39:52.600:Changing status: NMStatus.INITIALIZING --> NMStatus.WAITING_FOR_ASSOCIATED_LM
[nm03] 21:40:56.076:Changing status: NMStatus.WAITING_FOR_ASSOCIATED_LM --> NMStatus.WAITING_FOR_REGION_LMS
[nm03] 21:41:06.155:reconnecting connection_callback 3e5b9c46-5dcd-4c82-ade8-913947f00003
[nm03] 21:41:20.559:connection while waiting for region lm 3e5b9c46-5dcd-4c82-ade8-913947f00002
[nm03] 21:41:29.251:connection while waiting for region lm 3e5b9c46-5dcd-4c82-ade8-913947f00004
[nm03] 21:41:37.242:Changing status: NMStatus.WAITING_FOR_REGION_LMS --> NMStatus.RUNNING
```

(c) Output of NM 3. After finding the associated LM the RegisterMonitor motion is published and several connections by other components are received and handled.

Fig. 7.2.: Output of components hosted on *manul03* and *manul04* during startup of the system. The output is trimmed for readability.

Scenario 2: Component malfunction: NM

Description:
This scenario evaluates how the IDS handles malfunction of an NM. Specifically the total failure of an NM is simulated by forcefully shutting down NM 3.

Expectation:
Until the NM is disconnected the system behaves normal, analogous the the previous scenario. After NM 3 is disconnected, all components except NM 3 continue running. The LMs whose border region was watched by NM 3 continue running in a degraded state. All components that had a connection with NM 3 will generate an event about the unexpected connection loss. These components will attempt to reconnect with exponential backoff until the maximum number of retries is exceeded. After NM 3 is reconnected, it reintegrates itself into the system and normal operation is restored.

Execution:

1. Setup non-IDS components (PKI + Testbed)
2. Start IDS
3. After IDS is established and running, forcefully shutdown NM 5
4. Reconnect NM

Observations:
Figure 7.3 shows the output of components during the scenario. The system successfully recovers, and the stopped NM is reintegrated into the system. This matches the expectations. It can be noted that the outputs from LM 3 and LM 4 differ, the debug logging of LM 4 does not show any connection errors. This happens, because LM 4 did not actively use the connection with NM 3 for gossiping and therefore does not detect the loss of connection. As the order of gossip partners is randomized (cf. Section 5.4), this is normal behaviour and an error would have been logged if the NM would have still been disconnected when it is chosen as gossip partner.

```
[lm03] 22:47:40.376:missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113')]
[lm03] 22:47:40.378:Changing status: LMStatus.RUNNING --> LMStatus.DEGRADED
[lm03] 22:47:42.425:missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113')]
...
[lm03] 22:48:23.677:Connection error while connecting to OPC-UA Server. Retrying in 2 seconds
[lm03] 22:48:25.663:Changing status: LMStatus.DEGRADED --> LMStatus.WAITING_FOR_NM
[lm03] 22:48:25.738:Connection error while connecting to OPC-UA Server. Retrying in 4 seconds
[lm03] 22:48:29.820:Connection error while connecting to OPC-UA Server. Retrying in 8 seconds
[lm03] 22:49:07.399:Changing status: LMStatus.WAITING_FOR_NM --> LMStatus.RUNNING
```

(a) Output of LM 3 during the scenario. LM continues running in *degraded* state before the NM recovers and the connection is reestablished. The LM tried to gossip with the NM and received a connection error, triggering reconnection attempts with exponential backoff.

```
[lm04] 22:47:41.153: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113')]
[lm04] 22:47:41.154: Changing status: LMStatus.RUNNING --> LMStatus.DEGRADED
[lm04] 22:47:43.236: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113')]
...
[lm04] 22:48:54.427: missing clients: [UUID('3e5b9c46-5dcd-4c82-ade8-913947f11113')]
[lm04] 22:48:56.542: Changing status: LMStatus.DEGRADED --> LMStatus.RUNNING
```

(b) Output of LM 4, whose border region is watched by the stopped NM 3. After running in *degraded* state the NM reconnects and the LM transitions into normal *running* state.

