UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

LEHRSTUHL FÜR SOFTWARE-ENGINEERING,
INSB. MOBILE ANWENDUNGEN
PROF. DR. V. GRUHN

Universität Duisburg-Essen
Fakultät für Wirtschaftswissenschaften
Erlangung des akademischen Grades
DOCTOR RERUM NATURALIUM (Dr. rer. nat.)

Information Flow Monitoring in Cyber-Physical Systems

Nachvollziehen von Cascading Data Corruption in CPS

D I S S E R T A T I O N

Stefan Gries, M. Sc.
geboren in Duisburg, Deutschland

Essen, Januar 2021

1. Gutachter: Prof. Dr. Volker Gruhn,
Universität Duisburg Essen
2. Gutachter: Prof. Dr. Matthias Book,
Universität Island
Tag der mündlichen Prüfung: 26.05.2021

DOI: 10.17185/duepublico/74445
URN: urn:nbn:de:hbz:464-20210709-081420-8

Zusammenfassung

Cyber-Physical Systems (CPS) verbinden physische Prozesse der echten Welt mit digitalen Netzwerken und Rechensystemen. So wird es möglich, echtweltliche Prozesse mit Hilfe von Sensoren zu überwachen und Aktoren zu steuern. CPS bestehen dabei aus vielen einzelnen Systemen, die über ein Netzwerk verbunden sind und Daten austauschen. Jedes System hat hierbei eine eigene Aufgabe. Manche messen mit Hilfe von Sensoren Produktionsabläufe oder Umweltparameter, andere nutzen diese Daten, um daraus Entscheidungen und neue Informationen abzuleiten, während weitere auch Motoren oder sonstige Aktoren steuern, um in Prozesse der echten Welt physisch einzugreifen. Die Anwendungsdomänen hierbei sind vielfältig und reichen vom Smart Home, über die Industrie 4.0 mit intelligenten Fabriken, bis zu autonomen Fahrzeugen und moderner Medizintechnik.

Da diese verschiedenen Systeme vernetzt sind, zusammenarbeiten und Daten austauschen, werden die Systeme auch voneinander abhängig. Benötigt ein System beispielsweise Sensordaten eines anderen Systems, um Entscheidungen zu treffen, wirken sich ausgetauschte falsche oder fehlende Werte auch auf diese Entscheidungen aus. Solche Fehler können sich schnell im Netzwerk von System zu System ausbreiten. Der Fehler kaskadiert in Form von fehlerhaften Informationen durch das Netzwerk. Dieses Phänomen wird in dieser Arbeit mit dem Begriff *Cascading Data Corruption* bezeichnet.

Um dieses Kaskadieren des Fehlers im Netzwerk zu beheben, muss die Quelle des Fehlers identifiziert werden. Der Ort, an dem der Fehler sichtbar wird, beispielsweise an einer Maschine in der physischen Welt, ist jedoch oft nicht die Quelle. Stattdessen hat diese Maschine nur Befehle ausgeführt, die zuvor von vielen anderen Systemen ausgetauscht und verarbeitet wurden. Die Identifizierung der ursprünglichen Fehlerquelle ist schwierig, da CPS stark heterogen vernetzt sind und Datenflüsse weder immer gleich noch überhaupt bekannt sind.

Diese Arbeit stellt dazu das Konzept und Werkzeug des Information Flow Monitors (IFM) vor. Der Information Flow Monitor (IFM) ist in der Lage in dezentralen CPS-Netzwerken den Informationsaustausch der Komponenten zu überwachen und aufzuzeichnen. So wird es möglich, nach dem Auftreten eines Fehlers retrospektiv zu analysieren, welche Informationen genutzt wurden, um die als fehlerhaft identifizierte Information zu erstellen. Der IFM führt so dazu, dass Abhängigkeiten zwischen ausgetauschten Informationen bekannt sind und Fehlerquellen im Netzwerk identifiziert werden können. Durch den IFM kann so Ursache und Wirkung nachvollzogen werden.

Um dies zu erreichen, beinhaltet der IFM ein Protokoll, das die einzelnen Knoten im Netzwerk implementieren. Hierdurch wird es möglich, über Knoten hinweg Abhängigkeiten zwischen den ausgetauschten Informationen zu erfassen. Anschließend können Abhängigkeitsbäume für die zu untersuchenden Informationen automatisiert erstellt werden. Diese Abhängigkeitsbäume helfen bei der Identifizierung von Fehlerquellen im Netzwerk und stellen einen Ursache-Wirkungs-Graphen über die ausgetauschten Informationen dar.

Das IFM-Konzept ist zentraler Beitrag dieser Arbeit. Es wird in diesem Rahmen sowohl motiviert und spezifiziert als auch anhand von Beispielen, Anwendungsfällen und Experimenten evaluiert.

Abstract

Cyber-Physical Systems (CPS) connect physical processes of the real world with digital networks and computing systems. This makes it possible to monitor real-world processes using sensors and to control actuators. CPS consist of many individual systems that are connected and exchange data via a network. Each system has its own task. Some measure production processes or environmental parameters with the help of sensors, others use this data to derive decisions and new information, while some also control motors or other actuators to physically intervene in real-world processes. The areas of application are manifold and range from the smart home, to Industry 4.0 with intelligent factories, to autonomous vehicles and modern medical technology.

As these different systems are networked, work together and exchange data, the systems also become interdependent. For example, if one system needs sensor data from another system to make decisions, exchanged incorrect or missing information will also affect these decisions. Such errors can quickly spread from one system to another system in the network. The error cascades through the network in the form of incorrect information. This phenomenon is referred to in this thesis as *Cascading Data Corruption*.

To correct this cascading of the error in the network, the source of the error must be identified. However, the location where the error becomes visible, for example, on a machine in the physical world, is often not the source. Instead, this machine has only executed commands that were previously exchanged and processed by many other systems. Identifying the original source of the error is difficult because CPS are highly heterogeneously networked and data flows are neither always the same nor known at all.

This thesis introduces the concept and tools of the Information Flow Monitor (IFM). The IFM is able to monitor and record the information exchange of the

components in decentralized CPS networks. In case of an error, this makes it possible to retrospectively analyze, which information was used to create the information identified as faulty. The IFM thus reveals dependencies between exchanged information and helps to identify sources of errors in the network. The IFM thereby enables cause and effect to be traced.

To achieve this, the IFM contains a protocol that the individual nodes in the network implement. This makes it possible to capture dependencies between the exchanged information across nodes. It then becomes possible to automatically create dependency trees of the information to be examined. These dependency trees help to identify sources of error in the network and provide a cause-and-effect graph of the exchanged information.

The IFM concept is the central contribution of this work. It is motivated and specified as well as evaluated by examples, use cases and experiments.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel: Allgemeine Einführung in das Themengebiet der Cyber-Physical Systems (CPS) und der Abhängigkeiten in verteilten Netzwerken. Beschreibung der Motivation, Problemstellung und Zielsetzung der Arbeit.

Der Begriff Cyber-Physical System entstand vermutlich 2006 und wurde von Helen Gill an der National Science Foundation geprägt [1, S. 5][2]. Andere benennen Edward A. Lee als Schöpfer des Begriffs, der ihn zu einem ähnlichen Zeitpunkt benutzte [3].

Ausgangspunkt für die Verbreitung des Begriffs in Forschung, Wissenschaft und Wirtschaft waren die USA, die frühzeitig damit begonnen haben, in Forschungsprojekte zum Thema CPS zu investieren. 2007 wurde in den USA auf Empfehlung des President's Council of Advisors on Science and Technology (PCAST) ein Forschungsprogramm eingerichtet. Die National Science Foundation (NSF) führte in diesem Förderprogramm seit 2009 über 65 Projekte zum Thema CPS durch, die mit fast 60 Millionen US-Dollar gefördert wurden. Das PCAST verlängerte 2010 das Förderprogramm, da weiterer Förderbedarf für CPS festgestellt wurde. So wurden weitere 30 Millionen US-Dollar in die CPS-Forschung investiert. [3]

In den darauffolgenden Jahren wurde der Fokus auch in Deutschland auf die CPS gerichtet. Im Rahmen der Deutsche Akademie der Technikwissenschaften (acatech)-Studien [4, 5] wurde eine Forschungsagenda für CPS in Deutschland erarbeitet, die sowohl auf die wirtschaftliche als auch auf die gesellschaftliche Bedeutung eingeht. Hierbei wurde insbesondere auch der Forschungsbedarf hervorgehoben. In Deutschland entstanden anschließend mehrere Forschungsprojekte, beispielsweise CPS.HUB NRW [6], an dem auch die Universität Duisburg-Essen beteiligt war.

In Deutschland rechtfertigt sich die Forschung insbesondere durch den Bedarf an Digitalisierung im Mittelstand, der Automatisierung der Produktion (Industrie 4.0) sowie der sogenannten Smartifizierung der Gesellschaft (Smart Home / Smart City):

> „Durch die Möglichkeiten der vernetzten Datengewinnung und der interaktiven Begleitung von Kunden- und Nutzerprozessen ergeben sich große Potenziale für Innovationen und neue Geschäftsmodelle. Information samt ihrer intelligenten Verarbeitung wird zunehmend zum erfolgsentscheidenden Wettbewerbsfaktor. Dazu gehört die sinnvolle Nutzung von Daten und Informationen, wie sie in Cyber-Physical Systems entstehen." – Geisberger und Broy, AgendaCPS [4, S. 23]

CPS gelten als erfolgskritischer Faktor für moderne Geschäftsprozesse. Sie bilden die Grundlage für das Internet of Things (IoT) und Industrial Internet (Vereinigte Staaten) bzw. Industrie 4.0 (Deutschland) [7, 8]. Produktionsprozesse können stärker automatisiert und individualisiert werden, wenn CPS zum Einsatz kommen. Hierzu werden Daten aus physischen Prozessen digitalisiert und zur Entscheidungsfindung in der Prozesssteuerung eingesetzt.

Es gibt jedoch eine Kluft zwischen der digitalen Welt, in der CPS Informationen verarbeiten und austauschen, und der physischen Welt, in der wir leben. Diese Kluft soll durch CPS überbrückt werden, da Prozessautomatisierung nur dann gelingen kann, wenn physische Eigenschaften, die zur Steuerung dieser Prozesse benötigt werden, auch in digitaler Form vorliegen [9]. Dies geschieht in CPS durch Sensorik[1]. Der Rückkanal zur Steuerung der physischen Welt wird durch Aktorik[2] realisiert.

[1] Sensoren oder Sensorik im Allgemeinen bezeichnet Hardware, die genutzt wird, um Umweltparameter wie die Temperatur oder Luftfeuchtigkeit zu digitalisieren und so in Computern nutzbar zu machen. Der Begriff wird in Abschnitt 2.1.3 diskutiert.

[2] Aktoren sind Hardware, die in der physischen Welt Aktionen ausführen. Dies können beispielsweise digital steuerbare Motoren sein. Der Begriff wird in Abschnitt 2.1.3 beleuchtet.

CPS als Vision für neue Prozesse und Systeme

Basierend auf der aktuellen Geschwindigkeit der Veränderung von Technik ist davon auszugehen, dass binnen 25 Jahren die Computer in der Welt um uns herum hauptsächlich vernetzte Geräte sind, die miteinander kommunizieren [10, S. 71]. Es wird sich also nicht mehr um getrennte Systeme handeln, die Entscheidungen für sich selbst treffen. Sie werden interagieren und mit einer globalen Plattform verbunden sein. Heutige Autos oder Flugzeuge sind letztendlich kleine CPS, die jedoch nur sehr eingeschränkt mit einer globalen Plattform verbunden sind und größtenteils unabhängig agieren. Überträgt man die Funktionalität und Struktur solcher Systeme auf eine globale Ebene erhalten wir eine Vision von großen CPS [11]. CPS und Informationstechnik (IT) im Allgemeinen werden damit allgegenwärtig – auch in Branchen, die sich vorher nicht mit IT beschäftigt haben oder diese nur zur Prozessoptimierung genutzt haben. Nahezu jede Branche nutzt aber IT nicht nur zur Prozessoptimierung, sondern auch zur Produktoptimierung. Hierdurch werden CPS Teil von Produkten dieser Branchen. [5, S. 13]

Es ist davon auszugehen, dass die Allgegenwärtigkeit von Sensorik und Aktorik durch die Einbettung dieser Geräte sowie von Kommunikations- und Berechnungsinfrastruktur in alltägliche Umgebungen und Objekte der physischen Umgebung realisiert wird [12]. Der Trend zur Produkt- und Prozessoptimierung durch IT wurde bereits in den vergangenen Jahrzehnten immer sichtbarer [13]. Der hierdurch eingeleitete Wandel der Wirtschaft ist keine Wahlmöglichkeit, sondern erforderlich, um mit der Konkurrenz Schritt zu halten. Das breite Spektrum an erforderlichen Fähigkeiten wird durch CPS ermöglicht.

Durch ihre Allgegenwärtigkeit werden CPS daher die Art und Weise, wie Menschen untereinander, Menschen mit Maschinen und Menschen mit ihrer physischen Welt interagieren, nachhaltig verändern [12]. Gleichzeitig wird die Verbreitung von CPS nicht nur durch den Bedarf, sondern durch aktuell günstige Rahmenbedingungen vorangetrieben. Sensoren, Aktoren und Computerhardware im Allgemeinen werden immer kostengünstiger und leistungsfähiger. Gleichzeitig nehmen der Platzbedarf und Stromverbrauch ab. Energie kann besser in Akkus gespeichert werden. Drahtlose Kommunikation wird immer günstiger und Internetbandbreite erschwinglich. Hierdurch wird eine Verbreitung vieler kleiner Geräte und das Zusammenfügen zu einem CPS begünstigt. [12]

All dies gemeinsam führt dazu, dass CPS das Potenzial haben große Veränderungen bzgl. der Integration von Computern in den Alltag herbeizuführen [14]. CPS umfassen alle Fachdomänen wie beispielsweise medizinische Geräte, kritische Systeme, Wohnen, Verkehr, Sicherheit, Automobilität, Prozesssteuerung, Energieeinsparung, Umwelttechnik, Avionik, Steuerung kritischer Infrastrukturen (z. B. Strom und Wassernetze), Robotik, Verteidigung und Fertigung [14]. Einige Domänen werden stellvertretend in Abschnitt 2.1.5 vorgestellt.

CPS als Verknüpfung zwischen physischer und digitaler Welt

CPS versuchen die Kluft zwischen digitaler und realer Welt zu verringern, indem Informationen aus der realen Welt digitalisiert werden, um diese verarbeiten zu können [13]. Gleichzeitig können durch die Steuerung von Maschinen und Geräten, die echtweltliche Aktionen ausführen, digitale Entscheidungen auch an die reale Welt zurückgegeben werden. So ergibt sich zwischen der digitalen und analogen Welt ein Austausch großer Mengen an Daten, um CPS zu realisieren [15, S. 398].

Üblicherweise werden in Software-Systemen Objekte der realen Welt durch Modelle abgebildet. Bestimmte Eigenschaften eines physischen Objektes werden über Variablen an einen digitalen Zwilling, ein Modell, gebunden. Der digitale Zwilling repräsentiert so das realweltliche Objekt. Es gibt aber keine direkte Verknüpfung zwischen digitaler und realer Welt. CPS jedoch nutzen Sensoren und Aktoren, um reale Objekte in Software auswertbar zu machen. Bestimmte durch Sensorik erfassbare Daten werden aus der realen Welt direkt mit Software verknüpft, sodass die beiden Welten miteinander kommunizieren können. Informationsmodell und zugehörige physische Instanzen rücken nah zusammen und haben eine direkte Rückkopplung.

Definition von CPS

CPS und ihre Eigenschaften werden von unterschiedlichen Forschungsgruppen teils verschieden definiert. Auf Basis der zuvor beschriebenen Eigenschaften von CPS wird im Rahmen dieser Arbeit die folgende Definition für CPS als Grundlage verwendet:

> CPS sind vernetzte Systeme, die reale Objekte und Prozesse beobachten und beeinflussen. Sie sind in der Lage, sich der aktuellen realen Umgebung und ihren Bedingungen selbstständig anzupassen. Außerdem können CPS aus vielen heterogenen Komponenten bestehen. In manchen CPS verändern diese Komponenten dynamisch zur Laufzeit ihr Verhalten. Durch die Veränderung der Zusammenstellung autonomer Komponenten können manche CPS emergentes Verhalten zeigen und so zur Laufzeit neue Lösungsstrategien entwickeln.
>
> *Angelehnt an Hesenius und Gries et al.,*
> *Kerntechnologien für CPS* [16]

Die Definition wird in Abschnitt 2.1 mit Hilfe verschiedener in der Literatur genannter Eigenschaften von CPS und anderen CPS-Definition hergeleitet.

1.1 Motivation

Das Zusammenspiel aus Systemen in CPS und anderen Systemen, die über Netzwerke kommunizieren und mit echtweltlichen Objekten interagieren, findet nicht in einer kontrollierbaren Umgebung statt, auf die die Teilnehmer vollumfassend steuernden Einfluss haben. Stattdessen müssen die Benutzer, Entwickler und Systeme damit zurechtkommen, dass sich die Welt ohne ihr Zutun verändern kann und sie sich daran anpassen müssen. Für Menschen ist das noch recht einfach, Computer allerdings werden für einen definierten Anwendungsfall entwickelt, dessen Probleme sie lösen sollen. Verändert sich der Anwendungsfall oder die dazu zur Verfügung stehenden Informationen oder Ressourcen, können Computer diese Probleme oft nicht mehr selbstständig lösen. CPS müssen aber robust gegenüber Veränderungen in ihrer Umgebung sein [17], da die Interaktion mit der echten Welt immer einen gewissen Anteil an Ungewissheit über die Korrektheit und Sicherheit der vorliegenden Daten mit sich bringt [18]. Mangelnde Robustheit von CPS bzgl. Veränderungen im System führen zu einer reduzierten Zuverlässigkeit.

Keine Komponente ist aber in jedem Szenario zu 100% zuverlässig. Insbesondere bei CPS verstärkt sich dieses Problem durch die Einbeziehung der realen Welt mit ihren Unsicherheiten. Auch Plausibilitätsprüfungen oder andere herkömmliche

Methoden zur Steigerung der Robustheit können Fehler nicht vollständig vermeiden, da die Anbindung an die reale Welt zur Entwicklungszeit unvorhersehbare Bedingungen schaffen kann. Es stellt eine Herausforderung dar, dass die Welt zum einen nicht vollständig vorhersagbar ist, man zum anderen aber deterministische und zuverlässige Systeme benötigt. Dies bezieht sich natürlich nicht nur auf die korrekte Berechnung von Daten und Aktionen, sondern auch auf das korrekte Timing zur Ausführung. In verteilten Systemen, die Umwelteinflüssen gehorchen, ist nicht vorhersagbar, wann welche Daten zur Verfügung stehen werden. Dies ist eine Einschränkung für Systeme mit Echtzeitanforderungen. Diese Kombination aus teils unvorhersehbarem Timing, Rauschen in den Sensorwerten, Messfehlern und unvorhersehbaren Umwelteinflüssen ist für Softwareentwickler eine große Herausforderung, die beim Design der Systeme bedacht werden muss. [17]

Zuverlässigkeit und Robustheit sind also wichtige benötigte Eigenschaften von CPS. In diesem Bereich spielt auch die Wartbarkeit eine zentrale Rolle [10, S. 72], da die Wartung bestehender Systeme die Zuverlässigkeit und Robustheit nach dem Aufdecken von Fehlern verbessern kann. So führt eine Steigerung der Wartbarkeit im nächsten Schritt auch zu einer Steigerung der Zuverlässigkeit, da Wartbarkeit Fehlerbehebungen ermöglicht. Wartbarkeit wird vom ISO25010:2011 als eines der 8 Qualitätscharakteristika von Software definiert [19]. Zentraler Aspekt der Wartbarkeit von CPS ist die Sicherstellung der Beherrschbarkeit von Abhängigkeiten im Netzwerk. Dies ist die zentrale Problemstellung dieser Arbeit, die in Abschnitt 1.2 beschrieben wird.

1.2 Problemstellung

Ein hoher Grad an Abhängigkeiten zwischen datenaustauschenden Systemen ist ein wichtiger neuer Aspekt bei CPS, der bei alleinstehenden Computersystemen nicht auftritt. Diese logischen Abhängigkeiten erreichen ein hohes Maß an Komplexität und müssen beherrschbar gemacht werden, um Qualität sicherzustellen [20, S. 21]. Die Beherrschung von komplexen Abhängigkeitsstrukturen wird hierbei immer wieder als zentrale Herausforderung genannt [21][12][15, S. 406]. Gleichzeitig sind CPS besonders anfällig für die Verbreitung fehlerhafter Daten, da Eigenschaften der realen Welt gemessen, verarbeitet und weitergegeben werden [21]. Daher werden neue Konzepte und Werkzeuge benötigt, um mit

diesen Problemen umgehen zu können [12]. Werkzeuge können dann dabei helfen, fehlerhafte oder unerwünschte Interaktionen zu erkennen [10, S. 72].

Abhängigkeiten entstehen durch die Kooperation von Systemen im Netzwerk. Die einzelnen Systeme, die man in verteilten Netzwerken Knoten nennt, arbeiten zusammen und tragen ihre Arbeitsleistung zu einem Gesamtergebnis oder den Teilergebnissen anderer Komponenten bei. Insbesondere bei CPS interagieren Knoten und beeinflussen den jeweiligen Entscheidungsfindungsprozess durch die Zulieferung von Daten, beispielsweise in Form von Sensorwerten. Ohne einen korrekten Informationsfluss im Netzwerk werden zentrale Prozesse, die Aktionen bestimmen, gestört und das CPS kann seine Aufgaben nicht mehr erfüllen.

Hierdurch entsteht ein hoher Grad an Abhängigkeit zwischen den einzelnen Knoten und den von diesen ausgetauschten Informationen. Insbesondere Aktoren, die Aktionen in der realen Welt ausführen (vgl. Abschnitt 2.1.3), sind von zuliefernden Knoten abhängig, die ihnen ein Verhalten vorgeben. Letztendlich handelt es sich bei Aktoren nur um ausführende Instanzen des Netzwerkes, die allerdings selbst keine Entscheidungen treffen. Aggregatoren[3], die die Recheneinheiten des Netzwerkes darstellen und die Geschäftslogik enthalten (vgl. Abschnitt 2.1.3), haben bereits zuvor Informationen zusammengetragen und verarbeitet, um das korrekte, situationsabhängige Verhalten eines Aktors zu bestimmen. Betrachtet man diese Beziehung, wird klar, dass die Aktoren von der korrekten Funktionsweise der Aggregatoren abhängig sind. Zusätzlich zu dieser Abhängigkeit gibt es allerdings weitere Beziehungen zwischen den Aggregatoren untereinander. Die Verarbeitung von Daten findet in CPS in der Regel nicht zentralisiert, sondern verteilt statt. Hierdurch tauschen auch Aggregatoren untereinander verarbeitete Daten aus, die eigene Prozesse beeinflussen und das Ergebnis bestimmen.

Die Datenverarbeitung der Aggregatoren wiederum ist ihrerseits von Sensorik abhängig, die Informationen über die Umwelt erfassen und weitergeben (vgl. Abschnitt 2.1.3). Insgesamt ergibt sich eine Abhängigkeit der Aktoren von den Aggregatoren, die ihrerseits sowohl untereinander als auch von Sensorik abhängig sind.

[3] Aggregatoren sind Recheneinheiten im Netz, die vorliegende Daten kombinieren, um daraus neue Informationen abzuleiten. Der Begriff wird in Abschnitt 2.1.3 beleuchtet.

Bei CPS kann es sich sowohl um sehr kleine, lokal beschränkte, als auch global verteilte Netzwerke mit einer enormen Anzahl an Knoten handeln. Je größer diese Netzwerke werden, desto schwieriger lassen sich intuitiv Abhängigkeiten zwischen einzelnen Aktionen und Eingabedaten erkennen. Die genauen Abhängigkeiten entstehen durch das Zusammenspiel der einzelnen Knoten und können sich zur Laufzeit durch das autonome Verhalten der CPS-Komponenten verändern. Dem Architekten einzelner Bereiche eines CPS kann beim Entwurf nicht jede Abhängigkeit bekannt sein – es kann auch auf bereitgestellte und bereits aggregierte Daten anderer Knoten zugegriffen werden, um eigene Entscheidungen abzuleiten.

Insgesamt entsteht so zur Laufzeit des Systems eine unbekannte Abhängigkeitsstruktur zwischen Sensoren, Aggregatoren und Aktoren bzw. deren ausgetauschten Informationen. Die Aufdeckung dieser Abhängigkeiten ist für die Entwicklung und Wartung von CPS insbesondere in zwei Anwendungsfällen essenziell:

- **Ursache und Wirkung verstehen:** CPS sind verteilte Systeme, die Informationen zwischen verschiedenen Knoten austauschen und verarbeiten. Bei der Verarbeitung dieser Informationen kann jeder Knoten unabhängig von anderen mit den Daten verfahren und somit Inhalte ergänzen, verarbeiten oder verändern. Die genauen Aktionen sind nur dem verarbeitenden Knoten selbst bekannt und können nicht ohne Weiteres von anderen nachvollzogen werden.

  In vielen Szenarien müssen allerdings Entscheidungen eines CPS nachvollziehbar sein. Wenn eine Entscheidung, die ein Knoten eines CPS oder das CPS im Zusammenspiel getroffen hat, nicht mehr nachvollziehbar ist, können die Einflüsse auf die Entscheidung nicht mehr identifiziert werden. Tritt nun ein unerwünschtes Verhalten einer oder mehrerer Systemkomponenten auf, ist es nur schwer möglich, dieses Verhalten zu korrigieren, da deren Ursachen unbekannt sind.

- **Fehler zurückverfolgen:** Da wie bereits beschrieben nicht immer klar ist, wie Entscheidungen und Aktionen, die aus diesen Entscheidungen resultieren, zustande kommen, ist es ebenfalls nicht möglich, im Fehlerfall die Ursachen für diesen Fehler schnell und effizient zu identifizieren. Kommt es beispielsweise aufgrund falscher Sensorwerte zu einer nicht angemessenen Aktion eines Aktors, so muss es möglich sein, den Zusammenhang zwischen

Sensorwert und Aktion herzustellen. Durch die mehrfache Weitergabe und Aggregation von Sensorwerten und anderen Informationen wird die eigentliche Quelle von Daten verschleiert und ist beim ausführenden Aktor nicht sichtbar. Fehler werden jedoch oft erst beim Aktor sichtbar, da dieser eine echtweltlich wahrnehmbare Aktion ausführt.

Da CPS aus autonomen Komponenten bestehen und so zur Laufzeit neue Lösungsstrategien und somit auch Abhängigkeiten entwickeln, können diese nicht vor Inbetriebnahme des Systems modelliert werden. Es gibt bei CPS oft keinen Architekten, der alle Komponenten definiert – vielmehr gibt es viele Architekten, die das Verhalten der einzelnen Komponenten definieren und sich dabei an Schnittstellen orientieren. Das Zusammenspiel dieser Komponenten ergibt zur Laufzeit ein Gesamtverhalten des Systems. Bestimmt wird dieses Zusammenspiel auch durch zuvor definierte Schnittstellen der Komponenten. Die Perspektive jeder dieser Komponenten ist oft allerdings nicht systemweit, sondern lokal begrenzt. Die Komponenten kennen ihre Nachbarn und erhalten Daten von ihnen, haben selbst aber in der Regel keinen Bedarf zu erfahren, wie diese Daten zustande kamen. Die Komponenten müssen sich gegenseitig vertrauen. Nachbarn fungieren für den einzelnen Knoten also vielmehr als Datenquelle, die die Komponente selbst wiederum für andere darstellt.

1.3 Beispielproblem

Die in Abschnitt 1.2 beschriebene Problemstellung wird im Folgenden anhand eines Beispiels erläutert. Das Beispiel beschäftigt sich mit einem Intelligent Transportation System (ITS) aus dem Bereich der Verkehrstelematik. Hierin wird ein CPS beschrieben, das aus smarter Infrastruktur zur Verkehrssteuerung und -überwachung besteht.

Der für dieses Beispiel relevante Ausschnitt des ITS ist in Abbildung 1 dargestellt. Systeme sind hierbei als Rechtecke dargestellt, während einzelne Sensoren und Aktoren des Netzes als Ellipsen visualisiert sind. Technische Kommunikation zwischen Komponenten wird als Pfeile dargestellt. Das gezeigte CPS besteht aus vielen verschiedenen Infrastrukturkomponenten (Sensor-Boxen, Blitzer, Ampeln, Straßenschilder) und zentralen Steuerungssystemen (Zuflusssteuerung,

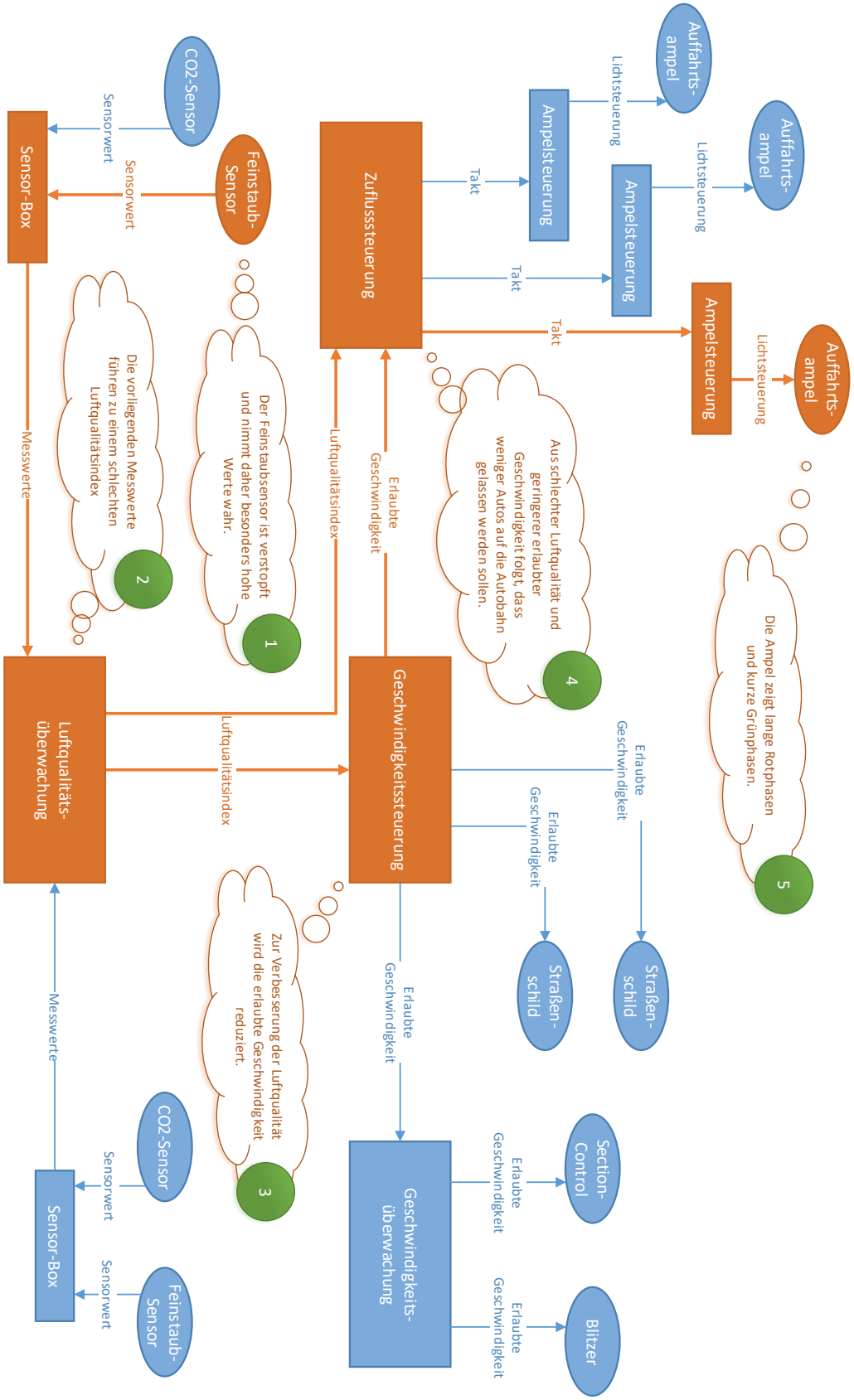


Abbildung 1: ITS als Beispiel eines CPS zur Überwachung und Steuerung von Sensoren und Aktoren im Straßenverkehr

Luftqualitätsüberwachung, Geschwindigkeitssteuerung). Die einzelnen Systeme kommunizieren miteinander, um einen Verkehrsfluss auf der Straße sicherzustellen. Die Kommunikation beinhaltet beispielsweise Informationen zur aktuellen Luftqualität sowie Informationen zu Ampelphasen oder Höchstgeschwindigkeiten.

Das Beispiel skizziert das Szenario der Steuerung eines stark frequentierten Autobahnabschnitts[4]. Die Autobahn ist zu Stoßzeiten staugefährdet. Daher existiert eine Zuflusssteuerung durch Ampeln, die den Zufluss von Fahrzeugen an den Auffahrten regelt. Hierdurch kann sichergestellt werden, dass der Verkehr nicht vollständig zum Erliegen kommt. Des Weiteren verläuft die Autobahn auch durch die Stadt, sodass eine Reduktion der Belastung der Luftqualität angestrebt wird. Dazu wird die Luftqualität mit Sensorik überwacht, um daraufhin die zulässige Höchstgeschwindigkeit und die Zuflusssteuerung anzupassen. Die Reduktion der zulässigen Höchstgeschwindigkeit führt in der Regel zu reduziertem Schadstoffausstoß, während Staus negativen Einfluss auf die Luftqualität haben.

Alle am Beispiel beteiligten Komponenten und Kommunikationspfade sind orange markiert (vgl. Abbildung 1). In diesem Beispiel kommt es zu einer Fehlfunktion an einem Sensor, der den Feinstaubgehalt der Luft überwacht. Der Sensor meldet daher zu hohe Werte (vgl. Sprechblase 1 in Abbildung 1). Der Sensor ist hierbei nicht selbst in der Lage den Fehler zu erkennen, da der falsch gemessene Wert bei hohem Verkehrsaufkommen plausibel wäre. Die Luftqualitätsüberwachung aggregiert verschiedene Sensorwerte verschiedener Messstationen und leitet als Durchschnittswert eine schlechte Luftqualität für den überwachten Autobahnabschnitt ab (vgl. Sprechblase 2 in Abbildung 1). Um die Luftqualität zu verbessern, wird die zulässige Höchstgeschwindigkeit reduziert (vgl. Sprechblase 3 in Abbildung 1) und die Zuflussteuerung der Autobahn den Gegebenheiten angepasst (vgl. Sprechblase 4 in Abbildung 1). Die Ampeln an den Auffahrten verlängern daher ihre Rotphasen und lassen weniger Fahrzeuge auf die Autobahn (vgl. Sprechblase 5 in Abbildung 1). So soll ein Stau vermieden und die Luftqualität verbessert werden.

[4] Dieses Beispiel ist inspiriert von der Autobahn A40, Abschnitt Essen-Zentrum bis Mülheim-Heißen. Auch hier existiert eine Zuflusssteuerung für die Autobahn und eine verkehrsabhängige Steuerung der zulässigen Höchstgeschwindigkeit.

In diesem Beispiel führt so also ein fehlerhafter Sensor zu einem unerwünschten Verhalten der Zuflusssteuerung der Autobahn. Die Ampeln der Zuflusssteuerung halten wartende Fahrzeuge nun unnötig lange auf, obwohl die Luftqualität nicht besonders schlecht und die Autobahn auch nicht überlastet ist. Diese Ungereimtheit kann Autofahrern und Ingenieuren des CPS einfach auffallen – der Fehler wird in der realen Welt sichtbar. Unsichtbar bleibt jedoch, auf welcher Datengrundlage die Entscheidungen getroffen wurden, die zu dieser Situation führten. Grundsätzlich ließe sich der Fehler auch an anderen Entitäten - beispielsweise den Straßenschildern, die andere Höchstgeschwindigkeiten anzeigen, oder den anderen Ampeln beobachten. Dieses Beispiel fokussiert den Fehler allerdings aus der Perspektive nur einer dieser Ampeln.

Die Begründung für das beschienene Szenario liegt in der Abhängigkeit der einzelnen Komponenten. Die Ampeln der Zuflusssteuerung sind von der korrekten Funktion des Sensors der Luftqualitätsüberwachung abhängig, um selbst korrekt und effizient arbeiten zu können. Betrachtet man die Situation nun aus Perspektive der Ampel (vgl. Abbildung 1), ist schwer erkennbar, warum diese zu wenige Fahrzeuge auf die Autobahn auffahren lässt. Es ist kein offensichtlicher Grund erkennbar.

Gäbe es Methoden, mit deren Hilfe transparent dargestellt werden kann, auf welcher Datengrundlage die Entscheidung der Ampelphasen getroffen wurde, könnte der fehlerhafte Sensor identifiziert werden. Wird hier sichtbar, dass der Sensor fehlerhafte Werte geliefert hat, ließe sich dieser Fehler beheben.

Die Abhängigkeit zwischen dem Sensor der Luftqualitätsüberwachung und der Ampel in diesem Beispiel ist von außen betrachtet nicht offensichtlich. Dies ist in CPS oft der Fall, da Informationen, die von einem Knoten zum anderen weitergegeben werden und immer wieder mit anderen Informationen gemeinsam aggregiert werden, nicht mehr zu ihrer Quelle zurückverfolgt werden können.

1.4 Zielsetzung und Forschungsvision

Die vorausgehenden Ausführungen beschreiben, dass CPS neue Strategien benötigen, um Wartbarkeit zur Laufzeit sicherzustellen. Dies betrifft insbesondere die Abhängigkeiten zwischen ausgetauschten Informationen, die in den CPS durch

die Verknüpfung von Komponenten entstehen. Im Fehlerfall müssen geeignete Werkzeuge zur Verfügung stehen, um die Fehlerquelle ermitteln zu können. Ohne diese Werkzeuge können die Mechanismen des Netzwerkes und die daraus resultierenden Abhängigkeiten nicht verstanden und Fehler nicht effizient behoben werden.

Zentraler Aspekt der Arbeit wird die Untersuchung der resultierenden Forschungsvision und den daraus abgeleiteten Forschungsfragen, die im Folgenden formuliert werden.

Forschungsvision

Ziel der Arbeit ist die Ermöglichung der folgenden Forschungsvision:

> Fehlentscheidungen und die Informationen, die zu diesen führten, können in verteilten Netzwerken über Aggregationen hinweg zu ihrer Quelle zurückverfolgt werden, um ihre Ursachen zu identifizieren und anschließend zu beheben.

Die Erreichung der Forschungsvision wird durch die folgenden Forschungsfragen geleitet:

Forschungsfrage I

> Wie kann der Informationsfluss in CPS überwacht und aufgezeichnet werden?

Forschungsfrage II

> Wie können Abhängigkeiten zwischen Informationen und Entscheidungen aus Datenfluss-Informationen ermittelt und visualisiert werden?

Forschungsfrage III

Wie kann man mit diesen Informationen das Auffinden von Fehlerursachen sowie die Instandhaltung und Pflege eines CPS unterstützen?

1.5 Lösungsansatz

Zentraler Baustein zur Lösung der Forschungshypothese ist das Konzept des Information Flow Monitors (IFM). Das Information Flow Monitoring ist ein Konzept mit zugehörigem Werkzeug, dem IFM, das dieses Konzept umsetzt. Mit Hilfe des IFM soll es möglich werden, Informationen, die im Netzwerk ausgetauscht werden, bezüglich ihrer Abhängigkeiten zu überwachen. Ziel ist es, dass hierdurch Informationsflüsse im Netzwerk über Aggregationen hinweg nachverfolgbar werden.

Teilt beispielsweise ein Sensor Sensorwerte mit einem anderen Aggregator-Knoten im Netzwerk, so muss dem IFM-Werkzeug bekannt sein, dass hier Informationen geflossen sind. Nutzt der Aggregator nun diese erhaltenen Sensorwerte, um daraus selbst neue Erkenntnisse abzuleiten, und teilt diese ebenfalls mit anderen Knoten, muss dieser Informationsfluss ebenfalls dem IFM bekannt sein. Zusätzlich soll die semantische Abhängigkeit zwischen den Sensorwerten und der berechneten Erkenntnis des Aggregators erfasst werden. So kann später nachverfolgt werden, dass die Erkenntnisse des Aggregators von den Sensorwerten des Sensors abhängig sind. Die Erkennung und Aufzeichnung dieser Abhängigkeiten ist zentrale Aufgabe des IFM und soll über mehrere Knoten im Netzwerk hinweg funktionieren.

Wichtige Rahmenbedingungen ergeben sich durch den Einsatz in CPS, die stark heterogen, autonom und dezentral sind. Ebenfalls werden CPS meist nicht von einem zentralen Service Provider betrieben, sondern von vielen verschiedenen Akteuren, die einzelne Knoten im Netzwerk bereitstellen. Dies führt dazu, dass es ein durchgehendes Werkzeug geben muss, das über Governance-Grenzen[5] hinweg funktioniert und von allen Akteuren mit den notwendigen Daten versorgt

[5] Als Governance-Grenze wird eine (virtuelle) Grenze zwischen Systemen bezeichnet, die von unterschiedlichen Parteien verwaltet werden.

werden kann. Die Umsetzung des IFM Konzepts und Werkzeugs werden in Kapitel 3 beschrieben. Weitere zentrale Anforderungen an den IFM werden in Abschnitt 3.1.1 dargestellt.

1.6 Aufbau dieser Arbeit

Diese Arbeit ist in fünf Kapitel gegliedert. Kapitel 1 begann mit einer allgemeinen Einleitung des Themas. Anschließend wurde die Motivation und Problemstellung der Arbeit diskutiert und daraus die Forschungsvision und die Forschungsfragen abgeleitet. Das Kapitel schloss mit einer kurzen Einführung in den Lösungsansatz.

Kapitel 2 beschäftigt sich mit dem für die Arbeit notwendigen Grundlagenwissen und erörtert Literatur zum allgemeinen Verständnis der Arbeit. Hierbei wird insbesondere auf die Besonderheiten von CPS eingegangen sowie der Abhängigkeitsbegriff geklärt. Zusätzlich werden verwandte Ansätze und Anwendungsgebiete für CPS dargestellt.

Kapitel 3 ist das zentrale Kapitel der Arbeit und stellt als Lösungsansatz den IFM vor. Hierbei werden zuerst die Anforderungen an das entsprechende Konzept und Werkzeug definiert, ehe daraus eine Architektur und ein Protokoll zur Kommunikation zwischen beteiligten Akteuren im Netzwerk definiert werden. Zusätzlich wird das Protokoll formal beschrieben und es werden Anwendungsbeispiele zum weiteren Verständnis des Konzepts dargestellt. Zentrale Fragestellungen um das Konzept (vgl. Abschnitt 3.2) sowie Erweiterungen (vgl. Abschnitt 3.3, Abschnitt 3.4) werden ebenfalls vorgestellt.

Kapitel 4 beinhaltet die Evaluierung des IFM. Dies geschieht mit Hilfe von zwei Evaluationsteilen. Teil 1 (vgl. Abschnitt 4.1) stellt ein großes CPS vor, anhand dessen die Funktionsweise des IFM erprobt wird. Teil 2 (vgl. Abschnitt 4.2) betrachtet die Performanz des IFM und misst zentrale Werte, die beim Einsatz in CPS eine Rolle spielen.

Im letzten Kapitel werden die Erkenntnisse der Arbeit zusammengefasst, reflektiert und der Beitrag der Arbeit kritisch beleuchtet. Zusätzlich werden offene Fragestellungen diskutiert, die Raum für weitere Arbeiten geben.

2 Grundlagen

In diesem Kapitel: Grundlagenwissen und Literatur zum allgemeinen Verständnis der Arbeit, des CPS-Begriffs und Abhängigkeiten in CPS sowie verwandte Arbeiten.

CPS sind geprägt durch eine enge Verknüpfung von eingebetteten Systemen, Echtzeitsystemen und verteilten Sensornetzen [12] zur Überwachung und Steuerung physischer Prozesse und Ressourcen. Die physischen Prozesse werden hierbei durch Sensorik erfasst und durch Aktoren beeinflusst und gesteuert. Durch Sensorik wird die physische Realität digitalisiert und so datenverarbeitenden Instanzen im Netzwerk verfügbar gemacht – diese können dann auf Basis dieser Daten Entscheidungen treffen und durch Aktoren ausführen lassen. Daher ersetzen CPS oft auch mechanische Steuerungen [11]. Es entstehen Ursache-Wirkungs-Ketten, da erfasste Sensordaten wiederum zu einer real ausgeführten Aktion eines Aktors führen können. An der Verarbeitung der dazu benötigten Daten können große Mengen an heterogenen Komponenten im Netz beteiligt sein. [20]

Die Kommunikation der Komponenten erfolgt üblicherweise über eine Mischung aus kabelgebundenen und kabellosen Netzwerken, die sich über große geographische Distanzen erstrecken können [12]. Die Anzahl der beteiligten Komponenten ist hierbei nicht beschränkt – die Größe von CPS reicht von einigen wenigen Komponenten im Smart Home bis hin zu abertausenden von Sensoren in Stromnetzen oder Netzen zur Erfassung von Wetterdaten. Durch eine hohe Menge an verschiedenen Komponenten ist eine Heterogenität der beteiligten Akteure bzgl. eingesetzter Hardware und Software und daraus resultierenden Fähigkeiten und Vorgehensweisen nicht zu vermeiden. Bei diesen Akteuren handelt es sich um eingebettete Systeme, die Komponenten wie Sensorik, Recheneinheit, Speicher und Netzwerkkommunikation bündeln. Hierbei kann es sich einerseits um sehr kleine, leichtgewichtige Systeme handeln, die jahrelang autonom mit einem Akku betrie-

ben werden können. Andererseits können auch komplex zu berechnende Aufgaben von Hochleistungsservern bewältigt werden, um empfangene Daten zu verarbeiten. Das hierdurch entstehende Zusammenspiel verschiedener Komponenten bringt neue Herausforderungen mit sich (s. Abschnitt 2.1.4).

