Anforderungsanalyse für ein verteiltes IDS und Entwicklung eines Prototypen auf Raspberry Pis

Bachelorarbeit

vorgelegt von:
Kai Oliver Großhanten

Matrikelnummer: 502824
Studiengang: B.Sc. Informatik

Thema gestellt von:
Prof. Dr. Anne Remke

Arbeit mit betreut durch:
Verena Menzel

Münster, 20. September 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

1 Einleitung

1.1 Motivation

Die Welt befindet sich im Wandel. Wie aktuell eindrucksvoll durch die Ukraine-Krise unter Beweis gestellt, ist in der heutigen Zeit Energie das Blut in den Adern der Gesellschaft. Kommt diese zum Erliegen, ist unsere gesamte moderne Lebensweise nicht mehr möglich. Für viele der Krisen, die die Welt in Atem halten, gibt es eine gemeinsame Lösung: Erneuerbare Energien.

Die erneuerbaren Energien haben u.a. den großen Vorteil, dass sie (bis auf wenige Ausnahmefälle) unabhängig von anderen Staaten genutzt werden können. Außerdem stoßen sie in der Regel über ihre gesamte Lebensdauer hinweg nur geringe Mengen an Treibhausgasen aus und bieten somit die Möglichkeit unseren aktuellen Lebensstil zumindest in Hinsicht auf den Energieverbrauch längerfristig abzusichern.

Doch auch die erneuerbaren Energien haben nicht nur Vorteile. Mit Ausnahme von Wasserkraft und Geothermie, haben insbesondere die verbreitetsten Formen - Wind- und Solarenergie - den Nachteil, dass diese stark vom Wetter und den Jahreszeiten beeinflusst sind. Im Winter sinkt die durchschnittliche erzeugte Leistung von Photovoltaikanlagen sehr stark ab, im Fall von Windkraftanlagen stellt eine Flaute ein Problem dar, was eine nicht zu unterschätzende Unberechenbarkeit ins Elektrizitätsnetz einbringt.[1, Kapitel 4.1.1] Diese Unberechenbarkeit steht dabei im Konflikt mit den Grundsätzen der Steuerung des Stromnetzes, da es das oberste Ziel dieser ist, die Netzfrequenz stabil zu halten. Um Stabilität herzustellen ist es notwendig zu jedem Zeitpunkt genau so viel Strom zu erzeugen, wie im selben Moment verbraucht wird.

Die Balance zwischen Erzeugung und Verbrauch zu gewährleisten ist Aufgabe der Netzbetreiber, die von ihren Leitständen aus jeden einzelnen Erzeuger steuern können. Traditionell war dies eine notfalls auch manuell schaffbare Aufgabe, da es wenige sehr große und träge Erzeuger (Kohle- und Atomkraftwerke) gab und bei erhöhtem Bedarf mit schnell reagierenden Gaskraftwerken ein Ausgleich gewährleistet werden konnte. Mit der Energiewende gibt es nun allerdings nicht mehr wenige Hundert bis Tausend Erzeuger, sondern Millionen von Einspeisepunkte, die alle unterschiedliche Eigenschaften aufweisen und jeder einzelne wetterabhängig über andere Produktionskapazitäten verfügt. Die fortschreitende Elektrifizierung belastet die Netze ebenfalls, wie zum Beispiel

der enorme Verbrauch, den ein modernes Elektroauto beim Laden verursacht und der besonders zu Stoßzeiten bedrohliche Ausmaße für die Stabilität des Netzes annehmen kann.

1.2 Forschungsgegenstand

Um das Stromnetz korrekt steuern zu können, müssen genaue und zuverlässige Informationen über den Zustand des Netzes zur Verfügung stehen. Wenn es einem potenziellen Angreifer gelingt die Sensordaten zu manipulieren, so kann dies zu Fehlentscheidungen bei der Steuerung führen, die verheerende Folgen nach sich ziehen können.

Um die Sicherheit der Steuerungssysteme zu erhöhen, wurde durch Justyna Chromik, Verena Menzel et. al. ein *Intrusion Detection System* (*IDS*)[1] entworfen, das unter Ausnutzung von physikalischen Gesetzen und einem verteilten, lokalen und hierarchischen Ansatz fehlerhafte Daten erkennen kann. Dieser den Kontext berücksichtigende Ansatz basiert darauf, dass im Netz verteilt sogenannte *Monitore* platziert werden, die die lokalen Sensordaten auf Plausibilität prüfen und bei Unstimmigkeiten Alarm schlagen. Untereinander kommunizieren diese Monitore die jeweiligen Sensordaten, sodass auch die *Border Regions* (*Grenzregionen*) zwischen einzelnen Netzabschnitten besonders gründlich geprüft werden können, da hier Sensordaten aus zwei unabhängigen Quellen zur Verfügung stehen.

Um dieses Konzept zu testen wurde ein ein größtenteils auf Python basierter Prototyp entwickelt. Da man zum Test des Ansatzes ein Stromnetz benötigt, ein echtes Stromnetz allerdings aus naheliegenden Gründen (Sicherheit, Finanzen, uvm.) nicht praktikabel ist, wurde ein *Testbed* entwickelt, das ein echtes Stromnetz simuliert. Dieses ist in der Lage die Topologie eines Stromnetzes aus Leitungen, Bussen, Verbrauchern, Erzeugern usw. abzubilden und zu jedem Simulationsschritt die “echte” Spannungs- und Stromstärke zu berechnen. Diese Daten werden dann über eine Schnittstelle zur Verfügung gestellt.

Das IDS besteht im Kern aus drei Komponenten. Es gibt zum einen den *Command and Control-Server* (*C2-Server*), welcher nur einmal existiert und als Zentraler Knotenpunkt anfangs die Verbindungen zwischen den einzelnen Monitoren herstellt. Des weiteren dient der C2-Server auch dazu die gefundenen Unstimmigkeiten (auch *Requirement Violations* genannt) zu protokollieren. Die anderen beiden Komponentenarten sind die *Lokal- und Nachbarschaftsmonitore* (*LM* und *NM*). LMs existieren jeweils einmal je Netzabschnitt und beziehen direkt lokal die Daten von den Sensoren. Diese Daten werden dann durch den Monitor anhand einer Reihe von Regeln auf Plausibilität ge-

prüft und eventuelle Verstöße an den C2-Server weitergeleitet. Die NMs haben die Aufgabe Grenzregionen zwischen Netzabschnitten zu überprüfen, wozu sie die Daten von zwei LMs benötigen. Dazu stellen die NMs eine direkte Verbindung zu den jeweils zuständigen LMs her und beziehen direkt von diesen die Daten.

Bisher sind die einzelnen Komponenten des IDS in *Docker-Container* isoliert und so logisch voneinander getrennt auf einem Desktop-Computer ausgeführt worden. Es wurde bis zum Beginn dieser Arbeit noch nicht getestet, inwiefern das System auf Hardware mit beschränkter Leistung und in einem echt verteilten Kontext arbeiten kann.

1.3 Forschungsfragen

Folgende Forschungsfragen werden in der vorliegenden Arbeit thematisiert:

1. *Welche technischen Anforderungen in Bezug auf Hard- und Software hat das System, wenn man es in der Realität einsetzen möchte?*
2. *Welche Änderungen sind notwendig, um das System auf einem Testaufbau aus Raspberry Pis lauffähig zu machen und welche Mindestanforderungen im Bezug auf Anzahl der Komponenten und Topologie des simulierten Netzes muss ein solcher Testaufbau erfüllen, damit es sinnvolle Ergebnisse erzielt?*
3. *Inwiefern beeinflusst die tatsächlich verteilte Ausführung auf getrennter Hardware die Funktionalität des Systems?*

1.4 Quellcode

Der Quellcode dieser Arbeit, die mit *plant-uml* erzeugten Diagramme und Protokolle der Leistungsmessung sind im Github-Repository `https://github.com/PhantomPhr3ak/bachelorthesis-distributed-ids` zu finden.

1.5 Aufbau der Arbeit

Der Entstehungsprozess des IDS Prototypen bis zum Beginn dieser Arbeit wird konzeptionell im Kapitel 2 geschildert. Eine technischere Darstellung des Systems wird dann in Kapitel 3 gegeben. Dabei wird der Fokus vor allem auf denjenigen Aspekten liegen, die für das Verständnis der späteren Kapitel relevant sind. Das Kapitel 4 widmet sich der Frage, unter welchen Bedingungen das IDS in der Realität eingesetzt werden könnte und welche Anforderungen erfüllt werden müssten, um einen sicheren Betrieb zu gewährleisten. Ein Schritt in Richtung eines realen Einsatzes wird dann vollzogen, indem das IDS auf einem Testaufbau bestehend aus mehreren Raspberry Pis getestet wird. Dies wird in Kapitel 5 beschrieben und schließlich werden in Kapitel 6 noch einmal die wichtigsten Erkenntnisse zusammengefasst.

2 Related Works

In diesem Kapitel wird ein Überblick über den Entstehungsprozess des Systems bis zum Beginn dieser Bachelorarbeit gegeben. Nachfolgend wird jeweils vorgestellt, welche Neuerungen mit der jeweiligen Arbeitseinheit ins System eingebracht wurden. Dabei werden vor allem konzeptionelle Aspekte geschildert. Die technische Umsetzung des aktuellen Systems wird im Detail in Kapitel 3 geschildert.

Zuerst wird die Entwicklung des ursprünglichen Proof of Concept durch Justyna Chromik et al. in Abschnitt 2.1 begonnen. Darauf aufbauend wurde durch Verena Menzel et al. das Konzept zu einem Intrusion Detection System erweitert (Abschnitt 2.2) und im Rahmen des Projektseminars um eine sichere Kommunikation und ein Deployment ergänzt (Abschnitt 2.3).

2.1 Process-aware SCADA traffic monitoring: A local approach

Im Rahmen ihrer PhD Thesis[1] wurde durch Justyna Chromik ein rein lokaler Ansatz zur Erkennung unsicherer Systemzustände in Stromnetzen entwickelt. Insbesondere für diese Bachelorarbeit relevant sind die Kapitel, die das Testbed betreffen, sowie das Prozess-Modell mit dem Regelkatalog, der Anforderungen an sichere Systemzustände eines Stromnetzes formuliert.

2.1.1 Konzept

Bei der Steuerung von industriellen Anlagen werden sogenannte *Supervisory Control and Data Acqusition* (*SCADA*) Systeme eingesetzt. Diese werden genutzt um Daten über Sensoren zu beschaffen, dann diese zentral auf einem Server zu sammeln, basierend darauf Entscheidungen zu treffen und diese in Form von Befehlen an Aktuatoren umzusetzen.

Das gesamte zu steuernde Netz wird in einzelne Teilnetze aufgeteilt. In jedem Teilnetz wird eine *Remote Terminal Unit* (*RTU*) platziert, was man ins Deutsche in etwa als Fernbedienungsterminal übersetzen kann. Diese kommuniziert direkt mit den Sensoren und Aktuatoren in ihrem Abschnitt, welche ihrerseits meist einfache Elektronische Bauteile ohne Internetzugang sind. Die RTU dient als Schnittstelle für den zentralen Server, um aus der Ferne Daten

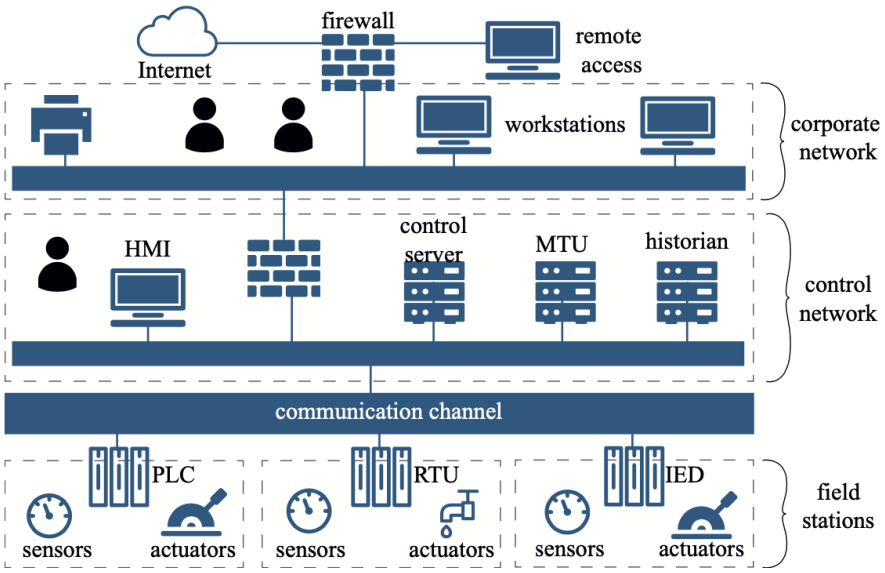

Abbildung 2.1: Grundlegende Topologie eines SCADA Systems.[1, Figure 2.4]

zu sammeln und Einfluss auf den entsprechenden Netzabschnitt zu nehmen.

Das Problem solcher Systeme liegt häufig darin, dass sie unter der Annahme entworfen wurden, dass die Kommunikation innerhalb der SCADA-Netze, also vor allem zwischen den RTUs und dem zentralen SCADA-Server getrennt von öffentlichen Netzen und bisweilen sogar über proprietäre Protokolle stattfindet. Dies hat dazu geführt, dass viele Betreiber von SCADA-Systemen historisch das Paradigma *security through obscurity* verfolgt haben. Diese proprietären Standards verwendeten dabei innerhalb ihres Netzes keine Authentifizierung und Verschlüsselung zur Absicherung der Verbindung. Daraus resultiert eine sehr hohe Anfälligkeit für Cyberangriffe aller Art, die eine große Gefahr für das gesamte SCADA System mitsamt aller dazugehörigen Infrastruktur darstellen.

Das Ziel von Chromik et al. war es eine Methode zu entwickeln, auch in SCADA Systemen in kritischen Infrastrukturen und insbesondere in Stromnetzen, fehlerhafte oder manipulierte Daten identifizieren zu können, da auch hier zuvor beschriebene Sicherheitsmängel vorkommen. Konkret sollen Angriffe erkannt werden, bei denen ein Angreifer die Daten zwischen einer RTU und dem SCADA-Server manipuliert. Dazu wird bei der Überwachung des Datenverkehrs die Tatsache genutzt, dass ein Stromnetz zum einen den Gesetzen der Physik folgt und man somit prüfen kann, ob die von der RTU stammenden Sensordaten plausibel sind. Zum anderen gibt es zusätzlich Sicherheitsanforderungen, die durch geplante Grenzwerte bei der Konstruktion entstehen (bspw. maximaler Stromfluss auf einer Leitung). Auch diese sollten eingehalten werden, um Schäden und übermäßige Abnutzungserscheinungen zu vermeiden.[5]

Diese Anforderungen wurden unter Zuhilfenahme eines Modells eines Stromnetzes ausformuliert und zu einem Regelkatalog zusammengestellt. Einen Auszug aus diesen Regeln wird in Unterabschnitt 2.2.1 dargestellt. Unter Verwendung eben dieser Regeln wurde auch ein *Self-aware Monitor* (*SAM*) implementiert. Der SAM generiert automatisch zur Teilnetz Topologie passende Regeln und ist in der Lage diese in Echtzeit anhand der Daten einer RTU zu prüfen.[1]

Für eine reale Umsetzung des Systems ist angedacht die Software des SAM auf separater Hardware auszuführen und jeweils eine solche Einheit an jeder RTU zu platzieren.

2.1.2 Simulator

Um den SAM testen zu können, benötigt man eine Datengrundlage. Da es aus Gründen des Datenschutzes, der Ausfallsicherheit und der Kosten nicht praktikabel ist ein solches System sofort an einem echten Elektrizitätsnetz zu testen, war es ein weiterer Bestandteil der Arbeit von Chromik et al., einen Simulator zu entwickeln, der es ermöglicht ein Stromnetz und die dazugehörigen RTUs zu simulieren.[6][7]

Die Kernanforderung an das Testbed besteht darin, ein reales Elektrizitätsnetz mit all seinen Komponenten in einer angemessenen Komplexität zu simulieren. Das heißt, dass man die wesentlichen Komponenten eines Stromnetzes abbildet, ohne dabei zu viele Details konfigurieren zu müssen und dabei trotzdem eine ausreichend präzise Simulation erhält.

Um dies zu erreichen wurde durch Chromik et al. das Testbed auf Basis der Co-Simulationsumgebung *Mosaik* entwickelt. Mosaik ermöglicht es eine Vielzahl von Simulatoren zu synchronisieren.

Das simulierte Netz basiert auf einer *Bus-Branch-Architektur*. Dabei gibt es einen *Bus*, der über *Branches* (*Leitungen*) mit anderen Bussen oder Simulatoren wie bspw. einem Haushalt, der Energie verbraucht, oder einer Photovoltaik Anlage, die Energie einspeist, verbunden wird. Auf Branches können *Switches* (*Schalter*) platziert werden, die je nach Zustand den Stromfluss unterbrechen. Die Komponente, die es uns ermöglicht Daten zu erfassen, sind die *Meter*, also Sensoren, die den aktuellen Strom und die Spannung an ihrer Position messen. Auch Meter werden, wie die Switches, auf Branches platziert.

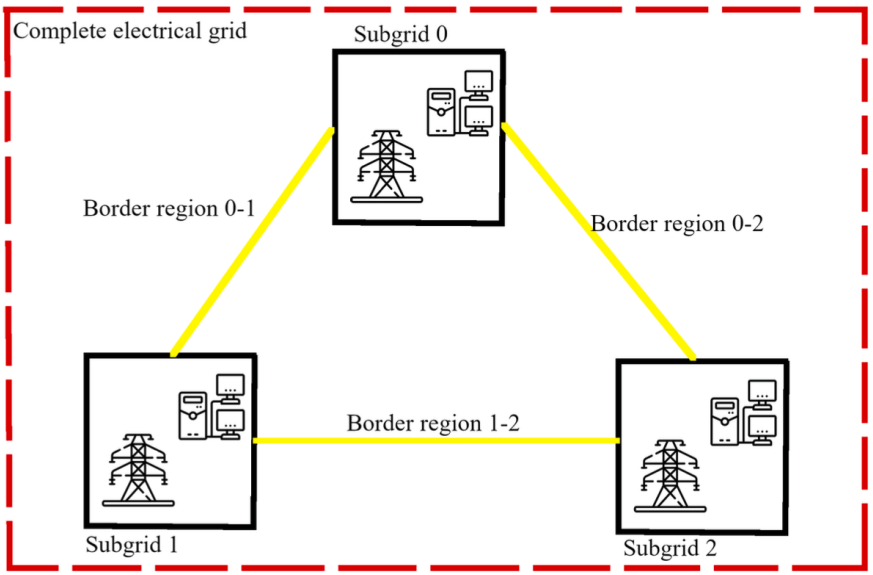

Abbildung 2.2: Schematische Darstellung eines gesamten Netzes (roter Kasten), das drei Teilnetze (subgrids, schwarze Kästen) beeinhaltet. Zwischen den Teilnetzen sind die Grenzregionen als gelbe Kanten eingezeichnet.[2, Figure 3.1]

2.2 A hierachical approach to monitoring SCADA networks

Die Masterarbeit von Verena Menzel[2] hat aufbauend auf das Testbed, den SAM und die Anforderungen, die durch Chromik et al. formuliert wurden, einen Prototypen des IDS entwickelt, dass als Neuerung nun über sogenannte Nachbarschaftsmonitore verfügt. Diese sind für die Überwachung von Border Regions zuständig und beziehen ihre Daten jeweils von zwei Lokalmonitoren.