```
[nm03] 22:48:32.210:connection_callback 3e5b0000-0000-0000-0000-000000000000
[nm03] 22:48:32.211:connection_callback from bootrapper
[nm03] 22:48:34.246:connection_callback 3e5b9c46-5dcd-4c82-ade8-913947f00002
[nm03] 22:48:34.248:buffering callback to 3e5b9c46-5dcd-4c82-ade8-913947f00002
[nm03] 22:48:37.866:Changing status: NMStatus.INITIALIZING --> NMStatus.WAITING_FOR_ASSOCIATED_LM
[nm03] 22:48:38.809:connection_callback 3e5b9c46-5dcd-4c82-ade8-913947f00003
[nm03] 22:48:38.809:connection_callback from associated LM
[nm03] 22:48:47.256:connection_callback 3e5b0000-0000-0000-0000-000000000000
[nm03] 22:48:47.257:reconnecting connection_callback 3e5b0000-0000-0000-0000-000000000000
[nm03] 22:48:48.195:Changing status: NMStatus.WAITING_FOR_ASSOCIATED_LM --> NMStatus.WAITING_FOR_REGION_LMS
[nm03] 22:48:48.201:connection_callback 3e5b9c46-5dcd-4c82-ade8-913947f00002
[nm03] 22:48:48.202:connection while waiting for region lm 3e5b9c46-5dcd-4c82-ade8-913947f00002
[nm03] 22:48:51.234:connection_callback 3e5b9c46-5dcd-4c82-ade8-913947f00004
[nm03] 22:48:51.236:connection while waiting for region lm 3e5b9c46-5dcd-4c82-ade8-913947f00004
[nm03] 22:49:02.081:Changing status: NMStatus.WAITING_FOR_REGION_LMS --> NMStatus.RUNNING
```

(c) Output of NM 3 after it is restarted. During initialization the NM already receives connections by other components which are buffered and handled later (cf. Section 6.4.4). The NM successfully recovers and reintegrates itself with the network.

Fig. 7.3.: Examplary output of different components during scenario 2. Output is trimmed for readability.

Scenario 3: Requirement violations

Description:
This scenario demonstrates the integration of a consumer into the running system. The consumer subscribes to requirement violation events from all monitors. No filter is provided, so the consumer subscribes to events generated by all monitors.

Expectation:
The consumer integrates with the IDS and registers with all monitors. If a violation is then detected, the consumer is actively notified via an event. The delay between the LM detecting a requirement violation and the consumer receiving it is minimal. As the consumer and the LM are connected using physical network links and located very close to each other, the delay should not exceed a couple of milliseconds. The event includes all relevant information about the violation.

Execution:
As the testbed by default does not produce requirement violations, the configuration file defining the maximum permissible values for a powerline is modified. Specifically the maximum allowed voltage on *node_b28*, which is connected to *sensor_209* and *sensor_207* is redefined to be 1337 Volt. This triggers a requirement violation in every monitoring step, as the voltage on the bus actually fluctuates around 10.5kV.

1. Setup non-IDS components (PKI + Testbed)
2. Start IDS
3. After the IDS is established, start consumer

Observations:
The consumer subscribes to requirement violation events from all monitors. After subscribing to LM 3, events are received. Events are received milliseconds after the LM detected the violation, as the push-based workflow minimizes delay (cf. Section 3.5). Figure 7.4 shows the logs generated by the consumer and LM 3.

```
[violationlogger]: 03:25:29.659: initializing
[violationlogger]: 03:25:29.659: connecting to bootstrapper
[violationlogger]: 03:26:06.608: motion caller 3e5b0000-0000-0000-0000-000000000000 is already known - nothing to do
[violationlogger]: 03:26:07.199: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f11113
[violationlogger]: 03:26:10.519: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f00006
[violationlogger]: 03:26:13.636: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f00005
[violationlogger]: 03:26:21.679: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f11112
[violationlogger]: 03:26:24.850: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f11115
[violationlogger]: 03:26:28.282: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f00003
[violationlogger]: 03:26:30.213: Event(['requirement:8', 'component_id:sensor_207', "EventId:b'794
    ↪ b5c4463604794bdb9d4e9f9914e59'", 'Time:01:26:30.198530', 'ReceiveTime:01:26:30.198545', "Message:LocalizedText(
    ↪ Locale=None, Text='Requirement 8 (local) violated! Voltage on meter sensor_207 exceeds max_voltage:
    ↪                     (10502.892622310828 >= 1337.0)')", 'Severity:1'])
[violationlogger]: 03:26:30.214: Event(['requirement:8', 'component_id:sensor_209', "EventId:b'148