CPS sind also eine Zusammenstellung aus heterogenen und teilweise autonom agierenden verteilten Systemen, die für gemeinsame und/oder eigene Zwecke kooperieren [22, 9]. Bei dieser Kooperation nehmen die Komponenten Aufgaben verschiedener Domänen wahr und ergänzen sich so [15, S. 398]. Bei den verteilten Systemen handelt es sich jedoch nicht um Strukturen, die einmalig geplant und entwickelt werden, sondern um Zusammenstellungen von Komponenten, deren Verhalten sich zur Laufzeit verändern kann. Ein sich verändernder Systemkontext[6], der durch die Sensorik wahrgenommen werden kann, kann auch zu einer Veränderung des Verhaltens des Netzwerkes führen. Dieses Verhalten ergibt sich insbesondere durch die Autonomie der einzelnen Komponenten.

Das Gesamtverhalten eines Gesamtsystems, kann daher nur aus der Beobachtung aller Komponenten erkannt werden. Die Funktion des CPS ergibt sich also aus dem Zusammenspiel aller Komponenten gemeinsam und lässt sich nicht durch die Betrachtung von Teilsystemen erklären. Dieses Verhalten wird Emergenz genannt. Klassische Client-Server-Systeme sind daher nicht emergent, da ein zentraler Server oder ein Teilsystem, bestehend aus mehreren zentralen Instanzen, die Hoheit über alle wichtigen Entscheidungen hält und die Clients wenig Einfluss auf die Gesamtfunktion des Systems haben.

Die beschriebenen Eigenschaften von CPS werden durch Software ermöglicht, die autonom in der Lage ist, auf geänderte Rahmenbedingungen zu reagieren. In großen Netzen lässt sich oft nicht überblicken, welche Änderungen welche Auswirkungen an jeder einzelnen Stelle im System haben können – die Systeme restrukturieren sich durch den Austausch von Daten selbstständig.

In diesem Kapitel werden Grundlagen zum weiteren Verständnis der folgenden Kapitel erläutert und verwandte Arbeiten zur Diskussion und Wissensbildung

[6] Der Systemkontext beschreibt den für ein System relevanten Teil der Umgebung und Umwelt, dessen Eigenschaften das System direkt betreffen (z. B. durch Sensorik oder ausgetauschte Daten) oder mit dem das System interagiert (z. B. durch Aktorik oder andere Systeme).

herangezogen. In Abschnitt 2.1 wird der Begriff CPS erläutert, eingeordnet und definiert. Dazu werden zentrale Eigenschaften, Bestandteile, Herausforderungen und Anwendungsgebiete von CPS beschrieben. Anschließend wird in Abschnitt 2.2 weiteres Grundlagenwissen zu Abhängigkeiten in CPS, auch aus der Literatur, zusammengefasst, um das Verständnis der Arbeit zu erleichtern. In Abschnitt 2.3 werden verwandte Arbeiten und Literatur vorgestellt, die für den Kontext der Arbeit relevant sind.

2.1 Grundlagenwissen zu Cyber-Physical Systems

Der Begriff Cyber-Physical System wurde in vorausgehenden Kapiteln bereits angerissen. Im Folgenden werden die drei Worte, aus denen sich der CPS-Begriff zusammensetzt, die jeweils eine Bedeutung beitragen, genauer beschrieben. Anschließend werden Eigenschaften von CPS zusammengetragen, um eine umfassende CPS-Definition abzuleiten, die für diese Arbeit gültig ist.

System

Der System-Begriff hat seinen Ursprung im Altgriechischen (altgriechisch sýstēma) und steht für ein aus mehreren Einzelteilen zusammengesetztes Ganzes [23]. Der internationale Standard ISO/IEC 15288:2008 [24] definiert ein System als eine Kombination aus interagierenden Elementen, die zur Erreichung eines oder mehrerer erklärter Zwecke organisiert sind [24]. Eine ähnliche Definition gibt das *Systems Engineering Handbook* des International Council of Systems Engineering (INCOSE):

> „[A system is] an integrated set of elements, subsystems, or assemblies that accomplish a defined objective. These elements include products (hardware, software, firmware), processes, people, information, techniques, facilities, services, and other support elements.“ – Systems Engineering Handbook [25]

Cyber

Der Cyber-Begriff leitet sich ursprünglich von cybernetic ab, das aus dem Altgriechischen am nächsten mit *etwas, dass geschickt im Lenken oder Regieren ist* übersetzt werden kann. Im CPS-Begriff bezieht sich Cyber insbesondere auf die Begriffe Cyberspace und Kybernetik [26].

Die Kybernetik, auch *Kunst des Steuerns* genannt, beschäftigt sich mit der Steuerung und Regelung von Maschinen. Wichtig hierbei ist immer die Rückkopplung zwischen Beobachtung und Handlungsweise, sodass Regelkreise entstehen, die aufgrund von Beobachtungen Entscheidungen treffen [27]. Typisches Beispiel für kybernetische Systeme sind Thermostate, die gemessene Temperaturen mit Soll-Werten vergleichen und so Heizungen oder Klimaanlagen steuern, um die Soll-Temperatur zu erreichen.

Cyberspace wiederum ist ein informeller Begriff und bezeichnet eine durch Computer erzeugte virtuelle Welt. Üblicherweise wird auch das Internet als ein Cyberspace bezeichnet – es steht aber nur stellvertretend für einen Datenraum.

Physical

Der Physical-Begriff bezieht sich auf Entitäten außerhalb des Datenraums. Fokus sind hierbei echtweltliche Objekte. Daher konzentriert sich der Phyiscal-Aspekt der CPS insbesondere auf die Bereiche Mechatronik und Sensorik [26]. Ein mechatronisches System ist hierbei der Kybernetik nicht unähnlich – mechanische Systeme werden von elektronischen Bauteilen und Software gesteuert und überwacht [28]. Sensoren und Aktoren sind hierbei von zentraler Bedeutung, da diese physikalisch-echtweltliche Komponenten beobachten bzw. digitalisieren sowie beeinflussen können.

Cyber-Physical System

Setzt man die drei zuvor genannten Begriffe zusammen, entstehen Cyber-Physical Systems. Hierbei handelt es sich demnach um aus mehreren Prozessoren und Komponenten zusammengesetzte Gebilde (*Systeme*), die über Netzwerke (*Cyberspace*)

miteinander kommunizieren, Daten austauschen und echtweltliche (*Physical*) Prozesse durch Rückkopplung zwischen Sensoren und Aktoren überwachen und steuern (*Kybernetik*).

2.1.1 Eigenschaften von CPS

CPS lassen sich nicht eindeutig von anderen Systemen und Begriffen wie System of Systems (SoS) [29] oder IoT [30] abgrenzen. Ziel ist daher in diesem Kapitel nicht die Abgrenzung der CPS von ähnlichen Domänen und Begriffen, sondern die Erörterung der Eigenschaften von CPS. Hieraus lässt sich anschließend eine Definition ableiten. In der Literatur lassen sich mehrere Definitionen zu CPS finden (vgl. Abschnitt 2.1.2), die sich jedoch in einigen Punkten unterscheiden. Um sich einer eigenen CPS-Definition zu nähern und die Essenz von CPS im Vergleich zu anderen Begriffen besser zu erfassen, werden im Folgenden zentrale Eigenschaften aus der Literatur [9, 22, 14, 17] [31, S. 2] [20, S. 22] und anderen Definitionen aufgelistet und erläutert.

Die Eigenschaften von CPS werden wie folgt zusammengefasst:

1. ***Interaktiv*** – Die Komponenten/Netzwerkknoten arbeiten zusammen und tauschen Informationen aus, um eigene Entscheidungen abzuleiten und zu teilen. Dies bringt gleichzeitig mit sich, dass CPS vernetzt sind und über das Netz miteinander kommunizieren.

2. ***Reaktiv*** – Klassische Systeme erzeugen eine Ausgabe nur dann, wenn diese angefordert wird. Dies geschieht in der Regel durch eine Eingabe, zu der eine Ausgabe berechnet werden soll. Reaktive Systeme erzeugen jedoch aus sich heraus Ausgaben, ohne dass diese explizit angefordert wurden. Dies geschieht durch die Wahrnehmung der Umwelt durch Sensorik sowie eigenen Daten, die die Systeme vorhalten können.

3. ***Parallel*** – In CPS werden ständig neue Sensorwerte aufgezeichnet und Daten ausgetauscht. Diese werden von allen beteiligten Systemen parallel verarbeitet. Die Systeme treffen hierbei selbst Entscheidungen auf den zugrunde liegenden Informationen und arbeiten echt parallel zu anderen Komponenten.

4. ***Flexibel*** – Das Gesamtsystem ist nicht starr, sondern verändert sich durch neue Software sowie neue und wegfallende Komponenten zur Laufzeit. Auch die Vernetzung kann sich zur Laufzeit ändern. Es handelt sich also nicht um einen Monolithen, der am Stück deployed wird, sondern um Einzelkomponenten die separat hinzugefügt oder verändert werden können.

5. ***Adaptiv*** – Komponenten arbeiten automatisiert und können auf geänderte Rahmenbedingungen reagieren – beispielsweise durch eine geänderte Funktionsweise. Geänderte Rahmenbedingungen können z. B. veränderte Ressourcen, Kommunikationspartner, durch Sensorik wahrgenommener Systemkontext oder neu deployte Software sein. Die Software der Komponente kann diese veränderten Rahmenbedingungen erkennen und das eigene Verhalten entsprechend anpassen.

6. ***Emergent*** – Das Verhalten des Gesamtsystems ergibt sich durch das Zusammenspiel der einzelnen Komponenten und lässt sich nicht durch die Betrachtung einzelner Teilsysteme ableiten. Das Gesamtverhalten des Systems und der einzelnen Komponenten kann sich so zur Laufzeit verändern, da die Komponenten auf veränderte Umweltparameter, Ressourcen oder andere Komponenten reagieren und ihr eigenes Verhalten anpassen.

7. ***Rollenbasiert*** – Jede logische Komponente im Netzwerk ist entweder Sensor, Aggregator oder Aktor (vgl. Abschnitt 2.1.3). Es kommt vor, dass einzelne Akteure im Netzwerk mehrere Rollen in einem physischen Gerät vereinen. Diese sind jedoch logisch getrennt und erfüllen füreinander bestimmte Aufgaben.

8. ***Autonom*** – Komponenten sind nicht von einer zentralen Instanz abhängig und entscheiden selbst, wie sie auf geänderte Rahmenbedingungen reagieren. Dies bedeutet nicht, dass es nicht Knoten gibt, die für die Gesamtfunktion des CPS wichtiger sind als andere. Die Gesamtfunktion eines CPS kann daher von einzelnen Knoten abhängen. Dies beeinträchtigt aber nicht die Autonomie der einzelnen Knoten.

9. ***Heterogen*** – Im Netzwerk existieren viele verschiedene Komponenten, deren Funktionsweise untereinander nicht bekannt sein muss. Die Komponenten können sich bzgl. Hersteller, Kommunikationsprotokollen, Hardware,

Software, Zweck, Kompatibilität, Kooperationsbereitschaft, Gutmütigkeit und vielen weiteren Eigenschaften unterscheiden.

10. ***Verteilt*** – Komponenten von CPS sind logisch und geographisch verteilt. Kommunikation wird über Netzwerke hergestellt. Bei diesen Netzwerken kann es sich um sehr unterschiedliche Typen handeln, beispielsweise Ethernet- oder Wireless Local Area Network (WLAN)-Netze sowie Ad-hoc-Netzwerke[7]. Durch die Verteiltheit können sich Rahmenbedingungen unterscheiden und verändern.

11. ***Eng integriert*** – Die Essenz von CPS ist die Integration physischer Prozesse in Berechnungen und Steuerungen. Hierdurch wird es möglich, physisch-analoge Prozesse mit Hilfe von Sensorik digital abzubilden.

12. ***Multifunktional*** – Der Einsatzzweck von CPS ist nicht auf ein bestimmtes Problem, eine Problem- oder Fachdomäne festgelegt. Durch ihre Generalität und die Abstraktheit des Konzeptes lässt es sich praktisch überall einsetzen, wo physische Prozesse eine Rolle spielen.

Viele CPS weisen einen großen Anteil dieser Eigenschaften auf. Es ist allerdings nicht notwendig, dass das System alle Eigenschaften aufweist, um als CPS zu gelten.

2.1.2 CPS-Definition

Verschiedene Forschungsgruppen und Institutionen haben sich mit der Fragestellung beschäftigt, inwiefern sich CPS von anderen IT-Systemen wie eingebetteten Systemen, SoS, Cloud-Systemen oder dem IoT unterscheiden. Hierbei existiert keine allgemein akzeptierte Definition. Stattdessen betrachten die verschiedenen Gruppen CPS aus ihrer Perspektive, um Unterschiede zu aus ihrer Sicht relevanten Fachdomänen zu beleuchten. Auch wenn die verschiedenen Definitionen nicht identisch sind, haben sie hohe Schnittmengen – insbesondere auch mit den zuvor beschriebenen Eigenschaften von CPS (vgl. Abschnitt 2.1.1).

[7] Ad-hoc-Netzwerke (lt. *ad-hoc*, für den Moment gemacht) sind (oft kabellose) Netze, die sich binnen kürzester Zeit bilden und auch wieder auflösen. Sie entstehen zwischen wenigen, oft geographisch nahen Komponenten, die einen vorübergehenden Kommunikationsbedarf haben.

Im Folgenden werden einige Definitionen und Beschreibungen von CPS anderer Forschungsgruppen zitiert und beschrieben. Abschließend wird eine eigene Definition von CPS vorgestellt, die im Rahmen des Forschungsprojektes CPS.HUB NRW [6] entstanden ist.

2.1.2.1 CPS-Definitionen anderer Forschungsgruppen

Bei der folgenden Definition handelt es sich um die Beschreibung der Charakteristika von CPS einer Forschungsgruppe der Universitäten Würzburg und Potsdam, die im Rahmen der Enzyklopädie der Wirtschaftsinformatik von Prof. Dr. Christian Janiesch erstellt wurde.

> „[...] Cyber-physische Systeme (CPS) [bezeichnen] die Kopplung von physischen, biologischen und/oder bautechnischen Komponenten, die über eine Recheneinheit integriert, gemonitort und/oder gesteuert werden. Integraler Bestandteil sind dabei informations- und softwaretechnische Komponenten mit mechanischen bzw. elektronischen Bestandteilen, die über eine Kommunikationsinfrastruktur wie bspw. das Internet in Echtzeit miteinander kommunizieren. Es handelt sich dabei um große, komplexe Systeme, die vollintegriert logische Berechnungen und physische Aktionen aufeinander abstimmen. [...] CPS können dabei weitestgehend automatisiert und autonom agieren. [...] Wahrnehmung und Ausübung von Funktionen ist [...] immer auf den jeweiligen Kontext angepasst. Die mechanischen bzw. elektronischen Teile eines CPS werden über sogenannte eingebettete Systeme realisiert, die mittels Sensoren ihre lokale Umwelt wahrnehmen und über Aktuatoren die physische Umwelt beeinflussen können.“ – Christian Janiesch, Enzyklopädie der Wirtschaftsinformatik [32]

In der Definition werden wesentliche zuvor beschriebene Eigenschaften aufgelistet, unter anderem die Interaktivität, Reaktivität und Autonomie. Fokus der Definition ist die Beschreibung der Vernetzung und der Interaktion zwischen softwaretechnischen Berechnungen und physischen Aktionen.

Manfred Broy, der als einer der führenden Akteure in der deutschen CPS-Forschung gilt, geht noch einen Schritt weiter und beschreibt CPS als Verschmel-

zung von eingebetteten Systemen, die sowohl mit ihrer physischen Umgebung als auch mit globalen Netzwerken verbunden sind:

> „The term Cyber-Physical Systems addresses a new type of systems which is the result of an amalgamation of embedded software systems, connected on one hand to their physical environment by sensors and actuators and global networks such as the Internet with its data and services. By Cyber-Physical Systems the Internet gets real world aware and embedded systems become location independent, can be connected on a global scale, and get access to global data and services.“ – Manfred Broy [33]

Durch die Integration von CPS bekommt das Internet eine Art Realitätsbewusstsein. Diese Integration der Beobachtung und Interaktion mit physischen Prozessen spielt in allen Definitionen eine zentrale Rolle:

> „Cyber-physical systems are integrations of computation, networking, and physical dynamics, in which embedded devices are networked to sense, monitor and control the physical world. “ – Xia et al. [34]

Die Bobachtung und Manipulation von physischen Prozessen durch CPS führt, wie im Folgenden beschrieben, auch zu neuen Verhaltensweisen, die sich nur verstehen lassen, wenn beide Welten gemeinsam betrachtet werden. Physische Prozesse, die teils außerhalb der Kontrolle des CPS stehen, beeinflussen das Verhalten des CPS stark. Die Definition aus der CIRP Encyclopedia of Production Engineering fasst Beschreibungen von Edward A. Lee und Raj Rajkumar zusammen. Gerade Edward A. Lee gilt in den USA als einer der Gründer der CPS-Forschung.

> „Cyber-Physical Systems (CPS) are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the internet. With other words, CPS can be generally characterized as ‘physical and engineered systems whose operations are monitored, controlled, coordinated, and integrated by a computing and communicating core‘[12]. The interaction between the physical and cyber elements is of key importance: ‘CPS is about the intersecti-

> on, not the union, of the physical and the cyber. It is not sufficient to separately understand the physical components and the computational components. We must understand their interaction‘[1].“ – László Monostori, CIRP Encyclopedia of Production Engineering [35]

2.1.2.2 Eigene CPS-Definition

Basierend auf den gesammelten Eigenschaften (s. Abschnitt 2.1.1) und Definitionen (s. Abschnitt 2.1.2.1) entstand eine eigene Definition von CPS aus der Perspektive des Software Engineerings, die bereits in Kapitel 1 dargestellt wurde. Die Definition fokussiert sich auf die wesentlichen Aspekte, die sich in vorhandenen CPS-Definitionen wiederfinden und für den Charakter und das Verhalten von CPS essenziell sind.

Diese Definition wird als Grundannahme in der folgenden Arbeit verstanden. Entscheidungen bzgl. Lösungs- und Evaluationskonzept sind auf Basis dieser Definition getroffen worden, um die Eigenheiten von CPS möglichst gut berücksichtigen zu können. Hierdurch wurde es möglich, ein auf CPS zugeschnittenes Konzept zu erarbeiten, da zu berücksichtigende Spezifika von CPS bekannt sind.

2.1.3 Sensorik und Aktorik

Zentraler Aspekt der vorgestellten CPS-Definition ist das Beobachten und Beeinflussen realer Objekte und Prozesse. Dies wird durch Sensoren und Aktoren realisiert.

Sensoren sind Komponenten im Netzwerk, die eine physische Größe der Umgebung messen. Aktoren sind das entsprechende Gegenstück, denn sie verändern eine physische Größe der Umgebung [1, S. 179]. Sensoren und Aktoren treten heute oft gebündelt mit Mikroprozessoren und Netzwerkinterfaces auf, sodass diese sowohl an das IoT angebunden als auch in CPS verwendet werden können. Sensoren messen und sammeln Daten, die mit ihrem aktuellen geographischen Standort zusammenhängen. Hierbei können Sensoren sowohl geographisch fest verankert sein (z. B. lokale Messstationen) als auch sich kontrolliert (z. B. in Autos,

Flugzeugen, Smartphones) oder unkontrolliert (z. B. durch Tiere, Wetterballons/-bojen) bewegen [15, S. 398].

Damit die analogen Werte eines Sensors von CPS verwendet werden können, müssen diese digital dargestellt werden. Der analoge Wert wird durch eine Menge an Stichproben repräsentiert, wobei eine Stichprobe zu einem bestimmten Zeitpunkt gültig war. Es handelt sich also um einen Strom aus Sensorwerten, bei denen die einzelnen Werte einen zeitlichen Abstand aufweisen [1, S. 180]. Die kontinuierlichen Informationen eines Sensors werden digital diskret dargestellt.

CPS entstehen erst durch die Interaktion von Hardware und Software – keiner der Teile allein kann die Anforderungen an ein CPS erfüllen. Es entsteht ein Wechselspiel zwischen Hardware und Software. Sensorik beispielsweise versorgt durch Software betriebene vernetzte Systeme mit notwendigen Informationen zur Entscheidungsfindung. Lee et al. [14] bezeichnen dieses Wechselspiel als Feedback-Loops, bei denen physische Prozesse, die durch Aktoren beeinflusst werden, durch Sensorik erfasst werden. Er bezeichnet die gemeinsame Dynamik von physischen Prozessen, Software und Netzwerken als die zentrale Eigenschaft von CPS. Sensorik sorgt dafür, dass physische Eigenschaften *cyberized* werden [14, S. 737]. Auf der anderen Seite werden digitale Eigenschaften durch Aktoren *physicalized* [14, S. 738].

Die Kommunikation der einzelnen Komponenten erstreckt sich hierbei über viele Teilsysteme hinweg. Mit durch Software getroffenen Entscheidungen wird wiederum Hardware gesteuert, die sogenannten Aktoren. In diesem Wechselspiel tragen echtweltliche Prozesse zur Entscheidungsfindung in Software bei, deren Resultate wiederum echtweltlich sichtbar werden können.

Die Vielfalt der in CPS eingesetzten Sensorik und Aktorik nimmt immer weiter zu. Im Jahr 2010 waren durchschnittlich bereits 35% des Wertes eines Automobils solche elektrischen Komponenten [36], die zunehmend Aufgaben von hydraulischen oder mechanischen Bauteilen übernehmen. Insofern ist mit einer weiteren steigenden Verbreitung elektronisch gesteuerter Bauteile zu rechnen. Hierdurch nehmen auch die Heterogenität und Komplexität der Komponenten im Netzwerk und des Netzwerks selbst stetig zu.

CPS bestehen aus drei Typen von Komponenten (Knoten) im Netzwerk: Sensoren, Aktoren und Aggregatoren. [1]

- Sensoren zeichnen echtweltliche Daten auf und geben diese an andere Knoten weiter. Selbst treffen sie keine Entscheidungen, sondern dienen als Sinnesorgane des Gesamtsystems. Sie sind eine Datenquelle für andere Komponenten im Netzwerk. Sensorwerte bergen eine gewisse Unsicherheit, da diese nur eine Annäherung an die Realität sind.

- Aggregatoren erhalten Daten von Sensoren und anderen Aggregatoren, um diese zu verarbeiten, zu aggregieren[8] und um Entscheidungen oder neue Informationen aus der Gesamtmenge der ihnen vorliegenden Daten abzuleiten. Aggregatoren sind von zuliefernden Komponenten bzgl. Korrektheit und Zuverlässigkeit abhängig. Aus den Daten abgeleitete Entscheidungen können durch Aktorik sichtbar werden.

- Aktoren erhalten Aktionsdaten von Aggregatoren und führen diese aus. Somit beeinflussen Aktoren wiederum echtweltliche Prozesse, die womöglich durch Sensorik wahrgenommen werden. Aktoren treffen selbst keine Entscheidungen. Sie sind reine Datensenken im Netzwerk und erstellen selbst keine neuen Informationen.

Sensoren, Aggregatoren und Aktoren können in CPS sowohl in getrennter als auch in gemeinsamer Hardware vorkommen. Es ist also möglich, dass ein System sowohl die Rolle eines Sensors als auch die Rolle eines Aggregators hat. Logisch lassen sich diese jedoch getrennt betrachten, obwohl sie in einer gemeinsamen Hardwareumgebung ausgeführt werden.

Durch die Weitergabe von Informationen im Netzwerk entstehen Abhängigkeiten zwischen den einzelnen Knoten. Entscheidungen eines Aggregators beruhen nicht mehr nur auf eigenen Daten, sondern auch auf Sensorwerten von entfernten Netzwerkknoten. Die durch den Aktor ausgeführte Aktion ist seinerseits also nicht nur von dem/den steuernden Knoten abhängig, sondern ebenfalls von dessen/deren Datenlieferanten. Dieser Abhängigkeitsbaum kann etliche Ebenen enthalten. An Aktoren sichtbar werdende Entscheidungen können daher oft nicht nachvollziehbar wirken, da der Abhängigkeitsbaum in der Regel nicht bekannt und nicht einfach konstruierbar/ableitbar ist.

[8] Der Begriff der Aggregation beschreibt das Zusammenführen von Daten und Informationen zu einer neuen, daraus abgeleiteten Information.

2.1.4 Herausforderungen

Da CPS aus vielen verschiedenen heterogenen Systemen bestehen und domänenübergreifend Probleme lösen, werden anders als bei bisheriger Software-Entwicklung auch andere Fachexperten benötigt. Bisher reichte es aus, wenn sich Software-Entwickler mit den Experten der jeweiligen Fachdomäne austauschten, in denen die Software eingesetzt werden soll. Bei der Entwicklung von CPS genügt dies jedoch nicht mehr, da zur Problemlösung nicht nur Software herbeigezogen wird, sondern auch der Physical-Teil des CPS entwickelt werden muss.

Die Entwicklung von CPS erfordert es daher, dass Informatiker und Netzwerker mit Experten aus den verschiedenen Disziplinen der Ingenieurwissenschaften zusammenarbeiten. Hierzu zählen beispielsweise Experten der Regelungstechnik, der Signalverarbeitung, des Bauingenieurwesens, der Biologie, der Chemie, Netzwerktechnik, Informations- und Kommunikationstechnik (IKT) und der hardwarenahen Informatik, die sich beispielsweise mit eingebetteten Systemen beschäftigt [12, 9]. Hierzu ist es notwendig, den Austausch der Fachdomänen in einem immer wiederkehrenden iterativen Prozess während der gesamten Entwicklungsphase aufrechtzuerhalten. Kämen Artefakte der unterschiedlichen Fachdomänen erst zum Ende der Entwicklung zusammen, sind Fehler und Inkompatibilitäten sehr wahrscheinlich [37].

Da jedoch Software Engineering und andere Engineering-Disziplinen zur Konzeption der Physical-Aspekte oft andere Engineering-Ansätze und Methoden verfolgen, da sie andere Probleme lösen, werden ebenfalls übergreifende Prozesse benötigt, um die Fachdomänen zueinander zu bringen [10, S. 71]. Diese Prozesse müssen auch den Langzeitbetrieb und die Wartbarkeit von CPS fokussieren [20, S. 26].

Zusätzlich zu den Fachexperten der verschiedenen Domänen benötigen diese Experten auch spezielle Kompetenzen zur Entwicklung von CPS. Geisberger und Broy fassen die notwendigen Engineering-Kompetenzen wie folgt zusammen:

> „Engineering-Kompetenzen umfassen erforderliche [...] Fähigkeiten und Fertigkeiten (Prinzipien, Verfahren, Methoden, Techniken und Best Practices sowie integrierte Werkzeugkonzepte verschiedener Disziplinen) für die zielführende Entwicklung, Gestaltung, Realisierung [...] von Cyber-Physical Systems [...]. Beispiele [...] sind Methoden

> der Modellierung, Anforderungsspezifikation, Konstruktion und der durchgängigen Qualitätssicherung [...], Verlässlichkeit, Betriebssicherheit, IT-Sicherheit oder Schutz der Privatsphäre.“ – Geisberger und Broy, agendaCPS [4, S. 28]

Durch CPS werden die Anforderungen an die Zuverlässigkeit, und somit auch an die Wartbarkeit, von Systemen ansteigen, da CPS bisherige ausgereifte Lösungen ersetzen, die fehlerfrei arbeiten. Nutzer erwarten bei Autos, Ampeln, Fernsehern, usw. keine Abstürze oder Fehlfunktionen. CPS steigern jedoch die Komplexität dieser Systeme und somit automatisch auch die Möglichkeiten für Fehler. Dies liegt insbesondere auch in der Integration der physischen Welt in die Systeme begründet. Die physische Welt ist nicht komplett vorhersagbar und Umstände, die beim Design eines Systems nicht berücksichtigt wurden, können später im Produktivbetrieb dennoch eintreten. [17]

Es gibt jedoch keine Systeme, die zu 100% zuverlässig und fehlerfrei arbeiten und die physische Welt wird immer Situationen hervorbringen können, die nicht vollständig bei der Entwicklung bedacht wurden. Bisher ist es der Software Engineer gewöhnt, dass er auf einer Abstraktionsschicht als Grundlage implementieren kann, die nahezu fehlerfrei[9] ist. Dies ist jetzt nicht mehr der Fall. Jede Abstraktionsschicht eines Systems muss mit Fehlern der darunterliegenden umgehen können, um Robustheit sicherzustellen. [17]

Bei Betrachtung der Korrektheit der Ausführung muss jetzt auch das gesamte System und nicht mehr nur die einzelne Komponente betrachtet werden. Benötigt ein Aggregator eine Information eines Sensors, der diese nicht rechtzeitig liefert, verhält sich das Gesamtsystem nicht mehr korrekt. Die einzelnen Systeme jedoch haben keinen Fehler gemacht, sondern sich nur im Gesamtzusammenhang so verhalten, dass das Gesamtsystem nicht mehr korrekt arbeiten kann. Es geht hier also sehr stark um die Integration von Komponenten und deren Kompatibilität. [17]

[9] Nur sehr wenige Systeme können als fehlerfrei belegt werden. In der Regel basiert Software jedoch auf erprobten Abstraktionsschichten, die über Jahre verbessert wurden und als ausgereift gelten.

Die aus den zuvor dargestellten Punkten zentral ableitbare Herausforderung ist die Komplexitätsbewältigung in einer großen, heterogenen Umgebung zur Steigerung der Wartbarkeit und Zuverlässigkeit. Hierzu liefert diese Arbeit einen Beitrag. Das in Kapitel 3 vorgestellte Konzept hilft Fehler und Verhaltensweisen in CPS über viele Komponenten hinweg nachzuverfolgen, um Fehler zu beheben und Verhaltensweisen zu verstehen.

2.1.5 Anwendungsgebiete für CPS

CPS lassen sich in verschiedenen Fachdomänen zum Einsatz bringen. Anwendungen reichen von kleinen Applikationen wie Smart Home Systemen bis zu kontinentalen oder globalen Energie- und Kommunikationsnetzen [12, 9]. Dieses Kapitel soll eine kurze Übersicht über die verschiedenen Anwendungsgebiete und ihren Nutzen geben. Innerhalb der Fachdomänen besteht die Möglichkeit, große Fortschritte bezüglich Funktionalität und Effizienz zu erreichen. Die daraus resultierenden wirtschaftlichen Auswirkungen dieser Applikationen können enorm sein [14].

Da CPS bei der Vernetzung von smarter werdenden Systemen eine zentrale Rolle spielen, haben sie auch Einfluss auf die Gestaltung von autonomer werdenden **Autos und anderen Fahrzeugen**, die mit Sensorik und Aktorik ausgestattet sind. Dies ist auch jetzt schon der Fall – neu ist allerdings sowohl die Menge der eingesetzten Sensorik sowie die Vernetzung mit anderen Teilnehmern und Verkehrsinfrastruktur. Die Perspektive der Entwicklung ist hierbei ein Verkehr mit weniger Unfällen, der durch stark optimierte Verkehrsflüsse Staus vermeidet und so Transport- und Wartezeiten reduziert. Die Routen, die die Fahrzeuge wählen, können hierbei mit anderen Verkehrsteilnehmern abgestimmt werden, um Überlastungen der Strecken zu vermeiden. Autonomes Fahren, Kollisionsvermeidung oder adaptive Routenberechnung werden durch CPS-Technologien ermöglicht. Sicherheit und Effizienz der Mobilität kann so gesteigert werden. [12] [13, S. 29] [14]

CPS bieten allerdings auch in der **Medizintechnik** sowohl die Perspektive der individuell zugeschnittenen Versorgung der Patienten als auch der Effizienzsteigerung durch Prozessoptimierung und Telemedizin. Telemedizin erlaubt die Überwachung von Vitalwerten durch medizinische Geräte und Sensorik von bei-

spielsweise Implantaten, die bei der Patientenbetreuung eine zentrale Rolle bei der Verbesserung der Diagnose- und Behandlungsmöglichkeiten einnehmen können [4, S.23]. Patienten können hierdurch proaktiv kontaktiert werden, sollten die Vitalwerte Grund zum Eingreifen geben. So kann sowohl der Betreuungsprozess der Patienten gestärkt und optimiert werden als auch die Qualität der Behandlung gesteigert werden. Auch in Krankenhäusern vor Ort können CPS unterstützen. Intelligente Operationssäle und Krankenhäuser oder bildgesteuerte Chirurgie sind nur zwei mögliche Beispiele.

Auch bei **Smart Grids** [38], die traditionelle Energienetze zunehmend ersetzen und als Netze mit hohen Abhängigkeiten zwischen den beteiligten Akteuren beschrieben werden, handelt es sich um CPS. Ein wichtiges Ziel bei der Forschung im Bereich der Smart Grids ist also die Sicherstellung von Zuverlässigkeit. Diese kann sowohl durch Fehler im System als auch durch Attacken von außen gefährdet sein. In unerwarteten Situationen können Fehler an einer Stelle im System dazu führen, dass sich Fehler ausbreiten und so das gesamte Netz kompromittieren. Diese Cascading Failures führen zu weitreichenden Stromausfällen [18]. Um diese Fehler zu vermeiden, können CPS den Zustand des Netzes überwachen und präventiv Maßnahmen, beispielsweise zur Wartung, anfordern. Dies kann zu einer ausfallfreien Stromversorgung beitragen [12].

Im Bereich der **Smart Factory / Industrie 4.0** ermöglichen CPS die Überwachung der Fabrik. Mit Hilfe von Sensorik können einzelne Produktionsschritte beobachtet, sowie Aktoren individuell gesteuert werden. CPS ermöglichen so flexible und kostengünstige Auftragsfertigung mit einem hohen Grad an Automatisierung und Individualisierung der Produkte.

Um individuell in der Smart Factory gefertigte Waren effizient zum Kunden transportieren zu können, werden intelligentere **Logistikprozesse** benötigt, die sowohl einen hohen Grad an Flexibilität und Effizienz mitbringen als auch kundenindividuelle Abläufe ermöglichen. Hierzu muss der gesamte Lebensprozess der Transportgüter erfasst, analysiert und gesteuert werden. Hierbei ist es notwendig, den aktuellen Standort der Transportgüter ständig zu erfassen. Dies geschieht über eindeutige Identifikatoren (IDs) an der Ware, wie beispielsweise Strichcodes oder Radio-Frequency Identification (RFID)-Tags. Durch die Identifizierung der Ware kann anschließend über Aktorik sichergestellt werden, dass jedes einzelne

Transportgut korrekt sortiert und verarbeitet wird. So kann eine Ware in Echtzeit verfolgt werden. [14]

In der **Smart Home** Fachdomäne geht es, wenn auch in kleinem Stil, ebenfalls um Prozessautomatisierung. Hierbei stehen alltägliche Dinge wie Licht- und Heizungssteuerung sowie Sicherheitsaspekte wie Zugangssteuerung und Überwachung im Mittelpunkt. Grundsätzlich ist das Ziel eines Smart Homes zum einen der Komfortgewinn der Bewohner, zum anderen aber auch das Sparen von Ressourcen und die Steigerung der Sicherheit. Sicherheit und Komfortgewinn spielt insbesondere auch für Senioren oder Menschen mit körperlichen Einschränkungen eine wichtige Rolle [12].

Das Sparen von Ressourcen kann auch im Bereich des **Umweltmanagements** eine zentrale Rolle spielen, wenn die Einsparung von fossilen Brennträgern durch Prozessoptimierung zum Umweltschutz beitragen kann. Zusätzlich kann die Umwelt mit beispielsweise Wettereigenschaften (Luftdruck, Temperatur, Niederschlag, Wind, ...) aufgezeichnet werden, um Prognosen und Handlungsempfehlungen abzuleiten. Das Monitoren der Umwelt bezieht sich aber beispielsweise auch auf Frühwarnsysteme für Naturkatastrophen [12].

Mobile Kommunikation ist ein Grundbaustein für erfolgreiche CPS. Netzwerke werden nicht mehr nur durch kabelgebundene Verbindungen realisiert, da mobile Geräte nicht mehr stationär installiert werden. Stattdessen besteht der Bedarf, Komponenten eines CPS mit sich zu führen sowie an Orten aufzustellen, deren Infrastruktur keine kabelgebundene Installation erlaubt. Dazu wird eine zuverlässige und leistungsfähige Kommunikationsinfrastruktur benötigt. Kommende 5G-Netzwerke können hierzu einen Beitrag liefern. [5, S. 14]

2.2 Grundlagenwissen zu Abhängigkeiten in CPS

Dass Abhängigkeiten in verteilten Systemen von zentraler Relevanz für diese Arbeit sind, wurde in vorausgehenden Kapiteln bereits beschrieben. Im Folgenden soll der Abhängigkeitsbegriff daher ausführlich behandelt werden. Dazu wird sowohl die Historie der Abhängigkeiten in Netzwerken als auch deren Relevanz für die heutige Zeit in CPS beleuchtet. Dieses Kapitel dient zum näheren Ver-

ständnis der Abhängigkeiten im Allgemeinen, deren verschiedener Typen sowie der Einordnung des Begriffs in den Zusammenhang dieser Arbeit.

2.2.1 Abhängigkeiten in verteilten Systemen

Ist ein System abhängig von einem anderen, beschreibt dies in der Regel die Notwendigkeit einer Eingabe[10] in dieses System durch ein anderes System, damit ersteres korrekt arbeiten kann. Das System wird abhängig von der Eingabe und dem Eingebenden. Diese Art der Abhängigkeit beschrieb DeMarco in der strukturierten Analyse [39] und spezifizierte Systeme und deren Komponenten durch Datenflussdiagramme (DFDs).

Diese Diagramme beschreiben ausschließlich die Weitergabe von Daten zwischen Elementen eines Gesamtsystems und keine Kontrollflüsse. DeMarco beschreibt Datenflussdiagramme als Netzwerkrepräsentation eines Systems [39, S. 47]. Übertragen auf CPS-Netze ist diese Beschreibung ebenfalls treffend, da sie sich auf die heterogenen topologischen Strukturen von CPS anwenden lässt. Er beschreibt hierbei sowohl datenerschaffende Komponenten wie Sensoren, datenverarbeitende Komponenten wie Aggregatoren als auch Datensenken wie Aktoren [39, S. 40].

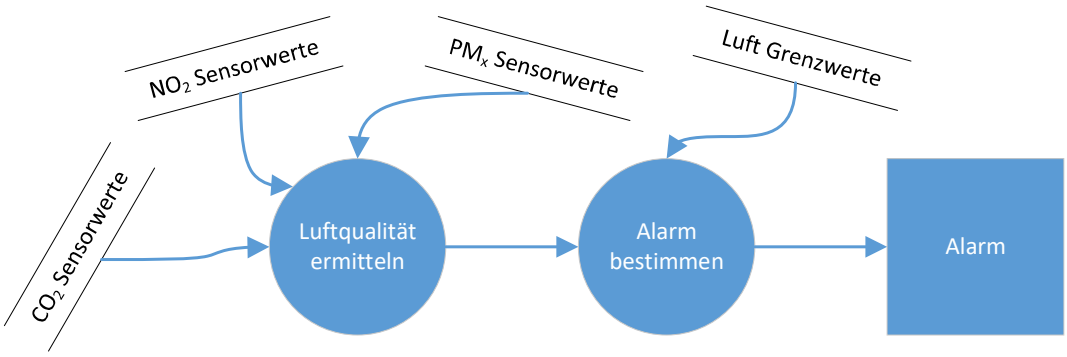


Abbildung 2: Fachliche Anforderungen an ein System, beschrieben durch ein DFD

[10] Eine Eingabe in ein System ist üblicherweise die Datenübertragung eines Systems an ein anderes. Das sendende System gibt dann Daten in das empfangende System ein. Eingaben können aber auch durch Benutzerinteraktion oder Sensorik entstehen.

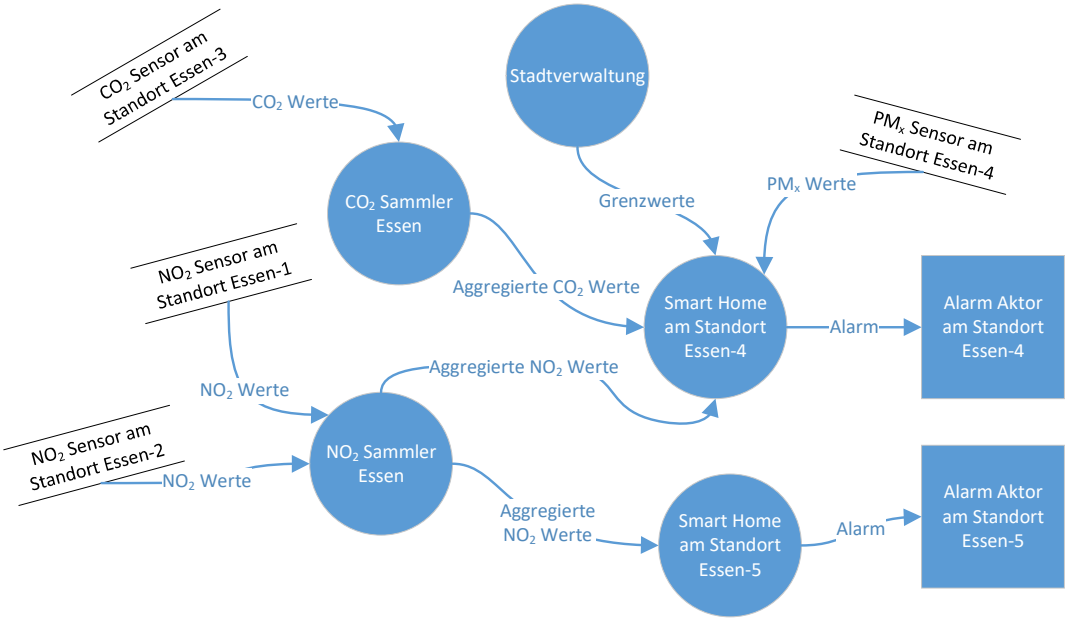


Abbildung 3: DFD einer möglichen Umsetzung eines Messsystems für Luftqualität, das auf den fachlichen Anforderungen basiert (vgl. Abbildung 2)

McMenamin und Palmer bedienen sich dieses Konzeptes und definieren zwei Sichten auf ein System: Die der fachlichen Anforderungen, genannt Essenz, und eine tatsächliche Umsetzung dieser Anforderungen in ein technisches System, genannt Inkarnation [40].

- **Fachliche Datenflüsse / Essenz**

  Fachliche Anforderungen an ein System sind die Anforderungen, die technologie- und systemunabhängig beschrieben werden können. Fachliche Datenflüsse lassen sich mit den beschriebenen Methoden [40, S. 16-22] mit Hilfe von DFDs spezifizieren. In Abbildung 2 ist ein Beispiel bzgl. eines Systems zur Messung von Luftqualität dargestellt, welches bei Überschreitung bestimmter Messwerte einen Alarm erzeugen soll. Die mögliche technische Umsetzung bleibt bei dieser Betrachtung völlig außen vor.

- **Technische Datenflüsse / Inkarnation**

  Technische Datenflüsse existieren nur in der tatsächlichen Umsetzung, also einer Realisierung eines Systems. Die Autoren sprechen hier gezielt von einer einzelnen technischen Umsetzung, die die fachlichen Anforderungen

> umsetzt. Die Realisierung stellt explizit eine Umsetzung dar, die aus einer Menge an möglichen Umsetzungen ausgewählt wurde (vgl. Abbildung 3).

Die oben genannten Autoren sehen also den Prozess des Requirements Engineering bis zur Erstellung des Systems als sequenziellen Prozess, bei dem genau eine gewählte technische Umsetzung die fachlichen Anforderungen erfüllt und zur Laufzeit unverändert bestehen bleibt. Zentraler Aspekt ist die Darstellung der sich durch Datenflüsse ergebenen Abhängigkeiten zwischen Komponenten des Systems. Allerdings bleiben diese Abhängigkeiten zur Systemlaufzeit auch konstant und verändern sich nicht, da das System starr ist.

Die hier dargestellten Aspekte lassen sich einfach in die heutige Zeit und auf CPS übertragen. Heutige Systeme jedoch verändern durch ihre Autonomie das Verhalten des Gesamtsystems zur Laufzeit. Sie entdecken neue Datenquellen, fügen Aktoren hinzu und kombinieren diese Komponenten über definierte Schnittstellen, um neue Prozesse zu entwickeln.

Somit ist ein CPS nicht nur eine einzige, zuvor festgelegte Implementierung einer Spezifikation, sondern kann zur Laufzeit verschiedene Formen annehmen, die die fachlichen Anforderungen erfüllen müssen. Insbesondere kann ein CPS zur Laufzeit neue Datenflüsse entwickeln und bestehende Datenflüsse verändern. Die Schlussfolgerung hieraus ist, dass sich Abhängigkeiten innerhalb der technischen Umsetzung ebenfalls verändern und sich nicht a priori spezifizieren lassen. Ein CPS hat also zu einer Essenz viele verschiedene Inkarnationen.

Datenflussdiagramme eignen sich, um sowohl fachliche als auch technische Abhängigkeiten in CPS zu spezifizieren. Ein Datenflussdiagramm, das sich allerdings auf technische Komponenten beziehen soll und nicht nur auf fachlicher Ebene agiert, spiegelt so immer nur eine mögliche Umsetzung eines Systems wider, aber nicht alle möglichen. Die Menge der möglichen Realisierungen eines Systems ist unabsehbar groß.

Hieraus resultierend können Abhängigkeiten zwischen Komponenten in CPS ebenfalls erst zur Laufzeit bestimmt werden und sind immer nur eine Momentaufnahme der aktuellen Konfiguration und Zusammenstellung des Gesamtsystems. Eine vollständige Modellierung der Abhängigkeiten vor Inbetriebnahme kann also nur einzelne Komponenten beschreiben, aber nie die Gesamtabhängigkeiten aller im System aktiven Komponenten. Insbesondere lassen sich Informationen, die

zur Laufzeit auszutauschen sind, nicht im Vorhinein zuverlässig und umfassend modellieren.