2.2.1 Scopes und Anforderungen

Während Chromik et al. in ihrer Arbeit einen rein lokalen Ansatz untersucht hat, d.h. ein Monitor benutzt stets nur die Daten einer ihm zugeordneten RTU, wird nun auch berücksichtigt, dass es Leitungen gibt, die zwischen den Teilnetzen verlaufen. Da Sensoren und Aktuatoren auf einer Leitung dabei einer von zwei unterschiedlichen RTUs zugeordnet sein können, ist es notwendig die Informationen von beiden Quellen zu beziehen. Dafür werden nun drei *Scopes* unterschieden. Nachfolgend werden jeweils die unterschiedlichen Scopes geschildert und diejenigen Regeln aufgelistet, die (mit Ausnahme von REQ13G) tatsächlich implementiert wurden.

Lokalscope

Das lokale Scope umfasst alle Anforderungen, die ausschließlich mit den Daten einer RTU geprüft werden können. Anders ausgedrückt, können also lediglich Anforderungen an Komponenten geprüft werden, die innerhalb eines einzelnen Teilnetzes verlaufen und somit nicht auf einem Branch liegen, der Busse in unterschiedlichen Teilnetzen verbindet. Die lokalen Anforderungen sind die folgenden:

- **REQ1** - Die Summe aller zufließenden Ströme an einem Bus muss gleich der Summe der ausgehenden Ströme eines Bus sein. (1. Kirchhoffsche Regel)
- **REQ2** - Alle gemessenen Spannungen an einem Bus sind gleich.
- **REQ3** - Es fließt kein Strom auf einer Leitung mit mindestens einem offenen Schalter.
- **REQ4** - Gemessene Spannung und Strom auf einem Branch sind an allen Messpunkten gleich.
- **REQ7** - Der Grenzwert für den maximal zulässigen Strom auf einem Branch wird an allen Messpunkten eingehalten.
- **REQ8** - Der Grenzwert für die maximal zulässige Spannung auf einem Branch wird an allen Messpunkten eingehalten.

Nachbarschaftsscope

Das *Nachbarschaftsscope* umfasst alle Anforderungen, die zur Überprüfung die Daten von genau zwei benachbarten RTUs benötigen. Dies ist überall dort der Fall, wo ein Branch von einem Teilnetz in ein anderes verläuft. Alle Branches, die von einem Teilnetz in ein anders verlaufen liegen gemeinsam in einer Border Region. Anforderungen die ausschließlich auf Nachbarschaftsebene geprüft werden können sind:

- **REQ3** - Es fließt kein Strom auf einer Leitung mit mindestens einem offenen Schalter.
- **REQ4** - Gemessene Spannung und Strom auf einem Branch sind an allen Messpunkten gleich.

Globalscope

Theoretisch existiert auch das *Globalscope*, also eine Ebene deren Anforderungen lediglich unter Verwendung von Daten aller RTUs und ihrer Sensoren geprüft werden können. Konkret wäre das die Anforderung:

- **REQ13G** Die gesamte verbrauchte Leistung entspricht der gesamten erzeugten Leistung.

Die globale Anforderung ist allerdings nicht in diesem System implementiert worden, da man hier mit zu großen Ungenauigkeiten bezüglich der Messzeitpunkte rechnen muss und somit eine Evaluation nicht über die typischen Sensoren geschehen würde. Vermutlich würde man in einem praktischen Einsatzszenario dies über eine Überwachung der Netzfrequenz realisieren.[8]

2.2.2 Testbed

Um den neuen Ansatz zu testen, wurde wieder ein Simulator benötigt, der diesmal auch in der Lage sein musste mehrere RTUs zu simulieren. Speziell sollten die zwei RTUs benachbarte Netzabschnitte aus dem gleichen übergeordneten Netz steuern. Dies wurde aufbauend auf den Simulator von Chromik et al. realisiert, indem nun zwei RTU-Simulatoren konfiguriert wurden.

2.3 Projektseminar - IT-Sicherheit in Stromnetzen

Beim Projektseminar hat sich Gruppe 3, bestehend aus Lisa Angold, Gelieza Kötterheinrich, Linus Lehbrink, Jan Speckamp und mir selbst, damit beschäftigt den IDS Prototypen von Menzel et al. um eine sichere und effiziente Kommunikationsinfrastruktur zu erweitern. Zuvor Bestand die Kommunikation zwischen den Lokal- und Nachbarschaftsmonitoren lediglich aus direkten Python-Funktionsaufrufen (siehe Abbildung 2.3). Das wohl größte Problem besteht dabei darin, dass die Implementierung von Menzel et al. nur in einer singulären Laufzeitumgebung ausführbar ist, was zwar eine sinnvolle Vereinfachung für ein erstes erproben des Konzepts darstellt, allerdings problematisch ist, wenn das System doch rein konzeptionell verteilt ausgeführt werden soll.[3]

Die Inter-Monitor-Kommunikation musste demnach auf die Verwendung eines echten Kommunikationsprotokolls umgestellt werden. Nach einem Vergleich mit anderen Protokollen, haben wir uns dabei für das *OPC Unified Architecture* (*OPC-UA*) Protokoll entschieden. Dieses sehr umfangreiche Protokoll wurde speziell für IoT-Anwendungen im industriellen Kontext entwickelt und ist konzipiert um den Datenaustausch zwischen industriellen Maschinen zu ermöglichen. Der Standard ist ausgesprochen flexibel gehalten und umfasst zahlreiche Module, um verschiedenste Anforderungen abzudecken. Der eigentliche Kommunikationsweg ist bei OPC-UA nicht vorgegeben, was ausgesprochene Flexibilität gewährleistet. Es wird sowohl Push-, als auch Pull-basierte Kommunikation unterstützt und auch eine Funktionalität in der Art eines *Remote Procedure Calls* (*RPC*) ist in dem Standard enthalten. Darüber hinaus

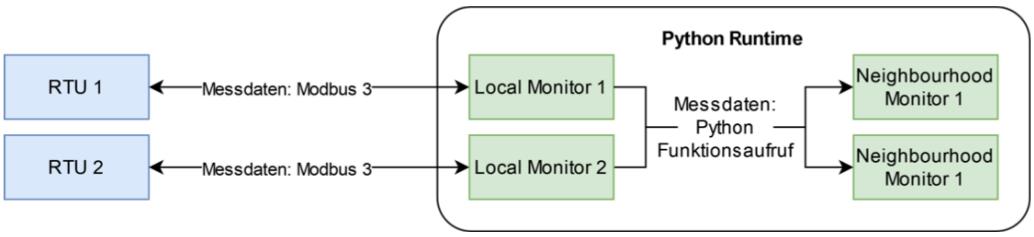

Abbildung 2.3: Übersicht Kommunikationsinfrastuktur in der Implementierung von Menzel [3, Abbildung 7]

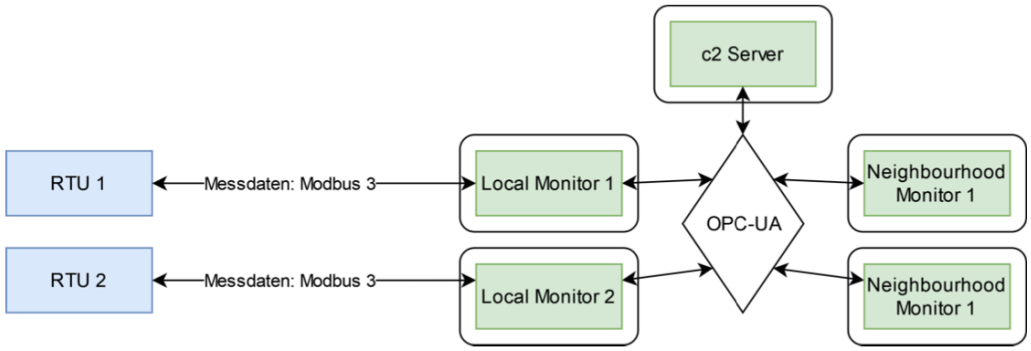

Abbildung 2.4: Übersicht Systemarchitektur nach dem Projektseminar [3, Abbildung 8]

wird nativ Authentifizierung von Komponenten und Verschlüsselung der Kommunikation unterstützt.[3][9]

Es existiert auch eine Implementierung des Standards für Python unter dem Namen *opcua-asyncio*[10], unter deren Verwendung wir die einzelnen Monitore von einander abkapseln konnten. Durch die so erreichte Trennung der Monitore wurde eine zusätzliche Komponente zur Überwachung und Steuerung des IDS benötigt, weshalb wir den Command and Control Server (C2-Server) eingeführt haben. Wie in Abbildung 2.4 zu sehen ist, befindet sich nun jede Komponente des Systems in einer eigenen Laufzeitumgebung. Desweiteren haben wir auch eine Visualisierung entwickelt, bei der Regelverstöße an der Topologie des Netzes veranschaulicht werden (Abbildung 2.5).

Um das komplizierte Ausführen des Systems unabhängiger von der Plattform zu ermöglichen, haben wir ein Deployment auf Basis von *docker compose* entworfen. Dieses wird im Detail in Abschnitt 3.3 vorgestellt.

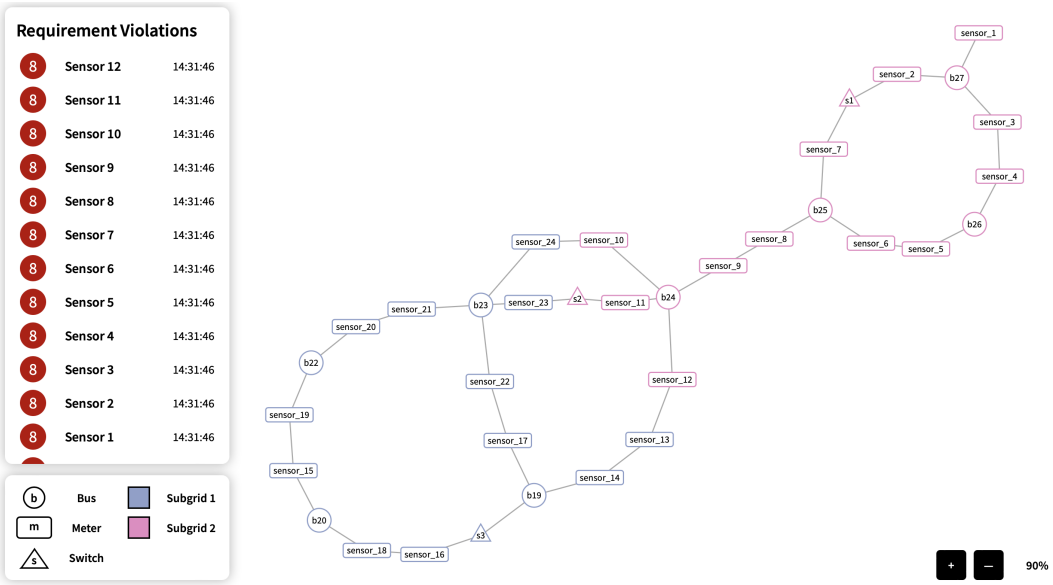

Abbildung 2.5: Visualisierung des IDS

3 System

In diesem Kapitel wird ein Überblick über die technische Seite des vorhandenen Systems gegeben. In Abschnitt 3.1 wird das Testbed beschreiben, danach wird die Implementierung des IDS in Abschnitt 3.2 vorgestellt und das Deployment in Abschnitt 3.3 dargestellt.

3.1 Testbed

Nachdem die Funktion des Testbeds und die Struktur des simulierten Netzes bereits zuvor in Unterabschnitt 2.1.2 beschrieben wurde, wird nun auf einige Details der Implementierung und Konfiguration mit Mosaik eingegangen.

3.1.1 Mosaik Architektur

Bei Mosaik[11] handelt es sich um eine Co-Simulationsumgebung, d.h. es wird eine Vielzahl unterschiedlicher Simulatoren synchronisiert. Konkret sind das die folgenden Simulatoren, deren Zusammenspiel in Abbildung 3.1 dargestellt wird:

- *Mosaik-HouseholdSim* simuliert Haushalte basierend auf Lastprofilen.
- *Mosaik-Py-Power* führt die Lastflussberechnungen durch.
- *Mosaik-RTUSim* stellt Daten seines Netzabschnitts über einen Modbus TCP Server bereit.
- *Mosaik-Web* stellt eine Visualisierung der Simulation auf dem *localhost* Port 9000 bereit. Man kann einzelne Knoten des Netzes anwählen und bekommt dann abhängig vom Typ einen Graphen mit Spannung bzw. verbrauchter Leistung angezeigt.(Abbildung 3.2)
- *Mosaik-CSV* kann benutzt werden um CSV Dateien mit aufgezeichneten Daten in den Simulator einzuspeisen.
- *Mosaik-HDF5* kann benutzt werden um HDF5-Datenbanken zu interagieren.

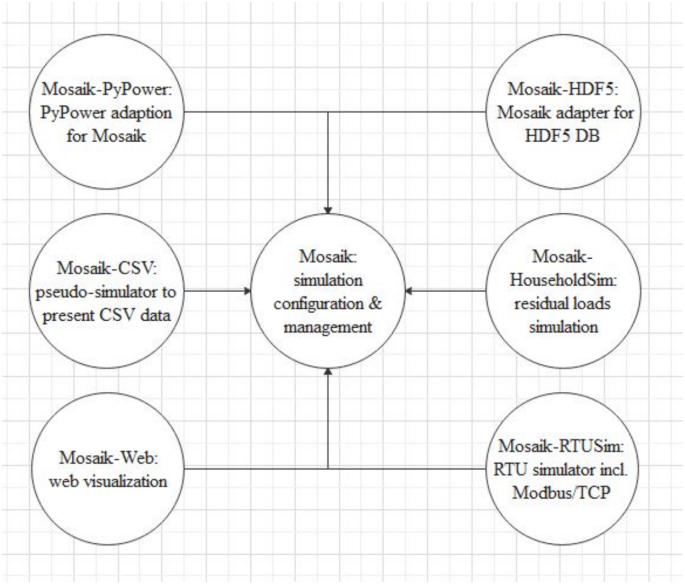

Abbildung 3.1: Mosaik Simulatoren, die im Testbed verwendet werden.[2, Figure 4.1]

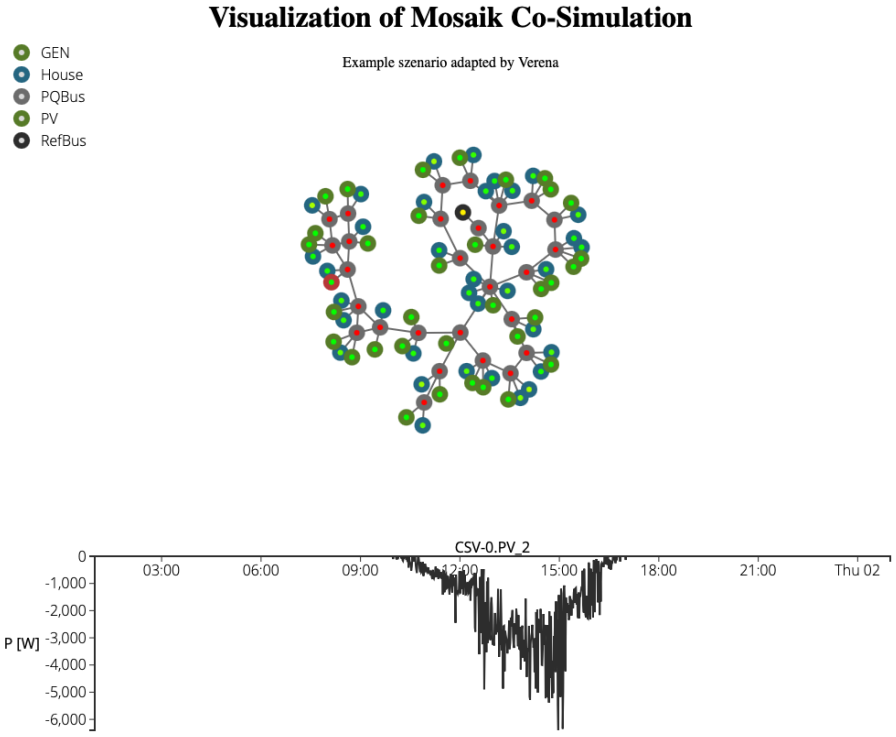

Abbildung 3.2: Web-Visualisierung von Mosaik. Oben ist die Topologie des simulierten Netzes zu sehen. Unten wird die eingespeiste Leistung einer Photovoltaik-Anlage über den Verlauf eines Tages hinweg dargestellt.

Alternative zum Testbed

Der Prototyp verfügt über 2 unterschiedliche Optionen:

- **Testbed** Dies ist der Standard-Modus. Per Mosaik werden live Daten erzeugt und über die RTU-Simulatoren ausgegeben.
- **CSV Replay** wurde durch mich im Rahmen dieser Arbeit hinzugefügt. Es handelt sich um ein Python-Skript, das dazu genutzt wird, zuvor aufgezeichnete Daten des Testbeds aus CSV-Dateien auszulesen und per Modbus auszugeben, genau wie es die RTU-Simulatoren im Livebetrieb tun. Weiteres hierzu findet sich in Kapitel 5.

3.1.2 Konfiguration

Generelle Konfiguration

In der Datei *ids/deployment/testbed/data/config.cfg* werden die grundlegenden Parameter der Simulation festgelegt. Es enthält Angaben zum Simulationzeitraum und die Dateipfade aller anderen wichtigen Konfigurationsdateien.

Gesamte Netztopologie

In der Datei *ids/deployment/testbed/config_files/demo_mv_grid.json* wird die Topologie des Netzes definiert. Nachfolgend die Eigenschaften, die konfiguriert werden können:

- **branch:** Name (String), Start (Bus), End (Bus), Branchtype, Länge (km), Status (true/false)
- **bus:** Name (String), Node type (REF oder PQ), Grund Spannung (kV)
- **branchtype:** Name (String), Längenspezifischer Widerstand (Ω/km), X (Ω/km), C (nF/km), max. Strom (A)
- **trafo:** Name (String), Primärseite (Bus), Sekundärseite (Bus), Trafotype, Status (true/false), tap
- **trafotype:** Siehe [12]

RTUs

Die RTU-Simulatoren werden in den Dateien *ids/deployment/testbed/data/-config_files/new_rtu_*.xml* konfiguriert. Da die RTU eine Modbus Schnittstelle zur Datenübermittlung anbietet, wird hier das Registersystem von Modbus definiert. Dabei werden unmittelbar einzelne Bytes im Speicher mit einem Index adressiert. Es wird dabei zwischen *Coils* und *Holding registers* unterschieden. Ein Coil speichert einen Boolean Wert und die Indizes werden ohne Sprünge

hochgezählt. Die Holding Registers hingegen besitzen eine Größe von 4 Byte, somit wird der Index hier für jedes weitere Holding Register um 4 erhöht. Ebenfalls wichtig ist, dass in dieser Datei die IP Adresse und der TCP-Port festgelegt werden, auf denen die RTU per Modbus erreichbar ist.