    ↪ b7ffd63e745158c0bcc9f4b7e36d1'", 'Time:01:26:30.202034', 'ReceiveTime:01:26:30.202043', "Message:LocalizedText(
    ↪ Locale=None, Text='Requirement 8 (local) violated! Voltage on meter sensor_209 exceeds max_voltage:
    ↪                     (10502.892622310828 >= 1337.0)')", 'Severity:1'])
[violationlogger]: 03:26:31.431: Subscribed to ReqViolationEvent from 3e5b9c46-5dcd-4c82-ade8-913947f11114
[violationlogger]: 03:26:32.299: Event(['requirement:8', 'component_id:sensor_209', "EventId:b'822
    ↪ d8fdea9544dc189809b8e69d6ccfe'", 'Time:01:26:32.278206', 'ReceiveTime:01:26:32.278221', "Message:LocalizedText(
    ↪ Locale=None, Text='Requirement 8 (local) violated! Voltage on meter sensor_209 exceeds max_voltage:
    ↪                     (10500.813269270247 >= 1337.0)')", 'Severity:1'])
[violationlogger]: 03:26:34.490: Event(['requirement:8', 'component_id:sensor_207', "EventId:b'2
    ↪ e6beb5992da4ba5b00c419b0b9787a8'", 'Time:01:26:34.325505', 'ReceiveTime:01:26:34.325516', "Message:LocalizedText(
    ↪ Locale=None, Text='Requirement 8 (local) violated! Voltage on meter sensor_207 exceeds max_voltage:
    ↪                     (10499.81157575161 >= 1337.0)')", 'Severity:1'])
[violationlogger]: 03:26:34.490: Event(['requirement:8', 'component_id:sensor_209', "EventId:b'299330563162441
    ↪ a8879dbf89039ec7a'", 'Time:01:26:34.328963', 'ReceiveTime:01:26:34.328971', "Message:LocalizedText(Locale=None, Text
    ↪ ='Requirement 8 (local) violated! Voltage on meter sensor_209 exceeds max_voltage:
    ↪ (10499.81157575161 >= 1337.0)')", 'Severity:1'])
```

(a) Output of Consumer. After publishing the *RegisterConsumer* motion monitors start connecting and the consumer subscribes to relevant events. Requirement violations for requirement 8 on *sensor_209* and *sensor_207* are received and logged.

```
[lm03] 03:26:36.379346: Requirement 8 (local) violated! Voltage on meter sensor_207 exceeds max_voltage:
    ↪ (10499.809044697846 >= 1337.0)
[lm03] 03:26:36.382700: Requirement 8 (local) violated! Voltage on meter sensor_209 exceeds max_voltage:
    ↪ (10499.809044697846 >= 1337.0)
[lm03] 03:26:38.453770: Requirement 8 (local) violated! Voltage on meter sensor_207 exceeds max_voltage:
    ↪ (10506.075668399544 >= 1337.0)
[lm03] 03:26:38.459349: Requirement 8 (local) violated! Voltage on meter sensor_209 exceeds max_voltage:
    ↪ (10506.075668399544 >= 1337.0)
```

(b) Output of LM 3 during the scenario. Requirement violations are detected and logged.

Fig. 7.4.: Output of the consumer and LM 3, which generates the violation events.

General observations

The MOSAIK testbed used for simulating the grid and providing the data only has a limited simulation time. After about 723 seconds the simulation finishes and all simulators are shut down. To allow for longer evaluation of the IDS, the testbed is then restarted automatically by docker. It takes some time until the testbed is set up again and reconnection is accepted. This intermittent loss of connection to testbed reflects in the logs of each LM, as they experience intermittent connection failures during this restart process. Figure 7.5 shows the log generated by the LMs. As this is a limitation of the setup, these error messages are benign and can be ignored. But it can be noted that intermittent loss of connection to the grid does not cause any issues, other than the lack of detected requirement violations caused by the lack of actual sensor data.

```
[lm02] 22:55:46.857: Modbus Error: [Input/Output] Server responded with bad
    ↪ response
[lm02] 22:55:48.863: Error connecting to Modbus Server'manul-master:10501'
```

Fig. 7.5.: Example output of LM when MOSAIK testbed is restarting

7.3 Results

The scenarios show that the new communication system is functional and can handle different scenarios. While these scenarios do not cover all possible conditions the system is subjected to in a real world use case, it forms a base for further research and evaluation.

It can be deduced that the new communication system works and the approaches developed in Chapter 5 can be successfully be used to develop an IDS that is fully distributed. The new communication system thereby remediates several of the problems and weaknesses which were present in previous prototypes. The most important factor is the removal of the central Command & Control server, whose centralized nature formed a big weakness in the overall IDS. Section 8.2 further elaborates on the results by answering the research questions.