Für die Arbeit ergeben sich hieraus insbesondere zwei zentrale Schlussfolgerungen:

- Abhängigkeiten innerhalb der fachlichen Darstellung eines CPS lassen sich durch DFDs gemäß [39, 40] spezifizieren. Die fachlichen Abhängigkeiten eines CPS verändern sich nicht durch Verhaltensweisen eines CPS, sondern nur durch nachträglich dem System hinzugefügten Anforderungen.

- Die Abhängigkeiten in der tatsächlichen Umsetzung eines CPS lassen sich nicht für alle Komponenten a priori modellieren, da sich diese zur Laufzeit verändern können. Jede erkannte Abhängigkeit ist nur eine Momentaufnahme des Systems. Diese Momentaufnahmen müssen zur Laufzeit ermittelt werden, da sich sowohl das System selbst als auch dessen Umwelt verändert. Anderenfalls verlieren CPS durch Modellierung an dieser Stelle ihren autonomen Charakter.

Aus den vorherigen Ausführungen wird die folgende für diese Arbeit gültige Definition für Abhängigkeiten abgeleitet:

> Existiert ein Datenfluss von einem System A zu einem System B, so ist System B von System A abhängig. Verwendet System B die empfangenen Daten, um selbst Ausgaben zu erzeugen, so sind auch diese Ausgaben von B von den verwendeten Ausgaben von A abhängig. Alle weiteren Systeme, die diese Ausgabe von B verwenden, sind somit nicht nur von der genannten Ausgabe von B, sondern auch von der genannten Ausgabe von A abhängig.

So entstehen direkte Abhängigkeiten zwischen einzelnen ausgetauschten Daten, aber keine grundsätzlich dauerhaft gültigen Abhängigkeiten zwischen Knoten.

Die zuvor beschriebene Abhängigkeit wird in der Literatur [41, S. 15] als Teil der Cyber-Abhängigkeiten benannt. Es wird jedoch zwischen weiteren Typen von Abhängigkeiten unterschieden, die zwischen den einzelnen Komponenten auftreten können:

- **Physische Abhängigkeit:** Eine Komponente produziert/verwaltet eine echtweltliche Ressource/Ware, die an andere Komponenten weitergegeben wird. Ohne die Ressource/Ware kann die abhängige Komponente nicht korrekt arbeiten.

- **Cyber-Abhängigkeit:** Eine Komponente hat eine sogenannte Cyber-Abhängigkeit, wenn der Zustand dieser Komponente von Informationen abhängig ist, die sie über das Netzwerk, z. B. von Informationssystemen oder Sensorik, erhält.

- **Geographische Abhängigkeit:** Eine Menge an Komponenten ist geographisch abhängig, wenn lokal begrenzte Ereignisse Einfluss auf diese Komponenten haben können. Dies impliziert insbesondere physische Ereignisse wie ein Feuer in einem Datenzentrum, Stromausfälle oder Erdbeben. Diese Ereignisse werden in der Regel nicht durch Sensorik der betroffenen Systeme erfasst, da dafür benötigte Sensorik üblicherweise für die Aufgabenbewältigung der Systeme nicht benötigt wird.

- **Logische Abhängigkeit:** Eine logische Abhängigkeit ist jede Abhängigkeit, die weder physisch noch geographisch und ebenfalls keine Cyber-Abhängigkeit ist. Diese Abhängigkeiten ergeben sich in der Regel durch menschliche Entscheidungen, die Einfluss auf eine technische Komponente haben. Das Versagen der IT-Infrastruktur eines Unternehmens und der gleichzeitige Kurseinbruch der Aktie dieses Unternehmens wäre beispielsweise logisch abhängig.

Andere Forschungsgruppen unterteilen die Arten von Abhängigkeiten ähnlich, bezeichnen sie jedoch anders [42]. Zimmermann und Zhang et al. bezeichnen die Cyber-Abhängigkeiten als funktionale [43, 44], Dudenhoefer et al. als informationelle [45] und Lee und Wallace als Input-Abhängigkeiten [46, 47]. Die Erkennung von Abhängigkeiten wurde von verschiedenen Forschungsgruppen als eine der größten Herausforderungen in CPS benannt. Miclea et al. bezeichnen die Evaluierung der gegenseitigen Abhängigkeiten zwischen Cyber- und Physical-Komponenten als die maßgebliche Herausforderung [48].

2.2.2 Cascading Data Corruption

Abhängigkeiten in verteilten Netzwerken haben in der Literatur bereits durch viele Forschungsgruppen Beachtung gefunden – auch speziell im Bereich von CPS. In Netzwerken mit voneinander abhängigen Komponenten unterscheidet man zwischen drei Arten von Fehlern [41, S. 22], die getrennt oder gemeinsam auftreten können:

- **Cascading Failure:** Man spricht von einem kaskadierenden Ausfall, wenn ein Ausfall einer Komponente zu einem Ausfall einer anderen Komponente führt, der sofort oder später durch sein oder das Verhalten anderer Komponenten sichtbar wird. In der Regel bezeichnet Cascading Failure den Totalausfall, also die Nichterreichbarkeit, einer Komponente.

- **Escalating Failure:** Ein vorhandener Fehler in einer Komponente führt dazu, dass sich ein vorhandener Fehler einer anderen Komponente intensiviert/verschlimmert.

- **Common Cause Failure:** In mehreren Komponenten treten gleichzeitig Fehler auf, die dieselbe Ursache haben.

Die Abhängigkeit einer Komponente von der korrekten Funktion des anderen Teilnehmers ist besonders schwierig zu behandeln. Das Problem hierbei ist, dass ein System zwar verfügbar ist und Daten liefert, diese allerdings korrumpiert sind. Es handelt sich hierbei um die zuvor beschriebene Cyber-Abhängigkeit.

Der Begriff *Cascading Data Corruption* wird im Rahmen dieser Arbeit eingeführt, um ein ähnliches Phänomen zu beschreiben, welches sich allerdings auf die Weitergabe fehlerhafter Daten bezieht. Fehlerhafte Daten in einer Komponente führen zu fehlerhaften Daten und Entscheidungen in anderen Komponenten. Fehlerhafte Daten breiten sich so im Netzwerk aus und sorgen bei vielen erreichten Komponenten für die Erzeugung neuer, davon abhängiger und ebenfalls fehlerhaften Daten. Die Komponenten bleiben, obwohl sie fehlerhafte Daten weitergeben, aber im Netzwerk verfügbar. Der Begriff lehnt sich an den Begriff der Cascading Failures an.

Der Fokus dieser Arbeit liegt, bezogen auf die vorgestellten Typen, insbesondere im Bereich der Cascading Failures bzw. des Spezialfalls Cascading Data Corruption ausgelöst durch Cyber-Abhängigkeiten.

Zentrale Eigenschaft von Cascading Data Corruption ist die Weitergabe fehlerhafter Daten im Netzwerk über viele Komponenten hinweg. Die Ursache für die fehlerhaften Daten, z. B. ein defekter Sensor, fehlende nicht vorliegende Informationen oder unausgereifte Software, kann hierbei sehr unterschiedlich sein. Die fehlerhaften Informationen werden an mehrere andere Komponenten weitergegeben und von dort aus jeweils weiter verteilt. Zusätzlich werden sie in anderen Komponenten mit neuen Informationen aggregiert und erneut weitergegeben. Der Fehler kaskadiert so, teils oder vollständig unbemerkt, durch das Netzwerk. Erst wenn ein bestimmter Punkt erreicht ist, fällt der Fehler auf. Durch seinen komplexen und verschachtelten Weg durch das Netzwerk ist die Quelle des Fehlers nun nicht mehr einfach zu identifizieren.

Tritt in einem CPS also ein Fehler auf, wodurch fehlerhafte Daten erzeugt wurden, passiert es oft, dass diese Daten an viele andere Knoten im Netzwerk weitergegeben werden. Auch diese verarbeiten die Daten weiter, aggregieren sie, und geben sie an weitere Komponenten weiter. Die fehlerhaften Daten erreichen so in verarbeiteter und unverarbeiteter Form immer mehr Komponenten, die auf Basis der fehlerhaften Daten selbst Fehler machen. Dieser Prozess wird Cascading Data Corruption genannt. Die Nichterkennung von Abhängigkeiten führt somit zu einer maßgeblichen Reduzierung der Systemzuverlässigkeit [49].

Cascading Data Corruption wird wie folgt definiert:

> Cascading Data Corruption bezeichnet einen Prozess, bei dem fehlerhafte Daten in einem Netzwerk von Knoten zu Knoten weitergegeben und verarbeitet werden, sodass am Ende des Ereignisses viele Knoten von fehlerhaften Daten betroffen sind und auch selbst von diesen Daten abhängige, neue fehlerhafte Daten erzeugen.

Beispiel

Die folgenden Abschnitte illustrieren die Abhängigkeiten am Beispiel aus Abschnitt 1.3. Ein weiteres Beispiel, dass sich an einem Szenario von Culler et al.

[50] orientiert, ist in der Literatur zu finden [51][11][52][12]. Alle Beispiele veranschaulichen das Problem der Abhängigkeiten und von Cascading Data Corruption in CPS.

Im hier behandelten Beispiel (vgl. Abschnitt 1.3) wurde ein Fehlverhalten an einer Ampel einer Autobahnzufahrt beobachtet. Wie es dazu kam, ist dem Ingenieur und Beobachter unklar, da die Datenflüsse, die zu diesem Verhalten führten, nicht bekannt sind. Um da Problem jedoch zu beheben, muss Wissen über die zuvor ausgetauschten Informationen im Netzwerk zur Verfügung stehen, damit die Fehlerquelle identifiziert werden kann.

Das sichtbare Fehlverhalten der Ampel wurde durch die Zusammenarbeit vieler Knoten im Netzwerk erzeugt. Wie genau die Sensordaten durch die Aggregatoren verarbeitet wurden, ist nicht bekannt und kann nicht bestimmt werden. Dementsprechend ist es auch nicht leicht festzustellen, wie das Ergebnis der Berechnung erzeugt wurde. Viele verschiedene Szenarien können zu einem Fehlverhalten führen. Letztlich kann jede einzelne am Prozess beteiligte Komponente ausfallen. Dies betrifft zum einen die Sensoren, die ausfallen oder fehlerhafte Daten liefern können. Zum anderen können auch die vom Aggregator gewonnenen Daten fehlerhaft verarbeitet werden. Es ist sehr schwierig, den Datenfluss nachzuvollziehen und zu verstehen, welche Informationen von welchen Sensordaten und Knoten abhängig sind. Die manuelle Analyse der Log-Dateien der einzelnen Knoten steht in größeren Netzwerken wegen des enormen Aufwands außer Frage. Durch Cascading Data Corruption hat die fehlerhafte Information schließlich den Empfänger erreicht – welche Komponente im Netzwerk die Ursache ist bleibt

[11] Dieser Beitrag wurde veröffentlicht in:
S. Gries, M. Hesenius und V. Gruhn. „Cascading Data Corruption: About Dependencies in Cyber-Physical Systems: Poster“. In: *DEBS 2017 - Proceedings of the 11th ACM International Conference on Distributed Event-Based Systems*. 2017, S. 345–346. DOI: 10.1145/3093742.3095092

[12] Dieser Beitrag wurde veröffentlicht in:
S. Gries, M. Hesenius und V. Gruhn. „Tracing Cascading Data Corruption in CPS with the Information Flow Monitor“. In: *New Trends in Intelligent Software Methodologies, Tools and Techniques - Proceedings of the 16th International Conference, SoMeT_17*. Hrsg. von H. Fujita, A. Selamat und S. Omatu. Bd. 297. Frontiers in Artificial Intelligence and Applications. Kitakyushu, Japan: IOS Press, 2017, S. 399–408. ISBN: 978-1-61499-799-3. DOI: 10.3233/978-1-61499-800-6-399

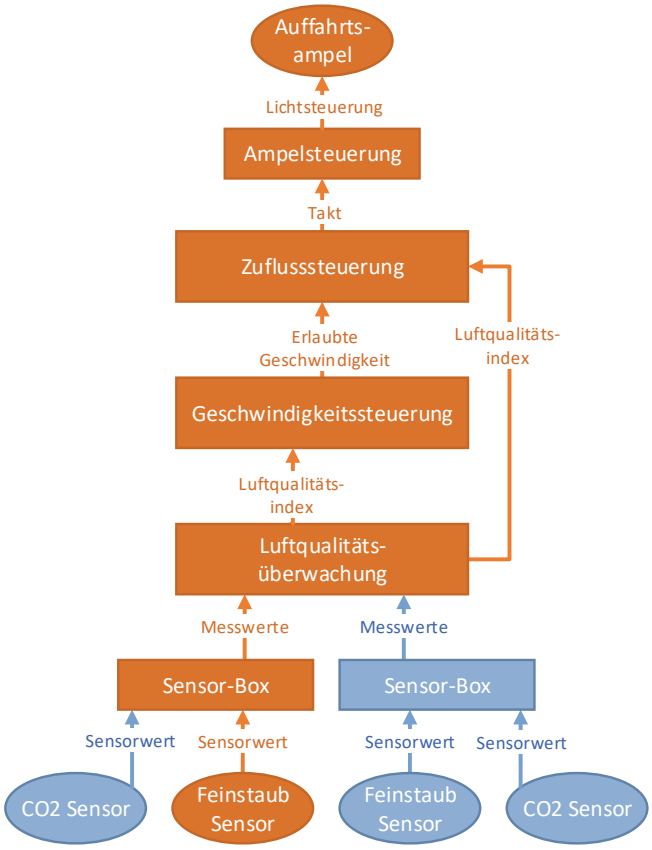


Abbildung 4: Abhängigkeiten zwischen Knoten im Netzwerk

unklar. Dies wird insbesondere dadurch erschwert, dass nicht einmal klar ist, welche Komponenten an der Erstellung der fehlerhaften Information beteiligt waren. Sowohl die beteiligten Knoten im Netzwerk als auch deren Abhängigkeiten sowie die ausgetauschten Informationen, bleiben unbekannt. Diese Werte wären jedoch notwendig, um die Fehlerquelle aufzudecken.

Aus der in Abschnitt 1.3 gezeigten Topologie des Netzwerkes lässt sich ein allgemeiner Abhängigkeitsgraph für dieses Beispiel konstruieren. Der Abhängigkeitsgraph (vgl. Abbildung 4) beinhaltet hierbei alle Knoten, die mittelbar und unmittelbar Informationen zur Ampelsteuerung zuliefern können. Es müssen sich jedoch nicht zu jedem Zeitpunkt alle Knoten an der Erzeugung einer Information im Netzwerk beteiligen. Sensoren und Aktoren sind hierbei als Ellipsen dargestellt, während Aggregatoren als Rechtecke visualisiert sind. Die Knoten sind hierbei analog zu Abbildung 1 (Seite 10) koloriert. Weitere Abhängigkeiten, auch wenn in diesem Beispiel nicht für den Fehler verantwortlich, sind blau dargestellt. Welche Knoten

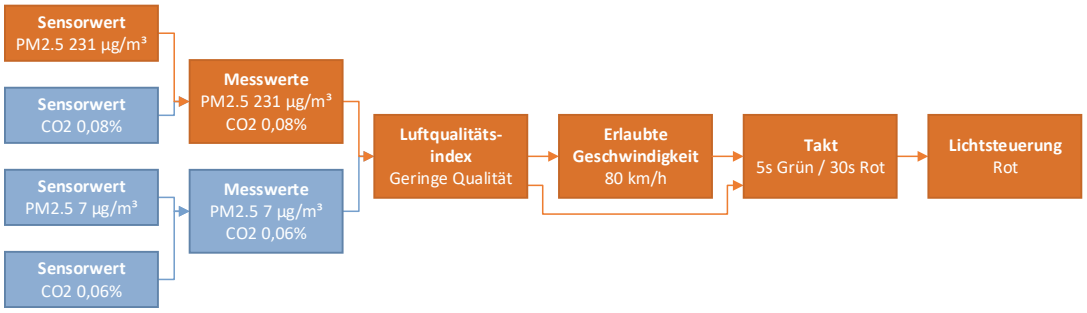


Abbildung 5: Abhängigkeiten zwischen ausgetauschten Informationen im Netzwerk

für einen Fehler verantwortlich sind, ist aber üblicherweise nicht bekannt und hier nur für dieses Beispiel illustriert.

Die Ampelsteuerung ist von allen gezeigten Knoten der Abbildung abhängig. Das Wissen darüber, welche Knoten als Fehlerquelle in Frage kommen, reicht jedoch noch nicht allein aus, um die Fehlerquelle zu identifizieren. Aus der Knotenabhängigkeit geht nicht hervor, welcher Knoten die fehlerhafte Information zugeliefert hat. Wäre allerdings nicht nur die Abhängigkeitsstruktur zwischen den datenzuliefernden Knoten bekannt, sondern auch zwischen den von diesen versendeten Nachrichten, könnte damit erkannt werden, woher die fehlerhaften Informationen stammen.

Sowohl das Wissen über die Abhängigkeiten zwischen Knoten als auch zwischen ausgetauschten Informationen liegt dem Nutzer nicht vor. Für Debugging[13]-Zwecke ist der eigentliche Informationsfluss, der alle ausgetauschten Daten enthält, notwendig, bleibt aber verborgen. Dieser ist aber erforderlich zur Identifizierung der Ursache des Problems. Ein solcher Informationsfluss, der Abhängigkeiten zwischen ausgetauschten Informationen beschreibt, ist in Abbildung 5 dargestellt. Mit dessen Hilfe ließe sich ein hoher zugelieferter Feinstaubwert als Fehlerquelle identifizieren. Der Zweig des Abhängigkeitsbaums, der in diesem Beispiel für den Fehler verantwortlich ist, ist in der Abbildung orange gefärbt.

[13] Debugging beschreibt den Prozess der Fehlersuche, üblicherweise in Software. Der Debugging-Prozess wird hierbei durch geeignete Werkzeuge unterstützt.

2.2.3 Daten und Informationen

Innerhalb dieser Arbeit wird viel von abhängigen Daten und Informationen gesprochen. Diese Begriffe sind zentraler Bestandteil dieser Arbeit. Da sie nicht synonym sind und die Unterschiede an verschiedenen Stellen von Bedeutung sind, werden im Folgenden die Begriffe differenziert beschrieben. [53, 54]

Daten

Alles, was in Netzwerken Verkehrslast verursacht oder auf Speichermedien Platz belegt, sind Daten. Diese können auch ungeordnet, unverständlich oder unlesbar sein – es handelt sich um Daten. Letztendlich sind Daten nichts anderes als eine Menge von Zahlen und Buchstaben oder Binärcode. Daten können von Computern gespeichert, versendet oder empfangen werden, ohne dass der Computer den Inhalt darstellen oder verarbeiten können muss.

Information

Daten können Informationen enthalten. Wenn die Daten in einen bestimmten Kontext gesetzt werden können oder mit einer bestimmten definierten Software verarbeitet werden, kann der Inhalt der Daten – die Information – sichtbar werden. Bei Informationen handelt es sich also um verstandene bzw. korrekt interpretierte Daten.

Wird als Datenbeispiel der Wert „638257380“ erhalten, kann zuerst nur eine Zahl erkannt werden. Mit entsprechendem Kontext, der zu dem Wissen führt, dass die Zahl als Zeitstempel zu identifizieren ist, lässt sich eine Information daraus ableiten. Die Daten entsprechen in diesem Fall dem Zeitpunkt 24.03.1990, 06:43:00 Uhr.

An bestimmten Stellen dieser Arbeit ist die Unterscheidung zwischen den Begriffen wichtig, an anderen Stellen treffen Aussagen sowohl für Daten als auch Informationen zu. Ist beispielsweise eine Information semantisch abhängig von einer anderen Information, sind auch die zugehörigen Daten und Pakete semantisch abhängig voneinander, da sie die entsprechende Information repräsentieren.

Zusätzlich taucht innerhalb der Arbeit auch der Begriff des Pakets immer wieder auf. Er kann aus zwei Ebenen des OSI-Protokollstacks heraus betrachtet werden. Aus dem Kontext der Netzwerkebene betrachtet, handelt es sich bei einem Paket um eine bestimmte Menge an Daten, die über Netzwerke übertragen werden. Hierin können mehrere Datensätze in einem Paket zusammengefasst werden. Große Datensätze wiederum können auch zur Übermittlung auf mehrere Pakete aufgeteilt werden. Der Kontext der Anwendungsebene, auf die sich diese Arbeit fokussiert, betrachtet jedoch nicht die einzelnen Pakete, sondern den übertragenen Datensatz. Dabei ist es unerheblich, ob auf Netzwerkebene mehrere Pakete verwendet wurden, um die Übertragung durchzuführen. In der Software kommen diese übertragenen Daten wieder als ein gemeinsamer Datensatz an und werden daher hier auch nur als ein Objekt betrachtet.

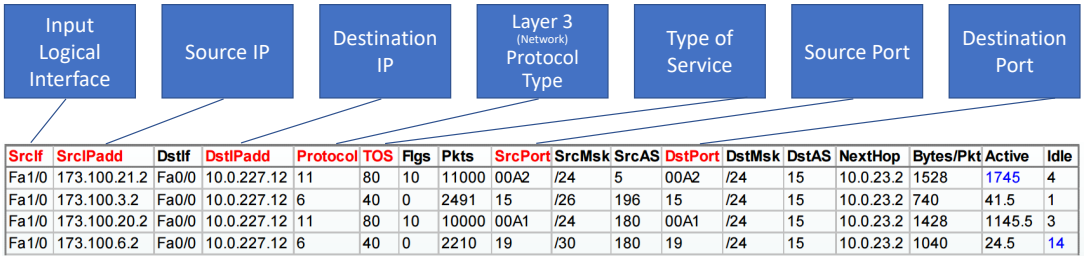
2.3 Weitere verwandte Fachgebiete

Vorausgehend wurde bereits Grundlagenliteratur zum Thema dieser Arbeit vorgestellt und diskutiert. Im Folgenden werden anhand von drei weiteren Fachgebieten verwandte Arbeiten zum Lösungsansatz IFM beschrieben. Die verwandten Fachgebiete beschäftigen sich mit ähnlichen Problemstellungen oder sind bei der Entwicklung des IFM hilfreich und werden daher im Folgenden dargestellt.

2.3.1 Network Monitoring

Methoden zur Laufzeitüberwachung von Netzwerken liegen mit den so genannten Network Monitoring Tools vor, die zur Systemlaufzeit Netzwerk-Datenpakete analysieren und quantifizieren. Zu den konventionellen Methoden gehören hierbei Werkzeuge wie Cisco NetFlow [55] oder GNU cflow [56]. NetFlow und cflow sind sich technisch sehr ähnlich und generell in der Lage eine Menge an Informationen über einen kontinuierlichen IP-Datenstrom zu erfassen und verkehrstechnisch zu analysieren.

NetFlow ist also in der Lage, Paket-Aktivitäten im Netzwerk zu erfassen, zu historisieren und statistisch auszuwerten. Hierbei arbeitet NetFlow auf Paket-Ebene (OSI Schicht 3, Vermittlungsschicht) und ist somit ausschließlich in der



SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	Pkts	SrcPort	SrcMsk	SrcAS	DstPort	DstMsk	DstAS	NextHop	Bytes/Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.0.23.2	1528	1745	4
Fa1/0	173.100.3.2	Fa0/0	10.0.227.12	6	40	0	2491	15	/26	196	15	/24	15	10.0.23.2	740	41.5	1
Fa1/0	173.100.20.2	Fa0/0	10.0.227.12	11	80	10	10000	00A1	/24	180	00A1	/24	15	10.0.23.2	1428	1145.5	3
Fa1/0	173.100.6.2	Fa0/0	10.0.227.12	6	40	0	2210	19	/30	180	19	/24	15	10.0.23.2	1040	24.5	14

Abbildung 6: Von NetFlow bereitgestellte Datenflussinformationen (in Anlehnung an [55])

Lage, technische, aber keine fachlichen Informationen zum Datenfluss zu erfassen. Fachliche Informationen können ausschließlich auf Schicht 7 (Anwendungsschicht) erfasst werden, da es nur hier möglich ist, Inhalte der versendeten Nachrichten zu verstehen und Abhängigkeiten zu beschreiben. Eine exemplarische Darstellung der von NetFlow erfassbaren Daten ist in Abbildung 6 dargestellt. Hierbei handelt es sich um die Rohdaten, die erfasst werden können. Aus diesen Daten lassen sich allerdings nur für Quality of Service (QoS)[14] relevante Informationen sowie weitere quantitative Statistiken ableiten, aber keine Aussagen über die inhaltlichen Abhängigkeiten der einzelnen Pakete erstellen. Diese aufgezeichneten Informationen umfassen Eigenschaften wie Datenmenge oder die Art der versandten Datenpakete. Sie können nicht genutzt werden um semantische Abhängigkeiten zu erkennen.

Die hier dargestellte Problematik macht diese Werkzeuge in der dargestellten Problemdomäne nur bedingt einsetzbar. Konzepte zur Datenerfassung lassen sich möglicherweise übertragen. NetFlow beispielsweise ist in die Netzwerkinfrastruktur integriert, sammelt dort Daten und übermittelt diese weiter an eine zentrale Instanz im Netzwerk, die Netz-Informationen empfängt, sammelt und auswertet. Die Datenerfassung erfolgt also dezentral, die Auswertung allerdings zentralisiert. Dieser Ansatz ist auch in CPS sinnvoll, da CPS keine zentralisierten Netze sind, die Auswertung von Abhängigkeiten aber möglichst zentral und übersichtlich dargestellt werden soll. Ähnliche Ansätze wie NetFlow verfolgen auch CeMon [57] und cSamp [58], diese sind aber ebenfalls nicht in der Lage inhaltliche

[14] Quality of Service (QoS), oder der eher unübliche deutsche Begriff der Dienstgüte, beschreibt die Güte eines Dienstes oder Systems aus Perspektive des Anwenders. Bezogen auf ein Netzwerk beschreibt QoS also insbesondere dessen Zuverlässigkeit und Geschwindigkeit. Formal beschreibt sie eine Menge von Qualitätsanforderungen.

Abhängigkeiten zu erkennen. Relevanz bzgl. dieser Arbeit haben sie insbesondere in Bezug auf ihre möglicherweise übertragbare Architektur.

In Kontrast zu diesen dargestellten Ansätzen versuchen Marashi et al. [49] Abhängigkeiten zwischen physischen und informationstechnischen Komponenten, also Hardware und Software, in CPS zu modellieren. Dieser Ansatz betrachtet nicht nur Netzwerk-Eigenschaften, sondern bezieht sich ebenfalls auf Inhalte. Durch die dynamische Struktur und die hier vor Laufzeit angewandte Modellierung von Abhängigkeiten wird der Ansatz der Emergenz in CPS allerdings nicht gerecht und erlaubt keine Identifizierung von Abhängigkeiten zur Systemlaufzeit. Für einzelne Abschnitte im CPS scheint die Modellierung von Abhängigkeiten jedoch sinnvoll (vgl. Abschnitt 3.3).

2.3.2 Informationsflussverfolgung

Einen verwandten Ansatz zu dieser Arbeit stellen Noak et al. mit der sogenannten Informationsflussverfolgung vor [59]. Die Autoren definieren hierfür Testfälle in Form von bekannten Ursache-Wirkungs-Graphen. Diese Graphen beschreiben die Zusammenhänge und Abhängigkeiten einzelner Nachrichten im Netzwerk, die allerdings bekannt sein müssen. Anschließend findet ein Network Monitoring statt, das versucht, einzelne Nachrichten mit dem Graphen in Beziehung zu setzen. Ein Testfall ist genau dann erfolgreich, wenn die durch Network-Monitoring ermittelten Nachrichten im Bezug des Ursache-Wirkungs-Graphen bzgl. Zeitpunkt und Häufigkeit korrekt sind. Die Überwachung der ausgetauschten Nachrichten findet hier ebenfalls auf Netzwerkebene statt. Eine Überwachung auf Applikationsschicht ist hier jedoch auch nicht notwendig, da bereits vor der Überwachung definiert wurde, welche Abhängigkeiten existieren. Zur Erkennung von Abhängigkeiten zwischen ausgetauschten Informationen eignet sich der Ansatz nicht.

Der Ansatz erkennt aber insbesondere an, dass so Knoten lokalisiert werden können, die Fehlverhalten ausweisen. Die Betrachtungsweise ist jedoch umgekehrt zum IFM-Konzept: Beim IFM sollen Abhängigkeiten ermittelt werden, um beobachtetes Fehlverhalten nachvollziehbar zu gestalten. Bei der von Noak et al. vorgestellten Informationsflussverfolgung werden Abhängigkeiten definiert und anschließend überprüft, ob diese tatsächlich wie gewünscht existieren oder ob Fehlverhalten vorliegt.

2.3.3 Topology Discovery

Die Kenntnis von Netzwerk-Topologien ist sowohl für das Netzwerk-Management als auch zur Netzwerk-Analyse essenziell. Unter anderem ist die Kenntnis der aktuellen Topologie des Netzes bei der Erkennung und Korrektur von Fehlern von zentraler Bedeutung. Heutige dynamische Netze, deren Topologie sich zur Laufzeit verändert, benötigen daher Methoden, um deren Topologie automatisch ermitteln zu können. [60]

Oft angewandte Protokolle zur Topology Discovery sind das Simple Network Management Protocol (SNMP) und das Internet Control Message Protocol (ICMP), teilweise auch Kombinationen aus diesen [61]. Beim SNMP handelt es sich um ein standardisiertes Protokoll, das von Netzwerkinfrastruktur wie Routern oder Switches unterstützt wird, nicht aber von Endgeräten. Auf Basis dieser Protokolle lassen sich Netzwerktopologien auf Hardware-Ebene ermitteln. Verschiedene Forschungsgruppen haben auf Basis dieser Protokolle Methoden entwickelt, mit deren Hilfe sich Topologien von dynamisch geprägten Netzen zur Laufzeit ermitteln lassen [60, 62].

Aus der ermittelten technischen Topologie resultieren die allgemein möglichen Abhängigkeiten zwischen Knoten auf Hardware-Ebene – die sogenannte physische Topologie. Die tatsächliche Kommunikation zwischen Knoten und deren Struktur – die logische Topologie – lässt sich hieraus jedoch nicht ableiten, da in Internet Protocol (IP)-basierten Netzen auch topologisch entfernte Knoten direkt miteinander kommunizieren können. Aus Software-Perspektive ist somit die Erkennung der Topologie der Hardware des Netzwerkes nicht von Belang, stattdessen muss die logische Topologie der Informationsflüsse ermittelt werden. Im Fall der Kommunikation von CPS über das Internet ist die Hardware-Topologie des verwendeten Netzwerkes zum einen enorm groß, zum anderen wenig aussagekräftig in Bezug auf Abhängigkeiten zwischen den einzelnen Komponenten des CPS. Dennoch können diese Methoden zu Debugging- und Monitoring-Zwecken auch im Rahmen dieser Arbeit nützlich sein.

2.4 Fazit

In den vorausgehenden Abschnitten wurden verwandte Arbeiten aus den Bereichen des Network Monitorings und der Informationsflussverfolgung vorgestellt. Keiner der vorgestellten Ansätze eignet sich für einen direkten Vergleich mit dem IFM, da entweder keine semantischen Abhängigkeiten, also Abhängigkeiten von Informationen, betrachtet werden, oder kein anwendbares Konzept, Protokoll und Werkzeug zur Verfügung steht, das verglichen werden könnte.

Das IFM-Konzept, dass in den folgenden Kapiteln vorgestellt wird, soll semantische Abhängigkeiten zwischen ausgetauschten Informationen aufzeichnen und auswertbar machen. So kann im Fehlerfall die ursprüngliche Quelle von Fehlinformationen, die zu einer Fehlerkaskade führt, identifiziert werden. Der IFM identifiziert die Quelle dabei nicht automatisch – soll aber als Debugging-Werkzeug wichtige Abhängigkeitsinformationen liefern, die eine Identifizierung ermöglicht und vereinfacht.

3 Information Flow Monitoring

In diesem Kapitel: Beschreibung, Definition und Beispiele zum IFM-Konzept, das zum besseren Umgang mit Abhängigkeiten und Cascading Data Corruption in CPS beitragen soll. Anwendungsbeispiele werden zur Illustration der Funktionsweise angeführt. Methoden und Erweiterungen zum Einsatz in verschieden ausgeprägten CPS werden vorgestellt.

3.1 Information Flow Monitor

Der in dieser Arbeit eingeführte Begriff des Information Flow Monitoring leitet sich vom Network Flow Monitoring ab. Als Network Flow bezeichnet man den Kommunikationsfluss zwischen zwei Knoten im Netzwerk. Ein Network Flow ist hierbei eine abstrakte Darstellung der Verbindung, beschreibt jedoch nicht den Inhalt der tatsächlich fließenden Daten [63], sondern nur Metadaten[15]. Fokus ist hier der Fluss von Daten und Informationen durch das Netzwerk und deren Veränderung.

Beim Network Flow Monitoring werden Metadaten über den Datenfluss im Netzwerk gesammelt, um Aktivitäten, (Miss-)Konfigurationen, Richtlinien-Verstöße, die Datenmenge und Nadelöhre zu erkennen oder aufzuzeichnen. Diese Metadaten beziehen sich insbesondere auf die Eigenschaften der Nachricht wie beispielsweise Größe, verwendetes Protokoll, Ports, IP-Adressen oder Informationen über Hops. Es handelt sich also insbesondere um Informationen, die zum Aufbau und zur

[15] Metadaten sind Daten, die Merkmale und Eigenschaften anderer Daten beschreiben, aber nicht deren eigentlichen Inhalt.

Wartung des Netzwerkes von Belang sind. Der eigentliche Inhalt der übermittelten Daten ist hierbei nicht von Belang.

Abhängigkeiten zwischen zwei Knoten können automatisch durch den Network Flow erkannt werden, da Informationen über kommunizierende Systeme in einem Network Flow enthalten sind. Abhängigkeiten zwischen übermittelten Informationen können jedoch nicht automatisch erkannt werden, da der Verarbeitungsprozess der Informationen durch die Knoten nicht von außen beobachtbar ist. Ein Knoten erzeugt Abhängigkeiten durch die interne Verarbeitung von Daten. Der Prozess der Verarbeitung dieser Daten ist im Netzwerk nicht sichtbar. Network Flow Monitoring wird oft von Netzwerkinfrastuktur wie Routern oder Switches geleistet, die keinen Zugriff auf den Inhalt der übermittelten Pakete haben. [64, 65]

Das Information Flow Monitoring verschiebt den Fokus vom Netzwerk zu den übermittelten Informationen. Das Information Flow Monitoring kann nicht wie beim Network Monitoring durch Netzwerk-Infrastruktur geschehen, da semantische Abhängigkeiten zwischen erstellten Informationen nicht automatisch erkannt werden können, sondern explizit benannt werden müssen. Bei einer Information ist nicht offensichtlich, ob der Sender die Primärquelle ist oder ob er sich auf andere beruft. Dies kann nur durch die Mitarbeit der teilnehmenden Knoten im Netzwerk ermöglicht werden, da nur diese jeweils für die durch sie erzeugten Informationen wissen, welche Informationen zur Generierung von neuen Informationen herangezogen wurden.

Das Hauptaugenmerk des Information Flow Monitorings ist die Software der einzelnen Knoten und die Nachvollziehbarkeit deren Verhaltens. Der Datentransport auf Netzwerkebene hat keinen Einfluss auf die inhaltlichen Abhängigkeiten der erstellten Informationen. Durch den Fokus auf die übermittelten Informationen werden beim Information Flow Monitoring die ersten vier Schichten des OSI-Modells verlassen, an denen das Network Flow Monitoring durchgeführt wird. Informationen werden jedoch in der Applikationsschicht verarbeitet und aggregiert.

Es wird also eine Lösung in der Applikationsschicht benötigt, da hier die Entscheidungen getroffen werden, die zu Abhängigkeiten zwischen Informationen führen. Während Network Monitoring also in der Regel auf der Transportschicht arbeitet, fokussiert Information Flow Monitoring die Applikationsschicht und ist

somit stark mit der datenverarbeitenden Software der einzelnen Knoten im CPS verknüpft.

In den vorherigen Kapiteln wurden CPS beschrieben, die sowohl teilweise emergentes Verhalten zeigen können als auch bzgl. ihrer Topologie in manchen Anwendungsfällen teilweise unbekannt sein können. Im Folgenden werden nur noch solche Netze betrachtet, die kein emergentes Verhalten zeigen und deren Topologie bekannt ist. Für diese Submenge der CPS wurde das IFM-Konzept entworfen und betrachtet ihre spezifischen Eigenschaften. Das bedeutet nicht, dass sich das IFM-Konzept nicht auch für emergente Netzwerke entwickeln lassen würde – ihre Anforderungen, insbesondere bzgl. autonomer Restrukturierung der Lösung, um den emergenten Veränderungen Schritt zu halten, sind jedoch andere. Dadurch, dass sich diese Submenge der emergenten CPS ständig in ihrer Topologie verändern werden Methoden benötigt, um auch den IFM ständig an diese Veränderungen anzupassen. Damit dies nicht manuell geschehen muss, werden Automatismen benötigt, die die selbstständige Anpassung des IFM sicherstellen.

Auch topologisch unbekannte Netzwerke haben andere Anforderungen, die insbesondere bei der Einbettung der Komponenten einer eigenen Information Flow Monitoring Lösung und deren Platzierung im Netzwerk beachtet werden müssten. Unbekannte Netze setzen insbesondere entweder voraus, dass durch geeignete Methoden die Topologie aufgedeckt werden kann oder das für die Umsetzung der Lösung die Topologie nicht bekannt sein muss.

Daher fokussiert diese Arbeit das Information Flow Monitoring in CPS in nichtemergenten Netzen mit bekannten Topologien[16]. In Abschnitt 5.3 wird diskutiert, wie das IFM-Konzept für emergente Netze erweitert werden könnte.

3.1.1 Anforderungen

In den vorausgegangenen Kapiteln wurden die Probleme bei der Nachverfolgung von Abhängigkeiten in CPS, die mit Hilfe von Information Flow Monitoring

[16] Die in Abschnitt 2.3.3 vorgestellten Methoden können genutzt werden, um Topologien aufzudecken oder besser zu verstehen.

behoben werden sollen, deutlich. Die zu lösenden Probleme sind wie folgt zusammengefasst:

- **Wirkung und Ursache sind entkoppelt und lassen sich nicht in Verbindung bringen:** Wie bereits in Abschnitt 2.2 beschrieben, existieren in CPS Abhängigkeiten zwischen ausgetauschten Informationen. Beim Beobachten einer Wirkung soll es möglich sein, die entsprechende(n) Ursache(n) zu ermitteln. Eine Ursache für eine Wirkung soll in Form einer Information, die im Netzwerk ausgetauscht wurde, ermittelbar sein.

  Eine Wirkung kann hierbei am einfachsten bei Aktoren in der physischen Welt beobachtet werden. Führt ein Aktor eine möglicherweise überraschende oder ungewollte Aktion in der realen Welt aus, ist es wichtig die Ursache hierfür zu ermitteln. Eine Wirkung kann allerdings, wenn auch deutlich schwieriger, beispielsweise in Datenbanken bei Aggregatoren beobachtet werden, wenn hier fehlerhafte Daten abgelegt werden. Eine Ursache zu diesen Wirkungen ist das Verhalten eines oder mehrere Akteure im Netzwerk, die die Wirkung beeinflussen. Die Ursache wird in Form einer erzeugten Information ermittelbar. Abgesehen von Fehlern muss es auch möglich sein, jegliche beobachtete Wirkung zurückzuverfolgen. Die Nachverfolgung von Informationen und die Aufdeckung derer Abhängigkeiten soll also immer und unabhängig vom Fehlerfall möglich sein.

- **Abhängigkeiten bleiben unverständlich und lassen sich nicht nachvollziehen:** In großen CPS werden enorme Mengen an Nachrichten ausgetauscht, die manuell nicht mehr zu handhaben sind. Hieraus resultiert auch eine entsprechend große Menge an erfassten Abhängigkeiten im System. Die Informationen über die Abhängigkeiten müssen daher so aufbereitet werden, dass ein einfaches Verständnis möglich ist. Eine reine Sammlung der Abhängigkeiten der ausgetauschten Informationen allein ist nicht nützlich.

- **Dezentralität des Netzwerkes muss bewahrt bleiben:** Klassische Cloud-Systeme haben üblicherweise eine Client-Server Architektur mit einer Sterntopologie. Dies bedeutet, dass alle Clients sternartig um einen zentralen Server angeordnet sind und mit diesem verbunden sind. Die wenigsten CPS sind jedoch vollständig zentralisierte Netzwerke mit einem allwissenden

Server [4] – stattdessen handelt es sich um Netze, die sehr unterschiedliche Topologien haben können.

- **Kompatibilität mit CPS-spezifischen Eigenschaften:** CPS unterscheiden sich in bestimmten Eigenschaften (vgl. Abschnitt 2.1.1) von anderen Systemen. Wesentliche Aspekte hierbei betreffen die Leichtgewichtigkeit einiger Komponenten sowie die Zusammenstellung heterogener Knoten im Netz. Leichtgewichtigkeit kann sich hierbei beispielsweise auf Rechenzeit oder Energiebedarf beziehen. Die Heterogenität bezieht sich beispielsweise auf die Verwendung von verschiedenen Protokollen in CPS und die Kompatibilität der einzelnen Knoten zu diesen Protokollen.

Aus den beschriebenen Aspekten lassen sich verschiedene Anforderungen ableiten, die ein Werkzeug zum Information Flow Monitoring erfüllen muss. Die folgenden Anforderungen sind als Grundlage zur Entwicklung des IFM zu verstehen.

- **Anforderung 1:** *Wirkung zur Ursache zurückverfolgen*

  Im Netzwerk beobachtetes Verhalten beruht immer auf einer zugrundeliegenden Information, die am Ort der Beobachtung vorliegt. Diese Information beruht oft auf anderen im Netzwerk ausgetauschten Informationen und wurde durch Aggregatoren erstellt. Zusammenhänge zwischen diesen Informationen sollen nachvollziehbar werden. Hierdurch werden Aggregationen nachträglich nachvollziehbar und es wird erkennbar, welche Informationen zusammengeführt wurden, um neue Informationen daraus abzuleiten.

- **Anforderung 2:** *Abhängigkeiten verständlich machen*

  Aufgezeichnete Abhängigkeiten von Informationen sind durch die reine Betrachtung der aufgezeichneten Daten für den Menschen unverständlich. Die Daten müssen aufbereitet und visualisiert werden, damit der Benutzer die Zusammenhänge zwischen den ausgetauschten Informationen einfacher und schneller verstehen kann. Dem Benutzer soll hierbei eine zentrale Anlaufstelle zur Verfügung gestellt werden, die alle erfassten Abhängigkeiten des überwachten Netzwerkes visualisieren kann. Einzelne Informationen sollen abgefragt werden können, um deren Abhängigkeiten anzuzeigen.

- **Anforderung 3:** *Dezentralität des Netzwerkes bewahren*

  Die Wahl der Topologie darf durch das Konzept zur Nachverfolgung von Abhängigkeiten nicht eingeschränkt werden. Dezentrale Netze müssen weiter möglich sein. Insbesondere darf nicht vorausgesetzt werden, dass alle Knoten mit einem zentralen Server verbunden sind, da in CPS oft Punkt-zu-Punkt Verbindungen über Funk eingesetzt werden und nicht jeder einzelne Knoten mit dem Internet verbunden ist.

- **Anforderung 4:** *Kompatibilität mit CPS-spezifischen Eigenschaften*

  Die Software einiger Knoten im CPS ist unveränderbar. Das eingesetzte System muss damit umgehen können, dass vereinzelt (aber nicht alle) Komponenten im Netzwerk nicht mit eigener Software ausgestattet werden können und daher das IFM-Protokoll nicht unterstützen. Es muss also zumindest möglich sein, dass einzelne Knoten im CPS nicht IFM-kompatibel sind, ohne die Gesamtfunktionalität des IFM zu stören und das Sammeln von Abhängigkeitsinformationen zu verhindern. Im Idealfall lassen sich auch diese Knoten in das IFM-Konzept integrieren.

- **Anforderung 5:** *CPS-adäquater Umgang mit Ressourcen*

  Die Ressourcen einiger CPS-Komponenten sind stark eingeschränkt. Dies betrifft Speicher, Energie, Verfügbarkeit oder Rechenleistung der Sensoren, Aktoren und Aggregatoren. Die Lösung muss hierauf Rücksicht nehmen.

Die Probleme und daraus resultierenden Anforderungen haben Einfluss auf das zu erstellende Werkzeug, das die Probleme berücksichtigen und die Anforderungen erfüllen soll. Die auf den Anforderungen basierende Architektur wird im Folgenden vorgestellt.

3.1.2 Architektur

Auf Basis der zuvor genannten Anforderungen wird in dieser Arbeit ein Konzept und Werkzeug entwickelt. Das Konzept trägt den Namen Information Flow Monitoring (IFM-Konzept) und wird durch das Werkzeug mit dem Namen Information

Flow Monitor (IFM) [52][17][66][18] umgesetzt. Im Folgenden werden Architektur, Komponenten und Funktionsweise des IFM beschrieben.

Durch die Heterogenität von CPS und die Verteiltheit der Systeme in CPS-Netzwerken kann nicht davon ausgegangen werden, dass jeder Knoten mit einer zentralen Instanz oder dem Internet verbunden ist [67]. Daher kann sich eine zentrale IFM-Instanz, die Daten sammeln und visualisieren soll, nicht an den Daten einer zentralen Komponente im Netz bedienen, da diese nicht existiert. Auch die Abfrage von Abhängigkeitsinformationen durch eine zentrale IFM-Instanz direkt bei den Knoten im Netzwerk ist nicht möglich, da nicht alle Knoten im Netzwerk mit dieser Instanz über beispielsweise das Internet verbunden sein müssen.