3.1.3 Bekannte Probleme

In der Vergangenheit sind bereits Ungenauigkeiten in den vom Testbed berechneten Daten aufgefallen. Diese entstanden durch Verzögerungen bei der Simulation und haben zu Falschalarmen geführt.[13][14]

3.2 Intrusion Detection System

Bei einem Intrusion Detection System handelt es sich um ein System, das das Ziel hat, eine Manipulation von Außen zu erkennen und in solchen Fällen Alarm zu schlagen.[15] Im Falle unseres Systems, soll es erkennen, wenn Sensordaten aus einem Stromnetz, dessen Topologie uns bekannt ist, nicht plausibel sind, d.h. entweder Manipulationen stattfinden oder Sensoren defekt sind.

In Unterabschnitt 3.2.1 werden einzelne Aufgaben der unterschiedlichen Komponenten ausformuliert und darauf folgend die Implementierung in Unterabschnitt 3.2.3 anhand von Beispielszenarien erläutert.

3.2.1 Komponenten

Das IDS setzt sich im Kern aus einem C2-Server und mehreren Lokal- und Nachbarschaftsmonitoren zusammen. Als zusätzlicher optionaler Bestandteil existiert auch eine Visualisierung.

Command and Control Server (C2)

Die Zentrale Komponente des IDS ist der Command und Control Server, der auch C2-Server genannt wird. Seine Aufgabe umfassen:

- Herstellen einer initialen Verbindung mit allen Monitoren
- Zuordnung von Lokalmonitoren zu Nachbarschaftsmonitoren
- Verwaltung aller aktiven Lokal- und Nachbarschaftsmonitore
- Protokollierung aller gefundenen Regelverstöße
- Überwachung des Zustandes der Monitore anhand eines Heartbeats
- Überwachung von CPU- und RAM-Auslastung der Monitore

- Bereitstellen eines Websockets mit gefundenen Regelverstößen für die Visualisierung

Lokalmonitor

Der Lokalmonitor ist für jeweils eine Steuerzone des SCADA-Systems, also eine RTU, zuständig. Er bezieht seine Messdaten unmittelbar von dieser RTU und überprüft alle Regeln mit lokalem Gültigkeitsbereich. Die Messwerte werden auch an alle zugeordneten Nachbarschaftsmonitore weitergeleitet. Daraus ergeben sich folgende Aufgaben für den LM:

- Beim C2-Server anmelden
- Zugeordnete Nachbarschaftsmonitore verwalten
- Daten per Modbus von der RTUs lesen
- Nachbarschaftsmonitor über Vorhandensein neuer Daten informieren
- Anforderungen prüfen
- Verstöße gegen Anforderungen an den C2-Server melden
- CPU- und RAM-Auslastung an C2-Server melden

Nachbarschaftsmonitor

Um die Anforderungen an eine Border Region zu prüfen empfängt der Nachbarschaftsmonitor die Daten von zwei RTUs. Dabei erfüllt ein NM folgende Aufgaben:

- Beim C2-Server anmelden
- Bei Lokalmonitoren anmelden
- Anforderungen prüfen
- Verstöße gegen Anforderungen an den C2-Server melden
- CPU- und RAM-Auslastung an C2-Server melden

Visualisierung des IDS

Die Visualisierung hat lediglich die Funktion eine gute Übersicht über das System zu liefern. Dazu benötigt sie zum einen die Topologie des Netzes und zum anderen die protokollierten Verstöße. Letztere müssen regelmäßig aktualisiert werden.

- Topologie des überwachten Netzes darstellen
- Abfragen der Anforderungsverstöße beim C2-Server
- Projektion der Verstöße auf die Darstellung des Netzes

3.2.2 Implementierung

Bei der Implementierung wurde ein Großteil des Prototypen von Menzel et al. überarbeitet. Nachdem die Wahl des Protokolls auf OPC-UA gefallen war, wurde unter der Verwendung der Open-Source-Python-Implementierung opcua-asyncio das gesamte IDS überarbeitet und neu strukturiert.

Die Implementierung des IDS ist im Verzeichnis *ids/implementation* zu finden. Monitore und C2-Server wurden jeweils in mehrere Bestandteile aufgespalten:

- Das Hauptprogramm, das durch das Deployment aufgerufen wird (z.B. *c2_server.py*) lädt die Konfiguration für die zu startende Komponente aus den Umgebungsvariablen aus. Diese Konfiguration wird dann an den Server übergeben, wenn dieser asynchron gestartet wird.

- Der OPC-UA-Bestandteil (z.B. *opc_c2server.py*) startet jeweils einen OPC-UA-Server. Diese Dateien enthalten zusätzlich auch *EventListener*, die auf Events aus anderen Teilen des IDS reagieren.

3.2.3 Kommunikation

In diesem Abschnitt wird anhand von Beispielszenarien das Zusammenspiel der Komponenten unter Verwendung der unterschiedlichen Protokolle erläutert. Da eine vollumfängliche Darstellung der einzelnen Protokolle für das Verständnis dieser Arbeit nicht notwendig ist, wird lediglich auf einige ausgewählte Aspekte eingegangen.

Start eines OPC-UA Servers

In Abbildung 3.3 ist der Startprozess des OPC-UA-Servers im Lokalmonitor in zusammengefasster Form dargestellt. Nachdem der Server erstellt wurde, wird gleich zu Beginn ein *UserManager* mit den Zertifikaten für C2-Server und Nachbarschaftsmonitor konfiguriert. Darauf folgt die Festlegung der Netzwerkeinstellungen des Servers, also der IP-Adresse und des Ports. OPC-UA kommuniziert über TCP.

OPC-UA bietet zwei verschiedene Modelle der Kommunikation an. Zum einen die *Client-Server Architektur*, bei der ein Server modellierte Daten zur Verfügung stellt und diese dann von Clients abgerufen werden können. Die andere Option ist das *Publisher-Subscriber-Modell*, bei dem Publisher Nachrichten an eine Middleware senden, die ihrerseits die Daten an beliebig viele Subscriber weiterleiten. Dabei kennt der Publisher selbst nicht die Empfänger. Dieser Modus ist speziell für Many-To-Many Anwendungen optimiert.[16]

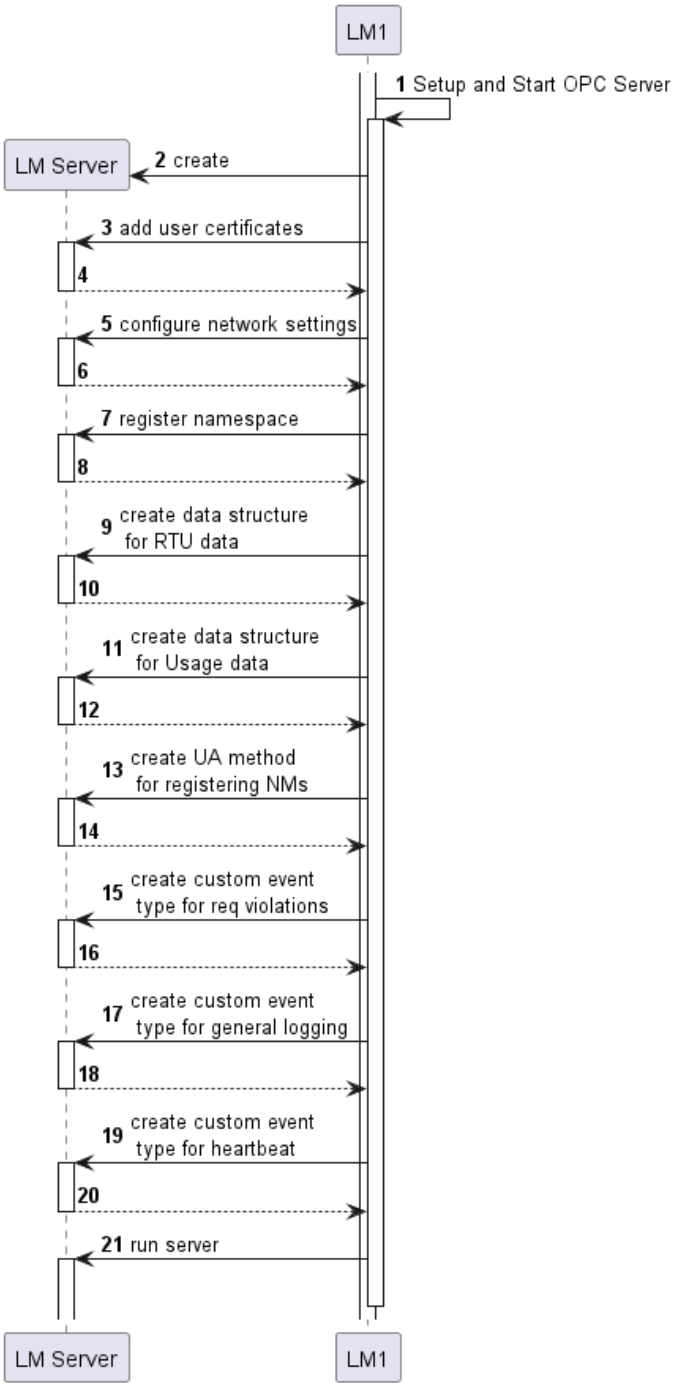

Abbildung 3.3: Start eines OPC-UA-Servers im Lokalmonitor. Der Lokalmonitor beeinhaltet einen OPC-UA-Server, der genutzt wird um Messdaten zu speichern und für Nachbarschaftsmonitore verfügbar zu machen. Die Grafik zeigt den Prozess der Konfiguration eines solchen Servers.

In Schritt 7 wird der *Namespace* des Servers registriert. Der Namespace eines OPC-UA-Servers dient der eindeutigen Identifikation des Ursprungs von Daten und ist in der Regel eine URL. OPC-UA unterstützt eine komplexe Form der Datenmodellierung, die darauf ausgelegt ist unterschiedlichste Arten von Informationen modellieren zu können.

Informationen werden in *Knoten* mit *Attributen* organsiert. Knoten gehören einer bestimmten *Knotenklasse* an, die spezifiziert welche Art von Information der Knoten beinhaltet. Die Knoten sind entweder *Typdeklarationen* oder Instanzen von Typen. Alle Knotenklassen, die Typdeklarationen definieren, werden in einer Vererbungshierarchie angeordnet.[17][18]

Die wichtigsten Knotenklassen sind:

- **ObjectType** - Typ eines Object-Knotens. Unterstützt hierachische Subtypen.
- **VariableType** - Typ eines Variable-Knotens. Spezifiziert Variable und definiert Vorlage zur Instanzierung von Variablen. (bspw. ein Satz Messwerte der aus einem Wert für Spannung und einem Wert für Stromstärke besteht)
- **DataType** - Typ eines Data-Knotens. Ist Attribut einer Variable und wird genutzt um die Art der Daten zu beschreiben. (also bspw. float)
- **Object** - Instanz eines ObjectType.
- **Variable** - Instanz eines VariableType.
- **Method** - Leichtgewichtige Funktion, die von einem Client aufgerufen werden kann, dann auf dem Server ausgeführt wird und ein Ergebnis an den Client zurückliefert.

In Schritt 9 bis 11 werden Informationsmodelle für Messwerte von Metern und Leistungsmerkmale (RAM-/CPU-Auslastung) dem Server hinzugefügt. Schritt 13 fügt eine Methode zur Registrierung von NMs hinzu. Es werden auch Event Typen für einen Heartbeat (Schritt 19) und generelles Logging (Schritt 17) angelegt.

Verbindung zwischen Client und Server

Damit der Lokalmonitor mit dem C2-Server interagieren kann, muss er einen OPC-UA-Client verwenden. Das Konfigurieren des Clients und die Anmeldung beim C2-Server wird in Abbildung 3.4 dargestellt. Der Client muss die korrekten Sicherheitseinstellungen des Servers kennen und über ein gültiges Zertifikat verfügen. Diese werden in Schritt 3 konfiguriert.

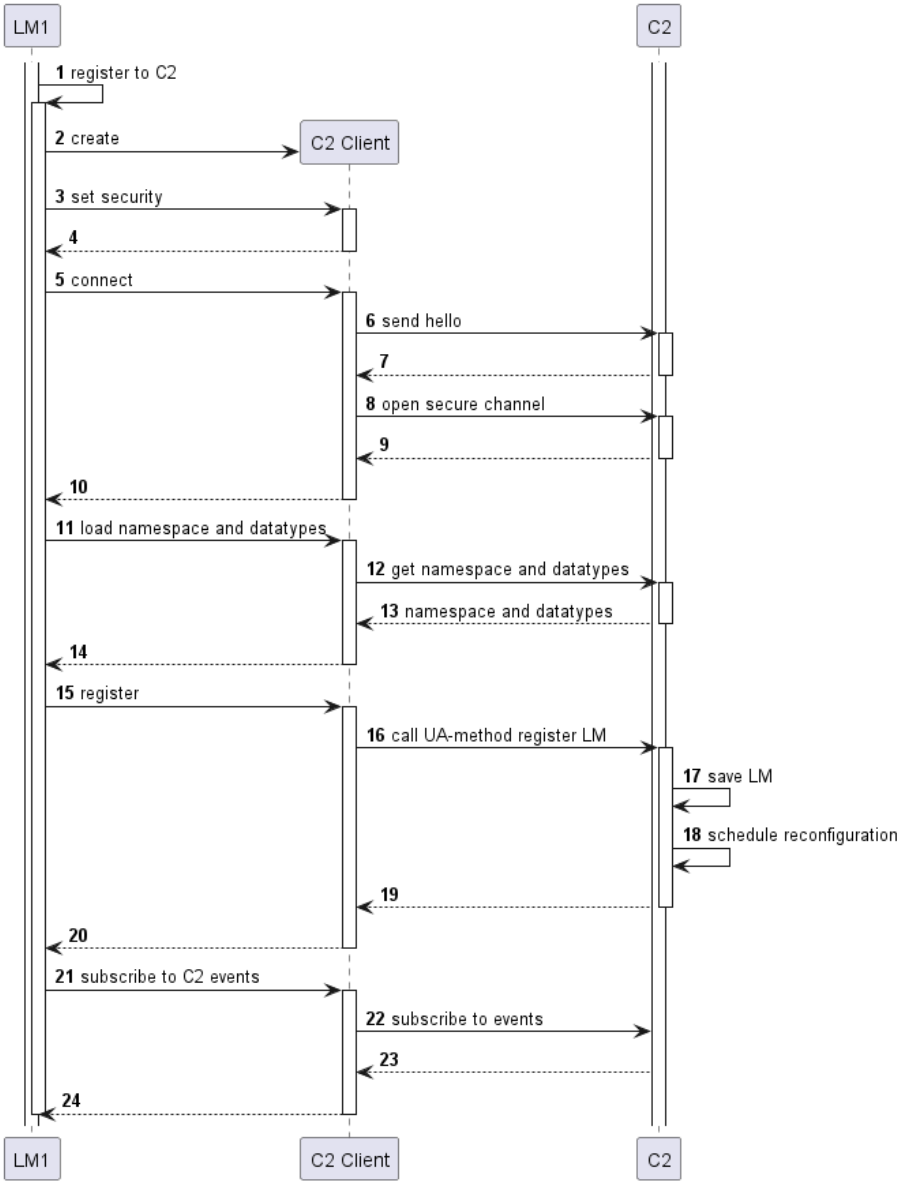

Abbildung 3.4: Herstellen der Verbindung zwischen Client im Lokalmonitor und OPC-UA-Server im C2-Server. Der C2-Client befindet sich im Lokalmonitor und stellt die Schnittstelle für die Kommunikation mit einem OPC-UA-Server dar. Es wird der initale Verbindungsaufbau zwischen Lokalmonitor und C2-Server veranschaulicht.

Anschließend wird die Verbindung zum C2-Server hergestellt (Schritt 5 bis 10). Es wird zuerst ein „Hello" gesendet. Wenn dies nicht fehlschlägt, wird ein sicherer Kanal aufgebaut, über den die weitere Kommunikation zwischen Client und Server abgewickelt wird.

Der Client fordert dann den Namespace und das Informationsmodell des C2-Servers an (Schritt 11-14). In diesem Modell wird dann die Methode *registerLM* aufgerufen. Hierbei handelt es sich um eine Funktionalität ähnlich zu einem Remote Procedure Call[19]. Die Funktion speichert im C2-Server die Referenz auf den Lokalmonitor. Darüber hinaus wird der Status des C2-Servers auf *should reconfigure* gesetzt. Jedes Mal, wenn ein neuer Lokalmonitor sich registriert, können auch neue Grenzregionen zwischen den Teilnetzen der Lokalmonitore entstehen und somit auch weitere Nachbarschaftsmonitore benötigt werden, sodass der C2-Server einmal alle Nachbarschaftsmonitore neu konfiguriert.

Zuletzt wird noch ein *EventHandler* für die Events des C2-Servers erstellt. Dieser hört im Falle des Lokalmonitors lediglich auf das Event, das er erfolgreich registriert wurde.

Neue Messwerte

In dem in Abbildung 3.5 sichtbaren Szenario, wird die Verarbeitung eines neuen Datensatzes an Messwerten dargestellt. Vereinfachend wurde hier auf die Darstellung der einzelnen OPC-UA-Clients und -Server verzichtet. Jedes Mal, wenn Daten angefordert werden oder ein Event ausgelöst wird, ist ein Client der Anfordernde/Auslösende und der Server der Sender der Daten bzw. der Empfänger des Events.

Die Schritte 1 bis 5 und 11 bis 12 stellen einen vollständigen Durchlauf der zyklischen Aktivitäten eines Lokalmonitors dar. Es wird in jedem Zyklus einmal ein Hearbeat (also ein Lebenszeichen, Schritt 1) an den C2-Server gesendet um zu bestätigen, dass der LM nach wie vor aktiv und erreichbar ist.

Der LM fordert via Modbus neue Daten von der RTU an (Schritt 2). Nach Erhalt dieser Messwerte (Schritt 3), werden sie in der Datenstruktur des internen OPC-UA-Servers hinterlegt (Schritt 4) und eine Benachrichtigung via OPC-UA-Event an die zugeordneten Nachbarschaftsmonitore gesendet (Schritt 5).

Sowohl der LM, als auch der NM prüfen dann die Messwerte auf Anforderungsverletzungen und erstatten per Event an den C2 Bericht, falls Verstöße detektiert werden. Die Schritte 6 bis 10 und 11 bis 13 können parallel ausgeführt werden, da sie in unterschiedlichen Prozessen durchgeführt werden.