8 Conclusion

Finalizing this thesis, this chapter gives an overall review of the thesis and evaluates the answers to the research questions. Furthermore, future work is outlined, pointing out future research topics that have been identified during work on this thesis.

8.1 Summary

After Chapter 1 introduces the general topic and motivates the thesis, Chapter 2 gives an overview over related work and the general background. The smart grid and the challenges it entails are described, establishing the operating environment and problems that need to be solved. Based on this, SCADA as well as IDS systems are introduced. Furthermore, the need for research in the context of resilient communication networks is outlined.

Chapter 3 then defines the distributed IDS that is examined further in this thesis. Special focus is given to the communication requirements of the IDS. Three different data archetypes that are present in the IDS are identified, and recommendations on how communication should be facilitated given.

The existing IDS prototype is then analyzed in Chapter 4. Based on the analysis and the recommendations given in the previous chapter, the existing prototype is critically evaluated. It showed that while the system works sufficiently in the limited test scenarios that were used in previous research, various problems and weaknesses exist when expanding these scenarios. Especially the centralization caused by the presence of a monolithic Command & Control server is a big weakness when the resilience of the IDS is investigated.

Chapter 5 subsequently develops a new communication network. The main goal is to remediate the issues identified in the previous chapter, while also improving overall maintainability and scalability of the system. First the assumptions this system makes, mainly requirements towards the operating environment, are clarified. Afterwards, the overall design goals that influence the development are outlined. Rumor-spreading algorithms are identified as a promising approach for

facilitating the communication, and different types are described. To fully utilize the rumor-spreading approach the concept of *motions* is introduced, which is used to standardize management of the network by containerizing each workflow into a dedicated container.

The next chapter describes the actual prototypical implementation of the system. For this, a small requirements specification is compiled, which later allows for evaluating whether the set goals were accomplished. The following sections describe the prototype, first organizational aspects and then different interesting and relevant implementation details are presented.

The developed prototype is evaluated in Chapter 7. The existing scenarios is expanded upon to more accurately reflect real conditions. The prototype is then subjected to the three scenarios and the actual behaviour is compared with the expected behaviour. Based on this an overall evaluation of the new system can be given, specifically in comparison with the previous prototype.

The last chapter concludes the thesis, summarizing the work and revisiting the research questions. Lastly, future research areas are identified and sketched out.

8.2 Research Questions

This thesis aimed to achieve the objective of developing a new and robust communication system for the IDS with the primary goal to achieve robustness against internal malfunctions as well as external threads. For this, the thesis posed three different research questions, which can be answered based on the results of the various chapters.

Question 1: *How should communication inside a hierarchically-organized distributed IDS be facilitated?*

This question has been thoroughly investigated in Chapter 3. The three data archetypes present in the system, configuration data, sensor data and event data each have differing requirements and should therefore be handled differently. Configuration data needs to be validated, but has no strict requirements towards time-criticality or minimization. It is therefore relatively indifferent to the communication mode chosen. Sensor and Event data on the other hand are very time-critical and should therefore always be transmitted using a push-based approach. Furthermore, as sensor data is transmitted very frequently, its size should be minimized by using a

binary-based encoding format. For the other types this does not matter as much as they are only transmitted infrequently.

Question 2: *How can the IDS organize itself without the need for centralized management?*

The problems of using a centralized management entity, such as the Command & Control server that was used by the previous prototype, have been elaborated on in Section 4.2. While various ways exist to implement self-organization of components, this thesis further explored the use of rumor-spreading algorithms (cf. Section 5.4). This methodology is very flexible and can handle the dynamic network topology that is present in the IDS, while providing very high redundancy without requiring external entities or complex mechanisms. By using a pseudorandom selection algorithm for choosing communication partners, a model of eventual consistency can be implemented while still guaranteeing acceptable bounds for time complexity. All of these properties make it a good fit for a fully self-organized distributed IDS.

When using such a rumor-spreading algorithm for distribution of messages through a network, it is important to ensure that the transmitted message contains all information that is necessary by any receiver to successfully handle the message. For this the concept of *motions* is introduced in Section 5.5. Motions form a standardized message container, encapsulating all information that is required for a specific workflow. Using this framework, all core workflows of the IDS can be handled in a efficient manner. Additionally, it can be guaranteed that all messages conforms to the same security guidelines.