Gleichzeitig besteht jedoch die Anforderung, dass Abhängigkeitsinformationen zentral abgefragt werden können, damit für Auswertungen alle notwendigen Informationen zur Verfügung stehen und gemeinsam visualisiert werden können. Insofern muss eine Instanz im Netzwerk existieren, die diese Informationen sammelt. Im Folgenden ist dieser Knoten im Netzwerk mit *IFM-Master* benannt. Wie die Sammlung der Informationen technisch abläuft, wird in den folgenden Kapiteln genauer beschrieben. Abhängigkeiten zwischen Informationen entstehen an jedem datenverarbeitenden Knoten im Netzwerk, unabhängig davon, ob dieser Knoten mit einer zentralen Instanz kommunizieren kann oder nicht. Das führt dazu, dass nicht jeder Knoten, an dem Abhängigkeitsinformationen entstehen, diese direkt auch selbst an den *IFM-Master* kommunizieren kann. Diese Eigenschaft muss im Architektur- und Protokollentwurf bedacht werden.

[17] Dieser Beitrag wurde veröffentlicht in:
S. Gries, M. Hesenius und V. Gruhn. „Tracing Cascading Data Corruption in CPS with the Information Flow Monitor“. In: *New Trends in Intelligent Software Methodologies, Tools and Techniques - Proceedings of the 16th International Conference, SoMeT_17*. Hrsg. von H. Fujita, A. Selamat und S. Omatu. Bd. 297. Frontiers in Artificial Intelligence and Applications. Kitakyushu, Japan: IOS Press, 2017, S. 399–408. ISBN: 978-1-61499-799-3. DOI: 10.3233/978-1-61499-800-6-399

[18] Dieser Beitrag wurde veröffentlicht in:
S. Gries, M. Hesenius und V. Gruhn. „Tracking Information Flow in Cyber-Physical Systems“. In: *Proceedings - 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)*. 2017, S. 2589–2590. DOI: 10.1109/ICDCS.2017.116

Es existiert aber eine Submenge von Knoten, die direkt mit dem Master verbunden sind und mit diesem kommunizieren können. Ihre Aufgabe ist es, Abhängigkeitsinformationen von umgebenen Knoten zu sammeln und diese an den Master zu melden. Diese Knoten werden im Folgenden *Spy-Knoten* genannt.

- In jedem Netzwerk existiert ein ***IFM-Master*** („Master“). Dieser Knoten sammelt alle Abhängigkeitsinformationen und bereitet diese auf, damit ein Benutzer Abhängigkeitsinformationen abfragen kann. Der Master ist später zentrale Anlaufstelle, um Abhängigkeiten auszulesen / zu ermitteln und zu visualisieren.

- Ein Teil der Knoten im Netzwerk sind direkt mit dem IFM-Master verbunden. Einige davon sind ***IFM-Spies*** („Spy“), sammeln Abhängigkeitsinformationen ihrer Umgebung und geben diese an den Master weiter.

- Andere Knoten, die weder Spy noch Master sind, werden ***IFM-Worker*** („Worker“) genannt. Sie zeichnen eigene Abhängigkeitsinformationen auf und geben diese, gemeinsam mit von anderen Workern erhaltenen Abhängigkeitsinformationen, an die Spies oder andere Worker ihrer Umgebung weiter.

Ein Datenfluss einer Abhängigkeitsinformation beginnt bei einem Worker und verläuft dann über andere Worker und genau einem Spy zum Master. Letzterer sammelt die Abhängigkeitsinformationen.

Eine Übersicht der im Folgenden beschriebenen Akteure ist anhand eines kleines, exemplarischen CPS in Abbildung 7 dargestellt. Das Kommunikationsprotokoll des IFM wird anschließend in Abschnitt 3.1.3 erläutert.

3.1.2.1 IFM-Worker

Worker sind Knoten im Netzwerk, die eine bestimmte Aufgabe im CPS erfüllen. Dies können sie als Sensor, Aggregator oder Aktor leisten. Zusätzlich hierzu erfüllen sie ihre IFM-spezifischen Aufgaben als Worker. Durch die Kommunikation der Worker und die Aggregation von Informationen durch Worker existieren Abhängigkeiten im Netzwerk. Abhängig davon, ob der Knoten ein Sensor, Aggregator oder Aktor ist, erhält der Knoten als Worker zusätzlich Aufgaben:

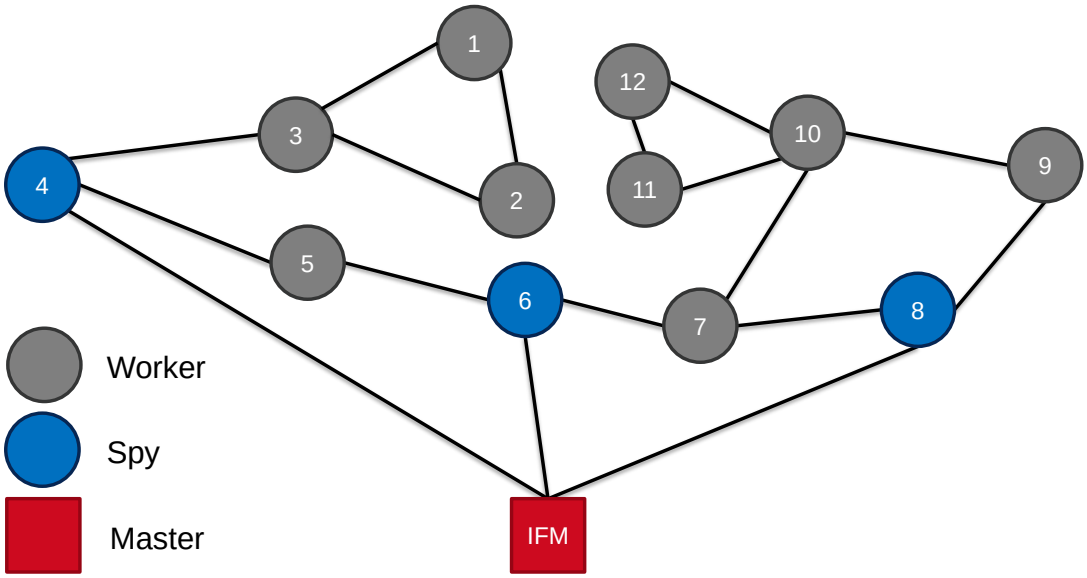


Abbildung 7: IFM Architekturübersicht

- **Sensoren** erzeugen neue Daten im Netzwerk. Um Abhängigkeiten beschreiben zu können, müssen diese Informationen identifizierbar sein. Ein Sensor markiert daher jeden erzeugten Sensorwert mit einem Identifikator, der zur Benennung einer Abhängigkeit einer anderen Information von diesem Sensorwert verwendet werden kann.

- **Aggregatoren** verwenden vorhandene Informationen ihrer Umgebung, beispielsweise Sensorwerte, um neue Informationen oder Erkenntnisse zu erzeugen. Dies geschieht durch Business-Logik, die nur diesem Aggregator bekannt ist. Daher muss der Aggregator bekannt geben, welche Informationen genutzt wurden, um zu einer bestimmten Erkenntnis zu gelangen. Emittiert der Aggregator eine neu erstellte Information, wird diese ebenfalls mit einem Identifikator versehen. Zusätzlich gibt der Aggregator bekannt, welche bereits bekannten Informationen genutzt wurden, um diese neue Information zu erzeugen. Der Verweis auf diese verwendeten Informationen geschieht über deren Identifikator.

- **Aktoren** sind Informationssenken im Netzwerk, die selbst keine Daten emittieren. Daher erzeugen diese auch keine neuen Abhängigkeiten, die erfasst werden müssten.

Gibt ein Knoten also Informationen weiter, ist jedes emittierte Informationsfragment über einen Identifikator eindeutig benennbar. Durch diese Benennbarkeit einzelner Informationen wird es möglich, dass Aggregatoren kommunizieren, welche Informationen genutzt wurden, um neue zu generieren. Emittiert ein Aggregator neue Informationen, so teilt er auch mit, welche anderen Informationen genutzt wurden, um diese neuen zu generieren. Dem Empfänger der neuen Information sind so die Abhängigkeiten der neuen Information zu anderen Informationen bekannt.

Verwendet ein Aggregator bereits aggregierte Informationen, um eine neue Information zu erzeugen, entsteht eine Hierarchie von Abhängigkeiten. In diesem Fall wird die gesamte Hierarchie mit emittiert und dem Empfänger mitgeteilt. Details zur technischen Umsetzung sind in Abschnitt 3.1.3 zu finden.

3.1.2.2 IFM-Spy

Ein Spy ist ein Worker, der mit dem Master verbunden ist. Er erhält zusätzlich zu seinen Aufgaben als Worker noch Spy-spezifische Aufgaben. Wichtigste zusätzliche Aufgabe ist die Weitergabe der Abhängigkeitsinformationen an den Master.

Spy-Knoten erhalten, wie alle anderen Knoten, Daten von anderen Knoten der Umgebung. Diese Daten werden dann genutzt, um eigene Erkenntnisse zu generieren und zu emittieren. Dieses Verhalten ist bereits durch die Worker bekannt. Die Menge der vorliegenden Abhängigkeitsinformationen steigt jedoch mit jedem Knoten, den die Informationen passieren, weiter an. Jeder verarbeitende Knoten fügt eine Hierarchieebene den Abhängigkeiten hinzu. Dies führt dazu, dass die Menge der mitgeführten Daten mit der Zeit stark ansteigt, da bei jeder Aggregation von Informationen die Abhängigkeitsmenge wächst. Bei der Aggregation mehrerer Informationen mit jeweils eigenen Abhängigkeiten werden die jeweiligen Abhängigkeitszweige zusammengefügt. Wird also eine neue Information C aus zuvor erhaltenen Informationen A und B erzeugt, enthält die Abhängigkeitshierarchie von C auch die Abhängigkeiten zu A und B, sowie deren Abhängigkeitshierarchie (vgl. Abbildung 8).

Jedes Mal, wenn Abhängigkeitsinformationen einen Spy-Knoten passieren, sendet der Knoten diese an den Master. Anschließend emittiert der Spy nur noch unvoll-

ständige, um die dem Master bereits mitgeteilten Abhängigkeitsinformationen bereinigten, Abhängigkeitsinformationen. Hierdurch reduziert sich jedes Mal, wenn Daten einen Spy-Knoten passieren, die Menge der mitgeführten Abhängigkeitsinformationen und deren Größe steigt nicht beliebig an. Dies geschieht nur, wenn auf dem Übertragungspfad der Information im Netzwerk auch Spy-Knoten liegen. Die hieraus resultierende Problematik der Platzierung der Spy-Knoten wird in Abschnitt 3.2 weiter beleuchtet.

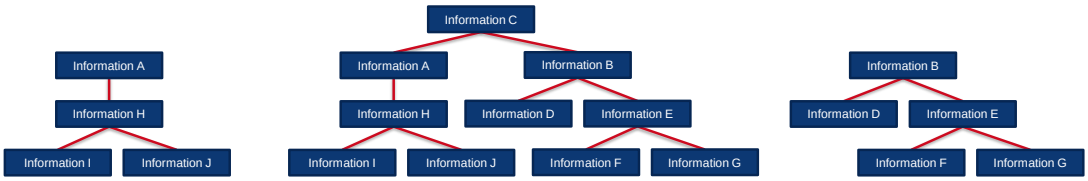


Abbildung 8: Zusammenführen von Abhängigkeiten

Es existieren Orte im Netzwerk, die Informationssenken darstellen. Dies betrifft einerseits alle Aktoren, die empfangene Informationen nur noch nutzen, um echtweltliche Aktionen auszuführen, andererseits aber auch manche Aggregatoren, die beispielsweise als Datenbank fungieren können. Erreicht eine Information eine solche Senke, gehen Abhängigkeitsinformationen verloren, bevor diese von einem Spy an den Master gesendet werden können. Dies ist immer dann der Fall, wenn ein Informationspfad nicht mit einem Spy endet. Um diesem Problem zu begegnen, muss sichergestellt werden, dass auch diese Abhängigkeiten über einen Spy an den Master gelangen. Daher werden diese Abhängigkeitsinformationen über denselben Pfad zurückgesendet, bis sie einen Spy erreichen. Dieses als BackPush bezeichnete Verfahren wird in Abschnitt 3.1.3.3 detailliert beschrieben.

3.1.2.3 IFM-Master

Der Master fungiert als zentrales Organ im IFM-Netzwerk. Das bedeutet, dass der Master zwar mit jedem Spy verbunden ist, um Abhängigkeitsinformationen zu sammeln, jedoch nicht mit jedem anderen Knoten im Netzwerk verbunden ist. Für die Funktion des CPS selbst spielt der Master keine zentrale Rolle. Der Master hat insbesondere drei Aufgaben:

- **Sammeln und Speichern** von erhaltenen Abhängigkeitsinformationen der Spies.

- **Aufbereiten und Zusammenfügen** der erhaltenen Abhängigkeitsinformationen.

- **Visualisierung und Bereitstellung** aller Abhängigkeiten eines zu analysierenden Informationsfragments.

Durch die beschriebene Architektur wird der Master zu einer zentralen Anlaufstelle zur Datenanalyse, ohne jedoch selbst bzgl. der Topologie des CPS Zentralität zu fordern.

Der Master erhält auf diesem Wege viele einzelne Informationen über Abhängigkeiten einzelner Informationen, die im Netzwerk erzeugt und ausgetauscht wurden. Seine Aufgabe ist es, diese Teile zu einem Gesamtbild zusammenzufügen. Zusammenhänge zwischen einzelnen Informationen können über deren Identifikatoren hergestellt werden. Der Master enthält auch ein Werkzeug, das Abhängigkeiten aufdecken soll. Daher hat er ein User Interface (UI), dass die Abhängigkeiten visualisiert und IFM-Visualizer genannt wird. Die Visualisierung geschieht mit Hilfe von Bäumen. Eine ausgewählte zu analysierende Information wird hierbei als Wurzel des Baumes dargestellt, während die Blätter die abhängigen Informationen darstellen.

Der gesamte Prozess von der Informationsgewinnung der Abhängigkeiten bis zur Visualisierung dieser im Master ist in Abschnitt 3.1.4 exemplarisch beschrieben.

3.1.3 IFM-Protokoll

Das IFM-Protokoll stellt sicher, dass alle Abhängigkeiten, die zwischen ausgetauschten Informationen im Netzwerk existieren, auch in der zentralen IFM-Master-Instanz zusammengetragen werden. Voraussetzung hierfür ist, dass auf dem Informationspfad, auf dem die Abhängigkeiten entstehen, mindestens ein Spy liegt. Als IFM-Protokoll werden hierbei alle Konventionen, Verhaltensweisen und Datenformate bezeichnet, die eingehalten werden müssen, damit das IFM-Konzept seine Funktion erfüllen kann. Eingesetzt wird es zur Kommunikation zwischen Workern, Spies und dem Master. Die Funktionsweise des Protokolls wird im Folgenden dargestellt.

CPS verwenden zur Kommunikation verschiedene Protokolle, beispielsweise Message Queuing Telemetry Transport (MQTT), Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), WebSockets oder andere. Diese Protokolle werden oft anhand verschiedener Anforderungen im CPS ausgewählt und erfüllen einen besonderen Zweck. Würde man diese Protokolle vollständig durch ein anderes ablösen, gäbe man gleichzeitig die Spezialisierung auf verschiedene Anwendungsfälle durch verschiedene Protokolle auf. Daher ersetzt das IFM-Protokoll diese Protokolle auf Applikationsschicht nicht, sondern ergänzt sie. Das IFM-Protokoll wird in die bereits bestehenden Protokolle eingebettet und immer gemeinsam mit einem Transportprotokoll eingesetzt. Letztendlich wird eine IFM-konforme Nachricht in eine Nachricht eines Transportprotokolls eingebettet. So kann eine IFM-Nachricht beispielsweise auch über MQTT übertragen werden.

Bei der Kommunikation zwischen Knoten in CPS werden Daten übertragen, deren Inhalte keine Rückschlüsse auf deren Ursprung oder deren Abhängigkeiten zulassen. Das IFM-Protokoll soll diese Abhängigkeitsinformationen hinzufügen, damit diese durch die Spies und den Master gesammelt werden können. Bei der Übergabe von Daten zwischen zwei Knoten im Netzwerk werden durch das Protokoll nicht mehr nur die Nutzdaten (*Payload*) übertragen, sondern zusätzliche Abhängigkeitsinformationen. Ähnlich dem Extensible Markup Language (XML)-Protokoll [68, 69] werden hierbei die eigentlich zu übertragenden Daten, der Payload, in einem Umschlag (*Envelope*) verpackt. Auf dem Umschlag werden hierbei Metainformationen wie Abhängigkeiten, Identifikator usw. abgelegt, während der Payload im Umschlag verpackt ist. Zur Definition des Umschlags wird die JavaScript Object Notation (JSON) verwendet. Der Umschlag mit Inhalt kann so anschließend beliebig transportiert werden.

3.1.3.1 Kommunikation zwischen Workern (inkl. Spies)

Die Kommunikation zwischen Spy und Spy, Worker und Worker sowie Spy und Worker wird durch das JSON-Schema in Abbildung 9 definiert. Das Schema beschreibt dabei den Umschlag, in den der Payload eingelegt wird. Jede Nachricht, die im CPS zwischen zwei Knoten ausgetauscht wird, muss sich an dieses Format halten. Im Folgenden sind der Aufbau und der Inhalt dieser Nachrichten weiter beschrieben.

```
{
  "type": "object",
  "title": "definition of a received packet",
  "required": ["id", "payload", "source"],
  "properties": {
    "id": {
      "$id": "#/properties/id", "type": "string", "title": "UUID",
      "examples": ["3-4-82f2318986..."], "pattern": "^(.*)$"
    },
    "timestamp": {
      "$id": "#/properties/timestamp", "type": "date-time", "title": "timestamp",
      "examples": ["2019-11-13T20:20:39+00:00"]
    },
    "payload": {
      "$id": "#/properties/payload", "type": "string", "title": "data payload",
      "examples": ["encoded payload"], "pattern": "^(.*)$"
    },
    "source": {
      "$id": "#/properties/source", "type": "integer", "title": "source ID", "examples": [3]
    },
    "destination": {
      "$id": "#/properties/destination", "type": "integer", "title": "destination ID",
          "examples": [4]
    },
    "lastspy": {
      "$id": "#/properties/id", "type": "string", "title": "last passed spy node",
      "examples": ["3-4-82f2318986..."], "pattern": "^(.*)$"
    },
    "history": {
      "$id": "#/properties/history", "type": "array",
      "title": "history information", "items": {
        "$id": "#/properties/history/items", "type": "object", "title": "dependent packet
            list",
        "default": null, "required": ["id", "source", "chunks"],
        "properties": {
          "id": {
            "$id": "#/properties/history/items/properties/id", "type": "string",
            "examples": ["3-4-82f2318986..."], "pattern": "^(.*)$"
          },
          "source": {
            "$id": "#/properties/history/items/properties/source",
            "type": "integer", "examples": [3]
          },
          "destination": {
            "$id": "#/properties/history/items/properties/destination",
            "type": "integer", "examples": [4]
          },
          "chunks": {
            "$id": "#/properties/history/items/properties/chunks", "type": "array",
            "title": "second level dependencies", "default": [],
            "items": {
              "$id": "#/properties/history/items/properties/chunks/items", "type": "string",
              "title": "ID of a dependent packet", "examples": ["2-3-fc6cf64943..."],
              "pattern": "^(.*)$"
            }
          }
        }
      }
    }
  }
}
```

Abbildung 9: JSON-Schema zur Kommunikation zwischen Workern (inkl. Spies)

***id* – Identifikator** (Abbildung 9, Z. 6)

Mit Hilfe des IFM sollen Abhängigkeiten ermittelt werden können. Damit diese auch benannt werden können, muss jede Nachricht, die im Netzwerk ausgetauscht wird, auch eindeutig benennbar/identifizierbar sein. Jede Nachricht kann hierbei eine oder mehrere Informationen enthalten. Daher erhält jede einzelne Nachricht, die im CPS verschickt wird, einen Identifikator. Die *id* ist eindeutig und dient als Primärschlüssel der Nachricht. Die *id* kann beispielsweise als Hash aus der kompletten Nachricht inklusive Metadaten (ohne *id*) und *payload* gebildet werden. Zusätzlich ist ein Zeitstempel sinnvoll, um die Kollisionswahrscheinlichkeit[19] zu reduzieren. Der Hash wiederum ist dann eine Zahlen-Buchstaben-Kombination. Die *id* wird hierbei durch den Knoten vergeben, der die Information erzeugt. Dies kann also entweder ein Sensor oder ein Aggregator sein, der zusätzlich Worker oder Spy ist.

***timestamp* – Zeitstempel** (Abbildung 9, Z. 10)

Der optionale *timestamp* dokumentiert den Erstellungszeitpunkt der versendeten Information. Für das Protokoll selbst besitzt dieses Feld keine Relevanz und wird für die Funktion des IFM nicht benötigt. Für Zwecke des Debuggings sowie für das Information Dependency Modeling (vgl. Abschnitt 3.3) kann der Zeitstempel jedoch genutzt werden.

***payload* – Nutzdaten** (Abbildung 9, Z. 14)

Der *payload* enthält die Information(en), die zwischen den beiden kommunizierenden Knoten ausgetauscht werden soll. Dieser *payload* kann auf beliebige Weise kodiert und verschlüsselt werden. Letztendlich handelt es sich beim *payload*-Feld um die Daten, die als einzige auch ohne den Einsatz des IFM-Protokolls verschickt würden. Es handelt sich also um den Inhalt des Umschlags, die eigentlichen Nutzdaten, wie bspw. Sensordaten, aggregierte Information, usw. Der *payload* enthält keine wei-

[19] Beim Hashing bezeichnet eine Kollision das Hashen zweier Werte, die zum selben Hash führen. Dann eignet sich der Hash nicht mehr zur eindeutigen Referenzierung auf die gehashten Werte. Die Kollisionswahrscheinlichkeit kann reduziert werden, wenn zum Hashen möglichst viele wahrscheinlich nur einmalig vorkommende Werte zum Hashing genutzt werden.

teren IFM-Metadaten und ist für die Aufzeichnung der Abhängigkeiten nicht relevant. Die Abhängigkeiten werden allein durch die Metadaten beschrieben.

source / destination – **Sender- und Empfänger-Identifikatoren** (Abbildung 9, Z. 18, 21)

Auch jeder Knoten erhält einen eindeutigen Identifikator. An dieser Stelle bietet sich zur Unterscheidung eine fortlaufende Nummerierung an. Bei Netzwerken, in denen sich die Knoten selbst eine ID geben, kann ebenfalls ein Hash verwendet werden, eine Kollision muss aber ausgeschlossen werden. Dies kann beispielsweise durch das mithashen von Geo-Koordinaten der Knoten sichergestellt werden, wenn es sich um stationäre Knoten handelt. Je nach Kommunikationsprotkoll bieten sich allerdings beispielsweise auch IP- und Media Access Control (MAC)-Adressen sowie Client IDs (z. B. bei MQTT) an, die gleichzeitig auch eine einfache Adressierung bzw. einen Lookup der Knoten ermöglichen.

Da jedes JSON-Dokument des gezeigten JSON-Schemas eine Nachricht repräsentiert, die von einem dedizierten Knoten gesendet wird, ist der Identifikator des Senders klar benennbar. Das Hinzufügen des Identifikators des Empfängers ist optional, da die Auswertung der Nachricht beim Empfänger erfolgt, der seinen Identifikator kennt. Ebenfalls kann es bei manchen Protokollen wie MQTT mehrere Empfänger geben, sodass diese nicht explizit benannt werden müssen. Diese beiden Felder werden also nicht genutzt, um den Transport der Information zum richtigen Empfänger sicherzustellen, sondern nur, um den Transport zu dokumentieren. Ist der Empfänger schon beim Absender benennbar, ist es jedoch sinnvoll diesen auch zu dokumentieren, da wenn der Absender ein Spy ist, diese Information den Master so früher erreicht.

lastspy – **Informationen über den letzten Spy** (Abbildung 9, Z. 24)

IFM-Spy-Knoten sind dafür zuständig, Abhängigkeiten an die zentrale Stelle, den IFM-Master, weiterzuleiten. Immer wenn das geschieht, hinterlässt der Spy seine ID an dieser Stelle im JSON-Dokument. Worker wiederum lassen dieses Feld unangetastet, so dass zu jedem Zeitpunkt

der letzte Spy des Pfades nachvollzogen werden kann. Werden mehrere Nachrichten zusammengefügt, kann hier ein beliebiger Spy aus einer der Nachrichten hinterlegt werden. Wenn möglich, ist es sinnvoll den am schnellsten erreichbaren Spy zu hinterlegen. Die Anzahl der notwendigen Schritte, die benötigt werden, um die *lastspy*-Kandidaten zu erreichen, lassen sich aus dem *history*-Array ableiten. Die Möglichkeit den letzten Spy zu identifizieren wird später für den IFM-BackPush-Mechanismus (vgl. Abschnitt 3.1.3.3) benötigt.

***history* – Abhängigkeitsinformationen** (Abbildung 9, Z. 28)

Erstellt ein Knoten eine Nachricht, für die Sensordaten anderer Sensoren verwendet wurden, so ist die erstellte Nachricht von einigen Nachrichten des Sensors abhängig. Im *history*-Array werden Kopien der *id*, *source* und *destination* genau dieser Nachrichten abgelegt, zu denen eine Abhängigkeit besteht. Ist der *payload* klein, kann auch dieser optional hinzugefügt werden, um später die Nachverfolgung zu vereinfachen. Bei großen Datenmengen bietet sich dies jedoch nicht an, da die Daten später im Master gespeichert werden müssen. Das Hinzufügen des *timestamp* ist ebenfalls optional.

Das *history*-Array enthält auch eine Kopie der aktuell versendeten Nachricht, um dessen Identifikator zu speichern. Dies muss im *history*-Array geschehen, da dieses später vom Master ausgewertet wird.

***chunks* – Liste abhängiger Informationen** (Abbildung 9, Z. 46)

Jede Nachricht im *history*-Array hat selbst ein Array, über welches dessen Abhängigkeiten angegeben werden können. Hierüber können andere Nachrichten im *history*-Array über deren *id* referenziert werden. Hierdurch kann eine beliebige Tiefe an Abhängigkeitshierarchien erzeugt werden. Die Informationen dieser referenzierten Nachrichten (*id*, *source*, *destination*, *chunks*, [*payload*, *timestamp*]) wurden ja ebenfalls in das *history*-Array aufgenommen. Es enthält also zuerst nur eine Kopie der eigentlich versendeten Nachricht, zzgl. *chunks*-Array. Da alle darin referenzierten Nachrichten auch ins *history*-Array rekursiv aufgenommen werden, wird hierin die gesamte Abhängigkeitsstruktur abgebildet. Die Entstehung einer solchen Struktur ist in Abschnitt 3.1.4 beschrieben.

Vorhergehend wurde die Struktur der ausgetauschten Nachrichten dargestellt. Der Umgang mit dieser Datenstruktur unterscheidet sich jedoch je nach Rolle der einzelnen Knoten im Netzwerk. Im Folgenden ist der Ablauf der Verarbeitung eingehender und ausgehender Nachrichten verdeutlicht.

Sensor

Ein Sensor selbst erhält keine Daten von anderen Knoten und muss dementsprechend auch keine eingehenden Nachrichten verarbeiten. Seine einzige Aufgabe ist es, Umweltdaten zu erfassen und digital zu emittieren. Dies geschieht über ausgehende Nachrichten, die sich an das beschriebene Datenformat halten müssen. Die Struktur dieser ausgehenden Nachrichten ist sehr überschaubar, da hierbei noch keine Abhängigkeitsinformationen erzeugt werden. Ein erfasster Sensorwert ist immer unabhängig von anderen ausgetauschten Datenfragmenten im Netzwerk. Auch wenn Aktoren, die durch Daten im Netzwerk gesteuert werden, über die reale Welt Sensoren beeinflussen können, besteht keine digitale Abhängigkeit zu anderen ausgetauschten Informationen. Abhängigkeiten, die durch Zusammenhänge außerhalb des Netzwerkes entstehen (Zusammenhänge in der physischen Welt) werden hier nicht betrachtet.

Erzeugt ein Sensor also eine Information, wird diese als Nachricht an einen oder mehrere Aggregatoren emittiert. Hierbei werden die Datenfelder der Nachricht (vgl. Abbildung 9) wie folgt belegt:

- Die ***id*** soll diese Nachricht eindeutig identifizieren. Der Hash soll eindeutig sein. Es kann beispielsweise ein Hash aus dem Sensorwert, den Metadaten und der Uhrzeit als Identifikator verwendet werden.
- Der ***payload*** kann beliebig gefüllt werden. Im Falle eines Sensors wäre der Sensorwert im *payload* enthalten.
- Der Sensorknoten hat selbst einen eindeutigen Identifikator. Mit diesem wird das Feld ***source*** belegt.
- Je nach Ausprägung des Netzwerkes kann auch der Empfänger-Aggregator der Nachricht über das ***destination***-Feld mit seinem Identifikator benannt werden.

- Das ***lastspy***-Feld wird mit der eigenen *id* befüllt, falls der Knoten selbst ein Spy ist. Anderenfalls bleibt das Feld leer.

- Das ***history***-Array enthält eine Kopie eines Teils der oben beschriebenen Felder, gekapselt in ein Objekt. Das Array enthält also ein Objekt, dass die Felder *id*, *source* und *destination* enthält. Da Sensorwerte in der Regel klein sind, kann auch der *payload* aufgenommen werden. Da ein Sensorwert noch keine Abhängigkeiten hat, entfällt der *chunks*-Array im Objekt bei Sensoren.

Aggregator

Anders als beim Sensor gibt es beim Aggregator drei unterschiedliche Fälle. Im ersten Fall emittiert der Aggregator Daten, die ebenfalls keine Abhängigkeiten haben. Dies ist beispielsweise dann der Fall, wenn der Aggregator Daten aus einer Datenbank emittiert, die nicht mit Hilfe anderer Knoten im Netzwerk gefüllt wurde. Dann haben die Daten in der Datenbank auch keine Abhängigkeit zu anderen im Netzwerk ausgetauschten Informationen. In diesem Fall geht der Aggregator bei der Emittierung der Daten so vor, wie der Sensor bei Sensordaten.

Im zweiten Fall verwendet der Aggregator zuvor von anderen Knoten im Netzwerk (Sensoren/Aggregatoren) erhaltene Informationen, um selbst eine neue Information zu erzeugen und zu emittieren. Hierbei entsteht eine Abhängigkeit der neuen Nachricht von allen verwendeten Informationen der zuvor empfangenen Nachrichten. Dabei werden die Felder wie folgt belegt (vgl. Abbildung 9):

- Der ***payload*** enthält die zu emittierende Information. Diese besteht aus aggregierten Daten.

- Die ***id*** ist wieder ein Hash. Er kann beispielsweise aus dem *payload*, den Metadaten und der Uhrzeit erzeugt werden.

- Das ***source***-Feld wird mit dem Identifikator des emittierenden Aggregators belegt.

- Das ***destination***-Feld wird analog zum Vorgehen beim Sensor belegt.

- Das ***lastspy***-Feld wird mit der eigenen *id* befüllt, falls der Knoten selbst ein Spy ist. Anderenfalls wird es mit einer beliebigen *id* eines *lastspy*-Feldes aus einem der aggregierten Nachrichten gefüllt. Sind diese ebenfalls alle leer, bleibt *lastspy* auch hier leer.

- Das ***history***-Array ist die Vereinigung[20] aller *history*-Arrays aller Nachrichten, die verwendet wurden, um den neuen *payload* zu erzeugt. Zusätzlich wird ein weiteres Objekt in das *history*-Array eingefügt, dass als Kopie die *id*, *source* und *destination* der aktuellen Nachricht – also der oben genannten Elemente enthält. Ist der erzeugte *payload* klein, kann auch dieser hinzugefügt werden. Dieses Objekt repräsentiert im *history*-Array die aktuelle Nachricht, die Abhängigkeiten zu allen anderen Elementen im Array hat. Daher wird der *chunks*-Array dieses Objekts mit einer Auflistung aller *id*s der zur Erzeugung des *payload*s genutzten Nachrichten – also der anderen Objekte im *history*-Array – gefüllt.

Im dritten Fall ist der Aggregator eine Datensenke und verhält sich wie ein Aktor.

Aktor

Anders als Sensoren und Aggregatoren emittiert ein Aktor keine Nachrichten, sondern ist eine Datensenke. Es kann jedoch vorkommen, dass ein Aktor eine Nachricht erhält, die ein nicht leeres *history*-Array enthält. Würde die Nachricht einfach konsumiert werden, gingen Abhängigkeitsinformationen verloren. Um dies zu verhindern, gibt es im IFM-Protokoll den BackPush-Mechanismus, der in Abschnitt 3.1.3.3 beschrieben wird.

Für alle drei Rollen (Sensor/Aggregator/Aktor) gilt, dass weitere Aktionen durchgeführt werden, sofern diese Knoten Spy-Knoten sind. Diese Aktionen werden in Abschnitt 3.1.3.2 beschrieben.

[20] Das *history*-Array kann hier als ungeordnete Menge verstanden werden. Die Vereinigung meint dann die mathematische Vereinigung mehrerer dieser Mengen.

3.1.3.2 Kommunikation zwischen Spy und Master

Die Spy-Knoten haben die Aufgabe Abhängigkeitsinformationen an den Master zu übermitteln. Die Abhängigkeitsstruktur einzelner Nachrichten wird durch deren *history* beschrieben. Die Spy-Knoten, unabhängig davon, ob sie Sensor, Aggregator oder Aktor sind, gehen dafür bei jeder Nachricht wie folgt vor:

Eingehende Nachrichten – Aggregatoren, Aktoren

Bei jeder eingehenden Nachricht erhält der Master eine Kopie des jeweiligen *history*-Arrays vom Spy. Damit sind dem Master alle Abhängigkeiten dieser Nachricht bekannt. Anschließend leert der Spy das *history*-Array der Nachricht zur weiteren Verarbeitung, da es nicht mehr benötigt wird und so die Verarbeitungs- und Kommunikationslast reduziert wird.

Ausgehende Nachrichten – Sensoren, Aggregatoren

Alle *history*-Arrays der Nachrichten, von denen die ausgehende Nachricht abhängig ist, wurden bereits an den Master geschickt. Daher müssen diese nicht mehr Teil des *history*-Arrays der ausgehenden Nachricht sein. Im *history*-Array wird jedoch die Kopie der Metadaten der ausgehenden Nachricht hinterlegt. Auch dieser neu erzeugte Bestandteil der History wird zeitgleich mit der Emittierung der neuen Nachricht an den Master gesendet.

Die Spy/Master Kommunikation entspricht also nicht dem gezeigten JSON-Schema (vgl. Abbildung 9). Letztendlich wird nur der Inhalt des *history*-Arrays verwendet, dessen Schema in Abbildung 10 dargestellt ist. Die Funktionsweise entspricht dabei der in Abschnitt 3.1.3.1 beschriebenen.

3.1.3.3 IFM-BackPush

Ein Informationspfad erstreckt sich üblicherweise über mehrere Knoten hinweg. Liegt auf diesem Pfad ein Spy, werden wie beschrieben Abhängigkeitsinformationen in Form der History an den Master übertragen und dort weiterverarbeitet. Bei diesem Schritt werden alle erfassten Abhängigkeiten zusammengetragen, die im Pfad vor dem Spy lagen. Liegen auf einem Pfad mehrere Spy-Knoten, wird die

```
{
  "type": "array",
  "title": "history information",
  "items": {
    "$id": "#/properties/history/items", "type": "object", "title": "dependent packet list",
    "default": null, "required": ["id", "source", "chunks"],
    "properties": {
      "id": {
        "$id": "#/properties/history/items/properties/id", "type": "string",
        "examples": ["3-4-82f2318986..."], "pattern": "^(.*)$"
      },
      "source": {
        "$id": "#/properties/history/items/properties/source",
        "type": "integer", "examples": [3]
      },
      "destination": {
        "$id": "#/properties/history/items/properties/destination",
        "type": "integer", "examples": [4]
      },
      "chunks": {
        "$id": "#/properties/history/items/properties/chunks", "type": "array",
        "title": "second level dependencies", "default": [],
        "items": {
          "$id": "#/properties/history/items/properties/chunks/items", "type": "string",
          "title": "ID of a dependent packet", "examples": ["2-3-fc6cf64943..."],
          "pattern": "^(.*)$"
        }
      }
    }
  }
}
```

Abbildung 10: JSON-Schema zur Kommunikation zwischen Spies und dem Master

History entsprechend etappenweise von jedem Spy an den Master gesendet, da sie sich auf den Pfaden zwischen zwei Spy-Knoten wieder füllt. Dieses Verhalten führt dazu, dass die auf der letzten Etappe eines Pfades gesammelten Abhängigkeiten nicht mehr den Master erreichen, falls der letzte oder vorletzte Knoten im Pfad kein Spy ist. Diese Problematik wird mit Hilfe des IFM-BackPush-Ansatzes gelöst. Hierbei werden dem Verhalten der Spy- und Worker-Knoten zusätzliche Aktionen hinzugefügt.

Die Kernidee besteht darin, dass ein Knoten erkennt, dass er für eine bestimmte Nachricht, die er erhalten hat, eine Datensenke darstellt. Dies bedeutet, dass die Nachricht nicht weiterverwendet wird, um ausgehende Nachrichten zu erzeugen. Dementsprechend hat die dieser Information beiliegende History keine Möglichkeit mehr, über einen Spy den Master zu erreichen. Hat ein Knoten dies erkannt, wird

```
{
  "type": "array",
  "title": "history information",
  "items": {
    "$id": "#/properties/history/items", "type": "object", "title": "dependent packet
        list",
    "default": null,
    "properties": {
      "lastspy": {
        "$id": "#/properties/history/items/properties/lastspy",
        "type": "integer", "examples": [2]
      },
      "id": {
        "$id": "#/properties/history/items/properties/id", "type": "string",
        "examples": ["3-4-82f2318986..."], "pattern": "^(.*)$"
      },
      "source": {
        "$id": "#/properties/history/items/properties/source",
        "type": "integer", "examples": [3]
      },
      "destination": {
        "$id": "#/properties/history/items/properties/destination",
        "type": "integer", "examples": [4]
      },
      "chunks": {
        "$id": "#/properties/history/items/properties/chunks", "type": "array",
        "title": "second level dependencies", "default": [],
        "items": {
          "$id": "#/properties/history/items/properties/chunks/items", "type": "string",
          "title": "ID of a dependent packet", "examples": ["2-3-fc6cf64943..."],
          "pattern": "^(.*)$"
        }
      }
    }
  }
}
```

Abbildung 11: JSON-Schema zur Kommunikation während des BackPush-Vorgangs

versucht einen erreichbaren Spy zu ermitteln, dem die History zur Weitergabe an den Master übergeben wird.

Um dies zu ermöglichen, muss ein Spy ermittelbar sein. Immer wenn ein Spy auf einem Pfad liegt, leert dieser, wenn ihn eine Nachricht erreicht, die History und übermittelt diese an den Master. Bei aktiviertem BackPush hinterlässt der Spy allerdings bei ausgehenden Nachrichten seinen eigenen Identifikator im Feld *lastspy*. Dieser Identifikator bleibt so lange im Objekt enthalten, bis der nächste Spy-Knoten es überschreibt. Erreicht die Nachricht allerdings keinen weiteren Spy-Knoten, sondern eine Datensenke, setzt der Datensenken-Knoten den BackPush-Mechanismus in Kraft. Dazu erstellt der Knoten eine neue Nachricht, die fast nur noch aus dem History-Array der sonst verloren gegangenen Nachricht

besteht. Zusätzlich wird ein Objekt in das Array eingefügt, dass als Ziel den *lastspy* aus der sonst verloren gegangenen Nachricht enthält. Diese BackPush-Nachricht folgt dem JSON-Schema in Abbildung 11. Der *lastspy* ist das Ziel des BackPush.

Über den *lastspy*-Identifikator ist ermittelbar, wo der letzte Spy des begangenen Pfades zu finden ist. Über die bis dahin gesammelten History-Informationen lässt sich ein Pfad zu diesem Spy-Knoten rekonstruieren. Dass die Spy-Knoten hierbei über ihre ID adressierbar sind, ist sicherzustellen (vgl. Abschnitt 3.1.3.1).

Die Datensenke ist so in der Lage, sowohl einen Spy-Knoten als auch einen Pfad zu diesem zu ermitteln. Daher wird die BackPush-Nachricht, die die sonst verloren gegangene History enthält, auf diesem Pfad bis zum nächsten Spy zurückgesendet. Alle beteiligten Knoten verwenden das in Abbildung 11 dargestellte JSON-Schema zur Kommunikation. Es ist dem Schema, das zur Kommunikation zwischen Spy und Master eingesetzt wird, sehr ähnlich, enthält aber zusätzlich das *lastspy*-Element als Ziel des BackPushs. Dieser *lastspy* ist Ziel des BackPush und ist in der Nachricht enthalten, sodass jeder Knoten auf dem BackPush-Pfad identifizieren kann, wohin die Nachricht gesendet werden soll. Der Inhalt der Nachricht, die auf dem Pfad zurück gepusht wird, wird von keinem der beteiligten Knoten verändert. Erreicht die Nachricht den benannten Spy-Knoten, übermittelt dieser die History an den Master. Dazu wird der *lastspy* entfernt und das für die Spy-Master-Kommunikation vorgesehene JSON-Schema (vgl. Abbildung 10) genutzt.

Durch dieses Verfahren ist sichergestellt, dass alle Abhängigkeiten auf allen Pfaden erfasst werden, auf denen sich mindestens ein Spy-Knoten befindet. Die Position des Spies auf dem Pfad spielt dabei durch das BackPush-Verfahren für die reine Funktion des IFM keine Rolle mehr. Je mehr Spy-Knoten sich jedoch auf dem Pfad befinden, desto effizienter kann das Protokoll arbeiten, da das *history*-Array über kürzere Wege transportiert werden muss. Ein Beispiel zu IFM-BackPush wird in Abschnitt 3.1.4 gegeben.

3.1.4 Protokollbeispiele

Im Folgenden werden zwei Ablaufbeispiele anhand eines kleinen exemplarischen Netzwerkes (vgl. Abbildung 12) gegeben, die die Funktionsweise des IFM-

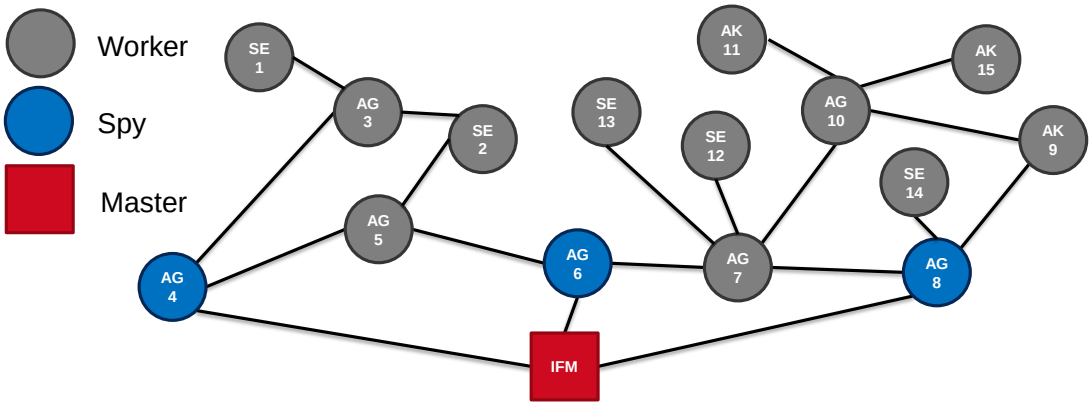


Abbildung 12: Exemplarisches Netzwerk für die Protokollbeispiele

Protokolls mit und ohne BackPush-Mechanismus beschreiben. Die Beispiele dienen dem besseren Verständnis des Protokolls. Alle darin enthaltenen Knoten sind IFM-konform und halten das beschriebene Protokoll vollständig ein. Die Sensoren in diesem Beispiel erzeugen unabhängig von einer realen Funktion in einem CPS jeweils Werte, der keinen echtweltlichen Bezug haben. Die Aggregatoren führen nur einfache mathematische Operationen als Aggregation aus, um deren Funktion nachvollziehbar zu machen.

Neben den folgenden konzeptionellen Beispielen ist ein in einen echtweltlichen Kontext eingebetteter Anwendungsfall in Abschnitt 4.1 dargestellt.

3.1.4.1 Protokollbeispiel I – ohne BackPush

Im ersten Protokollbeispiel soll der in Abbildung 13 dargestellte Informationsfluss bis hin zum Aktor AK9 mit Hilfe des IFM-Protokolls nachvollzogen werden. Die Nachrichten laufen hierbei baumartig auf den Zielknoten zu. Dabei werden Nachrichten verschiedener Knoten von Aggregatoren zusammengefasst. Die Abhängigkeiten, die hierbei entstehen, entstehen jedoch nicht direkt zwischen den einzelnen Knoten, sondern zwischen den Inhalten, also den Nachrichten, die die Knoten emittieren.

In Abbildung 13 ist also ausschließlich der Datenfluss dargestellt, nicht aber die Abhängigkeiten zwischen den einzelnen Knoten. Zwischen zwei Knoten können so beispielsweise auch mehrere Nachrichten ausgetauscht werden, die hier über nur eine Kante in der Abbildung visualisiert werden.