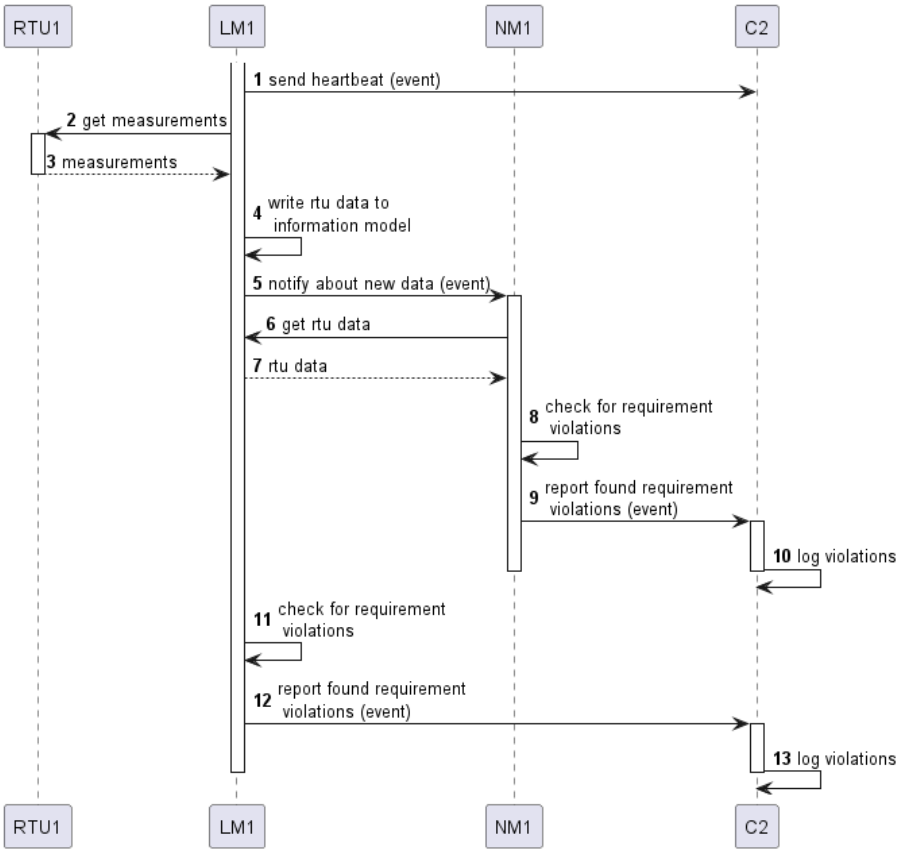

Abbildung 3.5: Neue Messwerte werden abgearbeitet. Im Dargestellten Szenario erhält der Lokalmonitor neue Daten von der RTU, die dann anschließend von Lokal- und Nachbarschaftsmonitor geprüft werden.

3.3 Deployment

3.3.1 Docker und Containervirtualisierung

Da das System als Ganzes ursprünglich bereits aus fünf (und nach dieser Arbeit sogar aus sechs) unterschiedlichen Arten von Komponenten besteht, war es nötig ein Deployment zu entwerfen. Im Projektseminar entschieden wir uns dazu *Docker* zu verwenden.[20]

Docker ist eine Software zur *Containervirtualisierung*. Dabei handelt es sich um eine Methode, um mehrere Instanzen eines Betriebssystems isoliert voneinander den Kernel eines Hostsystems nutzen zu lassen. Dabei schränkt das Hostsystem den Zugriff auf Systemressourcen durch die einzelnen Container ein. Im Vergleich zu anderen Methoden der Virtualisierung gilt Containervirtualisierung als besonders ressourcenschonend.[21]

Das hier geschilderte Deployment geht davon aus, dass das gesamte System auf einem einzelnen Hostrechner ausgeführt wird. Die Anpassungen, die nötig sind um es auf mehreren Rechnern verteilt auszuführen, werden in Kapitel 5 geschildert. Die durch uns im Projektseminar entworfene Struktur ist in Abbildung 3.6 zu sehen.

Eine aktive Docker Anwendung setzt sich aus sogennanten *Containern* zusammen. Diese Stellen jeweils eine isolierte Umgebung für die Ausführung von Anwendungen dar. Wie in dem Deployment Diagramm erkennbar, wurde für jede Komponente des IDS ein eigener Container genutzt. Jeder Container ist eine aktive Instanz eines sogenannten *Image*. Images sind Vorlagen, in denen das ,,Betriebssystem", in dem die Anwendung ausgeführt wird, konfiguriert wird. Häufig bauen Images wiederum auf anderen Images auf und fügen diesen dann zusätzliche Spezialisierungen hinzu.

Das Image wird im sogenannten *Dockerfile* definiert. Mithilfe des Befehls *docker build <Path of Dockerfile>* kann dann das Image gebaut werden, d.h. ein ausführbares System geschaffen werden. Dieses kann mithilfe des *docker run <Image name>* Befehls zu einem Container instantiiert und ausgeführt werden.

Auf dem Hostrechner läuft der sogenannte *Docker Daemon*. Dieser ist das Herzstück von Docker und stellt alle Funktionalitäten bereit. Er verwaltet die Images, die Container und realisiert die Funktionen des *Command Line Interfaces*.[20]

3.3.2 Dockerfiles

Beispielhaft ist in Quelltext 3.1 der Inhalt des Dockerfiles für die Monitore und den C2-Server zu sehen (Pfad: *ids/implementation/Dockerfile*). Alle Images in dem IDS Prototypen bauen auf das Image *python:3.7-slim-buster* auf. Wie am Namen erkennbar, ist dies ein Image, das speziell auf Python-Anwendungen der Version 3.7 ausgelegt ist. Die Namenszusätze *-slim* und *-buster* deuten darauf hin, dass es sich zum einen um eine Installation handelt, die nur die notwendigsten Packages vorinstalliert hat und zum anderen, dass die Installation auf den *Buster*-Release der Linux-Distribution *Debian* aufbaut.

Mit der *RUN* Anweisung werden Kommandozeilenbefehle ausgeführt. Wie man sieht, werden zuerst die im Image installierten Packages mithilfe des *apt* Packagemanagers aktualisiert und anschließend über den *pip* Packagemanager von Python alle vom IDS benötigten Pakete installiert. Die *COPY* Instruktion kopiert Datein vom Dateisystem des Hosts (auf dem der *build* Befehl ausgeführt wird) in das Dateisystem des Containers.

```
1  # syntax=docker/dockerfile:1.3
2
3  FROM python:3.7-slim-buster
4  WORKDIR /ids
5
6  RUN apt-get update
7  RUN apt-get -y install libffi-dev
8
9  RUN /usr/local/bin/python -m pip install --upgrade pip
10
11 COPY implementation/requirements.txt implementation/requirements.txt
12 RUN pip install -r implementation/requirements.txt
13
14 COPY . .
```

Quelltext 3.1: Dockerfile des C2-Servers

3.3.3 Docker Compose

Da der IDS Prototyp aus mehreren Containern besteht, ist es auch nach dem Bau von Images noch nicht handlich mit diesen umzugehen, da man jedes einzelne in einem eigenen Kommandozeilenfenster ausführen müsste, um Container zu erzeugen. Zusätzlich müsste man bei jedem Ausführen des *docker*

run Befehls auch noch spezifizieren, welche Ports freigegeben werden und somit nach außen hin sichtbar werden. Da dies mit zunehmender Größe eines Systems immer unübersichtlicher wird, gibt es das Tool Docker Compose.[22]

Docker Compose ist ein Werkzeug zur Definition von Multi-Container-Anwendungen. Es bietet Befehle zum starten, stoppen und neu bauen eines ganzen Systems. Außerdem gibt es Möglichkeiten den Status von laufenden Anwendungen zu überwachen und den Output des Loggings der Container einzusehen.

```
1  version: '3.4'
2  services:
3      c2:
4          image: itsis-blackout/ids_base:latest
5          build:
6            context: ../
7            dockerfile: implementation/Dockerfile
8          container_name: mosaik_ids_c2
9          env_file: config/environment/c2.env
10         command: python implementation/c2_server.py
11         ports:
12           - 8777:8777 # Websocket Port for connection to Visualization
13           - 4840:4840 # OPC C2-Server Port
14         volumes:
15           - ./config/certificates:/config/certificates
```

Quelltext 3.2: Auszug aus der *docker-compose.yml*. Zu sehen ist die Definition des C2-Servers.

In Quelltext 3.2 ist die *docker-compose.yml* zu sehen, die die Konfiguration des gesamten Deployments enthält. Nach der Definition der verwendeten Version von Docker Compose werden die *Services* dokumentiert. Der Auszug oben definiert den *c2* Service.

In der *Build* Sektion der Definition wird der *Context*, in dem der Build ausgeführt wird, als relativer Pfad im Verhältnis zur docker-compose.yml angegeben. Als relativer Pfad in Abhängigkeit des Contexts wird danach der Ort des Dockerfiles spezifiziert. Es wird der Name des Containers festgelegt, eine Datei mit Werten für Umgebungsvariablen festgelegt und anschließend der Befehl zur Ausführung des C2-Servers angegeben.

Unter *Ports* verbirgt sich die Möglichkeit eine Portweiterleitung aus dem internen Netzwerk des Docker Containers in das Netzwerk des Hostrechners

zu konfigurieren. Dies wird in diesem Fall für einen Websocket und den OPC-UA-Server durchgeführt. Das *Volume* definiert einen speziellen Speicher, der im Dateisystem des Hosts liegt und in den Container eingebunden wird.

Um das gesamte System zu bauen, kann nun in der Kommandozeile im Verzeichnis *ids/deployment* der Befehl *docker compose build* ausgeführt werden. Dies benötigt bei erstmaliger Ausführung mehrere Minuten. Mit *docker compose up* kann dann das System gestartet und mit *docker compose down* gestoppt werden.[22]

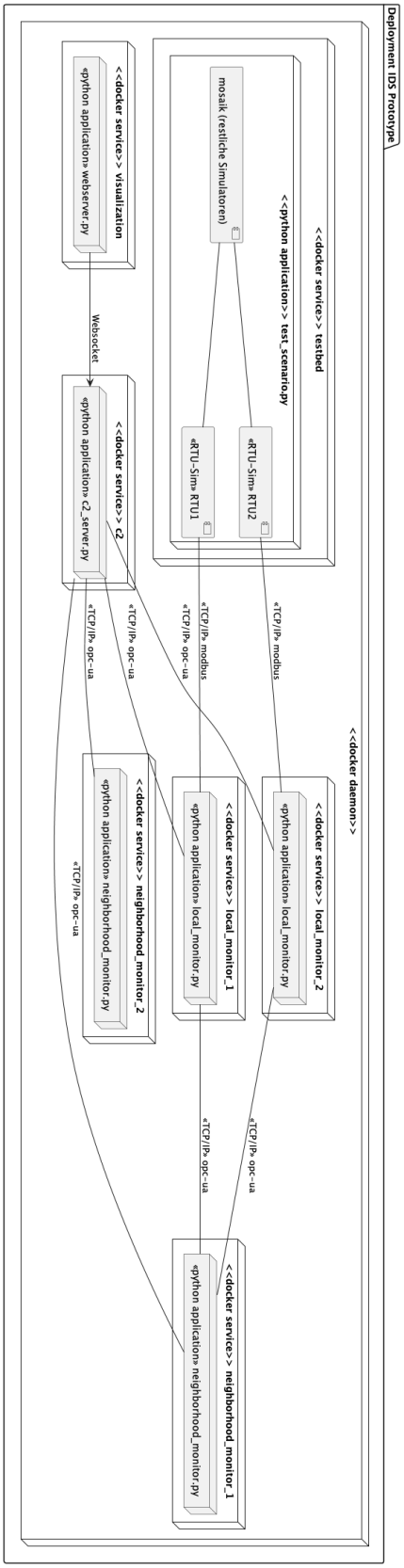

Abbildung 3.6: Deployment-Diagramm des gesamten Systems

4 Anforderungsanalyse

Das Konzept des IDS Prototypen hat, wie im Kapitel 2 geschildert, bereits eine Wandlung erlebt, die es Schritt für Schritt näher zu einem praktisch einsetzbarem System gebracht hat. Dieses Kapitel soll Aspekte herausarbeiten, die das System erfüllen muss, um noch näher an einen praktischen Einsatz kommen zu können.

Zu Beginn wird das Einsatzgebiet des IDS diskutiert (Abschnitt 4.1), danach werden Anforderungen bei der Auswahl von Hard- und Software in Abschnitt 4.2 bestimmt. Anschließend werden die zu berücksichtigenden Sicherheitsaspekte in Abschnitt 4.3 behandelt.

4.1 Rahmenbedingungen

Dieser Abschnitt beschäftigt sich damit, in welchen Szenarien das IDS eingesetzt werden kann. Dabei wird sowohl auf finanzielle, als auch auf organisatorische Aspekte eingegangen.

Grundsätzlich ist das System in der Lage, aufgrund seiner lokalen Struktur, bei der lediglich die initiale Herstellung einer Verbindung, sowie das Logging über den zentralen C2-Server abgewickelt wird, auch in sehr großem Maßstab ausgerollt zu werden. Es gibt allerdings einige limitierende Faktoren, die es erschweren das System im gesamten Elektrizitätsnetz inklusive aller Haushalte zu installieren.

Dichte an Messpunkten

Die wohl größte Hürde stellt die Dichte an benötigten Messpunkten dar. Viele der zu evaluierenden Regeln benötigen eine sehr hohe Dichte an Messgeräten, damit es überhaupt möglich ist sie prüfen. Beispielsweise setzt die erste Kirchhoffsche Regel voraus, dass an jeder Leitung, die mit einem Bus verbunden ist, der fließende Strom gemessen wird. Wenn auch nur über eine Leitung keine Information zur Verfügung steht, ist es nicht mehr möglich die Anforderung zu überprüfen.

Nicht nur die lokalen Regeln, sondern auch die Regeln der Nachbarschaftsmonitore sind hiervon betroffen. In den Grenzregionen sollten für höchste Sicherheit in jedem der Teilnetze, die verbunden werden, mindestens ein Mess-

gerät auf jeder Leitung installiert sein. Demnach sind auf solchen Leitungen sogar zwei Sensoren notwendig.

Die Anzahl an Messpunkten stellt dabei einen sehr großen Kostenfaktor dar, der nicht zu unterschätzen ist. Zusätzlich zu den Installations- und Anschaffungskosten der Sensoren sind die Wartungskosten bei einer derartigen Menge an Geräten massiv. Auch der Platzbedarf, den eine solche Menge an Sensorik aufweist, könnte die Kapazitäten in den vorhandenen Verteilerkästen sprengen.

Alternative

Alternativ zu einer Installation im gesamten Netz, wäre es möglich sich auf bestimmte Schlüsselstellen zu beschränken. Denkbar wäre es beispielsweise Sensoren an folgenden Stellen anzubringen:

- Rings um Transformatoren. Hier herrscht ein besonders großes Schadpotential, da bei einem Schaden ganze Stadtteile vom Stromnetz getrennt werden können. Gleichzeitig handelt es sich um größere Einrichtungen, bei denen typischerweise bereits umfangreiche Sensorik installiert ist.

- An den Hauptzuflüssen von Wohn- und Industriegebieten. So kann man mit weniger Aufwand trotzdem einen grundlegenden Schutz vor Manipulationen gewährleisten.

- An Grenzüberschreitenden Leitungen zwischen Ländern. Hier kann es von Interesse sein um Ketteneffekte zu verhindern, wenn bspw. Manipulationen in einem Land durchgeführt werden und man vermeiden möchte, dass durch dort entstandene Fehlentscheidungen Schäden am eigenen Netz auftreten.

Der aktuelle Prototyp nutzt die Haushaltsebene, da es bei einem Simulator keine nennenswerten finanziellen Auswirkungen hat mehr Messpunkte zu ,,installieren". Zukünftige Forschung könnte sich damit beschäftigen, wie es möglich ist das System auf wirtschaftliche Art zu implementieren und dabei die Funktionalität des IDS zu erhalten.

4.2 Systemanforderungen

Grundsätzlich gilt es im System zwei Komponentenklassen zu unterscheiden. Zum einen die Monitore (Lokal- und Nachbarschaftsmonitore), die verteilt ,,im Feld" eingesetzt sind und zum anderen den C2 Server, dessen Positionierung nicht an einen bestimmten Ort gebunden ist.

4.2.1 Monitore

Hardware

Bei der Auswahl von Hardware für die Monitore sollte darauf geachtet werden, dass man eine große Anzahl von ihnen benötigt. Der reine Leistungsbedarf eines Monitors hält sich dabei in Grenzen. Wenn möglich sollten Lokal- und Nachbarschaftsmonitor auf dem gleichen System ausgeführt werden, um den Wartungsaufwand gering zu halten.

Bei der Auswahl von Hardware und Betriebssystem ist aufgrund des Deployments darauf zu achten, dass es eine lauffähige Portierung des Docker Daemon geben sollte, damit der Prototyp unterstützt wird. Da Docker nativ eine Linux Anwendung ist und die MacOS- und Windows-Portierungen von Docker einen Hypervisor nutzen, empfiehlt es sich eine Linux-Distribution zu verwenden. Dies hat auch den Vorteil, dass diese in der Regel sehr gut an den Einsatzzweck anpassbar sind und somit ressourcenschonend eingesetzt werden können.

Wie in Kapitel 5 noch näher beleuchtet wird, genügt bereits ein leistungsschwacher Einplatinencomputer mit einem Gigabyte RAM aus, um die Monitore mit dem im Prototyp simulierten Netz auszuführen. Der nicht-volatile Speicherbedarf eines Monitors liegt bei weniger als 500 MB je installiertem Container. Hinzu kommt der Platzbedarf für das Betriebssystem und den benötigten Docker Daemon, sowie eventuelle weitere Software. Um hier ausreichend Flexibilität und Zukunftssicherheit zu gewährleisten empfiehlt sich eine Mindestspeicherkapazität von 10 Gigabyte.

Während im Bereich der Desktoprechner noch die x86-Architektur der Standard ist, so wird doch gerade bei Eingebetteten Systemen und Mobilgeräten mittlerweile die deutlich effizientere ARM-Architektur verwendet.[23] Neben den Vorteilen bei der Effizienz liegen auch die Kosten von ARM-Prozessoren deutlich niedriger als bei klassischen x86-CPUs. Eine Vielzahl unterschiedlicher *Kleinstrechner* (auch *Einplatinenrechner* genannt) ist kommerziell sofort in großen Stückzahlen verfügbar.

Konnektivität

Im Bezug auf Konnektivität muss man zum einen die Verbindung zur RTU und zum anderen die Verbindung zum C2-Server betrachten.

Da die Monitore in direkter räumlicher Nähe zu den RTUs platziert werden sollen, kann eine Verbindung zu diesen auf vielfältige Art und Weise erfolgen. Bisher wird über das Modbus-TCP-Protokoll, also einem Standard, der die Ethernet-Schnittstelle verwendet, kommuniziert. Sollte das System jemals über das Prototypen-Stadium hinaus entwickelt werden, so wäre es allerdings

auch denkbar eine andere Schnittstelle zur Dateneinspeisung zu verwenden. Viele RTUs verfügen bspw. auch über serielle Schnittstellen via RS232.[24] Man muss das IDS in diesem Fall an die bereits in Verwendung befindliche RTU anpassen.

Viele kommerziell verfügbare Kleinstrechner (Raspberry Pis, Arduinos, etc.) verfügen über die Möglichkeit mithilfe von Zusatzplatinen Schnittstellen hinzuzufügen, über die der Rechner standardmäßig nicht verfügt. So wäre es zum Beispiel möglich serielle Anschlüsse für die Verbindung zur RTU nachzurüsten. Auch ist es so möglich in einem SCADA Netz mit unterschiedlichen RTU Modellen den Monitor spezifisch an alle Modelle anzupassen, ohne dabei die Hardwareplattform des Monitors im Kern ändern zu müssen.