Question 3: *How can the IDS be resilient against malfunctions and outside attacks?*

A distributed system inherently has a large attack surfaces, as components often run in untrusted environments and due to the number of components many attack vectors exist. When evaluating attacks, two primary cases need to be addressed. The first case is the defense against attacks originating outside of the system. While this is certainly the most prominent attack vector, attacks originating form inside the system, e.g. through a compromised component, are another attack vector to be recognized.

Both of these factors are best addressed by integrating resilience into the system by design. The new communication system for the IDS is therefore fully distributed, each component is autonomous and configures itself. This minimizes the attack surface, as no external dependencies such as configuration services exist that could be compromised.

The communication network presented in this thesis also inherently incorporates some protection against attacks by malicious components that are already inside the system. In the rumor-spreading algorithm each motion is duplicated manyfold inside the network, with each duplicate taking a different route. Even if a malicious component is able to modify the motion, e.g. because a key got compromised, it can only modify the motions that pass through it. Motions that take a different route through the network cannot be modified. While this does not inherently protect against an intermediary component modifying motions, it makes it trivially easy to detect by comparing all received duplicates of a motion and ensuring they are the same.

Another factor impacting the security of the grid is ensuring continuous monitoring. As the power grid is continuously in operation, the IDS must also support this mode of operation. Requiring the IDS to temporarily shut down, e.g. when reconfiguration is necessary, would invalidate the whole approach of the IDS. An attacker could just wait until the next downtime for the attack, or possibly even inducing downtime themselves by a secondary attack.

The new communication network is certainly not the ultimate defense against all possible attacks, but it forms a solid base for further research into enhancing the capabilities and resilience of the IDS.

8.3 Future Work

As already described in Chapter 2, the evolution of the smart grid is still ongoing, requiring continuous research and innovation. While this thesis provided a starting point towards a resilient distributed IDS, other challenges remain unsolved. The following points form a short list of future research topics that have been identified during work on this thesis:

- **Additional consumers:**
  Due to time constraints this thesis only implemented a basic proof-of-concept consumer that logs requirement violations to a console. But various other consumers can be conceptualized.
  - Visualization of the IDS itself
  - Visualization of requirement violations
  - Monitoring of IDS components themselves

  - Fallback orchestration (e.g. automatically startup new components if other components are unavailable or crashed unrecoverably)

- **Defensive strategies:**
  This thesis only implemented the core workflows required for basic functionality of the IDS, but various other workflows are imaginable. As the communication system is designed to be very transparent, any change to the system state is implemented by publishing motions, it can easily be monitored. A component could integrate itself into the system and monitor the flow of motions to detect abnormalities.

- **Expand caller verification:**
  Due to time constraints this thesis only implemented encryption and cryptographic validation on the level of individual connections between components. As described in Section 5.5, the individual motions should also be signed and validated cryptographically. This allows for any receiver of the motion to ensure that the motion itself is valid, and was not modified by an intermittent component transmitting the motion.

Bibliography

[1] Verena Menzel. “A hierarchical approach to monitoring SCADA networks”. Master’s Thesis. WWU Münster, 2021 (cit. on pp. 1, 13–18, 20, 25, 30, 54, 73).

[2] Justyna J. Chromik, Carina Pilch, Pascal Brackmann, et al. “Context-aware local Intrusion Detection in SCADA systems: A testbed and two showcases”. In: *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm). Oct. 2017, pp. 467–472 (cit. on pp. 1, 12–14).

[3] Justyna Joanna Chromik. “Process-aware SCADA traffic monitoring: A local approach”. ISBN: 978-90-365-4801-4 Issue: 19-009 Series: DSI Ph.D. thesis series. PhD thesis. Netherlands: University of Twente, July 12, 2019 (cit. on pp. 1, 10, 13).

[4] Panagiotis I. Radoglou-Grammatikis and Panagiotis G. Sarigiannidis. “Securing the Smart Grid: A Comprehensive Compilation of Intrusion Detection and Prevention Systems”. In: *IEEE Access* 7 (2019), pp. 46595–46620 (cit. on pp. 1, 3).

[5] Kai Oliver Großhanten. “Anforderungsanalyse fur ein verteiltes IDS und Entwicklung eines Prototypen auf Raspberry Pis”. Bachelor’s Thesis. WWU Münster, Oct. 1, 2022 (cit. on pp. 1, 3, 25, 29, 32, 46, 47, 53–55).

[6] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Transactions on Programming Languages and Systems* (July 1982), pp. 382–401 (cit. on pp. 3, 30).