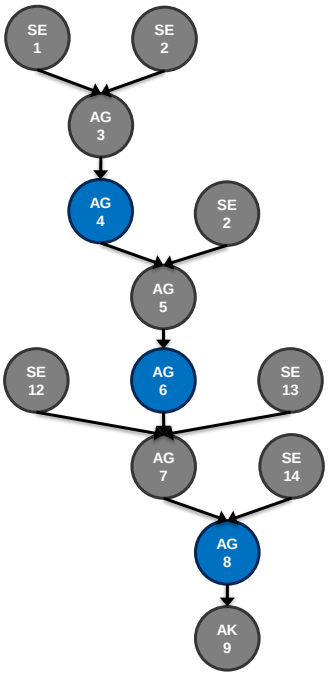


Abbildung 13: Datenfluss zwischen den Knoten im ersten Protokollbeispiel; Spy-Knoten in blau

Alle im Folgenden dargestellten Nachrichten, die im Netzwerk zwischen Knoten ausgetauscht werden, wurden gemäß dem IFM-Protokoll (vgl. Abschnitt 3.1.3) erstellt. Der Informationsfluss beginnt an den Knoten SE1 und SE2, die jeweils als Sensoren Daten der Umwelt erfassen. In diesem Protokollbeispiel geben sie diese Daten an den Aggregator AG3 weiter (vgl. Abbildung 13), der sie sammelt und weiterverarbeitet. Dazu sendet SE1 eine Nachricht mit erfasstem Sensorwert an AG3:

```
{
  "id": "fef728ac4ddf631553aa782acea453fe95b86517",
  "payload": "3.14159265359",
  "source": "SE1",
  "destination": "AG3",
  "lastspy": "",
  "history": [
    {
      "id": "fef728ac4ddf631553aa782acea453fe95b86517",
      "source": "SE1",
      "destination": "AG3",
      "chunks": []
    }
  ]
}
```

Diese Nachricht teilt AG3 den Sensorwert mit, sowie die Information, dass dieser keine Abhängigkeiten hat. Die Nachricht hat eine eindeutige *id* (Z. 2). Dass die Nachricht keine Abhängigkeiten hat, wird durch das leere *chunks*-Array (Z. 12) im zugehörigen Objekt mit der gleichen *id* (vgl. Z. 2, 9) im *history*-Array deutlich gemacht. In diesem Beispiel sendet SE1 nur die eine Nachricht. SE2 sendet eine sehr ähnliche an AG3:

```
1  {
2    "id": "57150854fe71d0293a494f31259170d2424a606b",
3    "payload": "2.71828182845",
4    "source": "SE2",
5    "destination": "AG3",
6    "lastspy": "",
7    "history": [
8      {
9        "id": "57150854fe71d0293a494f31259170d2424a606b",
10       "source": "SE2",
11       "destination": "AG3",
12       "chunks": []
13     }
14   ]
15 }
```

AG3 hat durch die beiden Nachrichten von SE1 und von SE2 je einen Sensorwert erhalten. Als Aggregator verarbeitet er diese Nachrichten. In unserem Beispiel wird von AG3 der Durchschnitt aller erhaltenen Sensorwerte berechnet. Das Ergebnis wird von AG3 an AG4 gesendet:

```
1  {
2    "id": "f273b5af8b98179d56e2e37bc13b5d3c0548ab70",
3    "payload": "2.92993724102",
4    "source": "AG3",
5    "destination": "AG4",
6    "lastspy": "",
7    "history": [
8      {
9        "id": "f273b5af8b98179d56e2e37bc13b5d3c0548ab70",
10       "source": "AG3",
11       "destination": "AG4",
12       "chunks": [
13         "fef728ac4ddf631553aa782acea453fe95b86517",
14         "57150854fe71d0293a494f31259170d2424a606b"
15       ]
16     },
17     {
18       "id": "fef728ac4ddf631553aa782acea453fe95b86517",
```

```
19      "source": "SE1",
20      "destination": "AG3",
21      "chunks": []
22    },
23    {
24      "id": "57150854fe71d0293a494f31259170d2424a606b",
25      "source": "SE2",
26      "destination": "AG3",
27      "chunks": []
28    }
29  ]
30 }
```

Die *history*-Arrays der aggregierten Nachrichten wurden hierbei zusammengeführt. Zeile 17-28 entspricht hierbei der Vereinigung der *history*-Arrays der erhaltenen Nachrichten. Zusätzlich wird das Array um ein eigenes Objekt ergänzt (Z. 8-16), dass über das *chunks*-Array (Z. 12-15) die Abhängigkeit der erstellen Nachricht (des Durchschnitts) zu den anderen Nachrichten (Z. 17-28) beschreibt.

AG4 erhält nun diese Nachricht. Als Aggregator nimmt auch dieser wieder Änderungen vor – in unserem Fall wird der Wert schlicht halbiert und an AG5 gesendet. Diese Halbierung entspricht letztendlich der Aggregation der eingehenden Daten anhand von Wissen oder bereits vorgehaltenen Daten. Da AG4 ein Spy ist, erzeugt dieser vor dem Senden der Daten an AG5 im ersten Schritt das neue *history*-Array separat und sendet dieses an den IFM-Master:

```
1  [
2   {
3     "id": "91d4c552f63b52a5d6f5124080a2018d6ce62b03",
4     "source": "AG4",
5     "destination": "AG5",
6     "chunks": [
7       "f273b5af8b98179d56e2e37bc13b5d3c0548ab70"
8     ]
9   },
10  {
11    "id": "f273b5af8b98179d56e2e37bc13b5d3c0548ab70",
12    "source": "AG3",
13    "destination": "AG4",
14    "chunks": [
15      "fef728ac4ddf631553aa782acea453fe95b86517",
16      "57150854fe71d0293a494f31259170d2424a606b"
17    ]
18  },
19  {
20    "id": "fef728ac4ddf631553aa782acea453fe95b86517",
```

```
21     "source": "SE1",
22     "destination": "AG3",
23     "chunks": []
24   },
25   {
26     "id": "57150854fe71d0293a494f31259170d2424a606b",
27     "source": "SE2",
28     "destination": "AG3",
29     "chunks": []
30   }
31 ]
```

Auch diese Nachricht ist die Vereinigung der erhaltenen Nachrichten zuzüglich eines neuen Objektes (Z. 2-9), welches die Abhängigkeiten der neu erstellten Nachricht zu bisherigen ausdrückt.

Da der IFM-Master nun bereits die bisherige History kennt, muss diese nicht mehr an AG5 weitergesendet werden. Das Ergebnis der Aggregation wird so mit reduziertem *history*-Array von AG4 an AG5 gesendet. Das Array enthält nur noch die Informationen über die gerade erzeugte Nachricht:

```
1  {
2    "id": "91d4c552f63b52a5d6f5124080a2018d6ce62b03",
3    "payload": "1.46496862051",
4    "source": "AG4",
5    "destination": "AG5",
6    "lastspy": "AG4",
7    "history": [
8      {
9        "id": "91d4c552f63b52a5d6f5124080a2018d6ce62b03",
10       "source": "AG4",
11       "destination": "AG5",
12       "chunks": [
13         "f273b5af8b98179d56e2e37bc13b5d3c0548ab70"
14       ]
15     }
16   ]
17 }
```

Diese Daten werden von AG 5 empfangen. Knoten AG4 hat darin auch vermerkt, dass dieser ein Spy (Z. 6) ist. Zusätzlich erhält AG5 auch Sensorwerte von SE2:

```
1  {
2    "id": "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
3    "payload": "2.71828182845",
```

```
  "source": "SE2",
  "destination": "AG5",
  "lastspy": "",
  "history": [
    {
      "id": "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
      "source": "SE2",
      "destination": "AG5",
      "chunks": []
    }
  ]
}
```


Die von SE2 erzeugte Nachricht hat ebenfalls keine Abhängigkeiten, da es sich um Sensorwerte handelt. AG5 aggregiert die erhaltenen Werte von AG4 und SE2 beispielsweise als Summe und sendet das Ergebnis an AG6:


```
{
  "id": "a4bc5d301cc1d3475fd198191c4748863353a78b",
  "payload": "4.18325044896",
  "source": "AG5",
  "destination": "AG6",
  "lastspy": "AG4",
  "history": [
    {
      "id": "a4bc5d301cc1d3475fd198191c4748863353a78b",
      "source": "AG5",
      "destination": "AG6",
      "chunks": [
        "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
        "91d4c552f63b52a5d6f5124080a2018d6ce62b03"
      ]
    },
    {
      "id": "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
      "source": "SE2",
      "destination": "AG5",
      "chunks": []
    },
    {
      "id": "91d4c552f63b52a5d6f5124080a2018d6ce62b03",
      "source": "AG4",
      "destination": "AG5",
      "chunks": [
        "f273b5af8b98179d56e2e37bc13b5d3c0548ab70"
      ]
    }
  ]
}
```

Über das *chunks*-Array (Z. 12-15) wird auch hier wieder die Abhängigkeit zu den Nachrichten von SE2 und AG4 beschrieben. Die Abhängigkeiten der Nachricht von AG4 (Z. 28) sind nicht mehr enthalten, da diese bereits dem IFM-Master mitgeteilt wurden. AG6 erhält diese Nachricht, verarbeitet diese durch Quadrieren, und sendet das Ergebnis an AG7. Da AG6 Spy ist, beginnt dieser wieder mit der Erstellung des *history*-Array, welches an den IFM-Master gesendet wird:

```
1  [
2    {
3      "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
4      "source": "AG6",
5      "destination": "AG7",
6      "chunks": [
7        "a4bc5d301cc1d3475fd198191c4748863353a78b"
8      ]
9    },
10   {
11     "id": "a4bc5d301cc1d3475fd198191c4748863353a78b",
12     "source": "AG5",
13     "destination": "AG6",
14     "chunks": [
15       "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
16       "91d4c552f63b52a5d6f5124080a2018d6ce62b03"
17     ]
18   },
19   {
20     "id": "b418e2c19a5936b99fc870434ff6cdcabfe1f705",
21     "source": "SE2",
22     "destination": "AG5",
23     "chunks": []
24   },
25   {
26     "id": "91d4c552f63b52a5d6f5124080a2018d6ce62b03",
27     "source": "AG4",
28     "destination": "AG5",
29     "chunks": [
30       "f273b5af8b98179d56e2e37bc13b5d3c0548ab70"
31     ]
32   }
33 ]
```

AG7 erhält wiederum von AG6 eine gekürzte Version des *history*-Array, den neuen *payload*, sowie den aktualisierten *lastspy*:

```
1  {
2    "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
3    "payload": "17.49958431872404",
```

```
  "source": "AG6",
  "destination": "AG7",
  "lastspy": "AG6",
  "history": [
    {
      "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
      "source": "AG6",
      "destination": "AG7",
      "chunks": [
        "a4bc5d301cc1d3475fd198191c4748863353a78b"
      ]
    }
  ]
}
```


Zusätzlich erhält AG7 weitere Sensorwerte von SE12 und SE13, die jeweils keine eigenen Abhängigkeiten haben:


```
{
  "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
  "payload": "1.0149416064096536",
  "source": "SE12",
  "destination": "AG7",
  "lastspy": "",
  "history": [
    {
      "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "source": "SE12",
      "destination": "AG7",
      "chunks": []
    }
  ]
}
```



```
{
  "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
  "payload": "1.08366",
  "source": "SE13",
  "destination": "AG7",
  "lastspy": "",
  "history": [
    {
      "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "source": "SE13",
      "destination": "AG7",
      "chunks": []
    }]
}
```

AG7 aggregiert die Daten von SE12, SE13 und AG6 und bildet die Summe der erhaltenen Werte. Das Ergebnis wird an AG8 gesendet:


```
{
  "id": "a9b91324bc1514507e06932c1403609881d8d880",
  "payload": "19.59818592513369",
  "source": "AG7",
  "destination": "AG8",
  "lastspy": "AG6",
  "history": [
    {
      "id": "a9b91324bc1514507e06932c1403609881d8d880",
      "source": "AG7",
      "destination": "AG8",
      "chunks": [
        "0093d82c3e14fbcc28e694cf3559821481dd5092",
        "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
        "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2"
      ]
    },
    {
      "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "source": "SE13",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "source": "SE12",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
      "source": "AG6",
      "destination": "AG7",
      "chunks": [
        "a4bc5d301cc1d3475fd198191c4748863353a78b"]
    }
  ]
}
```


Zusätzlich zu den Daten von AG7 erhält der Spy AG8 auch Sensordaten von SE14:


```
{
  "id": "025e014b082b384f08de94f50b84ee2f1b8b88b0",
  "payload": "1.1865691104156254",
  "source": "SE14",
```

```
  "destination": "AG8",
  "lastspy": "",
  "history": [
    {
      "id": "025e014b082b384f08de94f50b84ee2f1b8b88b0",
      "source": "SE14",
      "destination": "AG8",
      "chunks": []
    }
  ]
}
```

AG8 aggregiert die Daten von AG7 und SE14 als Summe. Da AG8 Spy ist, sendet dieser das aggregierte *history*-Array wieder zuerst an den IFM-Master:

```
[
  {
    "id": "973ff62ab613eeb33cbee8a02b62cb53a49fdf54",
    "source": "AG8",
    "destination": "AK9",
    "chunks": [
      "025e014b082b384f08de94f50b84ee2f1b8b88b0",
      "a9b91324bc1514507e06932c1403609881d8d880"
    ]
  },
  {
    "id": "025e014b082b384f08de94f50b84ee2f1b8b88b0",
    "source": "SE14",
    "destination": "AG8",
    "chunks": []
  },
  {
    "id": "a9b91324bc1514507e06932c1403609881d8d880",
    "source": "AG7",
    "destination": "AG8",
    "chunks": [
      "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2"
    ]
  },
  {
    "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
    "source": "SE13",
    "destination": "AG7",
    "chunks": []
  },
  {
    "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
    "source": "SE12",
    "destination": "AG7",
```

```
    "chunks": []
  },
  {
    "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
    "source": "AG6",
    "destination": "AG7",
    "chunks": [
      "a4bc5d301cc1d3475fd198191c4748863353a78b"
    ]
  }
]
```

Die Daten mit bereinigtem *history*-Array werden von AG8 an AK9 gesendet:

```
{
  "id": "973ff62ab613eeb33cbee8a02b62cb53a49fdf54",
  "payload": "20.78475503554932",
  "source": "AG8",
  "destination": "AK9",
  "lastspy": "AG8",
  "history": [
    {
      "id": "973ff62ab613eeb33cbee8a02b62cb53a49fdf54",
      "source": "AG8",
      "destination": "AK9",
      "chunks": [
        "025e014b082b384f08de94f50b84ee2f1b8b88b0",
        "a9b91324bc1514507e06932c1403609881d8d880"
      ]
    }
  ]
}
```

Da AK9 als Aktor eine Datensenke ist, endet an dieser Stelle der Informationsfluss. Da unmittelbar vor AK9 noch ein Spy platziert ist (AG8), haben alle Abhängigkeitsinformationen den IFM-Master erreicht. Diese wurden in 3 Stücken von den IFM-Spies AG4, AG6 und AG8 übermittelt. In Abschnitt 3.1.6 wird die Verarbeitung und Visualisierung dieser gesammelten Daten beschrieben.

3.1.4.2 Protokollbeispiel II – mit BackPush

Wie in Abschnitt 3.1.3.3 beschrieben, kommt der IFM-BackPush Mechanismus immer genau dann zum Einsatz, wenn nicht alle Abhängigkeitsinformationen an den Master gesendet werden können. Dies ist genau dann der Fall, wenn am

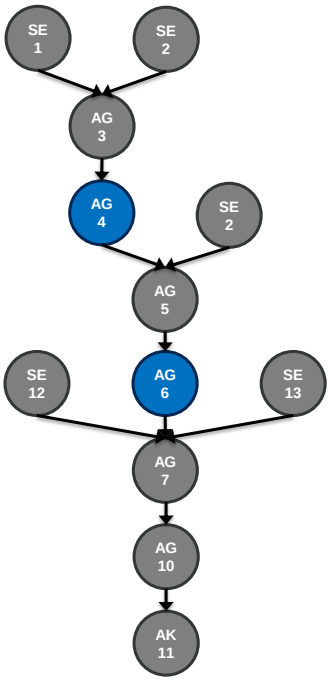


Abbildung 14: Datenfluss zwischen den Knoten im zweiten Protokollbeispiel; Spy-Knoten in blau

Ende des Informationspfades vor der Datensenke kein Spy platziert ist. Im ersten Protokollbeispiel (vgl. Abschnitt 3.1.4.1) ist unmittelbar vor dem letzten Aktor ein Spy, daher ist dort BackPush nicht notwendig. In diesem zweiten Beispiel wird ein Anwendungsfall skizziert, bei dem BackPush benötigt wird.

Der erste Teil des Informationsflusses ist zum ersten Beispiel identisch. Daher wird hier nur den hinteren Teil ab AG7 betrachtet. Der nun betrachtete Datenfluss ist in Abbildung 14 dargestellt.

Im ersten Beispiel übermittelt AG7 die Daten an AG8, in diesem Beispiel jedoch an AG10 (vgl. Abbildung 12):


```
{
  "id": "a9b91324bc1514507e06932c1403609881d8d880",
  "payload": "19.59818592513369",
  "source": "AG7",
  "destination": "AG10",
  "lastspy": "AG6",
  "history": [
    {
      "id": "a9b91324bc1514507e06932c1403609881d8d880",
      "source": "AG7",
```

```
      "destination": "AG10",
      "chunks": [
        "0093d82c3e14fbcc28e694cf3559821481dd5092",
        "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
        "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2"
      ]
    },
    {
      "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "source": "SE13",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "source": "SE12",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
      "source": "AG6",
      "destination": "AG7",
      "chunks": [
        "a4bc5d301cc1d3475fd198191c4748863353a78b"
      ]
    }
  ]
}
```


AG 10 ist ein Aggregator und halbiert den übermittelten Wert. Das Ergebnis wird an AK 11 geschickt:


```
{
  "id": "ad801198a9fab4e4ef79eb97624a4bf9c78b450a",
  "payload": "9.799092962566845",
  "source": "AG10",
  "destination": "AG11",
  "lastspy": "AG6",
  "history": [
    {
      "id": "ad801198a9fab4e4ef79eb97624a4bf9c78b450a",
      "source": "AG10",
      "destination": "AG11",
      "chunks": [
        "a9b91324bc1514507e06932c1403609881d8d880"
      ]
    },
    {
      "id": "a9b91324bc1514507e06932c1403609881d8d880",
      "source": "AG7",
```

```
      "destination": "AG10",
      "chunks": [
        "0093d82c3e14fbcc28e694cf3559821481dd5092",
        "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
        "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2"
      ]
    },
    {
      "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "source": "SE13",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "source": "SE12",
      "destination": "AG7",
      "chunks": []
    },
    {
      "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
      "source": "AG6",
      "destination": "AG7",
      "chunks": [
        "a4bc5d301cc1d3475fd198191c4748863353a78b"
      ]
    }
  ]
}
```


Bei AK11 endet der Informationsfluss, da dieser Knoten ein Aktor und somit eine Datensenke ist. AK11 ist in der Lage diesen Umstand zu erkennen, da der Knoten selbst keine Daten emittiert. Zusätzlich ist ersichtlich, dass nicht alle Abhängigkeitsinformationen an den IFM-Master gesendet wurden, da das *history*-Array mehr als das Element enthält, welches die aktuelle Nachricht repräsentiert. Aus diesem Grund setzt AK11 den IFM-BackPush-Mechanismus in Kraft.

Der erste Schritt ist das typische Verhalten eines Spies, obwohl AK11 selbst kein Spy ist. Der Knoten erstellt eine Nachricht, die fast nur noch aus den History-Informationen besteht. Die resultierende Nachricht ist die Vereinigung aus dem erhaltenen *history*-Array und ein Objekt, das den *lastspy* benennt:


```
[
  {
    "lastspy": "AG6"
  },
  {
```

```
    "id": "ad801198a9fab4e4ef79eb97624a4bf9c78b450a",
    "source": "AG10",
    "destination": "AG11",
    "chunks": [
      "a9b91324bc1514507e06932c1403609881d8d880"
    ]
  },
  {
    "id": "a9b91324bc1514507e06932c1403609881d8d880",
    "source": "AG7",
    "destination": "AG10",
    "chunks": [
      "0093d82c3e14fbcc28e694cf3559821481dd5092",
      "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
      "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2"
    ]
  },
  {
    "id": "0093d82c3e14fbcc28e694cf3559821481dd5092",
    "source": "SE13",
    "destination": "AG7",
    "chunks": []
  },
  {
    "id": "eb6bb7e7956a95264f00d304e2c396a4da3704dc",
    "source": "SE12",
    "destination": "AG7",
    "chunks": []
  },
  {
    "id": "553ae7da92f5505a92bbb8c9d47be76ab9f65bc2",
    "source": "AG6",
    "destination": "AG7",
    "chunks": [
      "a4bc5d301cc1d3475fd198191c4748863353a78b"
    ]
  }
]
```


Diese Nachricht ist denen, die von Spies an den IFM-Master gesendet werden, strukturell sehr ähnlich. Da AK11 jedoch kein Spy ist, kann dieser die Nachricht nicht direkt zum Master senden. Dem Knoten ist jedoch ein Spy bekannt, der *lastspy*. Ebenfalls ist der Knoten durch die History in der Lage eine Route zum Spy (*lastspy*, AG6) zu berechnen. Dies geschieht wie folgt:

- Aus Nachricht **553ae7da92f5505a92bbb8c9d47be76ab9f65bc2** geht hervor, dass eine Route zwischen AG6 und AG7 existiert.

- Aus Nachricht **a9b91324bc1514507e06932c1403609881d8d880** geht hervor, dass eine Route zwischen AG7 und AG10 existiert.
- Aus Nachricht **ad801198a9fab4e4ef79eb97624a4bf9c78b450a** geht hervor, dass eine Route zwischen AG10 und AK11 existiert.

Hieraus resultiert der BackPush-Pfad AK11 → AG10 → AG7 → AG6, den jeder auf dem Pfad liegende Knoten aus der übermittelten History schließen kann. Eine Adressierbarkeit der Knoten kann durch eine sinnvolle Benennung der Knoten-IDs sichergestellt werden (vgl. Abschnitt 3.1.3.1).

AK11 beginnt den BackPush dementsprechend damit, die oben dargestellte Nachricht an AG10 zu senden. Anschließend sendet AG10 es weiter an AG7 und dieser an AG6. Als Spy sendet abschließend AG6 die um das *lastspy*-Objekt bereinigte Nachricht an den IFM-Master. Das gezeigte Objekt wird auf dem Transport über die Worker zum Spy zum Master nicht modifiziert.

3.1.5 IFM-Protokoll in Kombination mit anderen Protokollen

Da das IFM-Protokoll immer mit anderen Transport- und Applikations-Protokollen eingesetzt wird, muss es in der Lage sein, sich in diese zu integrieren. Im Folgenden wird die Integration des IFM-Protokolls in zwei typische Protokolle, die im CPS-Kontext eingesetzt werden, beschrieben. Hierbei handelt es sich zum einen um das MQTT-Protokoll sowie das Representational State Transfer (REST)-Paradigma in Kombination mit HTTP oder HTTPS. Diese beiden Protokolle sind in CPS weit verbreitet. REST spielt insbesondere bei Serviceschnittstellen im Internet eine große Rolle. Grundsätzlich lässt sich das IFM-Protokoll jedoch mit jedem Transportprotokoll einsetzen, das Strings beliebiger Längen, und somit auch JSON-Objekte, transportieren kann.

3.1.5.1 IFM+MQTT

MQTT ist ein Nachrichtenprotokoll, das insbesondere für die Machine-to-Machine (M2M)-Kommunikation im IoT-Kontext entworfen wurde [70]. Daher eignet es sich auch für den Einsatz in CPS, um die Kommunikation der einzelnen Knoten

im Netzwerk zu realisieren. Der Einsatz von MQTT ist jedoch nicht in allen CPS möglich, da die Kommunikation über einen oder mehrere zentrale Server, sogenannte Broker, realisiert wird. Es muss also keine einzelne zentrale Instanz existieren, es besteht aber die Anforderung, dass es mindestens Teilgruppen im CPS gibt, die jeweils an eine zentrale Kommunikationsinstanz angebunden sind. Dies ist nicht in allen CPS gegeben.

Anders als bei vielen Kommunikationsprotokollen kommunizieren bei MQTT die beteiligten Knoten nicht 1:1 miteinander, sondern 1:n. Daher benennt der Versender einer Nachricht keinen Empfänger, sondern versendet die Nachricht nur auf einem bestimmten Kanal. Empfänger können diesen Kanal abonnieren. Der zentrale MQTT-Server (genannt *Broker*) verteilt dann eine auf diesem Kanal publizierte Nachricht an alle Abonnenten.

Das IFM-Protokoll (vgl. Abschnitt 3.1.3) sieht vor, dass in jeder Nachricht ein Sender und ein Empfänger benannt wird. Dies ist bei MQTT nicht möglich, da die Nachricht mglw. keinen oder mehrere Empfänger hat. Daher entfällt das optionale *destination*-Feld bei der Nutzung von MQTT. Hierdurch wird die Funktionalität der Abhängigkeitsnachverfolgung nicht eingeschränkt[21]. Hat eine Nachricht A beispielsweise eine Abhängigkeit von Nachricht B, so ist das *destination*-Feld von einer Nachricht von B gleich dem *source*-Feld einer Nachricht von A. Durch die Abhängigkeitshierarchie lässt sich also der Empfänger einer Nachricht rekonstruieren, da dieser die Nachricht weiterverarbeitet.

So kann das IFM-Protokoll auch mit MQTT eingesetzt werden. Da MQTT standardmäßig Strings versendet, kann das in Abschnitt 3.1.3 beschriebene JSON direkt übermittelt werden. Der Anwendungsfall in Abschnitt 4.1 beschreibt die Nutzung von MQTT in einem CPS.

[21] Dennoch ist es sinnvoll, das *destination*-Feld in anderen Anwendungsfällen, soweit möglich, zu nutzen. Es trägt dazu bei, dass Informationen über den Empfänger früher beim Master vorliegen und verhindert einen BackPush, wenn der vorletzte Knoten im Pfad ein Spy ist.

3.1.5.2 IFM+REST

Bei REST handelt es sich nicht um ein Protokoll, sondern um ein Programmierparadigma. Eingesetzt wird REST insbesondere bei Webservices in IP-Umgebungen, die über HTTP/HTTPS kommunizieren. REST ist daher sowohl für CPS relevant, die über das Internet kommunizieren, als auch für CPS, in denen Komponenten über Knoten-zu-Knoten-Verbindungen Informationen austauschen. Webservices auf der Basis von REST stellen im Internet neben dem Simple Object Access Protocol (SOAP) den am weitesten verbreiteten Ansatz zur Übertragung von Datenobjekten dar [71].

Auch bei REST liegt der Zweck schwerpunktmäßig auf M2M-Kommunikation. Es werden Schnittstellen definiert, über die Information bereitgestellt und abgefragt werden können. Anders als bei MQTT ergreift hierbei nicht der Versender einer Nachricht die Initiative, indem die Nachricht verschickt wird. Stattdessen wartet der Versender auf die Anfrage eines Empfängers, ehe die Nachricht erstellt und zur Abholung bereitgestellt wird. Je nach Implementierung der Schnittstelle muss sich der Empfänger hierbei identifizieren und/oder authentifizieren. REST-Schnittstellen stellen Daten oft als JSON oder XML bereit. [72, 73]

Da auch das IFM-Protokoll auf JSON aufbaut, ist hier eine einfache Implementierungsmöglichkeit gegeben. Identifiziert sich der Empfänger einer Information gegenüber dem Versender, kann dieser das definierte IFM-Protokoll vollständig umsetzen, da REST eine 1:1-Kommunikation darstellt. Entfällt die Identifikation muss ähnlich wie bei der MQTT-Implementierung das *destination*-Feld entfallen, da der Versender den Empfänger möglicherweise nicht benennen kann.

3.1.6 IFM-Master Datenhaltung und Visualisierung

Wie bereits in Abschnitt 3.1.2.3 beschrieben, erfüllt der IFM-Master zwei zentrale Aufgaben. Zum einen werden die von den IFM-Spies erhaltenen Abhängigkeitsinformationen gespeichert, zum anderen müssen diese aber auch verarbeitet und zugänglich gemacht werden, um Abhängigkeiten nachvollziehbar zu machen. In diesem Kapitel werden die Konzepte hinter der Verarbeitung und Visualisierung der gesammelten Abhängigkeitsinformationen beschrieben. Dazu werden zur Veranschaulichung die gesammelten Daten aus Abschnitt 3.1.4.1 genutzt.

3.1.6.1 Datenhaltung

Im Laufe des in Abschnitt 3.1.4.1 beschriebenen Prozesses werden insgesamt von drei IFM-Spies (AG4, AG6 und AG8) Daten an den Master gesendet. Die vom Master empfangenen Abhängigkeitsinformationen müssen in einer gemeinsamen Datenbank gehalten werden, um über alle empfangenen Daten hinweg Visualisierungen erstellen zu können. Grundsätzlich eignen sich hierfür sowohl relationale (z. B. Structured Query Language (SQL)) als auch dokumentenorientierte Datenbanken (Not only Structured Query Language (NoSQL)). Um die Struktur der *chunks* in einer gemeinsamen Tabelle mit den Abhängigkeitsinformationen darstellen zu können, bieten sich dokumentenorientierte Datenbanken an. Zusätzlicher Vorteil ist, dass dokumentenorientierte Datenbanken wie MongoDB [74] in der Lage sind, JSON-Daten nativ ohne Transformation und durchsuchbar abzulegen.

Nach Abschluss des im Beispiel (vgl. Abschnitt 3.1.4.1) beschriebenen Prozesses wurden die in Tabelle 1 dargestellten Daten in der IFM-Master-Datenbank abgelegt[22]. Die hierbei zusammengetragenen Daten, die sich in diesem Beispiel lediglich auf einen einzigen Informationsfluss beschränken, stammen aus den von den Spies an den Master gesendeten History-Daten. Sie sind bei zunehmender Datenmenge nicht mehr manuell auswertbar.

Bei der Ablage in der Datenbank werden die erhalten Abhängigkeitsinformationen nicht vorverarbeitet, sondern nur auf Duplikate gefiltert. In bestimmten Szenarien

[22] Die Datenbank kann auch weitere Felder, bspw. *payload* und *timestamp*, enthalten (vgl. Abschnitt 3.1.3). In der Tabelle sind nur die für dieses Beispiel notwendigen Felder dargestellt.

ID	Source	Destination	Chunks
91d4c552f63b52a5d6f5124080a2018d6ce62b03	AG4	AG5	['f273b5af8b98179d56e2e37bc13b5d3c0548ab70']
f273b5af8b98179d56e2e37bc13b5d3c0548ab70	AG3	AG4	['fef728ac4ddf631553aa782acea453fe95b86517', '57150854fe71d0293a494f31259170d2424a606b']
fef728ac4ddf631553aa782acea453fe95b86517	SE1	AG3	[]
57150854fe71d0293a494f31259170d2424a606b	AE2	AG3	[]
553ae7da92f5505a92bbb8c9d47be76ab9f65bc2	AG6	AG7	['a4bc5d301cc1d3475fd198191c4748863353a78b']
a4bc5d301cc1d3475fd198191c4748863353a78b	AG5	AG6	['b418e2c19a5936b99fc870434ff6cdcabfe1f705', '91d4c552f63b52a5d6f5124080a2018d6ce62b03']
b418e2c19a5936b99fc870434ff6cdcabfe1f705	SE2	AG5	[]
973ff62ab613eeb33cbee8a02b62cb53a49fdf54	AG8	AK9	['025e014b082b384f08de94f50b84ee2f1b8b88b0', 'a9b91324bc1514507e06932c1403609881d8d880']
025e014b082b384f08de94f50b84ee2f1b8b88b0	SE14	AG8	[]
a9b91324bc1514507e06932c1403609881d8d880	AG7	AG8	['0093d82c3e14fbcc28e694cf3559821481dd5092', 'eb6bb7e7956a95264f00d304e2c396a4da3704dc', '553ae7da92f5505a92bbb8c9d47be76ab9f65bc2']
0093d82c3e14fbcc28e694cf3559821481dd5092	SE13	AG7	[]
eb6bb7e7956a95264f00d304e2c396a4da3704dc	SE12	AG7	[]

Tabelle 1: Im IFM-Master gesammelte Daten, resultierend aus Abschnitt 3.1.4.1

kann es vorkommen, dass dieselbe Abhängigkeitsinformation mehrfach an den Master gemeldet wird. Dies ist dann der Fall, wenn ein Aggregator eine Information an mehrere Knoten weiterleitet und entsprechend jeweils auch die History mitsendet. Erreichen dann beide Informationsflüsse einen Spy, melden beide die Abhängigkeiten an den Master. Die doppelten Anteile der History werden dann nur einmal abgelegt. Es ist einfach zu prüfen, ob eine Abhängigkeit schon erhalten wurde, da jede Nachricht mit einer ID versehen wurde, die dann bereits in der Datenbank enthalten ist.

3.1.6.2 Datenauswertung

Bei den in der Datenbank abgelegten Abhängigkeiten wird pro Eintrag die Abhängigkeit einer Nachricht zu anderen Nachrichten beschrieben. Die Visualisierung dient nun dazu, einen Abhängigkeitsbaum zu konstruieren und anzuzeigen. Hierzu werden rekursiv zur zu untersuchenden Nachricht[23] die Abhängigkeiten durchlaufen und auch deren Abhängigkeiten untersucht. Es entsteht ein Baum mit der zu untersuchenden Nachricht als Wurzel. Ein Blatt eines Knotens ist eine

[23] Eine zu untersuchende Nachricht enthält eine oder mehrere Informationen und wird insbesondere dann ausgewählt, wenn diese entweder als fehlerhaft identifiziert wurde oder wenn aus anderen Gründen ihre Abhängigkeiten aufgedeckt werden sollen.

Nachricht, zu der eine Abhängigkeit besteht. Ein Ast endet, wenn eine Nachricht keine eigenen Abhängigkeiten hat.

Im Beispiel aus Abschnitt 3.1.4.1, das in den in Tabelle 1 dargestellten Daten in der IFM-Master-Datenbank resultiert, kann ein Abhängigkeitsbaum zu der von Aktor 9 erhaltenen Information konstruiert werden. Hierzu gibt es unterschiedliche Motivationen, die eine Rückverfolgung veranlassen:

Allgemeine Rückverfolgung von Abhängigkeiten
Fragestellung: Welche Knoten senden Informationen an den einen, zu untersuchenden Knoten?

Sollen in einem CPS allgemein die Abhängigkeiten von Informationen, die einen bestimmten Knoten erreichen, ermittelt werden, ermöglicht eine Benutzerschnittstelle (IFM-Visualizer, vgl. Abschnitt 3.1.6.3) die Auswahl dieser Informationen aus der Datenbank. Sind beispielsweise die Informationen in Aktor 9 relevant, gibt die Benutzerschnittstelle alle Informationen mit dem Empfänger (*destination*-Feld im JSON) AK9 aus. In diesem Beispiel ist das nur die Information mit der ID 973ff62ab613eeb33cbee8a02b62cb53a49fdf54, welche daher als Wurzel des Abhängigkeitsbaum dient. In anderen Anwendungsfällen können hier jedoch auch mehrere Informationen in Frage kommen.

Spezifische Rückverfolgung einer bestimmten Information
Fragestellung: Von welchen Informationen ist eine zu untersuchende Information abhängig? Wie kam es zu dieser Information?

Wurde in einem CPS ein bestimmtes auffälliges Verhalten beobachtet, muss ermittelt werden, welche Daten genutzt wurden, um dieses Verhalten auszulösen. Hierzu wird bestimmt, welche Information im auffälligen Knoten genutzt wurde, um das Verhalten auszulösen. In diesem Beispiel wurde die Information in AK9 mit ID 973ff62ab613eeb33cbee8a0 2b62cb53a49fdf54 bestimmt. Sie dient als Wurzel des Abhängigkeitsbaumes.

Für beide Szenarien wurde dieselbe Information mit der ID 973ff62ab613eeb3 3cbee8a02b62cb53a49fdf54 als Wurzel des Abhängigkeitsbaumes bestimmt. Wie aus der Datenbank hervorgeht (vgl. Tabelle 1), hat diese Information Abhän-

gigkeiten zu den Informationen mit den IDs 973ff62ab613eeb33cbee8a02b62cb53 a49fdf54 und a9b91324bc1514507e06932c1403609881d8d880. Daher werden diese beiden Informationen als Blätter an die Wurzel angehängt. Durch einen rekursiven Durchlauf der Abhängigkeiten dieser Informationen entsteht der anzuzeigende Abhängigkeitsbaum in Abbildung 15.

Abbildung 15: Abhängigkeitsbaum für eine Information von AK9, basierend auf Tabelle 1

3.1.6.3 Visualisierung

Mit Hilfe der Visualisierung ist es nun möglich, Abhängigkeiten einzelner Informationen nachzuvollziehen, um dann nach Fehlern innerhalb der Informationen suchen zu können. Um die Suche nach der Quelle der Fehler zu beschleunigen, ist die Aufnahme des Payloads in die History-Daten sinnvoll.

Anders als die zuvor beschriebenen Komponenten verfügt der Master über eine Benutzerschnittstelle (UI), die die Auswertung der Daten ermöglicht. Das UI trägt den Namen IFM-Visualizer.

Das UI realisiert die beschriebene Darstellung der Abhängigkeiten als Baumstruktur. Hierzu kann die ID der zu untersuchenden Information eingegeben werden. Anschließend werden die zugehörigen Metadaten aus der Master-Datenbank geladen. Von dort aus werden rekursiv die zugehörigen *chunks* geladen und in der Baumstruktur angezeigt (vgl. Abbildung 16[24]). Informationen eines Knotens werden hierbei immer mit der gleichen Farbe visualisiert. Um die Unterscheidung

[24] Diese Abbildung illustriert nur die Funktionsweise des IFM-Visualizers und bedient sich keiner Daten aus vorherigen Beispielen.

zu vereinfachen, werden hierzu Kellys 22 Farben des maximalen Kontrasts [75] verwendet.

Durch Auswahl der Information im Abhängigkeitsbaum werden auf der linken Seite der UI die Metadaten angezeigt. Wurde das optionale *payload*-Feld im *history*-Array ebenfalls gefüllt, erscheint auch dieses hier. Bei großen Nutzdatenmengen ist dies aus Effizienzgründen aber nicht immer Teil der vorliegenden Abhängigkeitsdaten. Beim Untersuchen der Abhängigkeiten kann der *payload* jedoch nützlich sein. Ohne vorliegende Nutzdaten ist die Analyse, in welchem Teilbaum der Fehler geschieht, nur durch manuelle Auswertung der einzelnen Knoten möglich. Dann unterstützt der Visualizer insbesondere durch Benennung der beteiligten zu untersuchenden Knoten.

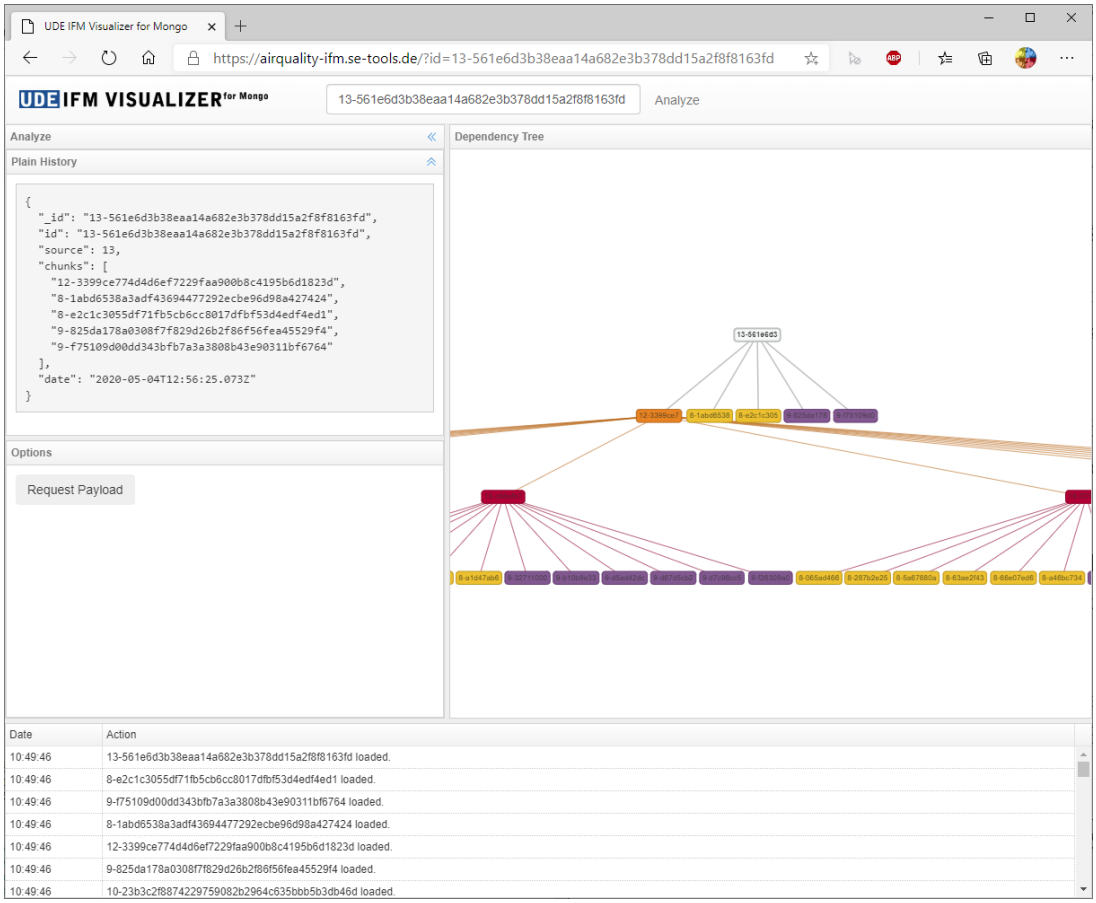


Abbildung 16: UI des IFM Visualizer

Liegt der *payload* jedoch nicht vor, besteht über das UI die Möglichkeit ihn von den einzelnen Knoten anzufordern und anschließend zu visualisieren. Dies kann natürlich nur geschehen, falls die Knoten die verschickten Informationen auch

selbst in einer Datenbank ablegen und die Knoten vom Visualizer aus erreichbar sind. Da dies in CPS oft nicht der Fall ist, ist die Platzierung der Nutzdaten in der History auf jeden Fall vorzuziehen. Daher ist die nachträgliche Anforderung der Nutzdaten durch den Visualizer auch kein Teil des IFM-Protokolls.

Die genaue Nutzung des Werkzeugs ist in Abschnitt 4.1.3 ausführlich anhand eines Fallbeispiels dargestellt.

3.1.7 Implementierung des IFM

Um den IFM in einem CPS nutzen zu können, müssen die in den vorherigen Kapiteln beschriebenen Konzepte vollständig umgesetzt werden. Dazu zählt insbesondere das IFM-Protokoll, die Bereitstellung eines IFM-Masters und die Integration der IFM-Worker- und IFM-Spy-Verhaltensweisen in die Sensoren, Aktoren und Aggregatoren. Die Best Practices bei der Umsetzung des IFM-Konzepts innerhalb eines CPS werden im Folgenden beschrieben.

Best Practices bei der Implementierung des IFM in ein CPS

Die Umsetzung der einzelnen benötigten Integrationsschritte ist je nach Komponente unterschiedlich aufwändig und benötigt verschiedene Schritte.

- Die Integration des **IFM-Masters** in das CPS erfordert keine Anpassung. Der Master kann schlicht als separater Knoten im Netzwerk aufgestellt werden. Bei der Planung ist darauf zu achten, dass der Master von möglichst vielen Knoten – insbesondere den späteren Spy-Knoten – gut erreichbar ist.

- Soweit wie möglich sollten alle Knoten im CPS zu **IFM-Workern** werden. Damit sie ihre Aufgabe erfüllen können, müssen die Worker den Überblick darüber behalten, welche eingehenden Nachrichten genutzt wurden, um ausgehende Nachrichten zu erzeugen. Dazu bietet es sich an, die Worker-Funktionalität in eine Komponente des jeweiligen CPS-Knoten zu kapseln, die sowohl diese Arbeit als auch die IFM-Protokoll-konforme Kommunikation erledigt. Zudem muss die Businesslogik des Knotens Aussagen darüber treffen, welche Abhängigkeiten zwischen ein- und ausgehenden Nachrichten existieren. Anderenfalls kann die genannte Worker-Komponente in der

IFM-Kommunikation keine Abhängigkeiten benennen. Ein Implementierungsvorschlag, wie die IFM-Komponente, im Folgenden IFM-Interface genannt, eines IFM-Worker-Knotens aussehen könnte, wird im Laufe des Kapitels beschrieben.

- Worker, die zusätzlich auch **IFM-Spies** sind, können die gleiche IFM-Interface-Komponente nutzen. Zusätzlich muss für die Spy-Knoten jedoch sichergestellt werden, dass diese direkt mit dem Master kommunizieren können.

IFM-Interface

In diesem Abschnitt werden Ideen zur Implementierung einer IFM-Interface-Komponente vorgestellt, die von Workern und Spies genutzt werden kann, um IFM-konform zu kommunizieren. Fokus ist die Kapselung der IFM-Funktionalitäten von der Businesslogik des Knotens. Eine solche Implementierung nutzt auch der in Abschnitt 4.1 beschriebene Anwendungsfall.

Das Interface hat für Aggregatoren insbesondere zwei Aufgaben:

- **Eingehende Nachrichten** werden verarbeitet, indem in einem ersten Schritt der Payload der Nachricht an die Businesslogik des Knotens übergeben wird. Diese hat kein Interesse an den anderen empfangenen Daten und benötigt diese auch nicht. Die Businesslogik kann so unabhängig von IFM-Funktionalität mit der Verarbeitung des Payloads der Nachricht beginnen. Die Interface-Komponente speichert in einer Datenbank die Zuordnung zwischen Nutzdaten (Payload) und empfangenen Abhängigkeitsinformationen (*history*).

- Um **ausgehende Nachrichten** zu verarbeiten, erhält das Interface von der Businesslogik sowohl die zu versendenden Nutzdaten, einen Identifikator des Empfängers, als auch eine Liste von Abhängigkeiten der erzeugten Nutzdaten von anderen, bereits empfangenen Nutzdaten. Diese Liste kann vom Interface genutzt werden, um die zugehörigen IDs der Nachrichten aus der eigenen Datenbank zu ermitteln und daraus ausgehende IFM-konforme Nachrichten zu generieren.