Bei weiterer Verwendung von Modbus-TCP bietet es sich an, für die Kommunikation mit dem C2-Server die gleiche Infrastruktur zu verwenden, die auch durch die RTU für die Kommunikation mit dem SCADA-Server verwendet wird. In vielen Fällen dürfte dies öffentliche Infrastruktur wie kabelgebundene DSL-Anschlüsse, Glasfaser oder eventuell auch Mobilfunk sein. Das legen von separaten eigenen Leitungen für die gesamte Infrastruktur wäre nicht wirtschaftlich und würde keinen Vorteil bringen, solange man nicht auch die RTU über das gleiche Netzwerk verbindet, da im Falle eines Ausfalls der Verbindung zwischen SCADA-Server und RTU ohnehin das IDS obsolet ist.

Je nach Relevanz eines Monitors kann man auch erwägen zwei voneinander unabhängige Verbindungen zu integrieren und somit eine gewisse Redundanz hinzufügen. Dabei sollte man darauf achten das man in dem Fall wirklich unabhängige Kommunikationswege wählt. Ein Beispiel ist in Abbildung 4.1 zu sehen: Bei Verwendung von DSL als Hauptanbindung und Mobilfunk als Backup, darf der Mobilfunkmast nicht über den gleichen Internet Service Provider (*ISP*) angebunden sein, denn sonst führt eine Störung im Netz des einen ISP dazu, dass auch die Backup-Infrastruktur mit ausfallen würde.

Der Bandbreiten Bedarf für Monitore ist nach heutigen Standards eher gering. Bereits einige Hundert Kilobit bis wenige Megabit die Sekunde genügen zur Versorgung eines Monitors. Die Latenz kann die Auswertung der Regeln beeinträchtigten, indem Messwerte eines Monitors schon wieder veraltet sind, bevor der andere Monitor überhaupt eine Chance hatte sie abzurufen.

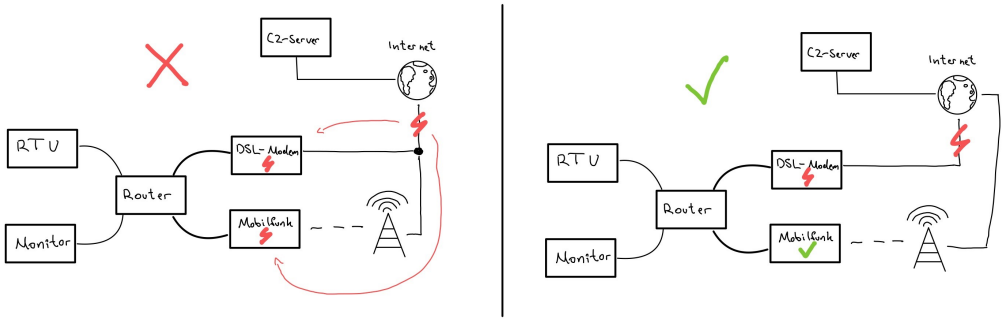

Abbildung 4.1: Redundante Netzwerkanbindung. Links eine unzuverlässige Umsetzung, rechts eine wirklich redundante Anbindung.

4.2.2 C2-Server

Hardware

Als einzige zentrale Komponente des IDS, ist der C2-Server die Komponente mit dem höchsten Leistungsbedarf. Während es bei den Monitoren neben der Verlässlichkeit auch auf Energieeffizienz ankommt, so ist dies beim C2-Server zweitrangig. Höchste Priorität hat, dass der C2-Server jeder Zeit genügend Kapazitäten hat, um auf Events aus dem Netzwerk der Monitore zu reagieren.

Somit kommt es bei der konkreten Auswahl von Hardware des C2-Servers je nach Anzahl der zu verwaltenden Monitore zu sehr unterschiedlichem Leistungsbedarf. In jedem Fall sollte hier allerdings auf Standard Serverhardware zurückgegriffen werden. Eventuell ist auch der Einsatz von Speicherfehler-Erkennendem *ECC*-RAM sinnvoll.

Die Anforderungen an nicht-volatilen Speicher sind nur gering, da kaum Daten lokal gespeichert werden sollten. Zur Steigerung der Performance empfiehlt sich eine SSD als Systemlaufwerk zu verwenden.

Als sicherheitskritische Komponente sollte der C2-Server jederzeit verfügbar sein, denn im Falle eines Ausfalls würden die gefundenen Verstöße nicht ausgewertet werden. Darum ist es von enormer Wichtigkeit, dass der C2-Server auch vor eventuellen lokalen Stromausfällen geschützt wird. Ein solches Schutzkonzept nennt man *Unterbrechungsfreie Stromversorgung* (*USV*)[25]. Diese kann je nach abzudeckender Dauer eines Ausfalls und der benötigten Leistung auf unterschiedliche Arten geschehen, üblich sind:

- Akkus werden für die gesamte Dauer eines Stromausfalls für die Versorgung genutzt. Vorteile: Keine Anlaufzeit. Nachteil: Hohe Anschaffungskosten.

- Notstromaggregat mit Benzin-/Dieselantrieb. Vorteile: Niedrige Anschaffungskosten. Nachteil: Treibstoff ist nicht ewig haltbar, Anlaufzeit von mehreren Sekunden (muss bspw. mit Akku überbrückt werden).

Konnektivität

Der C2-Server wird über das Internet mit den Monitoren verbunden. Diese Verbindung muss allerdings deutlich höheren Standards genügen, als dies bei den Monitoren der Fall ist. Die nachfolgend geschilderten Anforderungen an die Konnektivität gehen von einer Implementierung des IDS in großen Maßstab aus, bei dem ein enormes wirtschaftliches Risiko im Falle eines unbemerkten Angriffs auf das Stromnetz bestünde.

Während auf Seiten des Monitors Redundanz bei der Internetanbindung optional ist, so ist diese beim C2-Server unerlässlich. Auch hier gilt es zu beachten, dass mindestens zwei unabhängige Anbindungen existieren sollten (wie in Abbildung 4.1 für einen Monitor veranschaulicht). Zusätzlich sollten diese aber nicht nur von voneinander unabhängigen ISPs betrieben werden, sondern auch an geographisch möglichst weit auseinander liegenden Stellen in das Gebäude des C2-Servers eingespeist werden. So kann das Risiko einer unabsichtlichen Beschädigung bei Bauarbeiten oder durch Unfälle (Feuer etc.) reduziert werden.

Die zur Verfügung stehende Bandbreite am C2-Server sollte entsprechend der Anzahl der Monitore ausgewählt werden. Grundsätzlich sollten einige Hundert Megabit pro Sekunde als Minimum angesehen werden. Allerdings sollte die höchstmögliche Bandbreite gewählt werden, die der finanzielle Rahmen erlaubt.

Anders als auf Seiten des Monitors, spielt beim C2-Server die Latenz eine untergeordnete Rolle. Allerdings ist bei den zuvor empfohlenen Bandbreiten zumeist bereits eine ausreichend geringe Latenz zu erwarten.

Anmerkung

Vermutlich würde man zur Reduzierung der Kosten den C2-Server in einem bereits vorhandenen Rechenzentrum aufstellen. Dort sind in der Regel bereits entsprechende Internetanbindungen vorhanden, die den Ansprüchen des IDS bezüglich Redundanz und Leistung genügen. Auch die Notstromversorgung würde andernfalls einen enormen Kostenfaktor darstellen, der sich durch die Unterbringung im Rechenzentrum reduzieren würde. Gerade die Wartung der Notstromversorgung stellt einen wesentlichen Kostenpunkt dar, der im Rechenzentrum natürlich auf alle dortigen Server umgelegt werden kann.

4.3 Sicherheit

Der Daseinszweck des IDS besteht darin die Sicherheit bestehender SCADA-Systeme zu erhöhen. Dazu hat das System zumindest lesenden Zugriff auf Bestandteile der SCADA-Infrastruktur und somit Zugang zu sensiblen Informationen mit Bezug zu sicherheitskritischer Infrastruktur. Darum ist es im Zusammenhang mit einem realen Einsatz des Systems auch relevant zu untersuchen, ob das IDS-Schwachstellen enthält und somit selbst neue Angriffsvektoren auf das Elektrizitätsnetz eröffnet.

Das Grundkonzept der Informationssicherheit besteht im CIA-Prinzip.[26][27]

- **Confidentiality - Vertraulichkeit** Unterabschnitt 4.3.2
- **Integrity - Integrität** Unterabschnitt 4.3.3
- **Availability - Verfügbarkeit** Unterabschnitt 4.3.1

Ergänzend wird auch Physische Sicherheit (Unterabschnitt 4.3.4) kurz behandelt.

4.3.1 Verfügbarkeit

Risikoanalyse: Verfügbarkeit

Die IDS hat die Aufgabe manipulierte Sensordaten in Stromnetzen zu finden, somit ist es wichtig sicherzustellen, dass das System tatsächlich verfügbar ist und das SCADA System vor Attacken schützt.

(Distributed) Denial of Service Attacken

Eine häufig vorkommende Art von Angriffen in vernetzten Systemen ist ein sogenannter *Denial of Service* (*DoS*) Angriff. Dieser basiert darauf einen oder mehrere Endpunkt des Systems mit mehr Anfragen zu überhäufen, als der Endpunkt bewältigen kann. Dies führt dazu, dass die Kommunikation zwischen dem Endpunkt und dem Rest des Systems gestört wird und das Opfer der Attacke seinen Aufgaben nicht mehr oder nur in vermindertem Umfang nachkommen kann.[28]

Da die Durchführung derartiger Angriffe häufig eine enorme Bandbreite benötigt, ist in der Regel eine große Anzahl angreifender Rechner an einer Attacke beteiligt. Man spricht in diesem Fall von einer *Distributed Denial of Service Attacke* (*DDoS*). Die verwendeten Rechner für solch eine Attacke stammen zumeist aus einem *Botnet*, also durch Malware-infizierte Zombie-Rechner, die ohne das Wissen ihrer Besitzer durch Kriminelle ferngesteuert werden.[29]

Bei der Wahl einer Verteidigungsstrategie gegen einen DDoS-Angriff kommt es auf die genaue Art des verwendeten Angriffs an. Allgemein kann man kleine DoS-Attacken (mit nur einem oder wenigen angreifenden Rechnern) über eine Sperrliste mit den IP-Adressen der Angreifer abwehren. Desto mehr Rechner angreifen, desto aufwendiger werden die notwendigen Methoden zur Abwehr.

DDoS-Attacken auf einen Monitor

Der Monitor ist so klein, dass ein auftretender DDoS-Angriff größeren Ausmaßes kaum abzuwehren wäre. Realistisch ist eine korrekt konfigurierte Firewall und ein Grenzrouter (der Router für die Verbindung zum Internet), der Anfragen mit ungültigen Absenderadressen nach RFC 3704 filtert. Dies vermindert die Gefahr durch *IP-Spoofing*, also das vortäuschen einer falschen IP-Adresse zur Umgehung einer Sperrliste.[30]

Der C2-Server ist in der Lage einen DDoS-Angriff auf Monitore zu bemerken. Dies ermöglicht der Heartbeat, den alle Monitore in regelmäßigen Abständen an den C2-Server übermitteln. Implementiert wurde dieser über OPC-UA als Event. Auch die meisten anderen Attacken auf die Verfügbarkeit eines Monitors werden so erkannt.

DDoS-Attacken auf den C2-Server

Durch die Unterbringung in einem Rechenzentrum stehen dem C2-Server deutlich umfangreichere Verteidigungsstrategien zur Verfügung. Welche Optionen man tatsächlich implementiert, sollte man nach einer Risikoabschätzung entscheiden. Das Kosten/Nutzenverhältnis sollte gewahrt bleiben.

Möglichkeiten umfangreiche Attacken abzuwehren sind u.a.[31]:

- **Upstream filtering** Das umleiten des Datenverkehrs zu einem Dienstleister der auch sehr große Datenmengen filtern kann. Dazu benötigt dieser vorallem eine sehr große Bandbreite im Bereich vieler hundert Gigabis Terabit. Nachteil: Teuer und nur schwer mit Datenschutz vereinbar, da viele Anbieter solcher Dienste außerhalb der EU sitzen.
- **Blackholing** Das umleiten des Traffics an einen nicht-existierenden Server (*Blackhole*). Kann an ISP ausgelagert werden.
- **Intrusion Prevention Systeme** Identifizieren der Signaturen eines DDoS Angriffs, um rechtzeitig mit einer der obigen Methoden reagieren zu können.[32]

Unkonventionelle Angriffe

Den Effekt eines DDoS-Angriffs kann ein Angreifer potenziell auch hervorrufen, indem er den Monitoren bewusst auffällig manipulierte Messdaten zuspielt.

Die Monitore würden in Folge sehr viele Anforderungsverletzungen finden und diese an den C2-Server melden. Wenn genug Monitore dies gleichzeitig tun, ist es denkbar, dass in einem sehr groß dimensioniertem Netz mit entsprechend vielen Monitoren, ein DDoS-artiger Effekt auf Seiten des C2-Servers auftritt.

Da die zahlreichen Anfragen von legitimen, authentifizierten Monitoren stammen, erschwert dies den Umgang mit solchen Attacken. Erst einmal sind ja alle Meldungen an den Server valide, allerdings ist dieser ab einem gewissen Punkt nicht mehr in der Lage alle Anfragen abzuarbeiten. Dies könnte verwendet werden, um andere Manipulationen zu maskieren.

Gerade um derartige Angriffe zu verhindern, ist es notwendig den C2-Server ausreichend leistungsstark zu bestücken, sodass er auch in einem *worst case* Szenario noch funktionsfähig bleibt. Sollte das IDS zukünftig in solchem Maßstab implementiert werden, empfiehlt es sich zu untersuchen inwiefern solch eine Attacke realisierbar wäre und wie man das IDS dagegen wappnet.

4.3.2 Vertraulichkeit

Risikoanalyse: Vertraulichkeit

Vertraulichkeit bezieht sich im CIA-Prinzip darauf, dass alle Daten innerhalb des Systems vor unbefugtem Zugriff geschützt und nicht von Außen zugänglich sind. In Bezug auf das IDS sind folgende Daten besonders schützenswert:

1. Die Messdaten in den Lokalmonitoren
2. Die Messdaten in den Nachbarschaftsmonitoren
3. Die Konfigurationsdateien in den Monitoren
4. Die verwendeten Zertifikate und Privatschlüssel

Authentizität

Authentizität bezeichnet die Eigenschaft der *Echtheit*, Überprüfbarkeit dessen und der Vertrauenswürdigkeit. Den Prozess der Überprüfung nennt man *Authentifizierung*, zum Nachweis der Identität wird ein *Zertifikat* genutzt. Dieses Zertifikat wird von eine vertrauenswürdigen *Certification Authority* (*CA*) ausgestellt.[33][34]

OPC-UA

Der im IDS genutzte OPC-UA-Standard ist von vornherein auf Sicherheit ausgelegt worden. Wir verwenden den Modus *SignAndEncrypt* für die Absicherung der Kommunikation. Dies ist der sicherste Modus den OPC-UA anbietet und

nutzt Zertifikate für die Authentifizierung.

Das Bundesamt für Sicherheit in der Informationstechnik hat 2021 eine Sicherheitsanalyse des OPC-UA Standards in Auftrag gegeben. Dabei wurde der Standard ausführlich auf Schwachstellen untersucht. Die Autoren kommen zum Schluss, dass der durch das IDS verwendete *SignAndEncrypt*-Modus gegen alle betrachteten Angriffsszenarien mit Ausnahme eines DoS und des *Server Profilings* wirksamen Schutz bietet.[35]

Da die Übermittlung der Messdaten zwischen Lokal- und Nachbarschaftsmonitor über den OPC-UA Standard und somit verschlüsselt abgewickelt wird, kann die Sicherheit der Übertragung gegen *Eavesdropping* (Lauschangriffe) und damit verbundenes nach Außen dringen von Informationen als gewährleistet betrachtet werden.

Der aktuelle Prototyp benutzt für beide Lokalmonitore das gleiche Zertifikat, gleiches gilt auch für die Nachbarschaftsmonitore. Bei einer echten Anwendung des IDS sollten alle Komponenten eigene Zertifikate besitzen. Aktuell handelt es sich auch um selbstsignierte Zertifikate, diese würden in der Praxis von einer CA stammen.

Festplattenverschlüsselung

Bezüglich der Sicherheit von Konfigurationsdateien, Zertifikaten und Privatschlüsseln muss vor allem damit gerechnet werden, dass ein Angreifer versucht in den Besitz eines Monitors zu gelangen und versucht auf dessen Speicher zuzugreifen. Um zu verhindern, dass er solche empfindlichen Daten direkt auslesen kann, ist eine *Festplattenverschlüsselung* sinnvoll.[36]

Der Einsatz der Verschlüsselung sollte in Kombination mit einem *Trusted Platform Module* (*TPM*) verwendet werden. Dabei handelt es sich um einen speziellen Chip, der entweder bereits auf dem Mainboard verlötet oder aber als Zusatzplatine ergänzt werden kann. Definiert wird das TPM durch die ISO in [37]. Das TPM erfüllt unter anderem folgende Funktionen[38]:

- **Sicherer Speicher (Trust storage)** Zum Speichern von Schlüsseln. Begrenzte Kapazität.
- **Versiegelung (Sealing)** Binden von Daten an eine einzige System-Konfiguration aus Hard- und Software. Entschlüsselung der Daten ist an das TPM gebunden, bei Veränderungen sind die Daten nicht mehr zugreifbar.
- **Auslagerung (Binding/Wrapping)** Zur sicheren Speicherung von Datein und Schlüsseln außerhalb des Trust storage.

- **Sichere Zufallsgenerator** Für die Erzeugung von Zufallszahlen.

Gerade das sichere Speichern des eigenen Privatschlüssels ist ein Kernpunkt für die Gewährleistung von Sicherheit. Dies ist durch ein TPM möglich, weshalb es empfehlenswert erscheint bei der Auswahl einer Hardwareplattform für die Monitore auf vorhandensein eines solchen zu achten.

VPNs

Zur weiteren Erhöhung der Sicherheit kann zusätzlich auch ein *Virtuelles Privates Netzwerk* (*VPN*) verwendet werden. Dies würde die gesamte Kommunikation zwischen Monitor und C2-Server noch ein zweites Mal verschlüsseln. Vor eventuell existierenden noch unbekannten Schwachstellen im OPC-UA Standard wäre so zusätzlicher Schutz gewährleistet.

4.3.3 Integrität

Risikoanalyse: Integrität

Integrität wird im Zusammenhang mit IT-Sicherheit als Verhinderung unautorisierter Modifikation von Informationen definiert.[26] Im Falle des IDS ist vor allem die korrekte Übertragung von Daten zwischen den einzelnen Komponenten von hoher Relevanz. Ein Angreifer darf nicht in der Lage sein die Daten auf dem Weg zwischen einzelnen Komponenten unbemerkt zu verändern oder gar Daten selbst erzeugen und ins System einspeisen.

Man in the middle / Person in the middle Angriffe

Bei einer sogenannten *Man in the Middle*-Attacke (*MITM*) fängt ein Angreifer Datenpakete auf dem Weg zwischen zwei Komponenten ab und liest diese mit, modifiziert oder verwirft sie, sodass die Daten ihr Ziel niemals erreichen. Im Fall unseres Systems sind folgende Positionen für einen Angreifer möglich:

1. **Zwischen RTU und Monitor -** Hier könnten Messdaten verändert werden.

2. **Zwischen Monitoren -** Hier könnten Messdaten verändert werden.