[7] Peter Palensky and Friederich Kupzog. “Smart Grids”. In: *Annual Review of Environment and Resources* 38.1 (Oct. 17, 2013), pp. 201–226 (cit. on p. 6).

[8] Chendan Li, Tomislav Dragicevic, Nelson Leonardo Díaz Aldana, et al. “Grid architecture for future distribution system — A cyber-physical system perspective”. In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society. Oct. 2017, pp. 5235–5239 (cit. on p. 6).

[9] *COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE AND THE COMMITTEE OF THE REGIONS Smart Grids: from innovation to deployment*. 2011 (cit. on p. 6).

[10] Volker Berkhout, André Bisevic, Michael Claußner, et al. “Windenergie Report Deutschland 2018”. In: (2019). Publisher: Fraunhofer Verlag (cit. on p. 7).

[13] Katrin Schaber and Florian Bieberbach. “Redispatch und dezentrale Erzeugung: Alternativen zum Netzausbau?” In: (2015) (cit. on p. 7).

[14]Irfan Ahmed, Sebastian Obermeier, Martin Naedele, and Golden G. Richard III. “SCADA Systems: Challenges for Forensic Investigators”. In: *Computer* 45 (Dec. 1, 2012), pp. 44–51 (cit. on p. 8).

[15]Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. “A Survey on Cyber Security for Smart Grid Communications”. In: *IEEE Communications Surveys & Tutorials* 14.4 (2012). Conference Name: IEEE Communications Surveys & Tutorials, pp. 998–1010 (cit. on p. 8).

[16]Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, et al. “The Cybersecurity Landscape in Industrial Control Systems”. In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 1039–1057 (cit. on p. 8).

[17]Simon Duque Anton, Daniel Fraunholz, Christoph Lipps, et al. “Two decades of SCADA exploitation: A brief history”. In: *2017 IEEE Conference on Application, Information and Network Security (AINS)*. 2017 IEEE Conference on Application, Information and Network Security (AINS). Miri: IEEE, Nov. 2017, pp. 98–104 (cit. on p. 9).

[18]Teodor Sommestad, Göran N. Ericsson, and Jakob Nordlander. “SCADA system cyber security — A comparison of standards”. In: *IEEE PES General Meeting*. IEEE PES General Meeting. ISSN: 1944-9925. July 2010, pp. 1–8 (cit. on p. 9).

[19]Geeta Yadav and Kolin Paul. “Architecture and security of SCADA systems: A review”. In: *International Journal of Critical Infrastructure Protection* 34 (Sept. 1, 2021), p. 100433 (cit. on p. 9).

[20]A. Nicholson, S. Webber, S. Dyer, T. Patel, and H. Janicke. “SCADA security in the light of Cyber-Warfare”. In: *Computers and Security* 31.4 (June 1, 2012), pp. 418–436 (cit. on p. 9).

[21]Elisavet Grigoriou, Athanasios Liatifis, Panagiotis Radoglou Grammatikis, et al. “Protecting IEC 60870-5-104 ICS/SCADA Systems with Honeypots”. In: July 27, 2022, pp. 345–350 (cit. on p. 9).

[22]Vasiliki Kelli, Panagiotis Radoglou Grammatikis, Achilleas Sesis, et al. “Attacking and Defending DNP3 ICS/SCADA Systems”. In: May 1, 2022, pp. 183–190 (cit. on p. 9).

[23]Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. “Intrusion detection system: A comprehensive review”. In: *Journal of Network and Computer Applications* 36.1 (Jan. 1, 2013), pp. 16–24 (cit. on p. 9).

[24]David Tipper. “Resilient network design: challenges and future directions”. In: *Telecommunication Systems* 56.1 (May 1, 2014), pp. 5–16 (cit. on p. 10).

[27]David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. “Characterising the Digital Twin: A systematic literature review”. In: *CIRP Journal of Manufacturing Science and Technology* 29 (May 1, 2020), pp. 36–52 (cit. on p. 12).

[28]Robert Flosbach, Justyna Joanna Chromik, and Anne Remke. “Architecture and Prototype Implementation for Process-Aware Intrusion Detection in Electrical Grids”. In: *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. 2019 38th Symposium on Reliable Distributed Systems (SRDS). Lyon, France: IEEE, Oct. 2019, pp. 42–4209 (cit. on p. 12).

[29] Verena Menzel, Johann L. Hurink, and Anne Remke. “Securing SCADA networks for smart grids via a distributed evaluation of local sensor data”. In: *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2021, pp. 405–411 (cit. on p. 12).