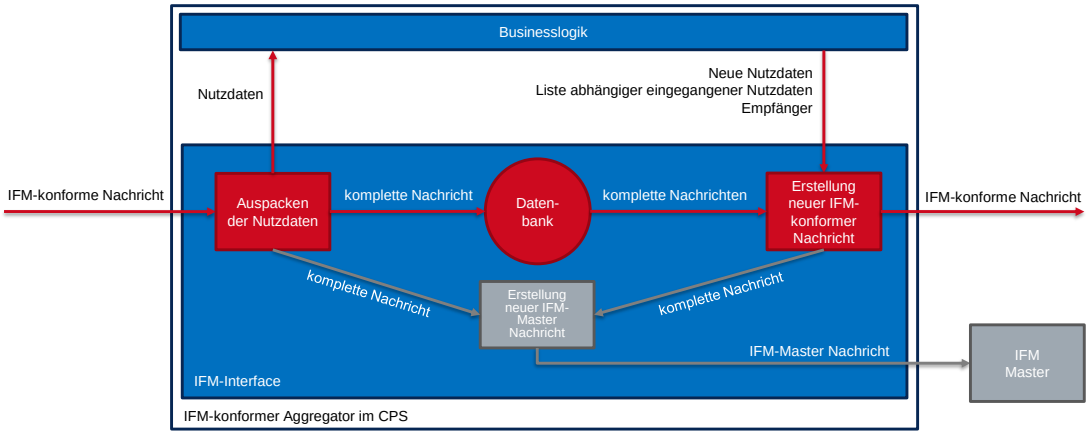


Abbildung 17: IFM-Interface für Aggregatoren

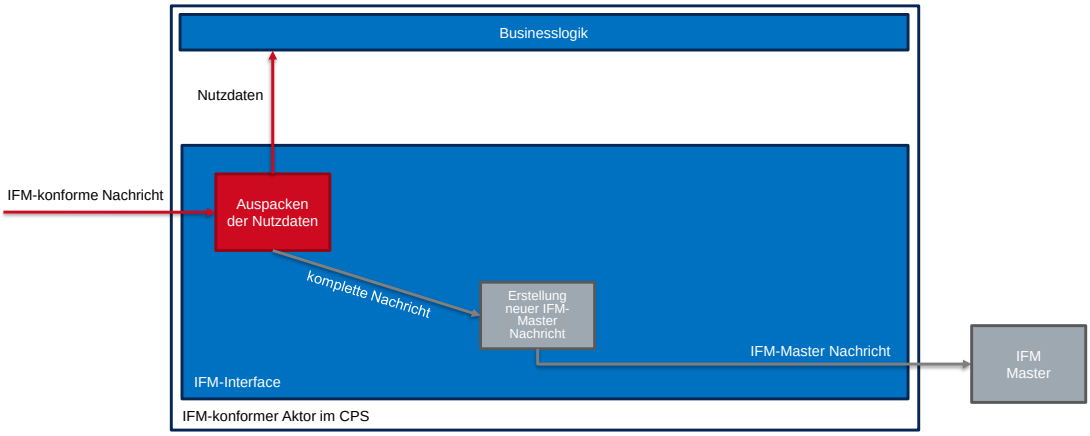


Abbildung 18: IFM-Interface für Aktoren

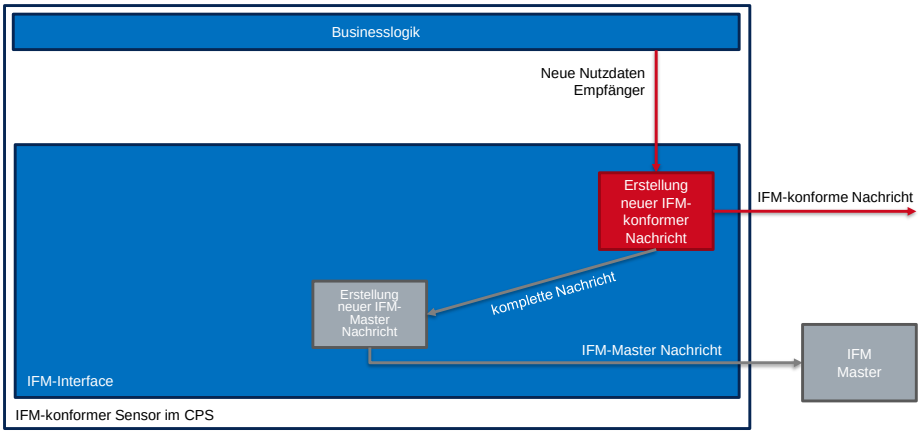


Abbildung 19: IFM-Interface für Sensoren

Ist der Knoten ein Spy, nimmt er des Weiteren die Aufgabe wahr, sowohl für eingehende als auch für ausgehende Nachrichten die History an den Master zu übergeben. Dazu wird im Interface die History vom Rest der Nachricht separiert und im zuvor beschriebenen Format an den Master geschickt.

Die beschriebene Architektur der Interface-Komponente ist in Abbildung 17 dargestellt. Der nur auf Spy-Knoten zutreffende Teil der Architektur ist in der Abbildung grau markiert. Links im Bild symbolisiert der Pfeil eine von einem anderen Knoten eintreffende Nachricht. Rechts im Bild werden an einen anderen Knoten verschickte Nachrichten ebenfalls über einen Pfeil visualisiert.

Für Sensoren und Aktoren fallen große Teile der Interface-Funktionalität weg. Für Aktoren (vgl. Abbildung 18) fallen die Datenbank und die Erstellung neuer IFM-konformer Nachrichten weg, während für Sensoren (vgl. Abbildung 19) das Auspacken der Nutzdaten und die Datenbank ersatzlos entfallen. Da von Sensoren generierte Nachrichten keine Abhängigkeiten haben, kann auch die Liste abhängiger eingegangener Nutzdaten entfallen.

In der dargestellten Architektur ist der BackPush-Mechanismus zur Steigerung der Übersichtlichkeit nicht enthalten, da dieser vollständig entkoppelt von der Businesslogik des jeweiligen Knotens ablaufen kann. Es empfiehlt sich, die IFM-Interface-Komponente als Bibliothek zu implementieren, sodass sie in den verschiedenen Knoten eingesetzt werden kann. Eine dem hier beschriebenen Konzept folgende Implementierung wurde auch in dem CPS in Abschnitt 4.1 eingesetzt.

Diskussion

Durch die vorgestellte Architektur einer möglichen Integration des IFM-Konzepts in die Sensoren, Aktoren und Aggregatoren wird es möglich, IFM-Komponenten in der Implementierung weitgehend von der Businesslogik zu trennen. Was allerdings Aufgabe der Businesslogik bleibt ist die Benennung von Abhängigkeiten. Dies ist jedoch nicht vermeidbar, da nur am Ort der Entstehung der Abhängigkeiten auch benannt werden kann, welche Informationen zur Erstellung neuer Informationen genutzt wurden. Da sich Netzwerkkommunikation, Datenverwaltung und -verarbeitung in das IFM-Interface auslagern lassen, ist der Aufwand durch das Hinzufügen der IFM-Funktionalitäten insbesondere davon abhängig, wie einfach es für die Businesslogik ist, Abhängigkeitsinformationen bekannt zu

geben. Dieser Aufwand steigt mit komplexer werdenden Datenverarbeitungen der Aggregator-Knoten an, während der Aufwand für Sensoren und Aktoren vernachlässigbar ist. Das führt dazu, dass das Nachrüsten des IFM in einem CPS mit hohem Aufwand verbunden ist, wenn es viele Aggregatoren enthält. In diesen Fällen ist es sinnvoll, das IFM-Konzept direkt bei der Planung des CPS zu integrieren. Keinesfalls lässt sich der IFM im Fehlerfall binnen kürzester Zeit in ein bestehendes Netz integrieren. Stattdessen sollte der IFM frühzeitig integriert werden. Soll der IFM das Netz nicht zu jeder Zeit überwachen, lässt sich bei einer erfolgten Integration auch das zügige Ein- und Ausschalten des Information Flow Monitorings realisieren, um den Mehraufwand des IFM-Einsatzes im Regelbetrieb zu vermeiden.

3.1.8 Praxisherausforderungen

In Bezug auf das IFM-Konzept existieren fünf zentrale Fragestellungen, deren Beantwortung zur Anwendbarkeit des Konzeptes und Protokolls beiträgt. Zwei dieser Fragestellungen wurden bereits zuvor diskutiert:

- **Wie kann ein Informationspfad vollständig überwacht werden, wenn ein Spy weit vorn im Pfad liegt?**
  Das Konzept des IFM-BackPush (vgl. Abschnitt 3.1.3.3) ist Teil des IFM-Protokolls und stellt sicher, dass ein einziger Spy auf einem Informationspfad ausreicht, um den gesamten Pfad zu überwachen.

- **Wie könnten Abhängigkeiten zur Wartung des CPS zugänglich gemacht werden?**
  Der IFM-Visualizer (vgl. Abschnitt 3.1.6) wurde als Teil des Masters geschaffen, um Abhängigkeiten als Baumstruktur zu visualisieren. Darin können auch übertragene Nutzdaten, wenn diese vorliegen, mit visualisiert werden, da diese zum Auffinden der Fehlerquelle oft nützlich sind.

Das BackPush-Konzept und der Visualizer wurden so Teil des IFM-Konzepts bzw. des Protokolls. Im Folgenden werden noch drei weitere zentrale Fragestellungen vorgestellt, deren Behandlung die Anwendbarkeit des Konzepts unterstützen soll:

- **Wo müssen Spy-Knoten platziert werden, damit der IFM die Abhängigkeiten erfassen kann?**
  Damit der IFM Abhängigkeiten erfassen kann, muss auf möglichst vielen Pfaden ein Spy-Knoten liegen. In manchen CPS ist die Platzierung von Spy-Knoten jedoch teuer und sie lassen sich nicht beliebig platzieren. Mit dieser Problematik beschäftigt sich das Kapitel zur Platzierung von Spy-Knoten (vgl. Abschnitt 3.2).

- **Wie können Knoten, die das IFM-Protokoll nicht unterstützen, in das CPS integriert werden?**
  Black-Box-Knoten, die das IFM-Protokoll nicht unterstützen, verursachen in IFM-überwachten Netzen Probleme, da diese Knoten Abhängigkeitsinformationen weder erstellen noch weitergeben. Mit der Abmilderung dieses Problems beschäftigt sich das Information Dependency Modeling (vgl. Abschnitt 3.3).

- **Wie kann das IFM-Protokoll in Mehrparteiensystemen eingesetzt werden, in denen kein Vertrauen herrscht?**
  In großen Netzwerken, in denen Systeme verschiedener Parteien zusammenarbeiten, herrscht üblicherweise kein uneingeschränktes Vertrauen zwischen den Systemen. Daher ist es sowohl notwendig, dass einzelne Parteien sowohl eigene Master-Instanzen nutzen können als auch das Abhängigkeiten manipulationssicher abgelegt werden. Die hieraus resultierenden Fragestellungen werden durch das IFM-Blockchain-Konzept (vgl. Abschnitt 3.4) behandelt.

3.2 Platzierung von Spy-Knoten

3.2.1 Motivation

Damit das IFM-Protokoll seinen Zweck erfüllen kann und alle notwendigen Abhängigkeitsinformationen in der zentralen IFM Master-Instanz zusammengetragen werden können, sind die Spy-Knoten essenziell. Befindet sich auf dem Pfad einer Nachricht kein Spy-Knoten, gehen die hier gesammelten Informationen über Abhängigkeiten verloren und werden nicht an den IFM-Master gesendet. Daher ist sicherzustellen, dass sich auf möglichst vielen Pfaden mindestens ein Spy-Knoten

befindet. Eine vollständige Abdeckung aller möglichen Pfade kann nur mit sehr hohem Aufwand erreicht werden, da auch sehr kurze Pfade, die nur wenige Knoten enthalten, abdeckt werden müssten. Ein Pfad ist abgedeckt, wenn sich auf diesem mindestens ein Spy befindet.

Das Problem der Komplexität der Nachverfolgung von Abhängigkeiten verschärft sich bei steigenden Pfadlängen. Für kurze Pfade lassen sich Abhängigkeiten noch teilweise manuell nachverfolgen. Mit steigender Pfadlänge nimmt die Komplexität der Abhängigkeiten stark zu, da die Anzahl der beteiligten Knoten ebenfalls steigt. Daher sind insbesondere auch lange Pfade abzudecken.

Einfach zu erreichen wäre dieses Ziel, wenn jeder Knoten im Netzwerk ein Spy ist. Dies führt einerseits zu einer perfekten Abdeckung aller Pfade, andererseits aber auch zu einer guten Performance, da History-Informationen nicht mehr über viele Knoten hinweg mittransportiert werden müssen, ehe diese zum Master gesendet werden. Lassen sich also einfach und günstig viele Spy-Knoten im Netz platzieren, ist dies zu präferieren. In vielen Anwendungsszenarien in CPS ist es jedoch nicht möglich, Spy-Knoten in beliebiger Menge kostengünstig einzusetzen. Dies ist insbesondere der Fall, wenn nicht alle Knoten mit der zentralen Master-Instanz kommunizieren können und die Schaffung einer Möglichkeit zur Kommunikation teuer ist. Dies ist insbesondere in Peer-2-Peer Netzen der Fall, in denen nicht jeder Knoten an ein größeres Netz wie das Internet angeschlossen ist und somit nur mit seinen Nachbarn kommunizieren kann. Gleichzeitig ist auch der Implementierungsaufwand für Worker im Vergleich zu Spies geringer – wenn auch nicht signifikant. Für Szenarien, in denen Spy-Knoten teuer sind, betrachten wir im Folgenden Methoden, wie die Menge der benötigten Spy-Knoten reduziert werden kann, sodass nicht jeder Knoten mit dem Master kommunizieren können muss.

Insgesamt soll also die Menge an durch Spy-Knoten abgedeckten Pfade mit minimalem Aufwand maximiert werden. Der Aufwand ist geringer je weniger Spy-Knoten eingesetzt werden müssen, um die gewünschte Abdeckung zu erreichen. Die Auswahl der Spy-Knoten ist keine triviale Aufgabe, da es sich hierbei um ein Optimierungsproblem handelt. Es soll mit möglichst wenigen Spy-Knoten eine möglichst große Menge an Pfaden abgedeckt werden. Dieses Problem intensiviert sich insbesondere in größeren Netzwerken.

Das Vorgehen ist wie folgt: Zuerst wird das sogenannte Spy-Cover-Problem formal (Abschnitt 3.2.2) definiert. Hierbei geht es um die Auswahl/Platzierung von Spy-Knoten im Netzwerk. Anschließend wird gezeigt, dass dieses Problem NP-schwer ist und es keinen effizienten Algorithmus geben kann, der das ideale Spy Set zu einem Netzwerk berechnet (Abschnitte 3.2.3 und 3.2.3.1). Daher ist es sinnvoll, dass ideale Spy Set nicht zu berechnen, sondern sich einer guten Lösung anzunähern. Dies wird auf Basis von Guidelines und Heuristiken zur Platzierung von Spy-Knoten geschehen (Abschnitt 3.2.4). Die vorgestellten Heuristiken werden mit Hilfe eines evolutionären Algorithmus optimiert und evaluiert (Abschnitt 3.2.5). Der evolutionäre Algorithmus soll hierbei Heuristiken finden, mit deren Hilfe Spy-Knoten-Mengen, sogenannte Spy Sets, ausgewählt werden können, die hohe Abdeckungen erreichen. Abschnitt 3.2.6 betrachtet abschließend die bei der Erstellung von Spy Sets entstehenden Kosten, die aus der Umwandlung von Knoten in Spies resultieren.

3.2.2 Problemdefinition

Im Folgenden modellieren wir ein CPS als einfachen Graphen $G = (V, E)$ mit der Knotenmenge V und der Kantenmenge E, wobei die Knoten den IFM-Workern und die Kanten den möglichen Kommunikationsverbindungen zwischen den Workern entsprechen. Einfache Graphen sind ungerichtet, haben keine Schleifen[25] und zwischen zwei Knoten existiert immer maximal eine Kante. Sei A die zugehörige Adjazenzmatrix.

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,m} \end{pmatrix} \tag{3.1}$$

[25] Als Schleife bezeichnet man bei Graphen eine Kante, die einen Knoten mit sich selbst verbindet. Kreise wiederum sind bei einfachen Graphen erlaubt.

Da der Graph ungerichtet ist, ist A symmetrisch. Die Anzahl der Pfade zwischen Knoten kann mit Hilfe der Potenz von A berechnet werden. Sei die Matrix A zur Potenz l wie folgt benannt:

$$A^l = \begin{pmatrix} b_{l,1,1} & \dots & b_{l,1,m} \\ \vdots & \ddots & \vdots \\ b_{l,m,1} & \dots & b_{l,m,m} \end{pmatrix} \tag{3.2}$$

In G gibt es $b_{l,i,j}$ Kantenfolgen der Länge l, die von Knoten i nach Knoten j führen. Die Summe aller Elemente der Matrix A^l entspricht dann der Anzahl der Pfade der Länge l, die im Graphen G existieren. Insgesamt gibt es q Pfade im Graph die die Länge zwischen 1 und $|V|$ haben.

$$q = \sum_{l=1}^{|V|} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} b_{l,i,j} \tag{3.3}$$

Sei U die Menge aller möglichen Pfade im Graphen G für die gilt, dass jeder Pfad jeden Knoten maximal einmal enthält. Ein Pfad besucht einen Knoten also nicht mehrfach, muss aber auch nicht alle Knoten besuchen. Alle Pfade $w \in U$ werden als geordnete Knotenfolgen verstanden und nicht als Kantenfolgen.

Die Menge aller Teilmengen von V wird auch Potenzmenge von V genannt, in Zeichen $\mathcal{P}(V)$. Jedes $S \in \mathcal{P}(V)$ ist ein potenzielles Spy Set. Ein potenzielles Spy Set ist also eine Teilmenge der Knotenmenge V.

Gesucht werden Spy Sets S, die mindestens den Anteil p mit $0 \leq p \leq 1$ der Pfade aus U abdecken, d. h.

$$|\{w \in U | \exists s \in S : s \in w\}| \geq p \cdot |U|. \tag{3.4}$$

Sei M_p die Menge aller S, für die (3.4) gilt. Es wird ein Spy Set $S_{min} \in \mathcal{P}(V)$ mit minimaler Größe gesucht, d. h.

$$|S_{min}| \leq |S| \text{ für alle } S \in M_p. \tag{3.5}$$

Zur Übersicht, auch für folgende Kapitel, sind hier noch einmal alle Variablen in einer Legende dargestellt:

G	Graph
V	Knotenmenge des Graphen G
E	Kantenmenge des Graphen G
A	Adjazenzmatrix des Graphen G
q	Anzahl der Pfade der Länge zwischen 1 und $\|V\|$ im Graphen G
U	Menge aller Pfade im Graphen G, die jeweils jeden Knoten maximal einmal enthalten
w	Ein Pfad aus U
S	Ein potenzielles Spy Set aus der Potenzmenge von V
p	Der Anteil an den Pfaden aus U, die ein Spy Set abdecken soll
M_p	Die Menge aller Spy Sets S, die mindestens den Anteil p der Pfade aus U abdecken
S_{min}	Das kleinste Spy Set aus M_p

3.2.3 Beweis der NP-Schwere

3.2.3.1 Komplexitätsklassen

In der Informatik bezeichnet man Fragen, deren Ergebnisse nur „ja“ bzw. 1 und „nein“ bzw. 0 sein können, als Entscheidungsprobleme. Probleme sind dann entscheidbar, wenn es einen Algorithmus gibt, der die Frage mit 0 oder 1 beantwortet. Es gibt jedoch ebenfalls nicht entscheidbare Fragen, für die keine Antwort gegeben werden kann, da kein Entscheidungsverfahren in Form eines Algorithmus existiert.

Entscheidbare Entscheidungsprobleme können unterschiedlich schwer durch einen Algorithmus lösbar sein. In diesem Zusammenhang unterscheidet man zwischen so genannten Komplexitätsklassen, die die Entscheidungsprobleme nach Schwere klassifizieren. Besondere Relevanz in der Informatik haben die Komplexitätsklassen P und NP.

Die Klasse P enthält hierbei alle Entscheidungsprobleme, die in Polynomialzeit durch einen Algorithmus lösbar sind. Dies bedeutet, dass die benötigte Rechenzeit zur Bestimmung der Antwort bei steigender Problemgröße maximal mit einer Polynomfunktion, insbesondere nicht exponentiell, wächst.

Die Klasse NP ist eine Obermenge von P und enthält zusätzlich alle Probleme, deren „ja“-Antwort in polynomialer Zeit überprüfbar ist. Nicht alle Probleme in NP sind also in Polynomialzeit entscheidbar. Für unseren Anwendungsfall sind insbesondere die NP-schweren und NP-vollständigen Probleme von Relevanz. NP-vollständige Probleme sind eine Untermenge von NP und lassen sich vermutlich nicht effizient lösen. Hieraus resultiert auch das sogenannte P-NP-Problem, dass sich mit dieser Frage der Effizienz beschäftigt. Das Problem wird bereits seit den 70er-Jahren diskutiert und ist in die Liste der Millennium-Probleme aufgenommen worden [76]. Bisher ist die Frage, ob P und NP gleich sind, nicht gelöst. Wären die Klassen gleich, wären NP-vollständige Probleme automatisch auch effizient lösbar. Hierfür existiert kein Beweis. Es wird aber in der Informatik allgemein angenommen, dass P und NP nicht gleich sind und somit NP-vollständige Probleme nicht effizient lösbar sind [77]. Neben der NP-Vollständigkeit ist die NP-Schwere[26] in Bezug auf diese Arbeit von Relevanz. NP-schwere Probleme sind mindestens so schwer wie alle Probleme in NP. NP-schwer sind also beispielsweise auch nicht-entscheidbare Probleme. NP-schwere Probleme sind genauso schwer oder schwerer als NP-vollständige Probleme.

Um im Rahmen dieser Arbeit zu zeigen, dass ein ideales Spy Set in einem CPS nicht effizient berechenbar ist, reicht dementsprechend der Beweis aus, dass das Problem NP-schwer ist. Daraus würde resultieren, dass Methoden gefunden werden müssen, um sich einer guten Lösung anzunähern.

Der Beweis der NP-Schwere wird im Folgenden erbracht.

3.2.3.2 Beweisstrategie

Der initiale Beweis der NP-Schwere für ein Problem ist oft sehr komplex und aufwändig. Daher bedient man sich in der Regel eines bekannten anderen Problems, für das schon gezeigt wurde, dass es NP-vollständig ist. Kann man nun zeigen, dass das eigene Problem mindestens so schwer ist, wie das Problem, für das

[26] NP-Schwere und NP-Härte sind im Deutschen Synonyme, wobei es sich bei NP-Härte vmtl. um eine Fehlübersetzung des englischen Begriffs *NP-hard* handelt. Für den englischen Begriff *NP-hard* sollte daher im Deutschen NP-schwer verwendet werden.

bereits bekannt ist, dass es NP-vollständig ist, ist das eigene Problem NP-schwer. Diesen Prozess nennt man Reduktion.

Bei einer Reduktion betrachtet man zuerst den Algorithmus, der das eigene Problem (im Folgenden Spy Cover genannt) löst. Für dieses Problem soll gezeigt werden, dass es NP-schwer ist. Bei einer Reduktion versucht man nun zu zeigen, dass dieser Spy-Cover-Algorithmus auch genutzt werden kann, um ein bereits als NP-vollständig bekanntes Problem (im Folgenden Vertex Cover) zu lösen. Kann gezeigt werden, dass der Spy-Cover-Algorithmus genutzt werden kann, um das NP-vollständige Vertex Cover zu lösen, so ist Spy Cover mindestens so schwer wie Vertex Cover. Daraus würde folgen, dass Spy Cover NP-schwer ist.

Ziel der Reduktion ist es also, Vertex Cover mit dem Spy-Cover-Algorithmus zu lösen. Hierzu gilt es die Ein- und Ausgaben in den Vertex-Cover-Algorithmus so vor- und nachzuverarbeiten, dass der Spy-Cover-Algorithmus damit arbeiten kann und das Problem löst. Es wird so ein Vertex-Cover-Algorithmus erstellt, der im Inneren nur aus dem Spy-Cover-Algorithmus sowie einer Datenvor- und Datennachverarbeitung besteht. Die erforderlichen Verarbeitungen sind somit zentraler Bestandteil der Reduktion und werden im Folgenden erläutert.

Die Menge S_{min} ist ein Spy Set und eine Menge an Knoten, die die gewünschte Abdeckung p der Pfade im Netzwerk erreicht. Das Set hat eine Mindestgröße, da es keine kleineren Sets gibt, die die gewünschte Abdeckung erreichen können. Wenn die Größe von S_{min} bekannt ist, ist es klar, dass es auch Knoten-Sets für alle größeren Größen gibt, die diese Abdeckung erreichen. Ein algorithmischer Ansatz zum Finden eines S_{min} könnte also sein, V als erstes Spy Set zu verwenden. V als Spy Set erreicht die gewünschte Reichweite in jedem Fall, da es 100% aller Pfade abdeckt. Anschließend wird die Größe des Spy Sets iterativ verringert, bis kein Set mit den gewünschten Eigenschaften gefunden werden kann. Dann ist die Größe des Spy Set S_{min} minimal.

3.2.3.3 Beweis

Spy-Cover-Algorithmus

Der Spy-Cover-Algorithmus bestimmt ein Spy Set für eine gegebene Menge an Pfaden $T \subseteq U$[27], das die Formel (3.4) (ersetze U mit T in der Formel) mit einer gegebenen Abdeckung p erfüllt. Das Spy Set soll maximal k Knoten enthalten. Ein Pfad gilt als abgedeckt, wenn die Schnittmenge aus Pfad und dem Spy Set nicht leer ist, d.h.:

$$\exists s \in S : s \in w$$

Deckt ein Spy Set einen Pfad ab liegt also mindestens ein Spy-Knoten des Spy Sets auf dem Pfad.

Der Spy-Cover-Algorithmus arbeitet wie folgt:

- Eingaben
  - Pfadmenge T, eine gegebene Menge an Knotenfolgen im Graph mit $T \subseteq U$
  - Knotenmenge V, die Menge aller Kanten im Graph G
  - Abdeckung p, der Anteil an allen Pfaden in T, der mindestens abzudecken ist
  - Spy Set-Größe k, die maximale Größe eines Spy Sets, das gefunden werden soll
- Fragestellung
  - Gibt es ein Spy Set $S \subseteq V$ mit $|S| \leq k$, dass $p \cdot |T|$ Pfade abdeckt? Wenn ja, welches?

[27] Aus verschiedenen Gründen kann es sinnvoll sein, die Menge der Pfade zu beschränken, ohne jedoch die Gesamtmenge auszuschließen. Gründe können z. B. Anforderungen an die Kommunikation sein, die aus den verschiedenen Rollen der einzelnen Knoten resultieren. So können beispielsweise Sensoren nicht untereinander kommunizieren. Der Beweis schließt aber U nicht aus, um möglichst übertragbar zu sein.

- Ausgaben
  - $\emptyset$, falls keine Lösung existiert. Dann gibt es kein Spy Set, dass maximal k Knoten beinhaltet und $p \cdot |T|$ Pfade abdeckt.
  - $S \subseteq V$, falls eine Lösung existiert. Dann ist S ein Spy Set das mindestens den Anteil p aller Pfade T abdeckt.

Der Spy-Cover-Algorithmus berechnet ein Spy Set mit der Größe k oder kleiner, das die gewünschte Abdeckung der Pfade T im Netzwerk erreicht. Im Folgenden wird bewiesen, dass wahrscheinlich kein effizienter Spy-Cover-Algorithmus existiert, da das Problem NP-schwer ist. Dies geschieht mit Hilfe des Vertex-Cover-Algorithmus und einer Reduktion.

Vertex-Cover-Algorithmus

Der Vertex-Cover-Algorithmus fragt, ob für einen gegebenen einfachen Graphen eine Knotenüberdeckung der maximalen Größe k existiert. Das bedeutet, ob es eine Teilmenge N der bestehenden Knotenmenge gibt, die aus maximal k Knoten besteht, so dass jede Kante des Graphen mit mindestens einem Knoten von N verbunden ist. Das Vertex-Cover-Problem gehört zu der Liste der 21 NP-vollständigen Probleme, deren Zugehörigkeit zu dieser Komplexitätsklasse Richard Karp 1972 [78] zeigte.

- Eingaben
  - Graph $G = (V, E)$
  - $k \in \mathbb{N}$
- Fragestellung
  - Gibt es eine Menge $N \subseteq V$ mit $|N| \leq k$, sodass alle Kanten ein Element aus N enthalten?
  - Informell: Gibt es eine Knotenmenge N der maximalen Größe k für die jede Kante des Graphen bei einem Element aus N beginnt oder endet?
- Ausgabe
  - Ja / Nein

Reduktion

Es wird im Folgenden gezeigt, dass das NP-vollständige Problem Vertex Cover auch durch Spy Cover gelöst werden kann. Wenn bewiesen werden kann, dass der Algorithmus, der Spy Cover lösen kann, auch Vertex Cover lösen kann, dann ist Spy Cover mindestens so schwierig wie Vertex Cover. Das bedeutet, dass Spy Cover NP-schwer ist.

- Vorverarbeitung
  - Transformiere alle Kanten aus E in Pfade der Länge 1 und definiere diese Pfadmenge als T
  - Übernehme die Knotenmenge V aus der Graphdefinition
  - Übernehme k
  - Setze p auf 1
- Nachverarbeitung
  - Ist das Ergebnis die leere Menge $\emptyset$, dann ist die Antwort *Nein*, anderenfalls *Ja*.

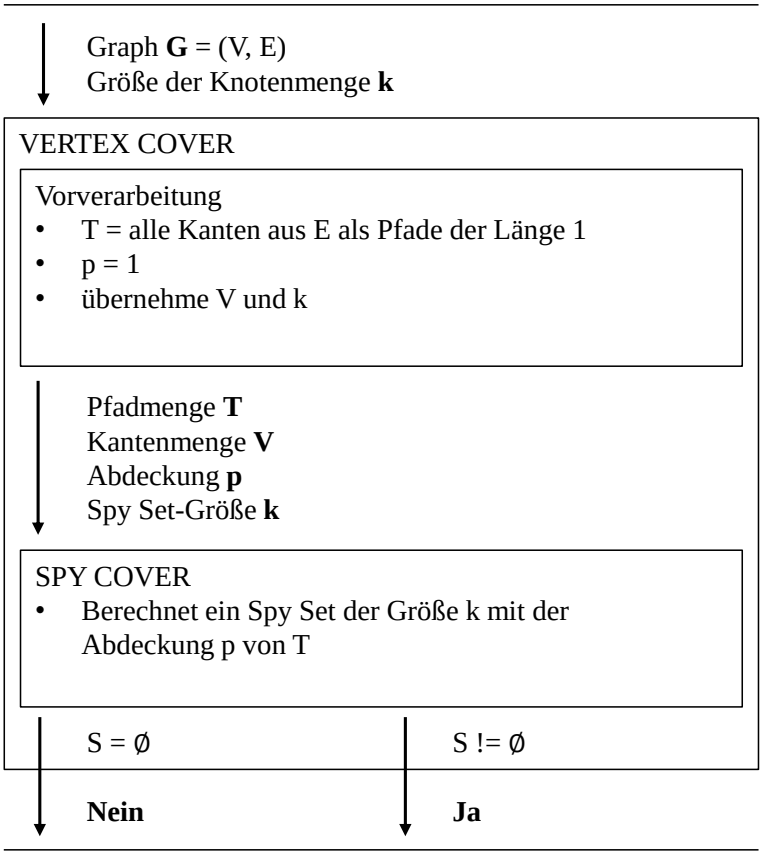


Abbildung 20: Visualisierung der Reduktion

Der gesamte beschriebene Reduktionsprozess wird in Abbildung 20 visualisiert. Darin zu sehen ist, dass der Vertex-Cover-Algorithmus (große Box) im Inneren durch eine Vorverarbeitung und den Spy-Cover-Algorithmus ersetzt wurde, aber nach außen weiterhin die gleichen Eingaben entgegennimmt und die gleichen Ausgaben erzeugt. Die Übergabe von Daten zwischen den einzelnen Komponenten und zur Ein- und Ausgabe sind mit Pfeilen visualisiert.

Ergebnis

Mit Hilfe der dargestellten Vor- und Nachverarbeitung ist der Algorithmus, der Spy Cover löst, auch in der Lage, Vertex Cover zu lösen. Da Vertex Cover NP-vollständig ist, ist hiermit bewiesen, dass Spy Cover NP-schwer ist.

3.2.4 Lösungskonzept zur Platzierung von Spy-Knoten

Wie gezeigt wurde, ist das Spy-Cover-Problem NP-schwer. Das bedeutet, es gibt wahrscheinlich keinen effizienten Algorithmus, der in der Lage ist, zu einem gegebenen Netzwerk und einer gewünschten Abdeckung das minimale Spy Set zu errechnen, welches die Abdeckung erreicht, ohne jede mögliche Spy-Set-Kombination zu prüfen.

Auch in mittelgroßen Netzwerken (vgl. Abbildung 23, Seite 123) ist die Anzahl der Möglichkeiten ein Set zu bilden enorm. Es existieren 1-267.650.600.228. 229.401.496.703.205.375 Möglichkeiten[28] im gezeigten Netzwerk mit 100 Knoten ein Spy Set zu bilden. Üblicherweise gibt es in CPS-Netzen Einschränkungen zur Platzierbarkeit von Spy-Knoten, die diese Menge einschränken. Die Menge bleibt jedoch unüberschaubar groß. Für jedes der möglichen Spy Sets müsste geprüft werden, welcher Anteil der im gezeigten Netzwerk vorhandenen 161.107 Pfade durch das Spy Set abdeckt wird. Der Aufwand ist insbesondere für noch größere Netze nicht zu leisten (vgl. Abschnitt 3.2.5).

Durch das Ausprobieren aller Möglichkeiten wäre es theoretisch möglich, die beste Lösung zu ermitteln. Dies ist aber nicht unbedingt notwendig. Eine sehr

[28] Die theoretisch maximale Menge an Spy Sets in einem Netzwerk der Größe x lässt sich durch die Formel $\sum_{m=1}^{x} \frac{x!}{(x-m)!\cdot m!}$ bestimmen.

gute Lösung ist völlig ausreichend, wenn sie mit viel weniger Aufwand ermittelt werden kann. Gleichzeitig gibt es auch nur eine einzige Beste, aber mehrere sehr gute Lösungen. Diese mehreren guten Lösungen geben Flexibilität und Auswahlmöglichkeiten bei der Platzierung der Knoten.

Die Idee bei der Positionierung der Spy-Knoten im Netzwerk ist daher nicht, direkt zu berechnen, welche Knoten im Netzwerk Spy-Knoten sein sollen. Stattdessen sollen Empfehlungen ausgesprochen werden, welche Eigenschaften ein Spy-Knoten erfüllen sollte, damit dieser zu einer hohen Pfadabdeckung des Spy Sets beiträgt. Die einzelnen Empfehlungen werden Guidelines genannt. Kombiniert man mehrere Guidelines entsteht eine Heuristik.

3.2.4.1 Guidelines und Heuristiken

Guidelines sind Anforderungen an ein Spy Set. Die Annahme ist, dass die Einhaltung der Guidelines zu Spy Sets führt, die mehr Pfade mit weniger Spy-Knoten abdecken, da andere Spy Sets den Anforderungen der Guidelines nicht entsprechen. Die im Folgenden vorgestellten Guidelines werden in Abschnitt 3.2.5 mit Hilfe eines evolutionären Algorithmus evaluiert. Guidelines werden benutzt, um damit die Spy Sets zu erstellen, die zu einer hohen Abdeckung von Kommunikationspfaden führen. Die Guidelines orientieren sich hierbei an verwandten Arbeiten [79, 80, 81, 82], die sich sowohl mit der Wichtigkeit von Knoten in Netzwerken, z. B. Zentralitätsmaßen, beschäftigen, als auch an Arbeiten, die Verbindungspfade in Netzwerken und die Bedeutung einzelner Knoten untersuchen. Auch die Berechnung der im Folgenden genutzten Zentralitätsmaße ist in diesen Arbeiten beschrieben.

Im Folgenden werden sieben Guidelines beschrieben, die jeweils unterschiedlich parametrisiert werden können. Eine Zusammenstellung aus den Guidelines (jeweils mit Parameter) wird Heuristik genannt und technisch durch eine Konfiguration in Form eines JSON-Arrays beschrieben. Eine Konfiguration beschreibt, welche Guideline in welcher Parametrisierung angewendet werden soll. Ziel ist es, eine gute, möglichst universell einsetzbare Heuristik in Form ihrer Konfiguration zu ermitteln. Aus den sieben Guidelines und ihren jeweiligen Parametrisierungen las-

sen sich 62.208 mögliche Konfigurationen[29] bilden. Die jeweilige Parametrisierung der Guidelines sind im Folgenden mit x beschrieben. Für $x = 0$ ist die Guideline in der Heuristik nicht enthalten bzw. deaktiviert. Die Guidelines lauten wie folgt:

0. ***DEGREE*** – *Jeder Knoten im Spy Set sollte mindestens x ausgehende Kanten haben.* – Parameter: $0 \leq x \leq 5$ mit $x \in \mathbb{N}$

   Eine größere Menge an Kanten an einem Knoten führt zu mehr Pfaden, die durch diesen Knoten verlaufen und so überwacht werden können.

1. ***DISTANCE*** – *Zwei Knoten im Spy Set sollten nicht über einen Pfad kürzer als x verbunden sein.* – Parameter: $0 \leq x \leq 5$ mit $x \in \mathbb{N}$

   Diese Regel definiert einen Mindestabstand zwischen Knoten. Ist $x = 2$, so sind beispielsweise keine benachbarten Knoten im Set zulässig. Durch benachbarte Knoten verlaufen viele gleiche Pfade. Hier entstehen sonst Redundanzen in der Überwachung, die unnötig sind, wenn möglichst wenige Spy-Knoten eingesetzt werden sollen.

2. ***BLIND*** (Blind Alley) – *Knoten des Spy Sets sollten nicht in einer Sackgasse liegen.* – Parameter: $x = 0 \vee x = 1$

   Eine Sackgasse ist ein unverzweigter Teilgraph des Netzwerkes, der nur auf einer Seite an den Hauptteil des Netzes angebunden ist und auf der anderen Seite endet (z. B. 91, 7, 61 am unteren Rand von Abbildung 23, Seite 123). [83]

3. ***DEG-CENT*** (Degree Centrality) – *Knoten des Spy Sets sollten zu den besten* $\frac{100}{x}\%$ *aller Knoten bzgl. Degree Centrality gehören.* – Parameter: $0 \leq x \leq 5$ mit $x \in \mathbb{N}$

   Das Maß der Degree Centrality ist eine Bezeichnung für den Grad eines Knotens. Hierbei wird angenommen, dass je mehr Kanten mit einem Knoten verbunden sind, desto zentraler ist dieser Knoten. Im Gegensatz zur DEGREE-Guideline wird hier jedoch kein absoluter Wert als Grad voraus-

[29] Die Anzahl der möglichen Konfigurationen ist das Produkt aus den jeweiligen Anzahlen der Möglichkeiten bei der Parametrisierung der Guidelines.

gesetzt, sondern verlangt, dass die Knoten zum besten Anteil bzgl. Degree Crentrality aller Knoten gehören.

4. ***CLO-CENT*** (Closeness Centrality) – *Knoten des Spy Sets sollten zu den besten $\frac{100}{x}$% aller Knoten bzgl. Closeness Centrality gehören.* – Parameter: $0 \leq x \leq 5$ mit $x \in \mathbb{N}$

   Als Closeness Centrality eines Knoten bezeichnet man den Kehrwert der Summe der Längen aller kürzesten Wege zwischen dem Knoten und allen anderen Knoten des Netzwerkes. Je größer der Wert, desto zentraler ist der Knoten.

5. ***BET-CENT*** (Betweenness Centrality) – *Knoten des Spy Sets sollten zu den besten $\frac{100}{x}$% aller Knoten bzgl. Betweenness Centrality gehören.* – Parameter: $0 \leq x \leq 5$ mit $x \in \mathbb{N}$

   Die Betweenness Centrality beschreibt, auf wie vielen der kürzesten Pfade aller möglichen Knotenpaare ein Knoten des Netzwerkes liegt. Knoten, die auf vielen dieser kürzesten Pfade liegen, werden als zentraler eingestuft. Je größer der Wert, desto zentraler ist der Knoten.

6. ***ROLES*** – *Sensoren (x =1), Aktoren (x=2) oder Sensoren und Aktoren (x=3) sollten nicht im Spy Set enthalten sein.* – Parameter: $0 \leq x \leq 3$ mit $x \in \mathbb{N}$

   Dadurch, dass Pfade an Sensoren und Aktoren immer enden oder beginnen, verlaufen keine Pfade durch diese Knoten hindurch. Dadurch können hier weniger Pfade überwacht werden.

Die vorgestellten Guidelines wurden anhand von drei Kriterien zusammengestellt:

- **Spezialisierung auf CPS:** Diese Arbeit fokussiert die Besonderheiten innerhalb von CPS. In den Netzen und deren Topologie spiegelt sich dies insbesondere in Form der Rollen und der daraus resultierenden Vernetzung wider. Innerhalb der Guidelines wird dies durch die *ROLES*-Guideline verkörpert.

- **Zentralität der Knoten:** Innerhalb der Literatur finden sich Arbeiten [84, 80, 85, 86, 87, 82], die sich mit der Zentralität von Knoten beschäfti-

gen. Zentrale Knoten erfüllen verschiedene Eigenschaften, die innerhalb der vorstehenden *BLIND*- und **-CENT*-Guidelines erläutert wurden. Grundannahme ist, dass zentrale Knoten öfter auf Pfaden liegen als andere Knoten und somit zu einer höheren Abdeckung beitragen.

- **Redundanz vermeiden:** Ziel bei der Zusammenstellung von Spy Sets ist es, diese möglichst klein zu halten. Weiter voneinander entferne Knoten haben weniger gemeinsame Pfade, die durch diese Knoten verlaufen. Ein Mindestabstand (*DISTANCE*-Guideline) soll diese unnötige Redundanz vermeiden.

Die vorgestellten Guidelines sind grundsätzlich anwendungsfallspezifisch erweiterbar.

Aus diesen Guidelines lassen sich anschließend Heuristiken mit ihren Konfigurationen bilden. Eine beispielhafte Konfiguration ist in Abbildung 21 dargestellt. Für die Konfigurationen existiert eine lange und eine kurze Schreibweise. Die lange Schreibweise wird in technischen Implementierungen genutzt und kann in die kurze Schreibweise zur besseren Lesbarkeit übersetzt werden. Im Verlauf dieser Arbeit wird daher insbesondere die kurze Schreibweise verwendet, obwohl technisch die lange Schreibweise umgesetzt wurde. In der langen Schreibweise ist der erste Wert jedes Subarray der Identifikator der Guideline und die zweite Zahl die Parametrisierung, also x. Die in der Abbildung dargestellte Konfiguration erlaubt dementsprechend nur Spy Sets, die keine benachbarten Knoten enthalten. Alle anderen Guidelines sind deaktiviert. In der kurzen Schreibweise werden nur aktivierte Guidelines mit Kurznamen und Parametrisierung aufgenommen. Nicht enthaltene Guidelines sind deaktiviert, was einem Parameterwert von $x = 0$ entspricht. Die beiden in Abbildung 21 dargestellten Konfigurationen beschreiben also die gleiche Heuristik.

$$[[0,0],[1,2],[2,0],[3,0],[4,0],[5,0],[6,0]]$$
$$[[DISTANCE,2]]$$

Abbildung 21: Eine Beispielkonfiguration in langer und kurzer Schreibweise

3.2.5 Evolutionärer Algorithmus

Wie die Heuristiken in welchen Szenarien konfiguriert werden müssen, um das Ziel der hohen Pfadabdeckung zu erreichen, ist nicht bekannt. Ebenfalls basieren die Heuristiken auf der zu überprüfenden Annahme, dass die enthaltenen Guidelines zu besseren Spy-Platzierungen führen.

Im Folgenden wird daher ein evolutionärer Algorithmus beschrieben und ausgeführt, der Guidelines zu Heuristiken kombiniert und evaluiert. Ziel ist das Finden von Heuristiken, deren Anwendung zu effizienteren Spy Sets führt, die hohe Abdeckungen erreichen. Da die perfekte Positionierung von Spy-Knoten im Netzwerk nur mit großem Aufwand berechnet werden kann, wird der evolutionäre Algorithmus genutzt, um Empfehlungen zur Platzierung von Spy-Knoten zu untersuchen.

Evolutionäre Algorithmen sind Optimierungsverfahren, die der Evolution von Lebewesen nachempfunden sind. Hierbei wird jede Evolutionsstufe genutzt, um die Problemlösungsstrategien anzupassen und zu evaluieren. Kann die leicht veränderte Strategie das Problem besser lösen, wird der Algorithmus mit dieser neuen Evolutionsstufe fortgesetzt. Kann die neue Evolutionsstufe das Problem nicht, oder schlechter als bereits bekannte Strategien, lösen, stirbt diese aus und wird in der nächsten Evolutionsstufe durch eine Mutation einer guten Strategie ersetzt. So werden Mutationen von Problemlösestrategien Teil neuer Evolutionsstufen, die bewertet werden können. Bessere Mutationen setzen sich gegen schlechtere durch. Je öfter eine neue Evolutionsstufe erreicht wird, desto besser kann das Problem gelöst werden. Bei evolutionären Algorithmen handelt es sich um Optimierungsverfahren die gute Lösungen anstreben, ohne den Anspruch zu haben das Optimum zu erreichen.