3. **Zwischen Monitoren und C2-Server -** Hier werden die Anforderungsverstöße übertragen.

4. **Zwischen Nachbarschaftsmonitor und C2-Server -** Hier werden Konfigurationen der Lokalmonitore und der Border Regions übertragen.

Resilienz

Durch die Sicherheitsfunktionen von OPC-UA wird die Integrität aller übertragenen Daten zwischen den Komponenten des IDS sichergestellt. Beispielsweise verhindert OPC-UA *Replay Attacken* (das aufzeichnen von Nachrichten durch den MITM und späteres nochmaliges einspeisen dieser Daten in den ursprünglichen Empfänger) dadurch, dass alle Nachrichten im OPC-UA Standard automatisch eine Sequenznummer beeinhalten, die es ermöglicht sofort eine wiederholte Nachricht zu erkennnen.[35]

Vorhandene Schwachstelle

Bereits im Rahmen des Projetkseminars ist eine Möglichkeit implementiert worden die Daten zwischen RTU und Monitor zu manipulieren. Dies ist aufgrund der Schwachstellen von Modbus ohne großen Aufwand möglich. Es ist kein ausreichender Schutz vorhanden.[3]

4.3.4 Physische Sicherheit

Als Teil eines Sicherheitskonzepts muss auch die Physische Sicherheit des gesamten Systems beachtet werden. Dazu zählen neben der Unzugänglichkeit der Hardware durch Dritte auch Aspekte wie Brandschutz. Man sollte hier ein gründliches Konzept ausarbeiten um eine sichere Funktion des Systems zu gewährleisten. Man kann sich beispielsweise an [39] orientieren.

4.3.5 Sonstiges

Man sollte nicht vernachlässigen, dass der Mensch häufig das schwächste Glied in der Kette ist. Es gilt bei der Einrichtung des IDS darauf zu achten, dass jede Person nur Zugang zu den für die Erfüllung ihrer Arbeit notwendigen Teilen des Sytems hat. Die Mitarbeiter sollten regelmäßig im Bezug auf Gefahren wie *Phishing* und *Social Engineering* geschult werden.[40]

Zur weiteren Aufrechterhaltung der Sicherheit ist es unerlässlich regelmäßig Sicherheitsupdates des Systems durchzuführen.

4.4 Sonstiges

4.4.1 Wartbarkeit

Beim Ausrollen des Systems in größerem Rahmen muss entschieden werden, wie die Wartung der Monitore durchgeführt werden soll. Die komfortabelste Lösung bestünde in der Einrichtung einer SSH Verbindung zu den Monitoren, allerdings stellt diese auch ein Risiko dar. Gerade Standard Passwörter sind

zu vermeiden und im Optimalfall ist die Authentifizierung über individuelle Zertifikate für jeden Monitor vorzunehmen. Ein Risiko besteht dabei im nach Außen dringen der Zugangsdaten, da ein Angreifer dann im schlimmsten an alle auf dem System gespeicherten Daten bekäme.

4.4.2 Anpassbarkeit

Um auch nachträgliche Änderungen zu ermöglichen eignet sich die bisherige Struktur bereits. Docker ermöglicht es bei Änderungen einfach einen neuen Container zu bauen und zu starten und somit auch nach der initialen Installation des Sytems Änderungen vorzunehmen.

5 Prototyp

Dieses Kapitel beschäftigt sich mit dem ausführen des Systems auf echt verteilter Hardware. In Abschnitt 5.1 wird die Struktur des Prototypen erläutert. Abschnitt 5.2 stellt alle notwendigen Anpassungen des Prototypen dar und erläutert die Bedienung des neuen Systems. Anschließend wird das Vorgehen bei der Prüfung der Funktionsfähigkeit des verteilteten Prototypen in Abschnitt 5.3 erläutert. Die Auswertung der Leistungsmessung wird in Abschnitt 5.4 vorgenommen.

5.1 Versuchsaufbau

5.1.1 Ziel des Prototypen

Der bisherige Prototyp soll so modifiziert werden, dass es möglich ist ihn nicht nur logisch in Docker-Container getrennt, sondern auch auf verteilter Hardware ausführen kann. Das Testszenario soll sich dabei nicht verändern. Es soll weiterhin jeweils zwei Lokal- und Nachbarschaftsmonitore, einen C2-Server und die Visualisierung des IDS geben. Auch das Testbed sollte bei dieser Gelegenheit separat ausgeführt werden.

Auf Seiten des Monitors sollen dabei bereits einige der Überlegungen aus der Anforderungsanalyse berücksichtigt werden. Es wäre wünschenswert, dass die verwendete Hardware möglichst klein und energieeffizient ist. Einer der verbreitetsten Rechner, der diese Anforderung erfüllt ist der *Raspberry Pi* (*Pi*). Es stehen bis zu sechs Raspberry Pis zur Verfügung, allerdings sollen im Optimalfall jeweils ein Lokal- und Nachbarschaftsmonitor zusammen ausgeführt werden und somit lediglich die Verwendung von vier Pis notwendig sein. Eine solche einheitliche Ausführung der Monitore wäre in der Realität erstrebenswert, da man so die Anschaffungs- und Wartungskosten, sowie den Platzbedarf des Systems senken kann.

Der C2-Server und das Testbed kann ebenfalls auf einem Raspberry Pi ausgeführt werden. Der Umfang des verwendeten Testszenarios ist klein genug, sodass auch die geringe Leistung eines solchen Kleinstrechners nicht vollständig ausgelastet wird.

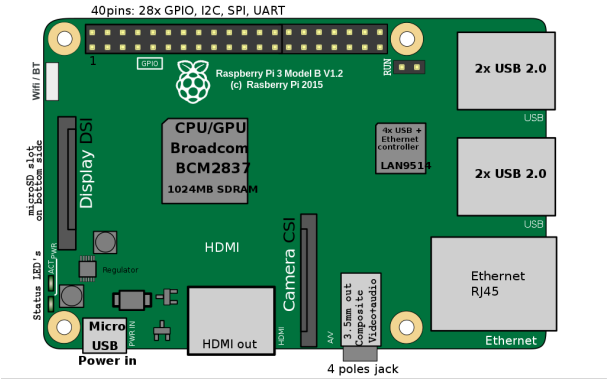

Abbildung 5.1: Raspberry Pi 3 Model B V1.2[4]

5.1.2 Raspberry Pis

Beim Raspberry Pi handelt es sich um eine Serie von Einplatinenrechenern, die durch die britische *Raspberry Pi Foundation* entwickelt wird. Die Raspberry Pi Foundation ist eine wohltätige Stiftung mit dem Ziel die Bildung im Bereich der Informatik zu fördern. Gegründet wurde sie im Umfeld der Universität Cambridge. Die Preise der Raspberry Pi Serie bewegen sich je nach Modell zwischen ca. 5 und 100 Euro.

Der für den Prototypen verwendete Pi ist das Modell *Raspberry Pi 3 Model B V1.2*. Es ist 2016 erschienen und hat eine Grundfläche, die der einer Kreditkarte entspricht. Die wichtigste technischen Daten im Überblick[41][42]:

- Broadcom *System on Chip* (*SoC*)
  - ARM Cortex-A53 CPU: 4 Kerne @ 1200MHz
  - Broadcom Dual Core VideoCore
  - 1GB RAM
- microSD Kartenslot
- 100 Mbit/s Ethernet über RJ-45 Buchse
- Broadcom BCM43143
  - Bluetooth 4.1 und Bluetooth Low-Energy
  - WLAN 2,4GHz b/g/n
- 800mA 5V Leistungsaufnahme über USB Typ micro B
- 40 Pin *General Purpose Input/Output* (*GPIO*)

Eine Besonderheit der Raspberry Pis besteht in den GPIO-Pins. Diese ermöglichen es Zusatzplatinen zu installieren, die unter anderem über i^2c mit dem Pi kommunizieren. Man nennt solche Platinen *Hardware Attached on Top* (*HAT*). Es gibt eine Vielzahl unterschiedlicher HATs, häufig werden den Pis Sensoren z.B. für Luftqualität oder zusätzliche Ausgänge (für Audiosignale etc.) hinzugefügt.

Einschränkungen der Raspberry Pis

Da es sich beim IDS um einen Teil der Sicherheitsinfrastruktur eines SCADA Systems handeln würde, kann es notwendig sein gesetzliche Vorschriften zu berücksichtigen. Derartige Regularien sehen häufig vor, dass Computergestützte Systeme einen *Audit-Trail* führen. Dabei handelt es sich um eine vor Manipulation geschützte Protokollierung aller wichtigen Ereignisse und Veränderungen am System. Die Raspberry Pis enthalten keine Echtzeituhr (*Real Time Clock, RTC*), sodass bei einem Ausfall der Internetverbindung nicht mehr garantiert werden kann, dass die echte Uhrzeit und das Datum für die Protokollierung zur Verfügung stehen. Es ist allerdings möglich eine *RTC* in Form eines *HATs* nachzurüsten.[41]

Ein weiterer schwerwiegender Nachteil der Raspberry Pis besteht darin, dass sie nicht über ein TPM verfügen (siehe Unterunterabschnitt 4.3.2). Dies hat zur Folge, dass sie standardmäßig nicht in der Lage sind die Privatschlüssel sicher zu speichern und den Speicher zu verschlüsseln, sodass bei Zugriff auf die Hardware ein Angreifer in Besitz wichtiger Informationen gelangen könnte. Auch ein TPM ist als HAT nachrüstbar, allerdings kann man dann nicht mehr alle anderen HATs zusätzlich verwenden, sodass man bspw. nicht in der Lage wäre andere Anschlussmöglichkeiten (RS-232 o.ä.) für die Verbindung zu einer RTU nachzurüsten.

5.2 Umsetzung

Dieser Abschnitt beschreibt die Umsetzung des zuvor geschilderten Versuchsaufbaus in die Praxis. Zuerst wird darauf eingegangen wie der Raspberry Pi eingerichtet wird, danach werden die notwendigen Änderungen im Code geschildert. Abschließend wird erläutert, wie man den Prototyp benutzt.

5.2.1 Konfiguration der Raspberry Pis

Installation des Betriebssystems

Bei einem Raspberry Pi wird das Betriebssystem auf einer Micro-SD-Karte gespeichert. Zur Installation des Betriebssystems kann der *Raspberry Pi Imager*, der durch die Raspberry Pi Foundation bereit gestellt wird, genutzt werden.

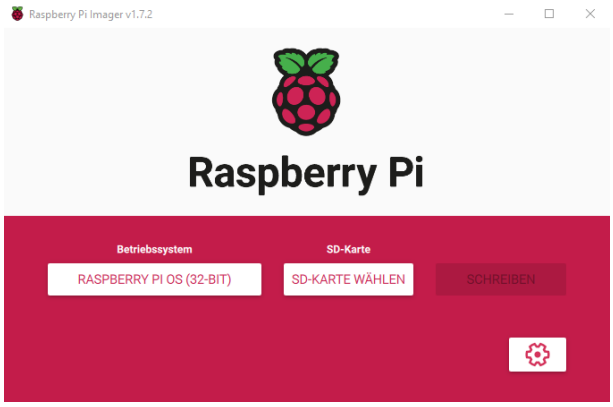

Abbildung 5.2: Hautpfenster des Raspberry Pi Imagers

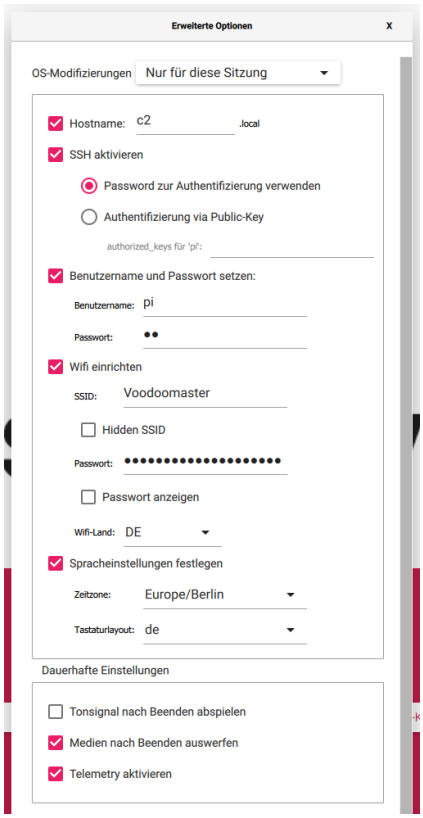

Abbildung 5.3: Erweiterte Einstellungen des Raspberry Pi Imagers

Dieses bietet eine einfach zu verwendende Grafische Benutzoberfläche an (Abbildung 5.3 und Abbildung 5.2). Die SD-Karte des Raspberry Pis wird dazu mit einem Computer verbunden, auf dem der Imager ausgeführt wird. Im weiteren Prozess wird die Speicherkarte einmal formatiert.[43]

Die Standardoption ist das Betriebssystem *Raspberry Pi OS (32-Bit)*, wobei es sich um eine speziell auf Raspberry Pis angepasste Version der Linux Distribution *Debian* handelt. Als Paketmanager steht standardmäßig *apt-get* zur Verfügung. Es empfiehlt sich dieses Betriebssystem zu verwenden, da es bereits alle passenden Treiber für einen Raspberry Pi enthält.[43]

Der Imager bietet die Möglichkeit bereits vor dem installieren des Betriebssystems einige Anpassungen vorzunehmen. Folgende Änderungen sollten vorgenommen werden:

- Hostname: Gerät korrekt benennen (*c2*, *monitor1*, *monitor2*, *testbed*)
- SSH aktivieren, mit Passwort authentifizieren: Sodass es möglich ist sich von einem anderen Rechner im Netzwerk aus per SSH zu verbinden
- Benutzernamen und Passwort festlegen: Zu Testzwecken beides *pi*
- WLAN vorkonfigurieren: SSID und Passwort eines Netzwerks, in das sich der Pi direkt nach dem starten einloggen soll
- Spracheinstellungen: Deutsche Zeitzone und Tastaturlayout einstellen

Installation von Software

Der Pi startet nach dem herstellen der Stromversorgung automatisch und verbindet sich mit dem zuvor konfigurierten WLAN Netzwerk. Die Verbindung mit dem Pi kann über SSH hergestellt werden. Unter MacOS würde man dazu den Befehl *ssh pi@hostname* verwenden und anschließend das Passwort *pi* eingeben. Über die SSH-Verbindung können die in Quelltext 5.1 aufgeführten Befehle auf dem Pi ausgeführt werden.

```
# use root user
sudo su

# install docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

# clone sourcecode from github using a access token
git clone
↪ https://@github.com/PhantomPhr3ak/bachelorthesis-distributed-ids.git

# cd into git repository
cd bachelorthesis-distributed-ids/ids/deployment/

# build docker containers
docker compose build
```

Quelltext 5.1: Konsolenbefehle auf dem Raspberry Pi, die einmalig nach der Installation des Betriebssystems ausgeführt werden.

5.2.2 Notwendige Anpassungen für eine verteilte Ausführung

Dieser Abschnitt widmet sich den Anpassungen, die am vorheringen Prototypen notwendig sind, damit die Software verteilt auf allen vier Raspberry Pis lauffähig wird.

Compilieren für ARM-Architektur

Wenn man versucht das Testbed in seinem ursprünglichem Zustand zu bauen, stellt man fest, dass dieses zuerst einmal nicht funktioniert. Der Build des Docker-Containers schlägt fehl und fordert dazu auf zusätzliche Pakete (z.B. Gnu Compiler Collection, Fortran Compiler, Rust Compiler, etc.) zu installieren. Dies hängt damit zusammen, dass der Python Paketmanager *pip* standardmäßig keine vorcompilierten Bibliotheken (sogenannte *Wheels*) für ARM-Architekturen findet.

Es werden auf der Webseite *www.piwheels.org* speziell für die ARM-Architektur des Raspberry Pi compilierte Pakete angeboten. Standardmäßig ist pip unter Raspberry Pi OS so konfiguriert, dass dieser sofort an dieser Quelle nach den Wheels sucht. Um pip auch im Docker-Container die kompatiblen Wheels

finden zu lassen, wurden die Dockerfiles um folgende Zeile ergänzt:

RUN echo "[global] extra-index-url=https://www.piwheels.org/simple" ≫ /etc/pip.conf

Anpassen der Konfigurationsdateien

Da im neuen Versuchsaufbau die einzelnen Komponenten nicht mehr über den Localhost erreichbar sind, muss jede Komponente die korrekte neue Adresse der jeweils anderen Komponenten kennen. Diese werden über Umgebungsvariablen konfiguriert, die in *.env*-Dateien im Verzeichnis *ids/deployment/config/environment* zu finden sind. Notwendige Änderungen sind bspw. im Lokalmonitor:

- **IDS_C2_ADDRESS = opc.tcp://c2:4840/freeopcua/server/**
  (Adresse des C2-Servers)
- **IDS_LM_OPC_ADDRESS = opc.tcp://monitor1:10808/freeopcua/server/**
  (Eigene externe Adresse des OPC-UA-Servers der Komponente)
- **IDS_RTU_MODBUS_HOST = testbed**
  (Adresse der RTU)
- **IDS_RTU_MODBUS_PORT = 10502**
  (Port der RTU)

Auch in der docker-compose.yml mussten die Portweiterleitungen angepasst werden, da diese zuvor nur für die beiden Visualisierungen bestanden haben. Nun müssen zusätzlich alle von OPC-UA-Servern genutzte Ports eine Weiterleitung erhalten.

Interne und externe IP-Adressen

Das nächste Problem zeigt sich darin, dass trotz korrekter Konfiguration aller Adressen der jeweiligen Komponenten, keine Verbindung untereinander erfolgt. Die Fehlerquelle besteht dabei darin, dass das System eine vielschichtige Architektur verwendet, bei der die internen IP-Adressen nicht unbedingt denjenigen Adressen entsprechen, die von außen für die anderen Komponenten sichtbar sind. In Abbildung 5.4 ist zu erkennen, wie viele Schichten jede Verbindung durchdringen muss, wenn sie zwischen zwei Komponenten verläuft.