[30] Tom Deuschle, Piet Adick, and Jan Speckamp. *Abschlussbericht Projektseminar: IT-Sicherheit für Stromnetze*. WWU Münster, 2022 (cit. on pp. 25, 29, 45, 46).

[31] Eric Brewer. *Spanner, TrueTime and the CAP Theorem*. 2017 (cit. on p. 30).

[32] Paul J. Leach, Rich Salz, and Michael H. Mealling. *A Universally Unique IDentifier (UUID) URN Namespace*. Request for Comments RFC 4122. Num Pages: 32. Internet Engineering Task Force, July 2005 (cit. on p. 31).

[33] Alexandros G. Dimakis, Anand D. Sarwate, and Martin J. Wainwright. “Geographic gossip: efficient aggregation for sensor networks”. In: *Proceedings of the 5th international conference on Information processing in sensor networks*. IPSN ’06. New York, NY, USA: Association for Computing Machinery, Apr. 19, 2006, pp. 69–76 (cit. on p. 35).

[34] Boris Pittel. “On Spreading a Rumor”. In: *SIAM Journal on Applied Mathematics* 47.1 (Feb. 1987), pp. 213–223 (cit. on p. 35).

[35] C. Avin and Robert Elsässer. “Breaking the log n barrier on rumor spreading”. In: *ArXiv* (Dec. 8, 2015) (cit. on p. 35).

[36] Alan Demers, Dan Greene, Carl Hauser, et al. “Epidemic algorithms for replicated database maintenance”. In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. PODC ’87. New York, NY, USA: Association for Computing Machinery, Dec. 1, 1987, pp. 1–12 (cit. on p. 35).

[37] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. “Resource discovery in distributed networks”. In: *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*. PODC99: ACM Symposium on Principles of Distributed Computing. Atlanta Georgia USA: ACM, May 1999, pp. 229–237 (cit. on p. 35).

[38] D. Kempe, A. Dobra, and J. Gehrke. “Gossip-based computation of aggregate information”. In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings. ISSN: 0272-5428. Oct. 2003, pp. 482–491 (cit. on p. 35).

[39] Josef Schindler, Asmaa Tellabi, and Karl Waedt. *Gossip protocol approach for a decentralized energy market with OPC UA client-server communication*. Accepted: 2021-01-27T13:33:39Z ISSN: 1617-5468. Gesellschaft für Informatik, Bonn, 2021 (cit. on p. 35).

[40] Despina I. Koukoula and Nikos D. Hatziargyriou. “Convergence acceleration of gossip protocols applied for decentralized distribution grid management”. In: *2015 IEEE Eindhoven PowerTech*. 2015 IEEE Eindhoven PowerTech. June 2015, pp. 1–6 (cit. on p. 35).

[41]Daniele Croce, Fabrizio Giuliano, Ilenia Tinnirello, et al. “Overgrid: A Fully Distributed Demand Response Architecture Based on Overlay Networks”. In: *IEEE Transactions on Automation Science and Engineering* 14.2 (Apr. 2017), pp. 471–481 (cit. on p. 35).

[42]R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. “Randomized rumor spreading”. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. Proceedings 41st Annual Symposium on Foundations of Computer Science. ISSN: 0272-5428. Nov. 2000, pp. 565–574 (cit. on p. 36).

[43]George Giakkoupis. “Tight bounds for rumor spreading in graphs of a given conductance”. In: *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Ed. by Thomas Schwentick and Christoph Dürr. Vol. 9. Leibniz International Proceedings in Informatics (LIPIcs). ISSN: 1868-8969. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 57–68 (cit. on p. 36).

[44]Petra Berenbrink, Robert Elsässer, and Thomas Sauerwald. “Communication Complexity of Quasirandom Rumor Spreading”. In: *Algorithmica* 72.2 (June 1, 2015), pp. 467–492 (cit. on p. 38).

[45]Benjamin Doerr, Tobias Friedrich, and Thomas Sauerwald. “Quasirandom Rumor Spreading”. In: *ACM Transactions on Algorithms* 11.2 (Oct. 30, 2014), 9:1–9:35 (cit. on p. 38).

[46]S. Rohjans and Michael Specht. “OPC UA: An Automation Standard for Future Smart Grids”. In: 2013 (cit. on p. 47).

[47]Justyna J Chromik, Anne Remke, and Boudewijn R Haverkort. “An integrated testbed for locally monitoring SCADA systems in smart grids”. In: *Energy Informatics* 1.1 (2018). Publisher: SpringerOpen, pp. 1–29 (cit. on p. 54).