Durch den evolutionären Algorithmus sollen die zuvor vorgestellten Guidelines zu Heuristiken kombiniert und überprüft werden.

Für alle vorgestellten Guidelines (vgl. Abschnitt 3.2.4.1) lässt sich begründen, wie diese zu einer effizienteren Auswahl von Spy Sets beitragen können. Ihr Wert kann sich jedoch je nach Ausprägung des Netzes unterscheiden. Die DISTANCE-Guideline fordert beispielsweise, dass Spy-Knoten im Netzwerk nicht unmittelbar nebeneinander positioniert werden sollen, weil nach Passieren eines Spy-Knotens

der nächste Spy-Knoten kaum neue Informationen beitragen kann. Es besteht also die zu überprüfende Annahme, dass wenn man alle Spy Sets ausschließt, die benachbarte Spy-Knoten enthalten, die verbleibenden Spy Sets im Durchschnitt eine höhere Abdeckung erreichen, als die ausgeschlossenen Spy Sets. Aus einer Menge Guidelines gilt es also, die besten Guidelines zu Heurstiken zu kombinieren und zu konfigurieren. Die Idee ist es, dass am Ende des Experiments eine Menge an zielführenden Heuristiken übrigbleibt, deren Anwendung zur Auswahl guter Spy Sets führt. So wird der evolutionäre Algorithmus nicht zur Ermittlung guter Spy Sets genutzt, sondern zur Ermittlung guter Heuristiken, also Guidelinekombinationen. Diese Heuristiken können später genutzt werden, um Spy Sets mit hohen Abdeckungen zu identifizieren.

Für unterschiedlich ausgeprägte Netzwerke, bei denen ähnliche im Folgenden zu sogenannten Netzwerkklassen zusammengefasst werden, liegt es nahe, dass die gleiche Heuristik unterschiedlich erfolgreich sein kann. Daher wurde der evolutionäre Algorithmus auf verschiedene Netzwerkklassen angewandt. Die Ergebnisse für alle Klassen sind in Abschnitt 3.2.5.4 dargestellt. Im Folgenden wird jedoch hier das Vorgehen anhand der Netzwerkklasse beschrieben, die aus 100 Kanten und 100 Knoten besteht, von denen 10 als Sensor, 10 als Aktor und der Rest als Aggregator definiert werden. Diese Klasse wird im Folgenden 10S10A (10 Sensoren / 10 Aktoren) genannt[30].

Das Vorgehen des evolutionären Algorithmus für diese Netzwerkklasse sieht dabei wie folgt aus:

1. Generierungsversuch von 100.000 zufälligen Spy Sets, der in 97.027 Spy Sets resultierte
2. Generierungsversuch von 150 zufälligen Netzwerken für die beschriebene Netzwerkklasse, der in 164 Netzwerken resultierte
3. Generiere alle Pfade in allen Netzwerken
4. Generiere 10 Heuristiken, die jeweils aus einer Submenge an Guidelines bestehen
5. Berechne für jede Heuristik, für jedes Netzwerk die minimale Größe, die ein Spy Set haben muss, sodass keines der generierten Spy Sets dieser oder größerer Größen weniger als 90% aller generierten Pfade abdeckt

[30] Im späteren Verlauf werden auch die Klassen 30S30A und 40S40A betrachtet.

6. Berechne für jede Heuristik den Durchschnitt der minimal erforderlichen Spy-Set-Größen aller Netze
7. Behalte die besten 50% der Heuristiken, verwerfe den Rest. Gute Heuristiken führen zu kleineren Spy Sets, die 90% der Pfade abdecken.
8. Mutiere die verbleibenden Heuristiken und füge zwei weitere zufällige Heuristiken hinzu
9. Zurück zu Schritt 5, um die mutierten Heuristiken zu evaluieren

Die einzelnen Schritte werden in den folgenden Abschnitten detailliert beschrieben.

3.2.5.1 Generierung zufälliger Spy Sets

Die NP-schwere des Problems führt dazu, dass später im Algorithmus für jedes Spy Set die prozentuale Abdeckung der Menge aller Pfade in jedem Netzwerk berechnet werden muss. Um dies technisch leisten zu können, wurde die Menge der Spy Sets auf 100.000 begrenzt. Dies ist jedoch unproblematisch, da es nicht das Ziel ist, das beste Spy Set für ein Netzwerk zu ermitteln, sondern eine ganze Klasse an Spy Sets zu ermitteln, die der Heuristik folgen und hohe Abdeckungen erreichen. Daher ist es ausreichend, eine große Menge an Spy Sets für den evolutionären Algorithmus zu verwenden, unter denen dann viele Spy Sets sind, die die Heuristiken erfüllen. Bei der Generierung wird die Menge der Spy Sets zuerst auf die unterschiedlichen möglichen Größen des Spy Sets aufgeteilt. Bei einem Netzwerk mit 100 Knoten kann ein Spy Set zwischen einem und 100 Spy-Knoten enthalten. Demnach wird für jede Größe versucht, $100.000/100 = 1.000$ Spy Sets zu erstellen. Dazu wird für jede mögliche Spy-Set-Größe a ein Vektor mit $100 - a$ Nullen und a Einsen erstellt. Die Indizes der Einsen geben hierbei die IDs der Knoten im Netzwerk an, die als Spy ausgewählt wurden. Um nun 1.000 verschiedene Vektoren pro Größe zu erhalten, werden die Nullen und Einsen im Vektor 1.000 Mal gemischt, sodass pro Größe bis zu 1.000 unterschiedliche Vektoren vorliegen. Für manche Spy-Set-Größen existieren jedoch keine 1.000 unterschiedlichen Möglichkeiten ein Spy Set zu bilden. Für die Spy-Set-Größe 100 existiert beispielsweise nur eine Möglichkeit, und zwar alle Knoten auszuwählen. Für die Größe 1 wiederum existieren genau 100 Möglichkeiten ein Spy Set zu bilden. Daher entstehen durch dieses Verfahren tatsächlich nicht 100.000 verschiedene Spy Sets, sondern nur 97.027 Spy Sets. Die Größenverteilung ist in Abbildung 22 dargestellt. Für die Spy-Set-Größen 4-96 existieren 1.000 Spy Sets, für die anderen Größen weniger.

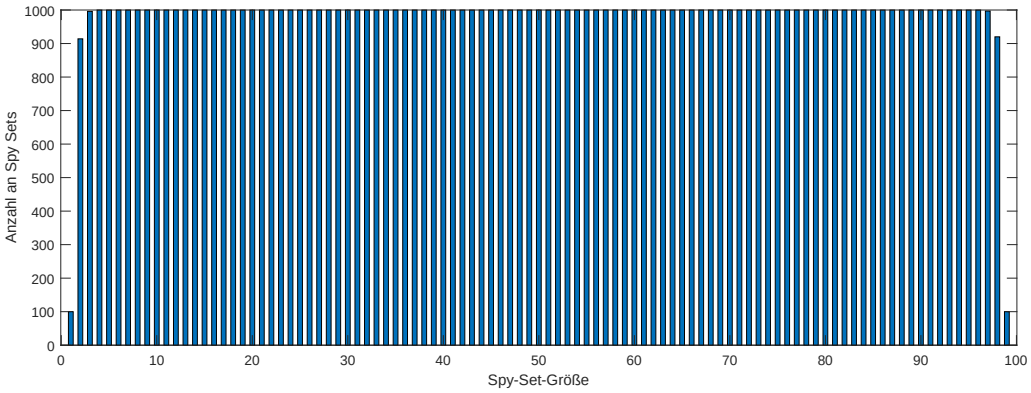


Abbildung 22: Verteilung der 97.027 Spy Sets nach ihrer Größe

3.2.5.2 Generierung zufälliger Netzwerke

Für das Experiment wird eine Menge von generierten Netzwerken genutzt. Diese Netzwerke sind zufällig erstellt, folgen aber bestimmten Regeln:

- *Knoten*
  Zuerst werden 100 Knoten erstellt. Jeder Knoten repräsentiert eine Komponente im Netzwerk, ist logisch, aber nicht unbedingt physisch, von anderen Knoten getrennt und kann mit anderen Knoten kommunizieren.

- *Sensoren*
  Aus der Menge von 100 Knoten werden 10 Knoten gezogen, die als Sensoren definiert werden.

- *Aktoren*
  Aus der verbleibenden Menge von 90 Knoten werden 10 Knoten gezogen, die als Aktoren definiert werden.

- *Aggregatoren*
  Alle übrigen Knoten werden als Aggregator definiert.

- *Kanten*
  Anschließend werden 100 ungerichtete Kanten erstellt. Dabei werden immer zwei Knoten zufällig gezogen und miteinander verbunden. Existiert eine Kante bereits, wird ein neues Knotenpaar gezogen. Ist keiner der beiden gezogenen Knoten ein Aggregator, wird ebenfalls ein neues Paar gezo-

gen, da Sensoren und Aktoren nicht direkt untereinander und miteinander kommunizieren.

- *Pfade*
  Anschließend werden alle theoretisch möglichen Pfade im Netzwerk mit maximaler Länge 100 erstellt, bei denen jeder Knoten auf seinem Pfad maximal einmal besucht wird. Längere Pfade können im Netzwerk aufgrund dieser Einschränkung nicht existieren. Pfade sind gerichtet. Zusätzlich gibt es Einschränkungen bzgl. der verschiedenen Rollen der Knoten. Sensoren dürfen nur Beginn eines Pfades sein und weder in der Mitte noch am Ende liegen. Kommt ein Sensor also im Pfad vor, ist dies der erste Knoten im Pfad. Aktoren dürfen nicht in der Mitte des Pfades liegen und auch nicht am Beginn. Kommt ein Aktor also im Pfad vor, ist dies der letzte Knoten im Pfad. Diese Vorgaben schränken die Pfadmenge stark ein. Es gibt keine Einschränkungen für Aggregatoren, da diese sowohl Daten verarbeiten und bereitstellen als auch speichern können. Aggregatoren können also sowohl am Anfang als auch am Ende und in der Mitte der Pfade liegen.

  Die Generierung der Pfade läuft technisch wie folgt ab: Zuerst werden alle Pfade der Länge 1 generiert. Dazu werden alle Sensoren und Aggregatoren ausgewählt und von ihnen jeweils ein Pfad zu allen ihren Nachbarn erzeugt. Die erzeugten Pfade bilden die Menge der Pfade der Länge 1. Andere Pfade der Länge 1 existieren nicht im Netzwerk. Um nun Pfade der Länge 2 zu generieren, werden alle Pfade der Länge 1 betrachtet. Für jeden dieser Pfade wird betrachtet, wie der Pfad der Länge 1 fortgesetzt werden könnte, um einen Pfad der Länge 2 zu erreichen. Pfade der Länge 1, die an einem Aktor enden, werden hierbei nicht betrachtet. Die Pfade der Länge 1 können nun mit allen Kanten, die am Knoten des Endes vom Pfad anliegen verlängert werden. Kanten, die zu einem bereits im Pfad enthaltenen Knoten führen, werden nicht beachtet. So entstehen aus den Pfaden der Länge 1 noch viel mehr Pfade der Länge 2. Dieses Verfahren wird iterativ fortgesetzt, bis alle Pfade aller Längen generiert sind. Es stellt sicher, dass am Ende des Prozesses für alle Größen zwischen 1 und 100 alle Pfade generiert wurden. Die Menge der resultierenden Pfade ist sehr groß, wenn auch nicht so enorm groß wie die Menge der möglichen Spy Sets (vgl. Abschnitt 3.2.4). Daher ist es hier möglich, tatsächlich alle Pfade, und nicht nur einen Ausschnitt zu betrachten.

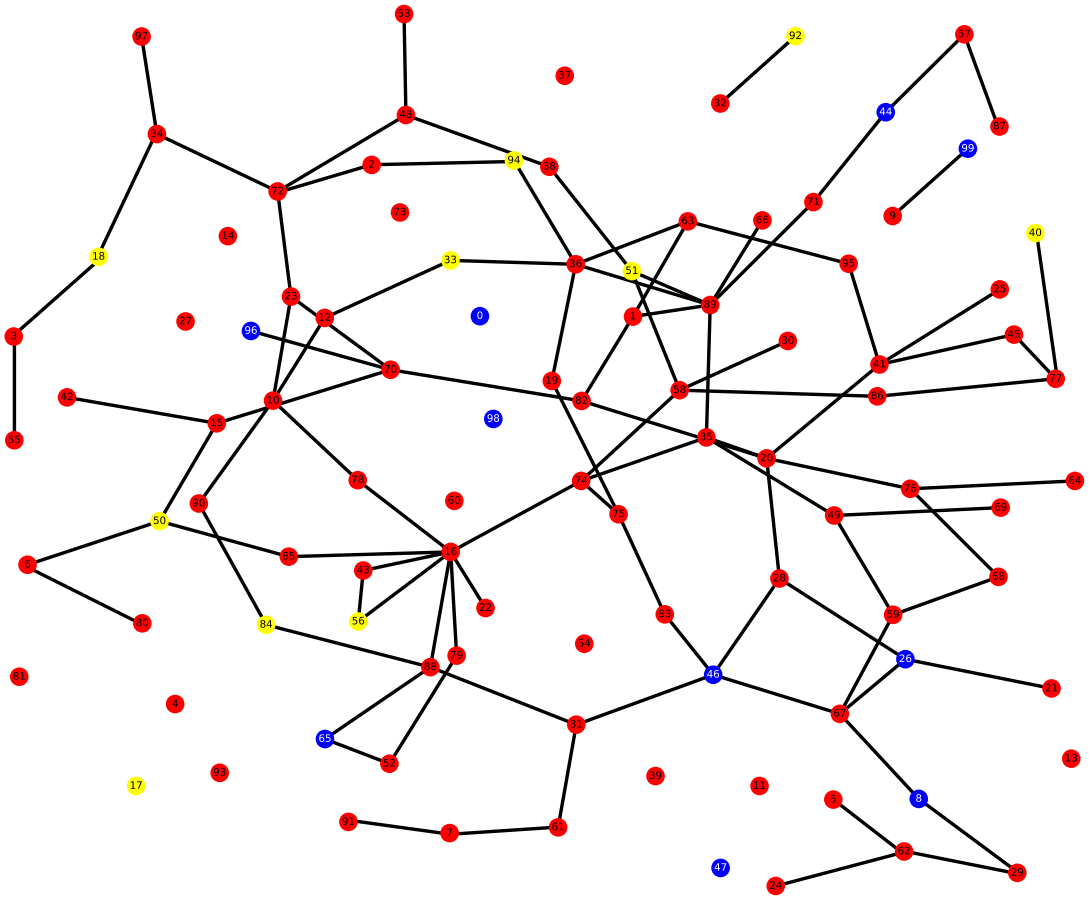


Abbildung 23: Netzwerk mit 100 Knoten, 100 Kanten, 10 Sensoren (blau), 10 Aktoren (gelb) und 80 Aggregatoren (rot)

Auf die zuvor beschriebene Weise wurden 164 Netzwerke generiert und abgespeichert. Die Netzwerke können sich hierbei bezüglich ihrer durchschnittlichen Pfadlänge deutlich unterscheiden – insbesondere, wenn die zufällige Generierung dazu führt, dass das Netzwerk aus Teilnetzen besteht, die wenig oder gar nicht miteinander kommunizieren. Bei Netzwerken mit stark unterschiedlichen Pfadlängen performen Heuristiken jedoch unterschiedlich gut[31]. Daher wurde darauf geachtet, dass für die verschiedenen durchschnittlichen Pfadlängen jeweils ähnlich viele Netzwerke generiert wurden.

Aus den Erkenntnissen mehrfacher Experimentdurchführungen ergab sich, dass mit einer Aufteilung nach durchschnittlichen Pfadlängen in drei Gruppen das

[31] Hierbei handelt es sich um Erfahrungswerte aus der mehrfachen Durchführung des Gesamtexperiments.

Optimierungsziel des evolutionären Algorithmus am besten erreicht werden kann. Für die durchschnittlichen Pfadlängen 5-8, 9-12 und 13-24 (jeweils *Bucket* genannt) wurden darum jeweils mindestens 50 Netze generiert[32]. Insbesondere hat sich bei vorherigen Experimentdurchführungen gezeigt, dass sich keine bzgl. Optimierungsziel gut performende Heuristik für den Fall finden lässt, dass es nur ein Bucket gibt, welches alle Netzwerke enthält. Netzwerke, deren durchschnittliche Pfadlängen nicht zwischen 5 und 24 waren, existierten nicht. Die Buckets können später einzeln zum Finden von Heuristiken untersucht werden, sodass je Bucket zugeschnittene Heuristiken erstellt werden können.

Die zuvor dargestellten Einschränkungen bei der Generierung der Netzwerke schränken die NP-schwere des Problems nicht ein, da die Reduzierung der Datenmenge (in diesem Fall der Pfade) keinen Einfluss auf die Schwere eines Problems hat. Des Weiteren existieren keine relevanten Einschränkungen für Pfade in Subnetzen, die ausschließlich aus Aggregatoren bestehen. Daher ist das Problem offensichtlich für diese Subnetze NP-schwer und somit auch für alle Netze, die diese Subnetze enthalten.

In Abbildung 23 ist exemplarisch ein Netzwerk dargestellt. Auffällig ist, dass durch diese Art der Erstellung sowohl isolierte Knoten als auch abgetrennte Teilgraphen entstehen. Diese Struktur ist in CPS nicht unüblich, da geographisch nahe Komponenten teils stärker vernetzt sind und so mit anderen Netzteilen wenig verbundene Teilnetze schaffen. In allen unseren generierten Netzwerken existiert allerdings ein dominierendes Kernnetzwerk, das einen Großteil der Knoten enthält. Um diese Kernnetzwerke herum existieren teils nicht vernetzte Knoten oder kleine Netze, die nicht mit dem Kernnetz verbunden sind. In der Abbildung sind Aggregatoren rot, Sensoren blau und Aktoren gelb. Sind die Knoten miteinander verbunden, können Pfade über diese Verbindungen verlaufen.

[32] Je mehr Netze untersucht werden, desto repräsentativer ist das Ergebnis für die jeweilige Netzklasse. Die Menge 50 wurde hier als Zielgröße gewählt, da mehr Netzwerke zu deutlich höheren Rechenzeiten des evolutionären Algorithmus führen. Für das Bucket 5-8 wurden 51, für das Bucket 9-12 56 und für das Bucket 13-24 57 Netze generiert. Insgesamt wurden also 164 Netze generiert. Die Streuung liegt in der zufälligen Generierung begründet, die Netze unterschiedlicher Buckets unterschiedlich häufig erstellt.

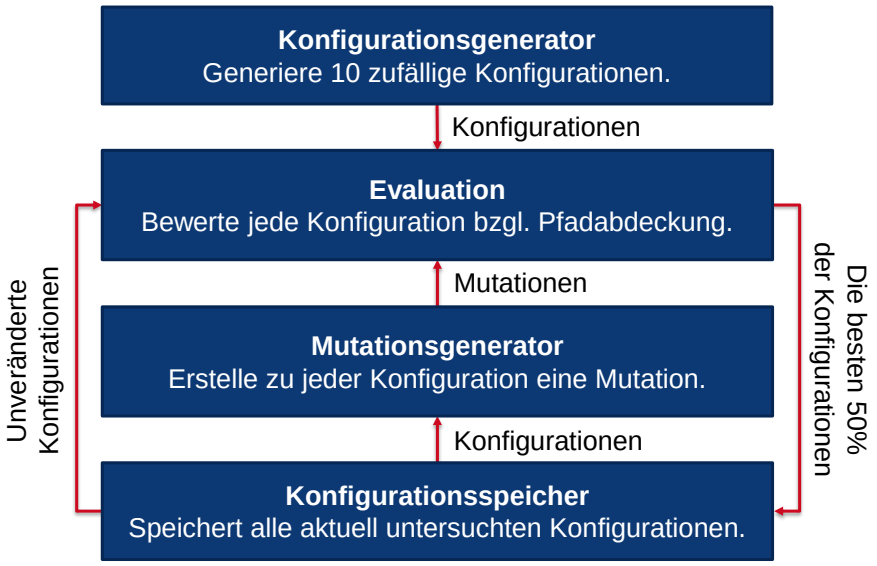


Abbildung 24: Visualisierung des evolutionären Algorithmus

3.2.5.3 Evolutionärer Algorithmus

Die in den vorausgehenden Abschnitten beschriebenen Schritte 1-3 (vgl. Übersicht der Schritte in Abschnitt 3.2.5) wurden als Vorarbeit für den evolutionären Algorithmus geleistet. Die eigentliche iterative Arbeit des Algorithmus findet in den Schritten 4-9 statt, die im Folgenden beschrieben werden und in Abbildung 24 visualisiert sind.

Das Ziel des evolutionären Algorithmus ist es, für die generierten Netze eine möglichst universelle Konfiguration von Guidelines zu finden, deren Anwendung bei der Auswahl von Spy-Knoten dazu führen soll, dass ein gewähltes Spy Set mit hoher Wahrscheinlichkeit[33] mindestens 90%[34] der Pfade abdeckt. Da die Netzwerkklassen und die Pfadlängen-Buckets innerhalb der Klassen sich strukturell unterscheiden, wird der evolutionäre Algorithmus auf jedes Bucket separat angewendet, um die dafür vielversprechendste Guideline-Konfiguration zu finden. Das

[33] In Abschnitt 3.2.5.6 werden die Ergebnisse des evolutionären Algorithmus in Bezug auf die Wahrscheinlichkeit der Erreichung von hohen Abdeckungen erläutert.

[34] In dieser Arbeit wird eine Abdeckung von 90% der Pfade als ausreichend betrachtet, da dies ein guter Kompromiss aus hoher Abdeckung und Kosteneffizienz ist. Dieser Wert kann bei Bedarf angepasst werden. Zu beachten ist, dass das Verhältnis zwischen benötigten Spies und erreichter Abdeckung nicht proportional ist. Eine kleine Steigerung der prozentualen Abdeckung jenseits 90% benötigt signifikant mehr Spies.

Vorgehen wird im Folgenden anhand des Beispiels des 57 Netzwerke umfassenden Buckets 13-24 der Klasse 10S10A vorgestellt.

Konfigurationsgenerator

Zuerst werden 10 zufällige Konfigurationen erstellt. Der Parameter x einer Guideline wird dabei gewürfelt, befindet sich allerdings je nach Guideline in einem anderen Bereich. Für manche Guidelines ist hier nur 0 oder 1 zulässig (aus oder an), für andere allerdings auch höhere Zahlen (vgl. Abschnitt 3.2.4.1). Diese Konfigurationen werden nur initial zu Beginn des Algorithmus erstellt.

Da die Konfigurationen zufällig erstellt wurden, ist nicht bekannt, ob diese dem Optimierungsziel entsprechen. Jede Konfiguration muss also im nächsten Schritt bewertet werden. Die Konfigurationen werden daher in der Evaluation (vgl. Abbildung 24) bewertet.

Evaluation

Jede erstellte Konfiguration wird innerhalb der Evaluation anhand von zwei Kriterien einzeln bewertet:

- *Primäres Kriterium*: Durchschnittlich benötigte Spy-Set-Größe

  Die Anwendung einer Konfiguration auf die 97.027 Spy Sets funktioniert wie ein Filter – es verbleiben die Spy Sets, die der Konfiguration entsprechen. Die verbleibenden Spy Sets haben unterschiedliche Größen und decken jeweils unterschiedliche Anteile der Pfade im Netzwerk ab.

  Gesucht ist die Minimalgröße dieser Spy Sets, für die nach Anwendung der Heuristik kein Spy Set verbleibt, das nicht 90% der Pfade abdeckt. Je kleiner die benötigte Set-Größe, desto besser.

  Für jedes der Netzwerke im betrachteten Bucket lässt sich so die für diese Konfiguration benötigte Spy-Set-Größe bestimmen. Das primäre Kriterium ist der Durchschnitt dieser Größen und sorgt für die Erreichung des genannten Optimierungsziels.

- *Sekundäres Kriterium*: Anwendungskomplexität der Guidelines

  Innerhalb einer Konfiguration können unterschiedliche Guidelines aktiv sein, die unterschiedlich schwer anzuwenden sind. Algorithmisch einfach anzuwendende Guidelines sollten entsprechend den schweren (**-CENT*-Guidelines) bevorzugt werden. Sind nur einfach anzuwendende Guidelines aktiv, sollten die Konfigurationen bevorzugt werden, die möglichst wenige Guidelines verwenden.

  Das sekundäre Kriterium sorgt so nicht für das Optimierungsziel, sondern für die einfache Anwendbarkeit der Heuristiken.

Für jede erstellte Konfiguration wird das primäre und sekundäre Kriterium berechnet. Dies ist der Abschluss der Evolutionsstufe 0. Es gibt jetzt eine Menge an bewerteten Heuristiken.

Anschließend werden die Konfigurationen von gut nach schlecht anhand des primären Kriteriums sortiert. Gibt es zwischen zwei Konfigurationen ein Patt, so entscheidet hier das sekundäre Kriterium über die Sortierung. Die 5 besten Konfigurationen werden im Konfigurationsspeicher abgelegt – der Rest wird verworfen.

Mutationsgenerator

Der Mutationsgenerator entnimmt die Konfigurationen der abgeschlossenen vorherigen Evolutionsstufe aus dem Konfigurationsspeicher und mutiert jede dieser Konfigurationen. Ziel ist es durch kleine, schrittweise Veränderungen sich einer optimalen Konfiguration anzunähern. Bei evolutionären Algorithmen ist es dabei üblich, sowohl bestehende Konfigurationen zu verändern als auch neue Konfigurationen hinzuzufügen, damit der Algorithmus auch völlig andere Daten bewerten kann. Letzteres verhindert, dass sich der Algorithmus an lokalen Optima festsetzen kann. Dazu geschehen im Mutationsgenerator zwei Schritte:

- *Schritt 1*: In genau einer Guideline jeder der 5 Konfigurationen wird ein Parameter zufällig neu gewählt. Alle anderen Parameter bleiben bestehen.
- *Schritt 2*: Der Mutationsgenerator generiert 2 neue Konfigurationen, entsprechend dem Vorgehen des Konfigurationsgenerators.

Der Mutationsgenerator gibt so insgesamt 7 Konfigurationen in die nächste Evaluationsphase. Gemeinsam mit den 5 unveränderten Konfigurationen besteht also Evolutionsstufe 1 aus 12 zu bewertenden Konfigurationen. Ab diesem Zeitpunkt entsteht ein Wechselspiel aus Mutationsgenerator und Evaluation, das sich über beliebig viele Evolutionsstufen fortsetzen lässt. Der Algorithmus bricht ab, wenn das Ergebnis des primären und sekundären Kriteriums gegen ein Optimum konvergiert, sich also über eine Menge an Evolutionsstufen nicht mehr verbessert. Dann sind die verbleibenden Konfigurationen die besten auffindbaren Konfigurationen.

3.2.5.4 Ergebnisse

Die im Folgenden vorgestellten Ergebnisse des evolutionären Algorithmus beziehen sich weiterhin auf die genannte Netzwerkklasse 10S10A. Ergebnisse für andere Netzwerkklassen finden sich im weiteren Verlauf des Kapitels.

Die Netzwerkklasse wurde hierbei in drei Buckets mit unterschiedlichen Pfadlängen unterteilt. Die Buckets betrachten die Pfadlängen 5-8, 9-12 und 13-24. Für jedes der Buckets wurde der evolutionäre Algorithmus einzeln eingesetzt, um jeweils die optimalen Konfigurationen zu ermitteln. Die durchschnittlichen Pfadlängen der Netzwerke der drei Buckets sind in Abbildung 25 bis 27 dargestellt.

Ergebnisevaluation mit 10.000.000 Spy Sets

Nachdem der evolutionäre Algorithmus für alle drei Buckets seine Arbeit getan hatte und Konfigurationen ermittelt hat, wurden diese zusätzlich noch einmal mit einer größeren Menge an Spy Sets überprüft. Während der evolutionäre Algorithmus mit 97.027 verschiedenen Spy Sets rechnete, um über die Evolutionsstufen hinweg eine angemessene Rechenzeit zu gewährleisten, erfolgte die Überprüfung mit fast 10.000.000 Spy Sets[35], deren Verteilung in Abbildung 28 dargestellt ist.

Im Folgenden werden exemplarisch die Ergebnisse des evolutionären Algorithmus und der Überprüfung der Ergebnisse des Buckets 13-24 der Netzklasse 10S10A genauer betrachtet und mit Daten belegt. Ergebnisse, die sich auf ein spezielles,

[35] Der Generierungsversuch von 10.000.000 Spy Sets resultiert in 9.456.680 Spy Sets (vgl. Abschnitt 3.2.5.1)

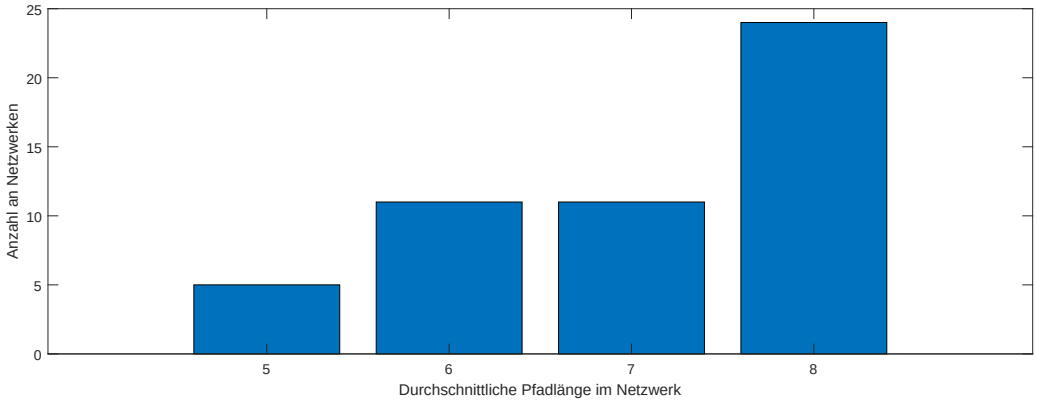


Abbildung 25: Durchschnittliche Pfadlängen der Netzwerke im Bucket 5-8 der Netzklasse 10S10A

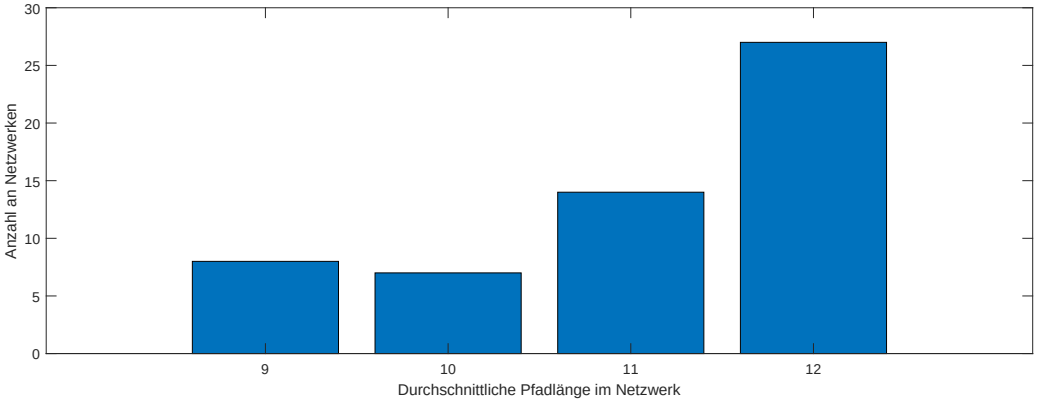


Abbildung 26: Durchschnittliche Pfadlängen der Netzwerke im Bucket 9-12 der Netzklasse 10S10A

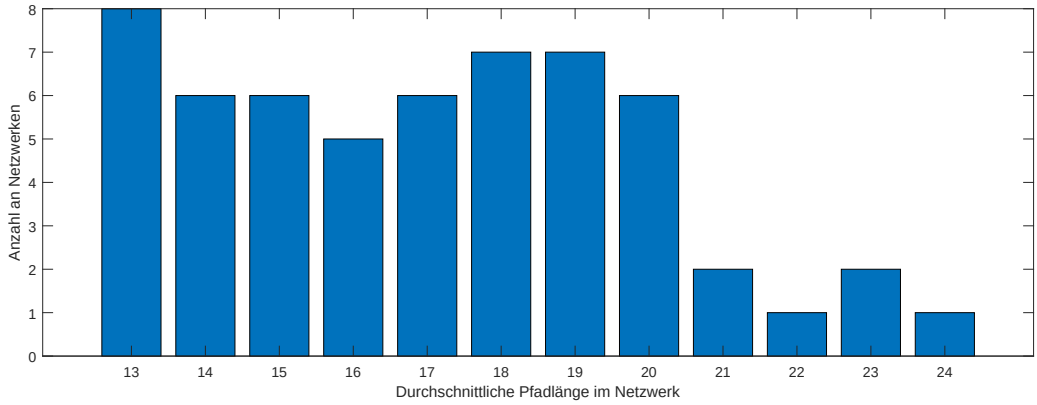


Abbildung 27: Durchschnittliche Pfadlängen der Netzwerke im Bucket 13-24 der Netzklasse 10S10A

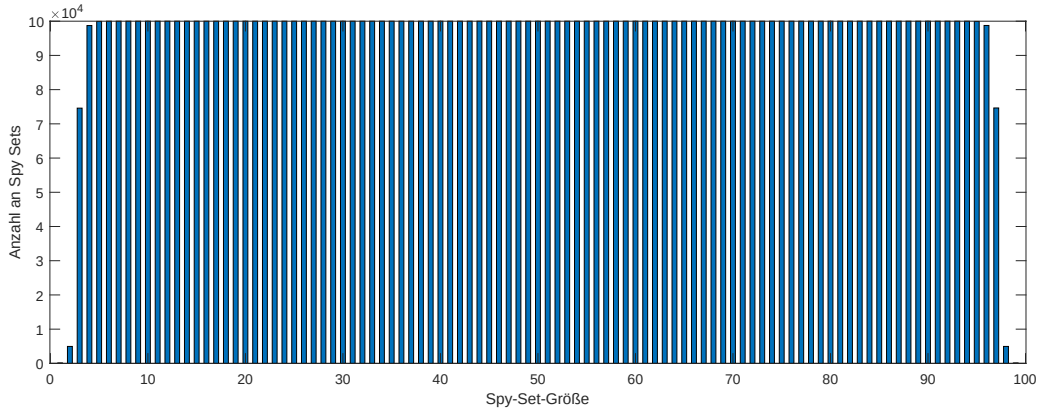


Abbildung 28: Verteilung der 9.456.680 Spy Sets nach ihrer Größe

exemplarisches Netzwerk beziehen, beziehen sich immer auf das Netzwerk aus Abbildung 23 (Seite 123), dass Teil des genannten Buckets ist.

Zur Referenz für das Ergebnis sind in Abbildung 29 die Ergebnisse der Abdeckung aller 9.456.680 Spy Sets – also ohne Anwendung einer Heuristik – für die Pfade des Netzwerkes aus Abbildung 23 dargestellt. Alle Spy Sets innerhalb dieses Netzes sind in Abbildung 29 in Box-Plots dargestellt. Auf der x-Achse wird hierbei die Größe des Spy Sets (Anzahl der enthaltenen Knoten) abgetragen während auf der y-Achse, die mit diesem Spy Set in diesem Netzwerk erzielte prozentuale Abdeckung der Pfade abgetragen wird. In jeder Spalte des Diagramms sind die mittleren 50% der Spy Sets in den blauen Boxen, während sich auf den Whisker (als Whisker bezeichnet man die gestrichelten, senkrechten Antennen der Boxen) etwa weitere 49,3 % der Spy Sets befinden. Ausreißer, etwa 0,3% der Spy Sets, sind mit einem roten Kreuz außerhalb der Boxen und Whisker markiert. Der Median ist mit einer roten Linie innerhalb der Box visualisiert.

Hier wurde noch keine Heuristik angewandt, die die Spy-Set-Auswahl einschränkt. Sets der Größe 1 erreichen hierbei sehr unterschiedliche Abdeckungen, wobei die Abdeckung im Median sehr gering ist. Wählt man ein Spy Set aus, dass aus nur einem isolierten Knoten besteht, so ist die Pfadabdeckung 0, da hierdurch keine Pfade verlaufen. Mit zunehmender Größe der Spy Sets nimmt auch die Abdeckung zu. Erst ab einer Spy-Set-Größe von 36 erreicht keines der Spy Sets mehr eine Abdeckung von weniger als 90%.

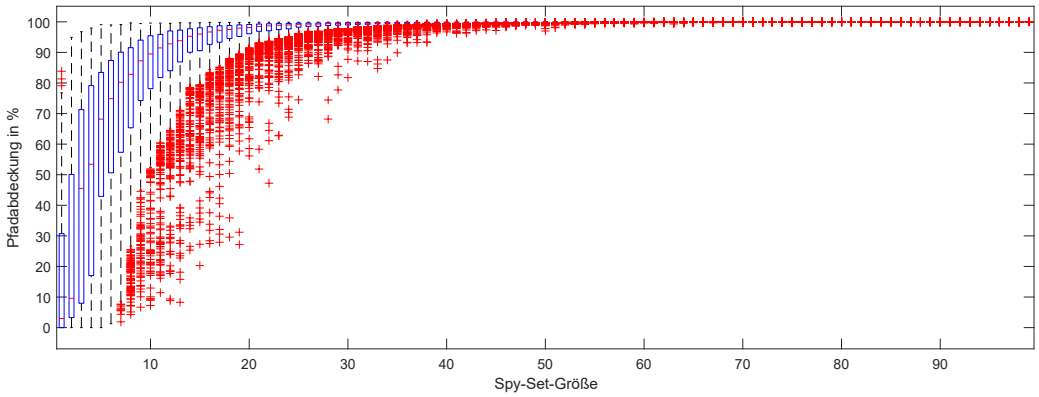


Abbildung 29: Abdeckungen der 9.456.680 Spy Sets ohne Nutzung von Heuristiken im Netzwerk aus Abbildung 23

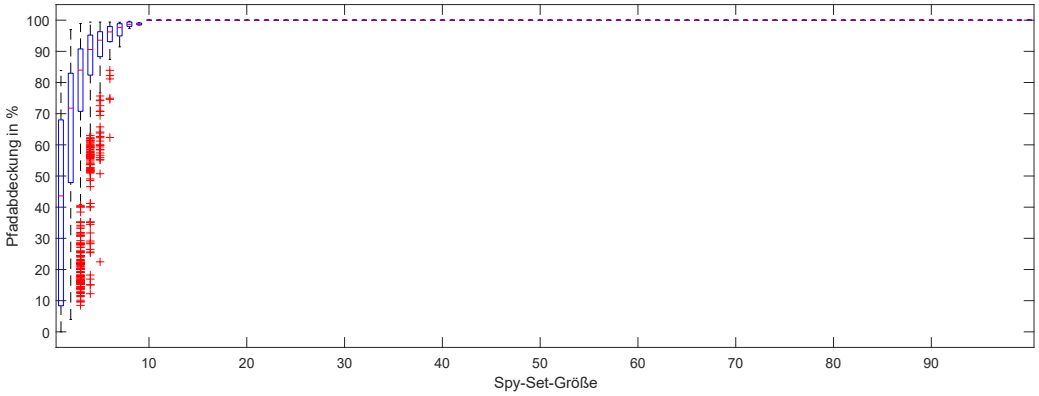


Abbildung 30: Abdeckungen der nach Anwendung der Heuristik [[DEGREE,2],[BLIND,1],[DEG-CENT,2]] verbleibenden Spy Sets im Netzwerk aus Abbildung 23

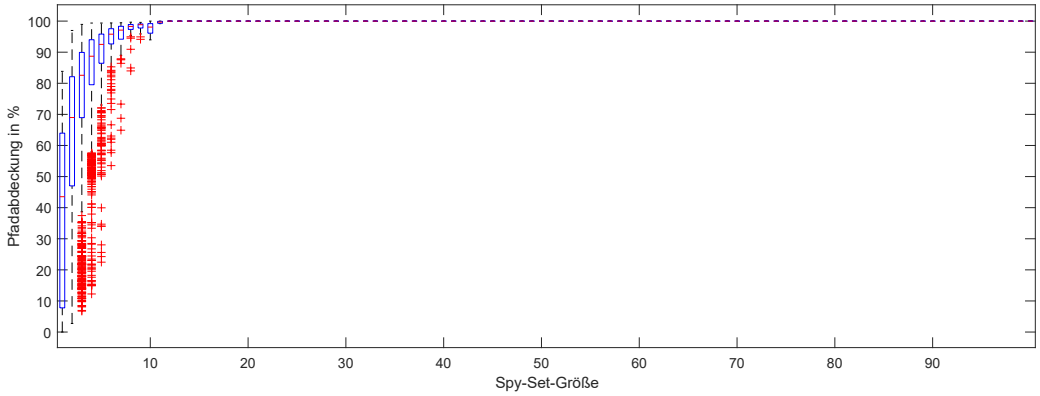


Abbildung 31: Abdeckungen der nach Anwendung der Heuristik [[DEGREE,2],[BLIND,1],[ROLES,3]] verbleibenden Spy Sets im Netzwerk aus Abbildung 23

Der evolutionäre Algorithmus hat für das genannte Bucket die beiden Konfigurationen [[DEGREE,2], [BLIND,1], [DEG-CENT,2]] und [[DEGREE,2], [BLIND,1], [ROLES,3]] als ähnlich gut ermittelt. Nutzt man diese Heuristiken, um die Menge der möglichen Spy Sets zu reduzieren, lässt sich das Ergebnis erheblich verbessern. Die Abdeckungen der nach der Anwendung der beiden Heuristiken verbleibenden Spy Sets sind in Abbildung 30 und 31 dargestellt. Wie in den Abbildungen zu sehen ist, wird in diesem Netzwerk nach Anwendung der Konfiguration [[DEGREE,2], [BLIND,1], [DEG-CENT,2]] nur noch eine Spy-Set-Größe von mindestens 7 statt 36 und nach der Anwendung der Konfiguration [[DEGREE,2], [BLIND,1], [ROLES,3]] nur noch eine Spy-Set-Größe von mindestens 9 statt 36 benötigt, um mindestens 90% der Pfade abzudecken. Die Ermittlung mehrerer ähnlich guter Heuristiken durch den Algorithmus bringt einen Flexibilitätsvorteil bei der Anwendung in der Praxis mit sich. Während die erste Heuristik leicht bessere Ergebnisse liefert, ist die zweite durch Verzicht auf die DEG-CENT-Guideline bzgl. Berechnungskomplexität einfacher anzuwenden.

Aussagekraft der Ergebnisse

Wie bereits erläutert, stellen die Konfigurationen Anforderungen an die Erstellung von Spy Sets. Die Konfigurationen führen also dazu, dass von den 9.456.680 generierten Spy Sets nur noch ein Bruchteil, der der Konfiguration entspricht, zulässig ist. Gleichzeitig wurden nicht alle möglichen, sondern nur dieser Ausschnitt der Gesamtmenge der Spy Sets untersucht. Daher ist nicht bewiesen, dass die Konfigurationen für alle möglichen Spy Sets gut performen. Nach Anwendung der Konfigurationen verbleiben in diesem Beispiel noch tausende von Spy Sets. Betrachtet man von diesen nur noch die mit den genannten Minimalgrößen 7 und 9, verbleiben noch 36 bzw. 30 Spy Sets. Dies würde bedeuten, dass wir 36 bzw. 30 Mal hintereinander zufällig aus der Menge aller Spy Sets ein Set gezogen haben, dass unserer Konfiguration entspricht und 90% aller Pfade abdeckt. Es befand sich kein Gegenbeispiel unter den gezogenen 36 bzw. 30 Spy Sets. Daher ist die Wahrscheinlichkeit, dass die Anzahl der Gegenbeispiele hoch ist, gering[36]. Somit ist davon auszugehen, dass die 9.456.680 Spy Sets repräsentativ für alle Spy Sets sind, sowie das die übrigbleibenden 30 bzw. 36 Spy Sets repräsentativ

[36] Eine genauere Wahrscheinlichkeitsbetrachtung ist in Abschnitt 3.2.5.6 dargestellt.

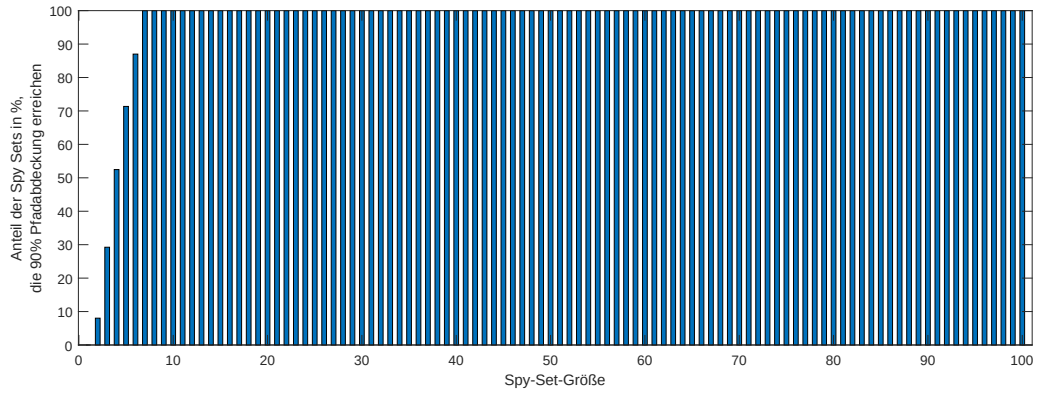


Abbildung 32: Anteile der Spy Sets aus Abbildung 30, die 90% Abdeckung erreichen

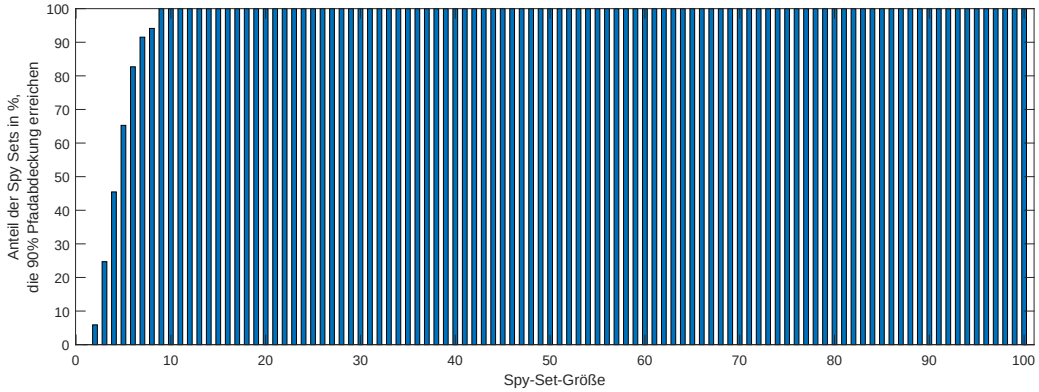


Abbildung 33: Anteile der Spy Sets aus Abbildung 31, die 90% Abdeckung erreichen

für alle durch die Konfiguration erlaubten Spy Sets sind. Zu beachten hierbei ist, dass auch hohe Anteile der Spy Sets von kleineren Größen als 7 und 9 bereits 90% Abdeckung erreichen können – ab den Größen von 7 und 9 erreichen jedoch alle verbleibenden Spy Sets die Abdeckung. Die Verteilung ist in Abbildung 32 und 33 dargestellt.