Extern sieht eine Komponente immer den Hostnamen des Zielrechners (bspw. monitor1 für den LM1 und NM1). Der Port des Zielrechners wird dabei über eine Port-Weiterleitung mit dem Docker-Container verbunden, dies ist allerdings aus externer Sicht nicht erkennbar. Intern muss jede Anwendung, die

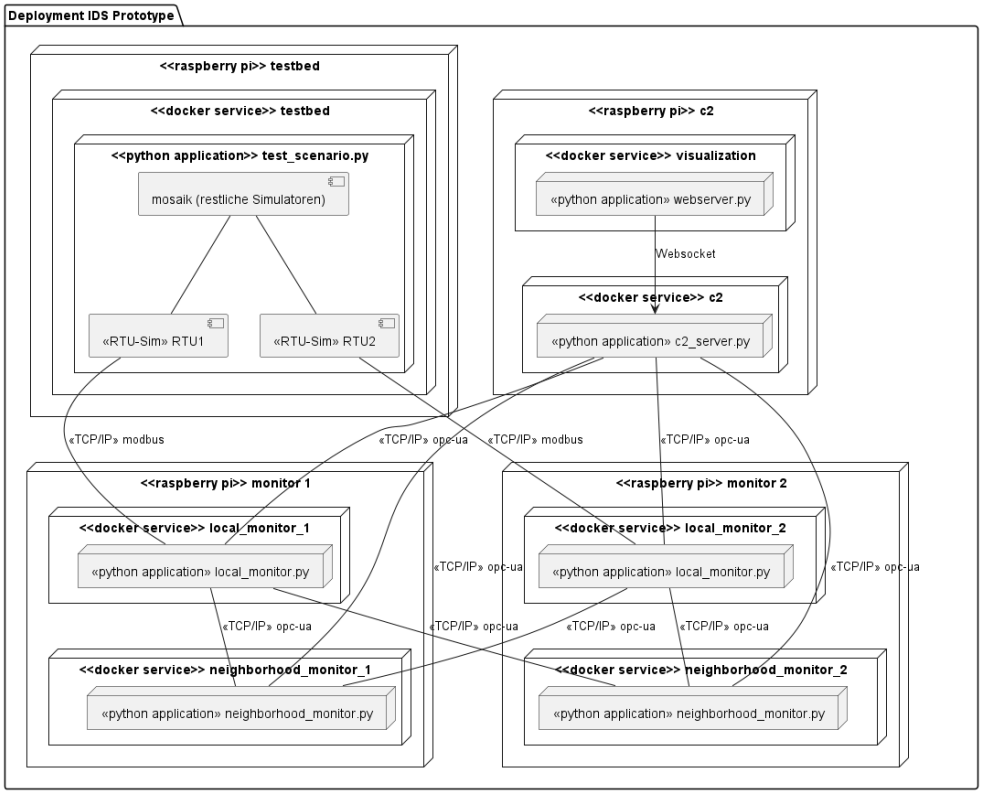

Abbildung 5.4: Deployment-Diagramm mit Raspberry Pis. Der Docker Daemon auf jedem Pi wurde der Übersicht halber nicht dargestellt.

von außen aus zugreifbar sein möchte, anstelle des üblichen Localhosts die IP-Adresse *0.0.0.0* benutzen.

Dank eines im November 2021 hinzugefügten Features, ist es möglich den OPC-UA-Server auf einer anderen Adresse und Port zu starten, als derjenigen Adresse, die der OPC-UA-Server den OPC-UA-Clients mitteilt.[44]

Fehlerhafte Regeln

Beim nochmaligen Testen des Prototypen ist unabhängig von der Ausführung auf Raspberry Pis aufgefallen, dass überdurchschnittlich viele falsch-positive Meldungen über Verstöße gegen die Kirchhoffsche Regel (Regel 2 im Lokalscope) auftraten. Bei gründlicherer Untersuchung der Implementierung dieser Regel fiel auf, dass die Sensoren für die Auswertung nicht korrekt ausgewählt wurden. Der Fehler konnte behoben werden.

5.2.3 Benutzung des Prototypen

Zur Steuerung des Prototypen sollte eine SSH-Verbindung zu jedem der Raspberry Pis bestehen. In den jeweiligen Fenstern können dann als root-User und im Verzeichnis *bachelorthesis-distributed-ids/ids/deployment/* die jeweils folgenden Befehle ausgeführt werden:

1. **c2:** *docker compose up c2*
2. **monitor1:** *docker compose up local_monitor_1 neighborhood*_monitor_1
3. **monitor2:** *docker compose up local_monitor_2 neighborhood_monitor_2*
4. Warte bis die Container gestartet sind.
5. **testbed:** *docker compose up testbed*

Das Stoppen der Container erfolgt genau wie das Starten, nur anstelle des *docker compose up* wird *docker compose down* verwendet.

5.3 Leistungsmessung

Da das Testbed bei jedem Neustart zufällig die Verbindungen zwischen Haushalten/PV-Anlagen und Bussen generiert, sind keine zwei Testdurchläufe gleich. Dies stellt ein Problem dar, wenn man prüfen möchte, ob der Testaufbau auf Raspberry Pis leistungstechnische Einbußen gegenüber der Ausführung auf einem einzelnen Hostrechner verursacht.

In der Vergangenheit wurde dieses Problem dadurch umgangen, dass die Messwerte aus einem Durchlauf des Testbeds aufgezeichnet und in einer CSV Datei gespeichert wurden. Die Lokalmonitore konnten im Prototypen von Menzel et al. direkt im Lokalmonitor ausgelesen werden. Da das auslesen der Messwerte aus den Dateien in beiden Lokalmonitoren synchron stattfinden muss, war dies nur deshalb ohne größeren Aufwand möglich, da zum damaligen Zeitpunkt noch das gesamte IDS in der selben Python Laufzeitumgebung ausgeführt wurde.[2]

Bei der Erweiterung des Systems um OPC-UA ist die Möglichkeit des Auslesens von CSV-Dateien nicht mehr mit implementiert worden, da dies nur mithilfe einer komplizierten Synchronisierung zwischen den Monitoren möglich gewesen wäre. Für das Erstellen von aussagekräftigen Leistungsmessungen des Prototypen war es daher notwendig eine Möglichkeit zu finden die ursprüngliche Funktionalität des Prototypen von Menzel et al. wieder herzustellen.

5.3.1 Mögliche Ansätze

Zur Auswahl standen folgende Optionen:

1. Testbed so modifizieren, dass Durchläufe reproduzierbar werden
2. Einlesen der CSV Dateien in Monitore integrieren
3. Neue Komponente entwerfen, die zuvor aufgezeichnete Daten per Modbus bereitstellt

Option 1 wird nach kurzer Recherche ausgeschlossen, da Mosaik und vor allem die darin eingebundenen Simulatoren nicht ohne weiteres zu modifizieren sind. Diese Option ist im Rahmen dieser Arbeit zu aufwendig.

Als nächste Option steht die Implementierung wie bei Menzel et al. zur Auswahl. Bei genauerer Überlegung wäre dies nur unter erheblichem Aufwand realisierbar, da Synchronität der Messdaten von höchster Wichtigkeit ist, um eine zuverlässige Prüfung der Daten im Monitor zu gewährleisten. Das mindeste wäre hier eine Sekunden genaue Synchronisierung der Systemzeiten in allen Komponenten gepaart mit vordefinierten Zeitpunkten, zu denen ein neuer Datensatz geladen würde. Dies wäre allerdings ebenfalls eine sehr aufwendige und fehleranfällige Option.

Die letzte zur Debatte stehende Option ist es in einer neuen Komponente zwei Modbus Server zu konfigurieren und diese als Ersatz für das Testbed Daten in die Lokalmonitore einspeisen zu lassen. Diese Option kommt ohne komplexe Synchronisierung aus, da die Datenstrukturen in den Modbus-Servern in einem Thread nahezu zeitgleich aktualisiert werden können. Voraussetzung für Option 3 war somit die Auseinandersetzung mit den auch im Testbed verwendeten *pymodbus*[45] und *mosaikrtu*[46] Bibliotheken. Die Implementierung dieser Option wird nachfolgend geschildert.

5.3.2 Anforderungen an das simulierte Netz

Damit es möglich ist alle Regeln des IDS zu testen, ist es notwendig die Topologie des simulierten Netzes sorgfältig auszuwählen. Es müssen mindestens

zwei Lokalmonitore verwendet werden, damit zwischen ihnen eine Border Region entstehen kann. Nachfolgend werden einige Rahmenbedingungen für die zwei Teilnetze und die Border Regions definiert.

Lokalscope

- Mehrere Busse ohne Leitungen, die in der Border Region liegen
- Mindestens eine Leitung mit mehreren Messpunkten
- Mindestens eine Leitung mit Schalter und mehreren Messpunkten
- Mindestens ein Bus mit jeweils mehr als einer ein- und ausgehenden Leitung

Nachbarschaftsscope

- Mindestens eine Leitung mit Messpunkten aus beiden Teilnetzen
- Mindestens eine Leitung mit Sensor und Schalter in einem Teilnetz und einem Sensor im anderen Teilnetz

Der von Menzel et al. ausgewählte Ausschnitt aus der Mosaik-Simulation wurde bereits so ausgewählt, dass er alle zuvor definierten Anforderungen erfüllt. Demnach können auch die Szenarien von Menzel[2] wiederverwendet werden.

5.3.3 Implementierung des Replay-Modus

Wie das Testbed, wurde auch der *Replay*-Modus in Python implementiert. Das Laden der Konfigurationsdateien, das Erstellen des Speichers, den die Server verwenden und das erstellen der Server wurde unter Verwendung des *rtu_model* aus der *mosaikrtu* Bibliothek umgesetzt. Nach den Durchlauf eines Szenarios startet nach 10 Sekunden Pause das nächste Szenario, solange bis alle vier Szenarien abgearbeitet sind. (Ablauf in Quelltext 5.2)

Um die neue Komponente zu verwenden, kann auf dem Raspberry Pi mit dem Hostnamen *testbed* nun anstelle des Befehls *docker compose up testbed* der Befehl *docker compose up replay_csv* verwendet werden.

```
# Vier Szenarien durchspielen
for s in [1,2,3,4]:

    # Beide RTUs erstellen
    for i in [0,1]:
        # RTU config laden
        config[i] = load_rtu(<rtu_config_i>)

        # CSV-Dateien laden
        scenario[i] = read(<scenario_s_subgrid_i>)

        # Speicher der Modbus-Server erstellen
        datablock[i] = create_datablock(config[i])

        # Modbus-Server erstellen
        server[i] = create_server(config[i], datablock[i])

        server[i].start()

    # Beide RTUs mit den aufgezeichneten Daten updaten
    for j in range(1, scenario[0].length):

        # Daten updaten
        for i in [0,1]:
            datablock[i] = scenario[i][j]

        # Warten
        wait(2sec)

    # Zwischen Szenarien  warten
    wait(10sec)
```

Quelltext 5.2: Skizzierter Ablauf bei der Wiedergabe von aufgezeichneten Szenarien aus CSV Dateien. (*ids/replay_csv/replay.py*)

5.3.4 Szenarien

Alle vier Szenarien wurden sowohl verteilt, als auch zentral (also auf einem einzelnen Rechner) ausgeführt. Die Logs wurden der Konsole entnommen und anschließend zur besserem Lesbarkeit aufgearbeitet. Der Log der zentralen Ausführung ist in *log_single_host* zu finden, das Protokoll des verteilten Ausführens in *log_raspberry_pi*.

Nachfolgend ein Überblick über die Szenarien (wie in [2]), sowie die Resultate der Ausführung:

Szenario 1 - Normaler Betrieb

Erwartung:
Es sollten keine Anforderungsverstöße durch das IDS gefunden werden.
Manipulationen:
Keine
Ergebnisse:
Sowohl bei der zentralen, als auch bei der verteilten Ausführung, treten wie erwartet keine Anforderungsverstöße auf.

Szenario 2 - Manipulation nur im lokalen Scope

Erwartung:
Es sollten durch die Lokalmonitore Anforderungsverstöße gegen die Regeln 2, 3, 7 und 8 gefunden werden.
Manipulationen:
Einzelne Messdaten von Sensoren wurden verändert, sodass die Messwerte insgesamt nicht mehr zusammenpassen. Die Veränderungen sind allein auf den Kontext der anderen Messwerte bezogen fehlerhaft und die Manipulationen finden allein im Bereich der Teilnetze statt, der nicht auch Teil der Border Region ist.
Ergebnisse:
Grundsätzlich sind sich beide Protokolle sehr ähnlich. In beiden Fällen treten keine Verstöße im Nachbarschaftsscope auf und die gefundene Anzahl an Verstößen im Lokalscope weicht nur minimal ab. Diese Abweichung besteht darin, dass die verteilte Ausführung drei Verstöße mehr findet, allerdings liegt die Ursache vermutlich darin, dass der erste Datensatz doppelt gelesen wird, da die ersten sechs Violations identisch sind.
Auffällig ist, dass bei beiden Ausführungen keine Verstöße gegen Regel 3 gefunden werden. Diese soll Alarm schlagen, wenn Strom auf einer Leitung fließt, auf der ein Schalter geöffnet ist. Worin der Fehler besteht konnte nicht festgestellt werden und muss in Zukunft geklärt werden.
Eine weitere Abweichung besteht darin, dass eigentlich keine Verstöße gegen

Regel 4 (lokal) auftreten sollten, jedoch bei beiden Varianten gemeldet werden. Auch diese Regel scheint Überarbeitung zu benötigen.

Szenario 3 - Manipulationen im Lokal- und Nachbarschaftsscope

Erwartung:
Es sollten Verstöße durch alle Monitore im Nachbarschafts- und Lokalscope gefunden werden. Lokal sollten die Regel 2 verletzt werden, in den Nachbarschaftsmonitoren die Regeln 4 und 3.
Manipulationen:
Einzelne Messdaten von Sensoren wurden verändert, sodass die Messwerte insgesamt nicht mehr zusammenpassen. Die Veränderungen sind allein auf den Kontext der anderen Messwerte bezogen fehlerhaft und die Manipulationen finden in der Border Region statt.
Ergebnisse:
Bei diesem Szenario wurden die Spannungen an den Sensoren 10 und 13 manipuliert, was dazu führen sollte, dass die Die Regeln 2 (Lokal) und 4 (Nachbarschaft) Auffälligkeiten zeigen. Beide Protokolle zeigen korrekt, dass die Spannung an diesen Stellen nicht plausibel ist. Das Protokoll der verteilten Ausführung enthält allerdings zusätzlich eine Vielzahl an Verstößen gegen die Regel 4N auf der Leitungen 31 gibt, obwohl die Sensoren dort nicht manipuliert wurden.
Die erwarteten Verstöße gegen Regel 3 (Nachbarschaft) wurden in beiden Protokollen nicht detektiert. Bei einer ersten Untersuchung des Problems zeigte sich, dass die Daten des Schalters nicht korrekt im Monitor verfügbar sind. Wo die genaue Fehlerquelle liegt, konnte nicht festgestellt werden.
Ein auffälliger Unterschied zwischen den beiden Protokollen besteht darin, dass bei der zentralen Ausführung die einzelnen Log-Einträge von den Komponenten immer abwechselnd auftreten, während sie bei der verteilten Variante Block-artig stattfinden(siehe Quelltext 5.3). Dies kann darauf hindeuten, dass mindestens eine der Komponenten an der Grenze ihrer Belastbarkeit arbeitet.

```
1  # Auszug zentrale Ausführung
2  [LM 2] | Requirement 2 violated! Voltage on bus b24 measured by sensor_10
   ↪  : 10499.48 (!= 10499.58)
3  [NM 1] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.48 (!= 10499.58)
4  [NM 2] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.48 (!= 10499.58)
5  [NM 1] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.48 (!= 10499.58)
6  [NM 2] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.48 (!= 10499.58)
7
8  # Auszug verteilte Ausführung
9  [LM 2] | Requirement 2 violated! Voltage on bus b24 measured by sensor_10
   ↪  : 10499.48 (!= 10499.58)
10 [NM 1] | Requirement 4N violated! Voltage on line branch_40 measured by
   ↪  sensor_12 : 10499.47 (!= 10499.35)
11 [NM 1] | Requirement 4N violated! Voltage on line branch_31 measured by
   ↪  sensor_11 : 10499.47 (!= 10499.35)
12 [NM 1] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.47 (!= 10499.35)
13 [NM 2] | Requirement 4N violated! Voltage on line branch_40 measured by
   ↪  sensor_12 : 10499.47 (!= 10499.35)
14 [NM 2] | Requirement 4N violated! Voltage on line branch_31 measured by
   ↪  sensor_11 : 10499.47 (!= 10499.35)
15 [NM 2] | Requirement 4N violated! Voltage on line branch_39 measured by
   ↪  sensor_10 : 10499.47 (!= 10499.35)
```

Quelltext 5.3: Auszug aus den Log Dateien. Bei zentraler Ausführung wechseln sich die beiden Nachbarschaftsmonitore ab, während bei verteilter Ausführung immer Blöcke an Verstößen protokolliert werden.

Szenario 4 - Manipulation nur im Nachbarschaftsscope

Erwartung:
Es sollten Verstöße ausschließlich in der Border Region durch die Nachbarschaftsmonitore gefunden werden. Die Lokalmonitore sollten keine Verstöße feststellen.
Manipulationen:
Der Zustand eines Schalters in der Border Region wurde verändert, sodass dieser scheinbar offen ist. Damit der Lokalmonitor dies nicht detektiert, wurde der gemessene Strom des Sensors auf der Leitung, die sich im gleichen Teilnetz befindet, auf 0 Ampere gesetzt. Der zweite Sensor auf der Leitung befindet sich im anderen Teilnetz und wurde nicht manipuliert, also misst dieser einen Stromfluss größer als 0 Ampere.
Zusätzlich wurde ein Datensatz in einem Teilnetz wiederholt abgespielt (Replay-Attacke). Da der kopierte Datensatz grundsätzlich plausibel ist, aber eben nicht mehr zu dem im anderen Teilnetz passt, sollte dies durch den Nachbarschaftsmonitor bemerkt werden.
Ergebnisse:
Beide Protokolle sind nahezu identisch und enthalten Einträge aus den Nachbarschaftsmonitoren, die Unstimmigkeiten bei der Spannung melden. Demnach schlägt die Regel 4 (Nachbarschaft) an. Regel 3 (Nachbarschaft) sollte theoretisch ebenfalls anschlagen, allerdings wurde bereits bei der Auswertung von Szenario 3 erläutert, dass diese Regel nicht funktionstüchtig ist.
Beide Protokolle enthalten jeweils vier Einträge eines Verstoßes gegen Regel 4 (Lokal) durch den Lokalmonitor 1. Dieser wird scheinbar durch eine Abweichung der Spannung ausgelöst, allerdings ist nicht klar, wodurch diese Abweichung entsteht.

5.4 Auswertung

In diesem Kapitel wurden alle notwendigen Schritte dargestellt, um den Prototypen echt verteilt auf einem Cluster aus Raspberry Pis ausführbar zu machen. Des weiteren wurde eine Möglichkeit geschaffen das IDS mit aufgezeichneten Daten zu testen, wozu die neue Komponente *replay_csv* eingeführt wurde.

Mit diesen Änderungen ist es nun erstmals seit den Änderungen im Projektseminars möglich, den Prototypen wieder vergleichbar zu testen. Dabei ist aufgefallen, dass Regel 2 im Lokalmonitor falsch implementiert war, womit die Vermutung aus dem Projektseminar widerlegt wurde, die darin bestand, dass das Testbed die falsch-positiven Meldungen im Normalbetrieb auslöste.

Nach der Korrektur der Regel 2, wurde unter Verwendung der Szenarien von Menzel[2] gezeigt, dass nach wie vor weitere Fehler in der Implementierung vorhanden sind, auch wenn die genaue Ursache noch nicht geklärt ist. Dies sollte in Zukunft untersucht werden.

Der Vergleich zwischen zentraler und verteilter Ausführung hat gezeigt, dass bei verteilter Ausführung eine Art Blockbildung im Log zu beobachten ist. Bei einer Wiederholung der Szenarien konnte nicht beobachtet werden, dass die CPU-Auslastung ein kritisches Niveau erreicht. Auch die Temperaturen der CPUs steigen nicht über 65 Grad Celsius und liegen damit im Akzeptablen Bereich. Der Arbeitsspeicher wird zu weniger als 50 Prozent ausgelastet. Somit ist die Ursache vermutlich nicht die zu geringe Leistungsfähigkeit der verwendeten Hardware.