Webpages

[@11]Timo Hartmann, Simon Bächle, and Ellen Szczepaniak. *Redispatch 3.0: bottleneck management through decentralised micro-plants?* URL: `https://www.adesso.de/en/news/blog/redispatch-3-0-bottleneck-management-through-decentralised-micro-plants-2.jsp` (visited on Aug. 9, 2023) (cit. on p. 7).

[@12]*Bundesnetzagentur - Redispatch*. URL: `https://www.bundesnetzagentur.de/DE/Fachthemen/ElektrizitaetundGas/Versorgungssicherheit/Netzengpassmanagement/Engpassmanagement/Redispatch/start.html` (visited on July 25, 2023) (cit. on p. 7).

[@25]tagesschau.de. *Kritische Infrastruktur: Politiker fordern bessere Sicherheit*. tagesschau.de. URL: `https://www.tagesschau.de/inland/sabotage-infrastruktur-sicherheit-101.html` (visited on July 25, 2023) (cit. on p. 11).

[@26]Thorsten Neuhetzki. *Hintergrund: Darum ist das Bahn-Netz ausgefallen*. inside digital. Oct. 10, 2022. URL: `https://www.inside-digital.de/news/hintergrund-darum-ist-das-bahn-netz-ausgefallen` (visited on Aug. 9, 2023) (cit. on p. 11).

Appendix

A

A.1 Requirements Specification

The following table shows all requirements towards the prototypical implementation developed during this thesis. The requirements each have one of three importance levels attached. The level *MUST* denotes that fulfillment is integral to the operation of the prototype. The level *SHALL* denotes requirements that should be fulfilled, but operation is possible without fulfillment or with partial fulfillment, possibly in a degraded way. The level *CAN* describes requirements that are optional.

ID	Description	Importance Level
	Operation	
F010	The system detects requirement violations for the local and neighborhood scope as defined by Menzel [1].	MUST
F011	The IDS provides a method to actively push requirement violations and other remarkable system events to relevant consumers	MUST
F020	The IDS is fully decentralized (external systems, e.g. networking equipment, are exempt from this requirement as it is not enforceable).	MUST
	Workflows	

F030	The system handles the core motions specified in Section 5.5. • Add Monitor • Remove Monitor • Add Consumer • Remove Consumer	MUST
	Security	
F200	All communication is encrypted using common cryptography methods. For authentication x509 certificates are used	MUST
F201	Certificates are validated using PKI. I.e. either OCSP or CRLs are supported.	SHALL
F202	Motions are signed with a cryptographic signature based on the certificate after creation	SHALL
F203	The authenticity and integrity of received motions is cryptographically checked upon receival	SHALL

Non-functional requirements

ID	Description	Importance Level
	Documentation & Maintainability	

NF001	All source code is documented using established documentation practices. All newly written code conforms to commonly used formatting and linting requirements. Python code adheres to the guidelines specified in *PEP 8*. Automatically generated code is exempt from these requirements.	SHALL
NF002	All code repositories include a readme file describing the contents. If external documentation exists it is linked there.	SHALL
NF003	The code is organized using git repositories and hosted on the zivgitlab.uni-muenster.de provider.	SHALL
	Licensing	
NF101	All implemented components and specifications are licensed under a common FLOSS license. Compatibility with used libraries and frameworks is ensured.	MUST
	Ease-of-Setup	
NF201	All software component are provided in a portable and standardized format (e.g. docker containers). All steps required to build the software are reproducible/deterministic given the documentation.	CAN

A.2 Data CD

A compact disk is attached to this thesis, containing the data listed below.

- Thesis:
  A PDF version of this thesis.
- Sourcecode:
  The sourcecode of the prototype. The sourcecode is copied from the GitLab repository at commit <>. The code can also be found at https://zivgitlab.uni-muenster.de/ag-sks/distributed-grid-ids/.

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration of Academic Integrity

I hereby confirm that this thesis, entitled

Developing a robust communication system for a distributed IDS

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. I am aware that plagiarism is considered an act of deception which can result in sanction in accordance with the examination regulations.

Münster, August 16, 2023

Jan Speckamp

I agree to have my thesis cross-checked with other texts to identify possible similarities and to having it stored in a database for this purpose.

I confirm that I have not submitted the following thesis in part or whole as an examination paper before.

Münster, August 16, 2023

Jan Speckamp