Bei der Untersuchung der Netzwerkaktivität der Docker Container zeigt sich, dass diese sich - wie bereits in der Anforderungsanalyse angegeben - lediglich im Bereich weniger 100 Kbit/s liegt. Dies kann demnach auch nicht die Ursache der Auffälligkeit sein. Da die Blöcke im Protokoll die Ausführung allerdings in der Funktion nicht beeinträchtigen, wird diese Arbeit sich nicht tiefer mit der Problematik befassen. Wo die tatsächliche Ursache liegt, gilt es in Zukunft herauszufinden.

Es kann festgehalten werden, dass nahezu alle Anforderungsverletzungen, die bei zentraler Ausführung auftreten, auch bei der verteilten Ausführung gefunden wurden. Dies deutet darauf hin, dass es kein grundsätzliches Problem bei der Anwendung des durch Menzel et al. entworfenen Konzepts in einer verteilten Umgebung gibt.

6 Fazit

6.1 Zusammenfassung

In der heutigen Zeit werden die hochgradig komplexen Elektrizitätsnetze durch SCADA-Systeme gesteuert. Diese stellen unter Betrachtung von Aspekten der IT-Sicherheit eine Gefahr dar, da sie häufig dem nicht mehr zeitgemäßen Paradigma *security through obscurity* folgen. Zur Erhöhung dieser Sicherheit wurde durch Menzel et al. das auch in dieser Arbeit behandelte Intrusion Detection System entworfen.

Kapitel 2 stellt den Entstehungsprozess des Systems bis zum Beginn der vorliegenden Arbeit dar. Begonnen wird mit der Entwicklung des ursprünglichen Konzepts, des SAM und des Testbeds durch Chromik et al., das auch im aktuellen IDS Prototypen noch auf lokaler Ebene angewendet wird. Danach wird auf die Änderungen durch Menzel et al. eingegangen, die zwei unterschiedliche Arten von Monitoren eingeführt haben und so noch mehr Angriffe detektieren konnten. Es wird an dieser Stelle auch der verwendete Regelkatalog vorgestellt. Der Prototyp des IDS wurde anschließend im Rahmen des Projektseminars so modifiziert, dass die einzelnen Monitore logisch getrennte Einheiten darstellen. Des weiteren wurde dabei eine sichere Kommunikationsinfrastruktur ergänzt, sodass nun die Komponenten des IDS geschützt kommunizieren können.

Das dritte Kapitel stellt mit einem Fokus auf die technische Umsetzung den Aufbau des IDS dar. Dabei wird eine kurze Übersicht über das Testbed gegeben, dann aber der Fokus auf die Funktionalität der einzelnen Komponenten des IDS gelegt. Es werden für alle Komponenten Aufgaben definiert und anschließend das Zusammenspiel der Komponenten untereinander, sowie die Funktion des verwendeten OPC-UA Protokolls anhand von Beispielszenarien erläutert. Das Deployment mit Hilfe von Docker wird darauf folgend thematisiert. Die verwendete Containervirtualisierung wird anhand des C2-Servers erläutert und abschließend ein Überblick über die Struktur des Deployments gegeben.

In der Anforderungsanalyse wird im vierten Kapitel eine Reihe an Bedingungen formuliert, die das IDS erfüllen sollte, wenn man es in der Realität einsetzen wollen würde. Dabei wird zuerst der Rahmen abgesteckt, in dem das IDS realisierbar wäre. Anschließend werden die Anforderungen an eine Umsetzung der Monitore und des C2-Servers erfasst. Dabei werden Hardware und Konnekti-

vität unter Berücksichtigung der Wirtschaftlichkeit erläutert. Darauf folgend werden die Anforderungen an die Sicherheit des IDS unter Zuhilfenahme des CIA-Modells untersucht. Bezüglich der Vertraulichkeit wird insbesondere auf die Wichtigkeit der sicheren Verwahrung der kryptografischen Schlüssel aufmerksam gemacht. Die Kommunikation ist bereits gegen die meisten Angriffe ausreichend geschützt.

Das fünfte Kapitel schildert die erstmalige tatsächlich verteilte Ausführung des IDS. Dazu wird auf Raspberry Pis zurückgegriffen und das IDS entsprechend an die verteilte Umgebung angepasst. Es wird hier auch eine neue Komponente vorgestellt, die bei der Durchführung einer Leistungsmessung das Testbed ersetzt und zuvor aufgezeichnete Daten wiedergibt. Zur Leistungsmessung wurden die Szenarien aus der Masterarbeit von Menzel wiederverwerwendet und ein Vergleich zwischen zentraler und verteilter Ausführung vorgenommen. Dies offenbarte, dass einige der Anforderungen nicht korrekt implementiert sind, allerdings grundsätzlich die verteilte Ausführung die Funktion des IDS nicht beeinträchtigt.

6.2 Beantwortung der Forschungsfragen

Welche technischen Anforderungen in Bezug auf Hard- und Software hat das System, wenn man es in der Realität einsetzen möchte?

Bei einer Anwendung in der Realität gilt es eine Vielzahl von Aspekten zu beachten. Ein solcher Einsatz wäre mit einer sehr viel größeren Anzahl Monitore verbunden. Bei der Auswahl der Monitore muss demnach ein Gleichgewicht zwischen Kosten und Nutzen gefunden werden, was sich vor allem auf die verwendete Hardware auswirkt. Diese kann sehr kompakt ausfallen und kommt auch mit geringer Rechenleistung und Netzwerk-Bandbreite aus. Der C2-Server im Gegensatz dazu sollte über gewisse Leistungsreserven verfügen, da er zu jedem Zeitpunkt in der Lage sein muss gemeldete Anforderungsverstöße zu verarbeiten. Er sollte in einem Rechenzentrum aufgestellt werden.

Bei der Sicherheit des IDS sind zwei Aspekte besonders relevant. Zum einen sollte das IDS eine hohe Verfügbarkeit aufweisen, was durch Redundanzen bei Energie- und Internet-Versorgung gewährleistet wird. Zum anderen sollte das IDS keine neuen Schwachstellen im System eröffnen. Da das System Informationen über die Architektur der SCADA-Netze und deren aktuellen Zustand beinhaltet, ist darauf zu achten, dass die Kommunikation innerhalb des Systems abhör- und manipulationssicher abgewickelt wird. Dazu muss berücksichtigt werden, dass ein Angreifer beim entwenden der Hardware nicht in den Besitz wertvoller Informationen wie bspw. kryptographischer Schlüssel gelangen kann.

Welche Änderungen sind notwendig, um das System auf einem Testaufbau aus Raspberry Pis lauffähig zu machen und welche Mindestanforderungen im Bezug auf Anzahl der Komponenten und Topologie des simulierten Netzes muss ein solcher Testaufbau erfüllen, damit es sinnvolle Ergebnisse erzielt?

Die bisherige Konfiguration des IDS war darauf ausgelegt, zentral auf einem Rechner ausgeführt zu werden. Die ARM Architektur der Pis macht es notwendig die Dockerfiles so zu erweitern, dass der Paketmanager *pip* passend compilierte Wheels finden kann. Alle Adressen der Komponenten müssen angepasst werden und darüber hinaus die OPC-UA-Server so abgeändert werden, dass sie zwischen einer internen und externen Adresse unterscheiden können.

Bezüglich der Anzahl an Komponenten wurde der bisherige Umfang des IDS beibehalten, sodass weiterhin jeweils zwei Lokal- und Nachbarschaftmonitore, der C2-Server, die Visualisierung und das Testbed im Prototypen ausgeführt wird. Allerdings wurde es durch die beschriebenen Änderungen ermöglicht, dass jeweils einen Lokal- und Nachbarschaftsmonitor zusammen ausgeführt werden können, sodass insgesamt lediglich vier Raspberry Pis benötigt werden.

Da das bisher simulierte Netz sorgfältig durch Menzel et al. ausgewählt wurde und bereits das Prüfen aller Anforderungen ermöglicht, war es nicht nötig dieses zu verändern.

Inwiefern beeinflusst die tatsächlich verteilte Ausführung auf getrennter Hardware die Funktionalität des Systems?

Bei der Ausführung der Test-Szenarien war zu beobachten, dass die grundsätzliche Funktionalität erhalten bleibt und das verteilte IDS nahezu alle Verstöße findet, die auch das Zentrale IDS detektiert. Auffällig ist, dass die Log-Einträge auf Seiten des C2-Servers in leicht veränderte Reihenfolge erfolgen und dabei Blöcke an Nachrichten der einzelnen Komponenten entstehen.

Einige der Regeln konnten nicht getestet werden, da sich herausstellte, dass sie fehlerhaft implementiert sind. Es ist allerdings nicht damit zu rechnen, dass diese von erhöhten Latenzen einer verteilten Ausführung anders beeinflusst wären, als es die funktionierenden Regeln sind.

Beim Auslesen der CPU- und Netzwerkauslastung der Raspberry Pis ist zu beobachten, dass sie weit unterhalb der Leistungsgrenzen liegen. Der Leistungsbedarf steigt allerdings mit der Anzahl der Komponenten an, sodass diese Aussage lediglich für das verwendete Szenario Gültigkeit besitzt.

Literaturverzeichnis

[1] Chromik, Justyna Joanna: *Process-aware SCADA traffic monitoring: A local approach.* Netherlands, University of Twente, Diss., Juli 2019. http://dx.doi.org/10.3990/1.9789036548014. – DOI 10.3990/1.9789036548014

[2] Menzel, Verena: *A hierarchical approach to monitoring SCADA networks*, WWU Münster, Diplomarbeit, 2021

[3] Deuschle, Tom ; Adick, Piet ; Speckamp, Jan: Abschlussbericht Projektseminar: IT-Sicherheit für Stromnetze / WWU Münster. 2022. – Forschungsbericht

[4] Efa: *Image published on wikipedia.* https://commons.wikimedia.org/wiki/File:RaspberryPi_3B.svg. – [Online; Stand 15. September 2022]; Lizenz: CC BY-SA 3.0 https://creativecommons.org/licenses/by-sa/3.0

[5] Chromik, Justyna J. ; Remke, Anne ; Haverkort, Boudewijn R.: What's under the hood? Improving SCADA security with process awareness. In: *2016 Joint Workshop on Cyber-Physical Security and Resilience in Smart Grids (CPSR-SG)* IEEE, 2016, S. 1–6

[6] Chromik, Justyna J. ; Remke, Anne ; Haverkort, Boudewijn R.: An integrated testbed for locally monitoring SCADA systems in smart grids. In: *Energy Informatics* 1 (2018), Nr. 1, S. 1–29

[7] Chromik, Justyna J. ; Pilch, Carina ; Brackmann, Pascal ; Duhme, Christof ; Everinghoff, Franziska ; Giberlein, Artur ; Teodorowicz, Thomas ; Wieland, Julian ; Haverkort, Boudewijn R. ; Remke, Anne: Context-aware local Intrusion Detection in SCADA systems: a testbed and two showcases. In: *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)* IEEE, 2017, S. 467–472

[8] Short, Joe A. ; Infield, David G. ; Freris, Leon L.: Stabilization of Grid Frequency Through Dynamic Demand Control. In: *IEEE Transactions on Power Systems* 22 (2007), Nr. 3, S. 1284–1293. http://dx.doi.org/10.1109/TPWRS.2007.901489. – DOI 10.1109/TPWRS.2007.901489

[9] Menzel, Verena u. a.: *Abstracts of the 11th DACH+ Conference on Energy Informatics*. http://dx.doi.org/10.1186/s42162-022-00215-6. Version: 2022

[10] *opcua-asyncio Github*. https://github.com/FreeOpcUa/opcua-asyncio/. – [Online; Stand 15. September 2022]

[11] *Mosaik Dokumentation*. https://mosaik.readthedocs.io/en/latest/overview.html

[12] Scherfke, Stefan u. a.: *mosaik-pypower 0.8.1 project description*. https://gitlab.com/mosaik/components/energy/mosaik-pypower/-/blob/0dcfd46eae15db69f37c20a18fa8c3a81f2361eb/README.rst

[13] Chromik, Justyna J. ; Pilch, Carina ; Brackmann, Pascal ; Duhme, Christof ; Everinghoff, Franziska ; Giberlein, Artur ; Teodorowicz, Thomas ; Wieland, Julian ; Haverkort, Boudewijn R. ; Remke, Anne: Context-aware local Intrusion Detection in SCADA systems: A testbed and two showcases. In: *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2017, S. 467–472

[14] Menzel, Verena ; Hurink, Johann L. ; Remke, Anne: 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, S. 405–411

[15] Radoglou-Grammatikis, Panagiotis I. ; Sarigiannidis, Panagiotis G.: Securing the Smart Grid: A Comprehensive Compilation of Intrusion Detection and Prevention Systems. In: *IEEE Access* 7 (2019), S. 46595–46620. http://dx.doi.org/10.1109/ACCESS.2019.2909807. – DOI 10.1109/ACCESS.2019.2909807

[16] *Unified Architecture - OPC Foundation*. https://opcfoundation.org/about/opc-technologies/opc-ua/. Version: Sep 2019. – [Online; Stand 1. September 2022]

[17] Graube, Markus ; Hensel, Stephan ; Iatrou, Chris ; Urbas, Leon: Information models in OPC UA and their advantages and disadvantages. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, S. 1–8

[18] *Reference of OPC-UA*. https://reference.opcfoundation.org/Core/Part3/. – [Online; Stand 1. September 2022]

[19] Wikipedia: *Remote Procedure Call — Wikipedia, die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Remote_

Procedure_Call&oldid=213190158. Version: 2021. – [Online; Stand 8. September 2022]

[20] *Docker Overview*. https://docs.docker.com/get-started/overview/. – [Online; Stand 1. September 2022]

[21] Wikipedia: *Containervirtualisierung — Wikipedia, die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Containervirtualisierung&oldid=225746071. Version: 2022. – [Online; Stand 29. August 2022]

[22] *Docker Compose Overview*. https://docs.docker.com/compose/reference/. – [Online; Stand 1. September 2022]

[23] Smith, Brad: ARM and Intel Battle over the Mobile Chip's Future. In: *Computer* 41 (2008), Nr. 5, S. 15–18. http://dx.doi.org/10.1109/MC.2008.142. – DOI 10.1109/MC.2008.142

[24] Hamra, Sam: *Ethical hacking: Threat modeling and penetration testing a remote terminal unit*. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1517798&dswid=-2939. Version: 2020

[25] Wikipedia contributors: *Uninterruptible power supply — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Uninterruptible_power_supply&oldid=1099030980. Version: 2022. – [Online; accessed 19-September-2022]

[26] Europäische Kommission and Generaldirektion Informationsgesellschaft und Medien: *Information Technology Security Evaluation Criteria (ITSEC) : Provisional harmonised criteria*. Publications Office, 1994

[27] von Solms, Rossouw ; van Niekerk, Johan: From information security to cyber security. In: *Computers & Security* 38 (2013), 97-102. http://dx.doi.org/https://doi.org/10.1016/j.cose.2013.04.004. – DOI https://doi.org/10.1016/j.cose.2013.04.004. – ISSN 0167–4048. – Cybercrime in the Digital Economy

[28] Informationstechnik, Bundesamt für Sicherheit in d.: *DDoS-Anfriffe im Cyber-Raum*. https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Gefaehrdungen/DDoS/ddos_node.html. Version: 2022. – [Online; Stand 11. September 2022]

[29] Bertino, Elisa ; Islam, Nayeem: Botnets and Internet of Things Security. In: *Computer* 50 (2017), Nr. 2, S. 76–79. http://dx.doi.org/10.1109/MC.2017.62. – DOI 10.1109/MC.2017.62

[30] Baker, F. ; Savola, P.: *Ingress Filtering for Multihomed Networks*. https://www.rfc-editor.org/info/rfc3704. Version: 2004. – [Online; Stand 12. September 2022]

[31] Wikipedia contributors: *Denial-of-service attack — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=1107569369. Version: 2022. – [Online; accessed 19-September-2022]

[32] Radoglou-Grammatikis, Panagiotis I. ; Sarigiannidis, Panagiotis G.: Securing the Smart Grid: A Comprehensive Compilation of Intrusion Detection and Prevention Systems. In: *IEEE Access* 7 (2019), S. 46595–46620. http://dx.doi.org/10.1109/ACCESS.2019.2909807. – DOI 10.1109/ACCESS.2019.2909807

[33] Shirey, R.: *RFC 4949*. https://www.rfc-editor.org/rfc/rfc4949. Version: 2007. – [Online; Stand 11. September 2022]

[34] Wikipedia: *Authentizität — Wikipedia, die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Authentizit%C3%A4t&oldid=224983937. Version: 2022. – [Online; Stand 11. September 2022]

[35] Dorp, Johannes vom ; Merschjohann, Sven ; Meier, David ; Patzer, Florian ; Karch, Markus ; Haas, Christian: *Sicherheitsanalyse (2021) Open Platform Communications Unified Architecture (OPC UA)*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/OPCUA/OPCUA_2022.html. Version: 2021. – [Online; Stand 11. September 2022]

[36] Casey, Eoghan ; Stellatos, Gerasimos J.: The Impact of Full Disk Encryption on Digital Forensics. In: *SIGOPS Oper. Syst. Rev.* 42 (2008), apr, Nr. 3, 93–98. http://dx.doi.org/10.1145/1368506.1368519. – DOI 10.1145/1368506.1368519. – ISSN 0163–5980

[37] International Organization for Standardization: Information technology — Trusted platform module library — Part 1: Architecture / International Organization for Standardization. 2015. – Standard

[38] Wikipedia: *Trusted Platform Module — Wikipedia, die freie Enzyklopädie*. https://de.wikipedia.org/w/index.php?title=Trusted_Platform_Module&oldid=224193152. Version: 2022. – [Online; Stand 11. September 2022]

[39] Baker, Paul R. ; Benny, Daniel J.: *The complete guide to physical security*. Boca Raton : CRC Press, 2013. – ISBN 9781420099638

[40] Sykosch, Arnold ; Doll, Christian ; Wübbeling, Matthias ; Meier, Michael: Generalizing the Phishing Principle: Analyzing User Behavior in Response to Controlled Stimuli for IT Security Awareness Assessment. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. New York, NY, USA : Association for Computing Machinery, 2020 (ARES '20). – ISBN 9781450388337

[41] Wikipedia contributors: *Raspberry Pi — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=1110318678`. Version: 2022. – [Online; accessed 15-September-2022]

[42] Raspberry Pi Foundation: *Raspberry Pi Documentation*. `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-3-model-b-2`. – [Online; accessed 15-September-2022]

[43] Raspberry Pi Foundation: *Raspberry Pi Imager*. `https://www.raspberrypi.com/software/`. – [Online; accessed 19-September-2022]

[44] *Github Issue: Unable to set custom url as endpoint*. `https://github.com/FreeOpcUa/opcua-asyncio/issues/232`. Version: 2021. – [Online; Stand 15. September 2022]

[45] Github Contributors: *Github - pymodbus Project*. `https://github.com/riptideio/pymodbus`

[46] Chromik, Justyna: *Github - mosaik-cosim*. `https://github.com/jjchromik/mosaik-cosim`

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *„Anforderungsanalyse für ein verteiltes IDS und Entwicklung eines Prototypen auf Raspberry Pis“* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Kai Oliver Großhanten, Münster, 20. September 2022

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Kai Oliver Großhanten, Münster, 20. September 2022