Gesamtergebnisse für das Bucket 13-24 der Netzklasse 10S10A

Zuvor wurde ein exemplarisches Netzwerk des Buckets 13-24 der Netzwerkklasse 10S10A betrachtet, um den Ursprung der einzelnen Werte zu erläutern. Im Folgenden soll jetzt das gesamte Bucket mit seinen Durchschnittswerten betrachtet werden, da die genannten durch den evolutionären Algorithmus gefundenen optimalen Konfigurationen für das gesamte Bucket gelten.

Die Durchschnittsanzahl der nach Anwendung der Konfigurationen verbleibenden Spy Sets über alle Netzwerke dieses Buckets ist 31,4 für die Konfiguration [[DEGREE,2],[BLIND,1],[DEG-CENT,2]] und 54,4 für die Konfiguration [[DEGREE,2],[BLIND,1],[ROLES,3]] und bewegt sich somit in einem ähnlichen Rahmen, wie dem zuvor beschriebenen Einzel-Netzwerk, bei dem die Werte bei 36 und 30 lagen. Die hohe Wahrscheinlichkeit, dass die Spy Sets repräsentativ sind, ist also auch für das gesamte Bucket gegeben. Die durchschnittliche Mindestgröße dieser Spy Sets beträgt 8,8 für die Konfiguration [[DEGREE,2],[BLIND,1],[DEG-CENT,2]] und 9,3 für die Konfiguration [[DEGREE,2],[BLIND,1],[ROLES,3]]. Der größte beobachtete Wert ist bei beiden Konfigurationen 11. Das bedeutet, unter den 57 Netzwerken im Bucket gibt es keines, in dem nach Anwendung einer dieser beiden Konfigurationen auf die 9.456.680 Spy Sets ein Spy Set verbleibt, dass die Mindestgröße 11 hat und weniger als 90% der Pfade abdeckt.

Die beiden Heuristiken lassen sich in Handlungsempfehlungen zur Platzierung von Spy-Knoten übersetzen. Für die Heuristik [[DEGREE,2],[BLIND,1],[ROLES,3]] sieht diese wie folgt aus:

Wenn in einem Netz der Klasse 10S10A des Buckets 13-24 ein Spy Set gebildet werden soll, und für dieses Set gilt, dass

- *... das Spy Set keine Sensoren oder Aktoren enthält*
- *... alle Knoten im Set mindestens zwei ausgehende Kanten haben*
- *... keiner der Knoten im Set in einer Sackgasse liegt*
- *... das Set aus mindestens 11 Knoten besteht*

..., dann deckt das Spy Set mit hoher Wahrscheinlichkeit mindestens 90% aller Pfade ab.

Ergebnisse weiterer Buckets und Netzwerkklassen

Bei der weiteren Untersuchung verschiedener Netzwerkklassen und ihrer Buckets sind wichtige Zusammenhänge klar geworden: Je weniger Aggregatoren im Netz sind, desto geringer ist die durchschnittliche Pfadlänge im Netz. Je geringer die durchschnittliche Pfadlänge im Netz, desto mehr Spy-Knoten werden tendenziell

benötigt, um die gewünschte Abdeckung zu erreichen. Dies geht teilweise so weit, dass der evolutionäre Algorithmus für beispielsweise die Netzwerkklasse 40S40A im Bucket 1-4 die benötigte Spy-Set-Größe von 79 berechnet. In allen Netzen gibt es jedoch eine triviale Lösung, die 100% aller Pfade abdeckt: Die Definition aller Aggregatoren als Spy. Durch die beschriebene Vernetzung ist sichergestellt, dass kein Pfad ohne Aggregator existiert. Die triviale Lösung lässt sich durch die Konfiguration [[ROLES,3]] und der Menge der Aggregatoren im Netz als Zielgröße des Spy Sets beschreiben.

Ist also das Ergebnis des evolutionären Algorithmus schlechter, als die triviale Lösung, so ist die triviale Lösung vorzuziehen. Der evolutionäre Algorithmus selbst findet diese Lösung in der Regel nicht, da es keine Lösungsklasse, sondern nur eine einzelne Lösung ist, die nur durch ein einziges Spy Set repräsentiert wird. Dieses ist in der Regel nicht in der Menge der 97.027 zufälligen Spy Sets enthalten. Der Algorithmus kann jedoch nur Konfigurationen finden, für die auch eine gewisse Menge an Spy Sets existiert.

Die Ergebnisse für die Netzklassen 10S10A, 30S30A und 40S40A sind in Tabelle 2 dargestellt. Ergebnisse des evolutionären Algorithmus, die schlechter als die Triviallösung waren, wurden durch diese ersetzt und sind entsprechend markiert. Die Anwendung dieser Regeln lässt sich je nach Netzwerk unterschiedlich realisieren. Bei der Planung eines CPS können Spy-Knoten strategisch platziert werden, wobei die Einhaltung der Regeln der jeweiligen Heuristik jederzeit zu überprüfen ist. Gleichzeitig gibt es je nach Bucket auch mehrere Möglichkeiten für die Konfiguration, die ähnliche Abdeckungen gewährleisten. Je nach Anwendungsfall kann hier ebenfalls eine möglichst einfach umsetzbare Heuristik gewählt werden. Bei Einhaltung der Heuristik bei der Platzierung der Spy-Knoten ist die Erreichung einer hohen Pfadabdeckung sehr wahrscheinlich.

3.2.5.5 Diskussion

Die in diesem Beitrag vorgestellten Heuristiken können dazu verwendet werden, Spies in CPS zu platzieren, um eine gewünschte Abdeckung mit hoher Wahrscheinlichkeit zu erreichen. Dabei wurden unterschiedliche Netzklassen untersucht, die verschiedene CPS-Typen repräsentieren. Diese unterscheiden sich durch einen unterschiedlichen Anteil von Sensoren, Aktoren und Aggregatoren. Die Verteilung

Netzklasse	Bucket (Pfadlängen-durchschnitte)	Bucketgröße (Netzanzahl)	Konfiguration	Max. benötigte Spy-Set-Größe	Durchschn. benötigte Spy-Set-Größe	Durchschn. verbleibende Spy Sets ≥ benötigter Größe
10S10A	5-8	51	[[ROLES,2]]	55	50,1	343,3
			[[ROLES,3]]	42	34,3	13,6
	9-12	56	[[ROLES,3]]	36	31,3	91,3
			[[ROLES,3], [DEGREE,1]]	27	22,3	17,0
	13-24	57	[[ROLES,3], [DEGREE,2], [BLIND,1]]	11	9,3	54,5
			[[DEGREE,2], [DEG-CENT,2], [BLIND,1]]	11	8,8	31,4
30S30A	1-8	120^{B}	$[[ROLES,3]]^{A}$	40^{C}	40^{C}	1^{D}
	9-12	53	[[ROLES,2]]	22	17,3	10,5
40S40A	1-99	110^{B}	$[[ROLES,3]]^{A}$	20^{C}	20^{C}	1^{D}

[A] Bei diesem Bucket dominierte die Triviallösung die vom evolutionären Algorithmus gefundene Konfiguration.

[B] Hier wurden mehrere Buckets, die die gleiche Konfiguration als Lösung haben, zusammengefasst.

[C] Bei der Triviallösung entspricht die Größe des Spy Sets der Anzahl der Aggregatoren.

[D] Die Triviallösung besteht aus nur einem Spy Set.

Tabelle 2: Überblick über die Ergebnisse für die verschiedenen Netzwerkklassen

bezieht sich hierbei auf die logische Trennung der genannten Rollen – ein Sensor und ein Aggregator können sich physisch jedoch einen Netzwerkknoten teilen. Dies führt insbesondere auch dazu, dass hohe Anteile an Aggregatoren in Netzwerken nicht ungewöhnlich sind. Unterstützt wird dieser Aspekt durch den in CPS weit verbreiteten Ansatz der Service-Mashups und Micro-Services, die ebenfalls zu vielen (kleinen) Aggregatoren führen. Die beiden genannten Aspekte führen dazu, dass hierbei längere Pfadlängen entstehen und im Ansatz auch betrachtet werden sollten.

In den ermittelten Heuristiken wird oft nur eine kleine Auswahl an Guidelines verwendet, obwohl alle Guidelines zur effizienteren Platzierung von Spies beitragen können. Die Guidelines sind allerdings nicht überschneidungsfrei, sodass beispielsweise ein Maß für Zentralität in einer Heuristik ausreicht. Dies kann sich aber je nach Netzklasse und Bucket unterscheiden.

Das zeigt aber auch, dass die Auswahl und Zusammenstellung von Spy Sets nicht intuitiv ist und dass der evolutionäre Algorithmus bessere Heuristiken erstellt als bei einem manuellen, intuitiven Vorgehen. Insbesondere fällt auf, dass der Mindestabstand zwischen zwei Knoten keinen Einfluss auf die gute Auswahl der Spy-Platzierungen hat. Dies wäre jedoch intuitiv, da nahe beieinander liegende Spy-Knoten ähnliche Pfade überwachen könnten und daher redundant sind. Dies ist jedoch nicht der Fall. Ebenfalls auffällig ist, dass teils die DEGREE-Guideline, teils die DEG-CENT-Guideline und teils beide Guidelines angewandt werden. Intuitiv lässt sich nicht nachvollziehen, wann welche dieser Zentralitätsguidelines zu besseren Ergebnissen führen. Zudem wählt der Algorithmus niemals die Betweeness oder Closeness Centrality. Auch dies lässt sich intuitiv nicht nachvollziehen. Dies gilt auch für die Konfiguration der ROLES-Guideline, die teilweise keine Sensoren und Aktoren zulässt, teilweise aber auch nur keine Aktoren. Dass diese Erkenntnisse nicht intuitiv nachvollziehbar sind, aber überprüfbar zu besseren Ergebnissen führen, lässt erkennen, dass ein Ingenieur auf die Heuristiken angewiesen ist, um gute Spy Sets zu erstellen.

Für die Anwendbarkeit der Heuristiken durch Ingenieure bei der Erstellung von Konfigurationen kann Tabelle 2 konsultiert werden. Hierbei kann sich der Ingenieur an einer möglichst ähnlichen dargestellten Netzklasse zum Finden einer angemessenen Konfiguration orientieren. Wenn die betrachteten CPS aber

signifikant andere Eigenschaften haben als die in Abschnitt 3.2.5.2 beschriebenen, sind die Ergebnisse möglicherweise nicht übertragbar.

Zudem bleibt zu bedenken, dass die Heuristiken auf eine Abdeckung von 90% der Pfade in den Netzwerken optimiert wurden. Für Werte über 90% steigt die Größe der Spy Sets nicht linear, sondern rapide an, sodass ein schlechtes Kosten-Nutzen-Verhältnis existiert. In den meisten Fällen reicht jedoch eine Abdeckung von 90% aus, da diese insbesondere auch lange Pfade enthält, für die eine manuelle Nachverfolgung besonders schwierig und aufwändig ist.

3.2.5.6 Aussagekraft der Ergebnisse

Die Ergebnisse lassen sich mit hoher Wahrscheinlichkeit auf andere Netzwerke gleicher Klassen und Buckets übertragen, da es sehr unwahrscheinlich ist, dass sich anderenfalls unter den 164 Netzwerken kein Gegenbeispiel finden lässt. Der evolutionäre Algorithmus hat mit Hilfe von 97.027 zufällig aus der Menge aller Spy Sets gezogenen Spy Sets die genannten Heuristiken entwickelt. Diese wurden mit Hilfe von 9.456.680 aus der Menge aller Spy Sets gezogenen Spy Sets überprüft. Die Menge der nach Anwendung der Heuristiken verbleibenden Spy Sets, die 90% der Pfade abdecken, wurden dementsprechend aus der Menge aller Spy Sets gezogen, ohne dass hierbei ein Gegenbeispiel gefunden wurde, das nicht 90% Abdeckung erreicht.

Die Berechnung der Wahrscheinlichkeit wird im Folgenden für das zuvor bereits intensiv betrachtete Bucket 13-24 der Netzklasse 10S10A durchgeführt. Dazu wird nur noch zwischen guten (Spy Sets, die 90% Abdeckung im Bucket erreichen) und schlechten Spy Sets (Spy Sets, die diese Abdeckung nicht erreichen) unterschieden.

Nach der Anwendung der Heuristik [[ROLES,3],[DEGREE,2],[BLIND,1]] auf die 57 Netze dieses Buckets verbleiben noch durchschnittlich 54,5 Spy Sets der Mindestgröße 11. Diese guten Spy Sets wurden zufällig aus der Gesamtmenge aller Spy Sets gezogen, da die Spy Sets zufällig generiert wurden. Es wurde kein schlechtes Spy Set, also kein Gegenbeispiel, in der nach Anwendung der Heuristik verbleibenden Menge gefunden.

Die Grundlage der folgenden Berechnung ist also, dass es gelungen ist, aus einer Gesamtmenge durchschnittlich etwa 54 Spy Sets zufällig zu ziehen unter

denen es kein einziges schlechtes Spy Set gab. Auf Grundlage eines linksseitigen Binomialtests [88] mit Poisson-Approximation[37] lässt sich die obere Grenze für den Anteil der Gegenbeispiele, also der schlechten Spy Sets, in der Gesamtmenge aller Spy Sets abschätzen.

Dies gelingt mit der Formel $e^{n-p_0} \leq \alpha$, wobei n die Anzahl der gezogenen guten Spy Sets, α das Signifikanzniveau und p_0 der gesuchte Anteil ist. Mit $n = 54$ und $\alpha = 0{,}1$ ergibt sich unter der Annahme von unabhängig identisch verteilten Ergebnissen $p_0 = 0{,}043$. Dieses Ergebnis lässt sich textuell wie folgt übersetzen:

> Mit 90 % Wahrscheinlichkeit ist der Anteil der schlechten Spy Sets in der Gesamtmenge aller Spy Sets 4,3 % oder geringer.

Die Wahrscheinlichkeit von 90 % ergibt sich hierbei aus dem gesetzten Signifikanzniveau $\alpha = 0{,}1$. Aus diesem Ergebnis lässt sich ableiten, dass es mit hoher Wahrscheinlichkeit gelingt ein Spy Set zu erstellen (aus der Gesamtmenge zu ziehen), das eine hohe Abdeckung erreicht, wenn Mindestgröße und Konfiguration eingehalten werden.

3.2.6 Kosten bei der Platzierung von Spy-Knoten

Für einzelne Knoten im Netzwerk kann es unterschiedlich teuer/aufwändig sein, diese zu einem Spy zu machen. Manche Knoten lassen sich schlicht gar nicht in einen Spy-Knoten umwandeln. Dies liegt insbesondere darin begründet, dass Spy-Knoten zwingend mit der zentralen IFM-Master Instanz verbunden sein müssen um direkt kommunizieren zu können. In CPS sind manche Knoten allerdings physisch nur mit benachbarten Knoten verbunden. Je nach Geographie oder infrastrukturellen Gegebenheiten kann sich der Preis für die Ausstattung eines Knotens als Spy sowie deren prinzipielle Möglichkeit deutlich unterscheiden. Die Ermittlung eines realistisch umsetzbaren Spy Sets kann den Ingenieur so vor große Herausforderungen stellen.

[37] Für die Poisson-Approximation gelten je nach Literatur andere Voraussetzungen. Eine übliche, in diesem Fall erfüllte, Voraussetzung ist $n > 10$ und $p_0 \leq 0{,}05$.

Vorteil bei den zuvor vorgestellten Heuristiken ist jedoch, dass diese nicht ein einzelnes Spy Set zur Umsetzung vorschlagen, sondern stattdessen ganze Klassen von Spy Sets erlauben, die den in der Heuristik enthaltenen Guidelines entsprechen. Unter diesen Spy Sets können sich nun unterschiedlich teuer umzusetzende Sets befinden. Um möglichst günstig ein Spy Set umzusetzen bieten sich verschiedene Optionen an:

- Bei der Implementierung der Spy-Knoten, die der Heuristik entsprechen, vermeidet der Ingenieur intuitiv die Auswahl teurer Spy Sets. Er stützt sich dabei auf seine Fachkenntnis des Netzes.
- Der Ingenieur erstellt nicht nur ein einzelnes Heuristik-konformes Spy Set, sondern direkt mehrere. Wird jeder Knoten im Netzwerk mit einem Kostenwert versehen, die die jeweilige Umwandlung quantifiziert, kann berechnet werden, welches Spy Set am günstigsten umzuwandeln ist.
- Der Ingenieur berechnet alle für die Heuristik möglichen Spy Sets, um anschließend über den Kostenwert der einzelnen Knoten das günstigste Spy Set zu berechnen.

Die Optionen beginnen also bei einem intuitiven Vorgehen und reichen bis zu einer vollständigen Berechnung. Sollen nur einzelne teure Knoten vermieden werden, reicht ein intuitives Vorgehen aus. Sollen die Kosten allerdings stark optimiert werden, muss eine Berechnung durchgeführt werden. Dies findet im Folgenden exemplarisch für das bereits zuvor genutzte Netzwerk (vgl. Abbildung 23, Seite 123) sowie die für das zugehörige Bucket ermittelte Heuristik ([[DEGREE,2],[DEGCENT,2],[BLIND,1]], vgl. Tabelle 2) statt. Bei diesem Netzwerk verbleiben 36 Spy Sets nach Anwendung der Heuristik, die die notwendige Mindestgröße zur Abdeckung von 90% der Pfade haben.

Zur Abbildung der Kosten für die Ausstattung der Knoten als Spy-Knoten wurde zufällig ein Vektor mit 100 Elementen generiert, bei dem jedes Element die Kosten für einen zugehörigen Knoten repräsentiert. Die Einträge sind hierbei ganze Zahlen von -1 bis 10, wobei 0 bis 10 die Kosten von gratis bis teuer angibt, während -1 diesen Knoten als nicht umwandelbar definiert. Die für dieses Beispiel generierten Knotenpreise sind in Abbildung 34 dargestellt. Knoten 1 hat hierbei einen Preis von 6, Knoten 2 einen Preis von 2, Knoten 22 kann kein Spy sein, und so weiter. Der Preis für die Umsetzung eines Spy Sets ist die Summe der

$$[6, 2, 6, 7, 10, 1, 8, 9, 4, 5, 1, 7, 10, 2, 9, 3, 10, 1, 7, 9, 0, -1, 4, 3, 9, -1, 0, 4, 9, -1, 9, 7, 10,$$
$$3, 9, -1, 10, 8, 3, 7, 10, 9, 2, 6, 4, 1, 3, 9, 9, 10, 7, 6, 3, 9, 10, 5, 4, 3, 10, 9, 9, 9, 10, 8, 3, 1, 8,$$
$$-1, 5, 6, 7, 5, 7, 10, -1, 7, 7, 1, 3, 5, 5, 1, 6, 9, 0, 6, 7, 7, 9, 4, 6, 4, 3, 2, 7, 1, 6, 2, 3, 0]$$

Abbildung 34: Generierte Knotenpreise

Preise für die Umwandlung der darin enthaltenen Knoten. Enthält ein Spy Set einen nicht umwandelbaren Knoten (Preis = -1), so ist auch der Gesamtpreis für die Umwandlung -1, da dieses Spy Set nicht umsetzbar ist.

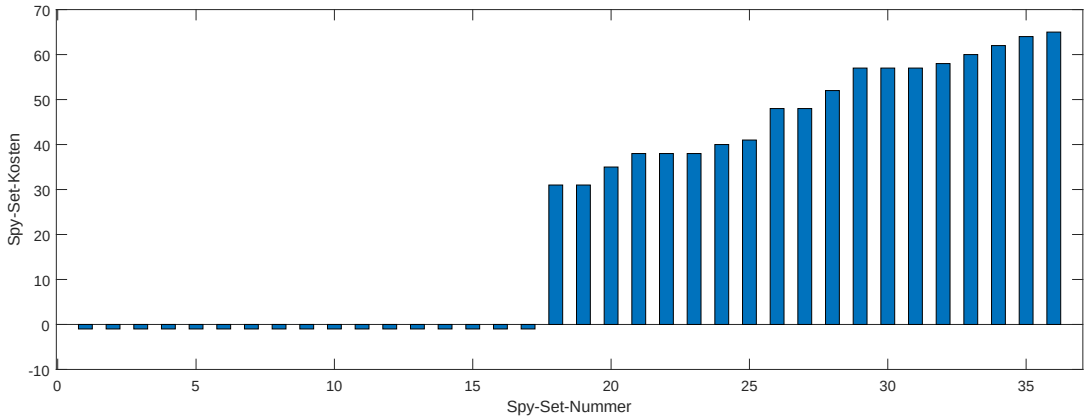


Abbildung 35: Gesamtpreise der unterschiedlichen Spy Sets

Durch das beschriebene Verfahren wurde der Gesamtpreis der Umsetzung für jedes der 36 Spy Sets berechnet. Das Ergebnis ist in Abbildung 35 dargestellt. Insgesamt gibt es 6 Knoten, die nicht zu Spies werden können (vgl. Abbildung 34). Diese 6 Knoten führen dazu, dass 17 der 36 Spy Sets nicht umsetzbar sind, da diese einen der nicht umwandelbaren Knoten enthalten. Große Mengen an Knoten, die kein Spy werden können, führen also zu einer erheblichen Reduktion der möglichen Spy Sets. Aus den verbleibenden Spy Sets, deren Umsetzungskosten nun bekannt sind, kann ein Spy Set mit möglichst geringen Kosten ausgewählt werden.

3.3 Information Dependency Modeling [89][38] [90][39]

Neben dem Konzept zur Platzierung von Spy-Knoten ist das Information Dependency Modeling ein weiteres Konzept, das zur Anwendbarkeit des IFM in CPS beiträgt. Hierbei handelt es sich um eine Erweiterung des IFM, welche die Modellierung von Abhängigkeitsstrukturen zwischen Informationen zulässt. Das Konzept wird im Folgenden motiviert und erläutert.

3.3.1 Problemmotivation

Verschiedene Produkte unterschiedlicher Hersteller werden mit Hilfe von verschiedenen Strategien, Engineering-Prozessen und Werkzeugen entwickelt. Diese Produkte müssen in bestehende Lösungen integriert werden können. Dies gilt auch für Komponenten in CPS. [91, S. 30]

Aggregatoren verwenden eingehende Daten zur Bestimmung ausgehender Daten. Nur diese verarbeitenden Knoten wissen, welche Daten für die Erzeugung einer ausgehenden Information verwendet wurden. Daher kann eine Abhängigkeit nicht automatisch ohne Hilfe des Knotens aufgezeichnet werden. Wenn also ein Knoten im Netzwerk das IFM-Protokoll nicht unterstützt, können Abhängigkeiten, die durch die Verarbeitung von Daten in diesem Knoten entstehen, nicht vom IFM erfasst werden.

In der Praxis ist es nicht auszuschließen, dass einige Knoten nicht dem Protokoll entsprechen. Das ist insbesondere darauf zurückzuführen, dass Komponenten/Knoten in CPS wiederverwendet werden. Dies geschieht insbesondere in Form von

[38] Dieser Beitrag wurde veröffentlicht in:
S. Gries, M. Hesenius und V. Gruhn. „Embedding Non-Compliant Nodes into the Information Flow Monitor by Dependency Modeling“. In: *Proceedings - 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS 2018)*. 2018, S. 1541–1542. DOI: 10.1109/ICDCS.2018.00163

[39] Dieser Beitrag wurde veröffentlicht in:
S. Gries, J. Ollesch und V. Gruhn. „Modeling Semantic Dependencies to Allow Flow Monitoring in Networks with Black-Box Nodes“. In: *Proceedings of the 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS'19)*. SEsCPS '19. IEEE Press, 2019, S. 14–17. DOI: 10.1109/SEsCPS.2019.00010

Commercial off-the-shelf (COTS)-Komponenten. Hierbei handelt es sich um Komponenten, die nicht individuell für einen Anwendungsfall entwickelt wurden, sondern in der Regel extern eingekauft werden und eine Standardkomponente zur Lösung eines Problems darstellen. Dies wird insbesondere bei Sensoren und Aktoren genutzt, da diese einen Hardwarefokus haben und nicht für jeden Anwendungsfall individuell entwickelt werden. Zudem lassen sich Sensoren und Aktoren einfach auf verschiedene Anwendungsszenarien anwenden, während die Aggregatoren mit Softwarefokus oft individuell für einen Anwendungsfall entwickelt werden.

Diese COTS-Komponenten können oft nur in geringem Umfang oder gar nicht angepasst werden. Diese Knoten werden im Folgenden als Black-Box-Knoten bezeichnet. Black-Box bedeutet in diesem Zusammenhang, dass das IFM-Protokoll vom Knoten nicht implementiert werden kann. Wissen über die Geschäftslogik dieser Knoten liegt jedoch vor.

Hierdurch ergeben sich zusammenfassend folgende zu betrachtende Problemstellungen:

- Die Software von Black-Box-Knoten ist nicht veränderbar. Daher ist es nicht möglich, die Software des Knotens so zu modifizieren, dass der Knoten das IFM-Protokoll unterstützt.
- Ohne das Mitwirken des Knotens bei der Aufzeichnung von Abhängigkeiten ist es unmöglich von außen Abhängigkeiten zwischen ein- und ausgehenden Informationen zu erkennen.

Für Sensoren und Aktoren lässt sich dieses Problem trivial lösen:

- Hinter jedem **Sensor**, der ein Black-Box-Knoten ist, wird ein Aggregator ins Netz eingefügt. Der Aggregator erhält als einziger die Sensordaten und vertritt den Sensor im Netzwerk. Alle anderen Knoten im Netz beziehen ihre Daten von diesem Aggregator statt direkt vom Sensor. Der Aggregator kann die IFM-Kompatibilität problemlos herstellen, da Sensorwerte keine Abhängigkeiten haben, die der Aggregator wissen müsste.
- Vor jedem **Aktor**, der ein Black-Box-Knoten ist, wird ein Aggregator ins Netz eingefügt. Der Aggregator vertritt den Aktor im Netzwerk und reicht

von empfangenen IFM-konformen Nachrichten nur die Nutzdaten an den Aktor weiter. Da am Aktor keine neuen Abhängigkeiten entstehen, kann der Aggregator alle IFM-Aufgaben für den Aktor übernehmen.

Ist die Software der Black-Box-Knoten unveränderbar, ist dies also nur für Aggregatoren ein Problem. Von diesen muss entsprechend der Algorithmus, mit dem aus eingehenden ausgehende Informationen erzeugt werden, bekannt sein. Anderenfalls ist die Benennung von Abhängigkeiten von außen unmöglich.

Ziel des Information Dependency Modeling ist es, eine durchgehende Nachverfolgbarkeit von Abhängigkeiten über Black-Box-Aggregatoren hinweg zu ermöglichen. Würde ein Ast eines Abhängigkeitsbaums ohne das Dependency Modeling an einem Black-Box-Knoten enden, soll dieser durch Modellierung überbrückt werden. Es soll dementsprechend das Verhalten der Businesslogik bzgl. der Verbindung zwischen Ein- und Ausgaben durch Modellierung beschrieben werden. Hinter dem Black-Box-Knoten können dann im Visualizer weitere aufgezeichnete Abhängigkeiten dargestellt werden, während der Ast in der Visualisierung sonst beim Black-Box-Knoten enden würde. Zu beachten hierbei ist, dass das Dependency Modeling nur in einem sehr begrenzten Maße eingesetzt werden kann, da es die eigentliche Funktion – das Sammeln von Abhängigkeiten – für die modellierten Knoten nicht in vollem Umfang ersetzen kann. Einschränkungen und Einsatzgebiete werden in Abschnitt 3.3.5 diskutiert.

3.3.1.1 Beispielszenarien

Die folgenden Beispiele beziehen sich auf das in Abbildung 36 dargestellte Netzwerk und illustrieren die Probleme mit dem Umgang mit Black-Box-Knoten in IFM-Netzwerken.

3.3.1.1.1 Beispielszenario A

In diesem Beispiel agieren Knoten 1 und 2 (vgl. Abbildung 36) als Datenquellen und senden Informationen an Knoten 3. Dieser aggregiert die empfangenen Informationen und sendet ein Ergebnis an 4. Dieser wiederum sendet ein Kalkulationsergebnis an 5. Gleichzeitig ist 4 auch Spy und sendet Abhängigkeitsinformationen an den IFM-Master. Knoten 5 ist ein Black-Box-Knoten, der Operationen auf

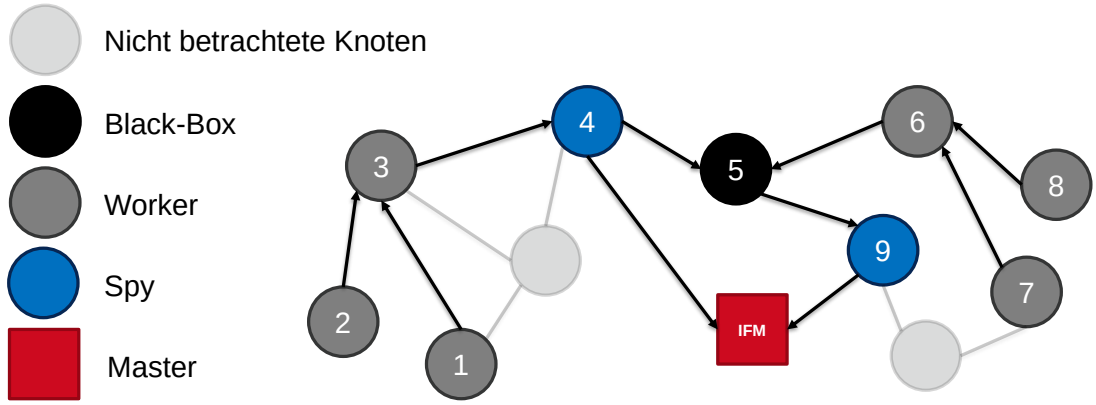


Abbildung 36: Beispielnetzwerk mit Black-Box-Knoten

empfangenen und bereits vorliegenden Informationen ausführt. In diesem Szenario hat 5 die Aufgabe, den Mittelwert der letzten 10 Werte für die von 4 empfangenen Informationen zu berechnen. Diese Information über die Businesslogik muss dem Modellierer bekannt sein. 5 verwendet nur die Daten von 4, obwohl 5 auch Zugriff auf Informationen von 6 hat. Der Mittelwert wird dann an Knoten 9 weitergeleitet. In diesen versendeten Informationen sind keine Abhängigkeitsinformationen enthalten, da 5 das IFM-Protokoll nicht unterstützt. Es ist jetzt nicht mehr nachvollziehbar, zu welchen Informationen die von 9 empfangenen Informationen Abhängigkeiten haben. Es ist nicht entscheidbar, ob und welche Informationen von 4 oder 6 von 5 benutzt wurden, um ausgehende Informationen zu erzeugen. 9 ist ein Spy, hat aber keine Abhängigkeitsinformationen vom Black-Box-Knoten 5 erhalten. 9 kann nicht wissen, welche Daten von welchem Knoten verwendet wurden, um die empfangenen Daten zu berechnen.

3.3.1.1.2 Beispielszenario B

Dieses Beispiel ist Beispiel A sehr ähnlich. Unterschied ist, dass Black-Box-Knoten 5 zusätzlich zu den Informationen von 4 auch Informationen von Knoten 6 erhält, wobei die Knoten 6-8 eine ähnliche Funktion wie die Knoten 1-3 erfüllen. So wird der Knoten 5 nun von zwei Seiten mit Informationen versorgt. Seine Funktion ist immer noch die Aggregation der letzten 10 empfangenen Werte, jetzt aber von den Knoten 4 und 6. Das Problem ist nun, dass die Existenz der Knoten 6, 7 und 8 dem IFM nicht einmal bekannt ist, da es in diesem Teil des Netzwerks keinen Spy gibt. Keine einzige Information, die mit 6-8 berechnet wurde, ist für das IFM sichtbar.

3.3.2 Lösungsansatz

Die Idee des Information Dependency Modeling ist es, das Verhalten eines Knotens nicht aufzuzeichnen, sondern zu modellieren. Bei der Aufzeichnung der Abhängigkeiten teilt ein Knoten, der Daten versendet, mit, welche Abhängigkeiten zu eingehenden Daten bestehen. Es findet also durch Mitwirkung des Knotens eine Verknüpfung von ein- und ausgehenden Daten statt. Da ein Black-Box-Knoten nicht an der Aufzeichnung von Abhängigkeiten mitwirkt, kann die Verknüpfung dieser Informationen nicht aufgezeichnet werden, aber modelliert werden. Voraussetzung hierfür ist genaue Kenntnis über die Funktionsweise der Businesslogik des Black-Box-Knotens. Des Weiteren muss bekannt sein, welche Nachbarn des Black-Box-Knotens mit diesem interagieren, damit dessen Eingaben mit den Ausgaben verknüpft werden können.

Mit Hilfe der Modellierung soll so ein Zusammenhang zwischen ein- und ausgehenden Informationen des Black-Box-Knotens hergestellt werden. Dies muss an einem zentralen Ort geschehen, der die im Netzwerk versendeten Nachrichten kennt. Hierzu eignet sich der IFM-Master. Um sicherzustellen, dass alle eingehenden und ausgehenden Nachrichten des Black-Box-Knotens im Master bekannt sind, sollten unmittelbar vor und nach dem Black-Box-Knoten zusätzliche Spy-Knoten platziert werden. Ein ganzes Cluster von Black-Box-Knoten sollte hierbei logisch als ein Black-Box-Knoten betrachtet werden. So kann beobachtet werden, welche Informationen vom Black-Box-Knoten konsumiert und erzeugt werden.

Für alle in den Black-Box-Knoten eingehenden Informationen sind die Abhängigkeiten durch das IFM-Protokoll bekannt. Zu jeder ausgehenden Information müssen nun die Abhängigkeiten zu den eingehenden modelliert werden. Hierzu kann auf die Datenbank des IFM-Masters zurückgegriffen werden. Durch Modellierung wird beschrieben, zu welcher definierten ausgehenden Information, die durch die Spy-Knoten auch mit einer ID versehen wurde und in der Master-Datenbank erfasst ist, Abhängigkeiten zu anderen Informationen in der Master-Datenbank existieren.

3.3.3 Modellierung und technische Umsetzung

Zu jeder ausgehenden Information eines Black-Box-Knotens wird so eine Teilmenge der Datenbank des IFM-Masters als abhängig definiert. Dies geschieht auf Grundlage der Kenntnis der Funktionsweise des Black-Box-Knotens. Um Teilmengen an Daten aus Datenbanken auszuwählen, eignen sich Data Query Languages (DQLs). Letztendlich sind alle DQLs, die in Datenbankservern zur Selektierung von Daten eingesetzt werden, recht ähnlich und austauschbar. SQL- und MongoDB-Queries sind beispielsweise recht einfach ineinander überführbar [92]. Da sich für die Master-Datenbank dokumentenorientierte Datenbanken besonders eigenen (vgl. Abschnitt 3.1.6.1), wird die Modellierung im Folgenden anhand von MongoDB Queries erläutert.

Durch die Modellierung wird immer genau einer Information eine Menge an Informationen zugeordnet, zu denen eine Abhängigkeit besteht. Basis der Zuordnung ist eine durch den Black-Box-Knoten erzeugte Information mit ihren Datenfeldern (vgl. Abschnitt 3.1.3, Abschnitt 3.1.6). Die Information, zu der Abhängigkeiten definiert werden sollen, wird im Folgenden als *givenMessage* bezeichnet und ist eine ausgehende Nachricht des betrachteten Black-Box-Knotens. Sie repräsentiert alle ausgehenden Nachrichten des Knotens. Die Abhängigkeit einer Information wird hierbei durch die Selektion von anderen Informationen der IFM-Master-Datenbank beschrieben. Innerhalb der Selektion ist *givenMessage* eine Variable, die als Filter eingesetzt werden kann.

3.3.3.1 Rekonstruktion der Abhängigkeiten im Visualizer

Bei der Anzeige des Abhängigkeitsbaums nutzt der IFM-Visualizer die vorliegenden Modellierungen, um Black-Box-Knoten zu überbrücken, sollten Nachrichten dieser Knoten Teil des Abhängigkeitsbaums sein. Der Algorithmus der Visualisierung läuft hierbei wie folgt ab:

1. Gibt ein Benutzer eine zu untersuchende Information in den IFM-Visualizer ein, kann dieser anhand von aufgezeichneten vorliegenden Informationen den Abhängigkeitsbaum visualisieren. Die Äste des Abhängigkeitsbaums enden an Nachrichten, die selbst keine aufgezeichneten Abhängigkeiten mehr haben. Eine Teilmenge dieser Nachrichten stammt von Black-Box-Knoten.

Diese Nachrichten können Abhängigkeiten haben, sie wurden jedoch nicht aufgezeichnet.

2. Liegt für den erzeugenden Knoten einer solchen Nachricht eine Modellierung vor, setzt der Visualizer diese Nachricht als *givenMessage* und führt den zugehörigen DQL-Query aus. Der Query läuft auf der Master-Datenbank und liefert eine Menge an Nachrichten zurück – diese Nachrichten sind als Abhängigkeit zur *givenMessage* definiert worden und werden als Blätter im Abhängigkeitsbaum an die *givenMessage* angefügt.

3. Die angefügten Blätter können wiederum selbst aufgezeichnete Abhängigkeiten haben. An dieser Stelle kann der Visualizer normal fortgesetzt werden und die aufgezeichneten Abhängigkeiten der Blätter normal angezeigt werden.

Die Modellierung in Schritt 2 ermöglicht so die Überbrückung des Black-Box-Knotens und die durchgehende Visualisierung von aufgezeichneten Abhängigkeiten (vgl. Abbildung 57, Seite 193) sowohl vor als auch nach dem Black-Box-Knoten.

3.3.4 Beispiele

Im Folgenden werden zwei Beispiele zur Modellierung von Abhängigkeiten gegeben, um die vorausgehenden theoretischen Erläuterungen zum Umgang mit dem Information Dependency Modeling zu veranschaulichen. Die Beispiele beziehen sich auf die beiden Szenarien aus Abschnitt 3.3.1.1.

3.3.4.1 Modellierungsbeispiel A

Alle Informationen, die in diesem Beispiel vom Black-Box-Knoten 5 genutzt werden, stammen von Knoten 4 und sind dem IFM-Master bereits bekannt. Da Knoten 4 ein Spy ist, meldet dieser auch alle ausgehenden Daten direkt an den Master. Um die Abhängigkeiten der aus 5 ausgehenden Informationen zu den eingehenden Informationen, die 4 versendet hat, zu modellieren, wird ein MongoDB Query verwendet.

```
1 {
2   find:"history",
3   filter:{
4     "date":{$lte:givenMessage.date},
5     "source":4
6   },
7   sort:{"date":-1},
8   limit:10
9 }
```

Abbildung 37: Abhängigkeitsdefinition mit MongoDB zu Beispielszenario A

Da in diesem Beispiel Knoten 5 immer die letzten 10 von Knoten 4 erhaltenen Informationen aggregiert (vgl. Abschnitt 3.3.1.1.1), wählt der Query genau diese Informationen zu einem gegebenen *givenMessage* aus. Das *givenMessage* ist die Information, die von 5 emittiert wurde, zu der die Abhängigkeiten modelliert werden sollen. Zuerst werden hierzu alle Informationen ausgewählt, die älter als die gegebene Information sind, (vgl. Abbildung 37, Z. 4) und von Knoten 4 erzeugt wurden (vgl. Abbildung 37, Z. 5). Alle von Knoten 4 versendeten Informationen gelangen auch zu Knoten 5 (vgl. Abbildung 36). Danach werden die letzten 10 vor Erzeugung des *givenMessage* empfangenen Informationen ausgewählt (vgl. Abbildung 37, Z. 7-8). Insgesamt returniert der Query so die in Abschnitt 3.3.1.1.1 als abhängig beschriebenen Informationen zu einer gegebenen Information.

3.3.4.2 Modellierungsbeispiel B

In diesem Beispiel verarbeitet Knoten 5 zusätzlich zu den Informationen von 4 auch Informationen von 6 (vgl. Abbildung 36). Der komplette Informationsfluss zwischen den Knoten 6, 7 und 8 wird hierbei von keinem Spy erfasst. Die von 6 an 5 gesendeten Nachrichten sind daher dem IFM-Master nicht bekannt und nicht in dessen Datenbank enthalten. Hierzu gibt es zwei Möglichkeiten, mit dieser Problematik umzugehen.

Platzierung zusätzlicher Spy-Knoten

Damit die Abhängigkeiten zwischen den Informationen, die der Black-Box-Knoten emittiert und den Informationen, die er konsumiert, modelliert werden können, müssen eingehende und ausgehende Informationen dem IFM-Master bekannt sein. Einfachste Möglichkeit dies zu gewährleis-

```
{
  find: "history",
  filter:{
    "date":{$lte:givenMessage.date},
    $or:[
      {"source":4},
      {"source":6}
    ]
  }
  sort: {"date": -1},
  limit: 10
}
```

Abbildung 38: Abhängigkeitsdefinition mit MongoDB zu Beispielszenario B [90]

ten ist die Umwandlung von Knoten 6 in einen Spy-Knoten. Anschließend können die aus der beschriebenen Logik (vgl. Abschnitt 3.3.1.1.2) resultierenden Abhängigkeiten modelliert werden. Dies geschieht analog zum vorherigen Beispiel, mit dem Unterschied, dass Abhängigkeiten jetzt nicht mehr nur zu Informationen von 4 existieren können, sondern zu Informationen von 4 und 6 (vgl. Abbildung 38).

Erzeugung von Platzhalter-Informationen

Nicht immer besteht die Möglichkeit, Spy-Knoten beliebig zu platzieren. Daher ist im Folgenden eine Alternative skizziert, wie Abhängigkeiten auch ohne zusätzlichen Spy-Knoten in diesem Szenario beschrieben werden können. In diesem Fall kann in die Master-Datenbank eine History-Information eingefügt werden (vgl. Abbildung 39), die alle von 6 versendeten Nachrichten repräsentiert. Dies geschieht bei Planung des Netzes durch den Ingenieur. Dieser Platzhalter wird alle Nachrichten, die von 6 versendet werden, repräsentieren. Anschließend kann die Abhängigkeit zu diesem Platzhalter und den Informationen von 4 modelliert werden. Bei der Auswertung ist die Abhängigkeit zu Informationen von 6 somit nicht mehr versteckt, muss aber bei Bedarf manuell untersucht werden. Die weiteren Abhängigkeiten von den Informationen von 6 werden hierdurch nicht erfasst. Die Platzierung eines weiteren Spy-Knoten ist somit in jedem Fall vorzuziehen. Einziger Vorteil bei diesem Vorgehen ist, dass im IFM-Visualizer nun auch Knoten 6 sichtbar ist, der zuvor komplett fehlte.

```
1  {
2    insert: "history",
3    documents: [{source: 6}]
4  }
```

Abbildung 39: Einfügen einer Platzhalter-Information in die Master-Datenbank

3.3.5 Diskussion

Das Ziel des Information Dependency Modeling ist es, dass bei der Visualisierung von Abhängigkeiten im IFM-Visualizer Black-Box-Knoten nicht mehr dazu führen, dass Abhängigkeitsbäume an diesen Knoten enden, sondern auch aufgezeichnete Abhängigkeiten hinter diesen Knoten mit in den Baum aufgenommen werden können. Das Dependency Modeling deckt also keine echten Abhängigkeiten zwischen Informationen auf, sondern verknüpft nur andere aufgezeichnete Abhängigkeiten, damit diese gemeinsam visualisiert werden können. Dieses Problem bezieht sich zudem nur auf Aggregatoren – für Sensoren und Aktoren gibt es bessere Lösungen (vgl. Abschnitt 3.3.1).

Damit das Dependency Modeling zu einer durchgehenden Visualisierung sinnvoll beitragen kann, müssen folgende Eigenschaften bedacht werden:

- Es dürfen im Netz insgesamt nur **wenige Black-Box-Aggregatoren** vorhanden sein. Anderenfalls verknüpft der Master durch viele Modellierungen wenige aufgezeichnete Abhängigkeiten, die dann keine Aussagekraft mehr haben.

- Das Dependency Modeling benötigt an verschiedenen Stellen im Netzwerk **zusätzliche Spy-Knoten**. Da Spy-Knoten einerseits teuer sein können und andererseits auch nicht immer beliebig platziert werden können, kann dies ein Problem darstellen.

- Um Abhängigkeitsstrukturen für einen Black-Box-Knoten modellieren zu können, müssen **präzise Informationen zur Businesslogik vorliegen**. Anderenfalls lassen sich die darin entstehenden Abhängigkeiten nicht modellieren. Zeitgleich kann im Dependency Modelling nicht auf die Inhalte der Nachrichten zugegriffen werden – diese können jedoch neben den Metadaten auch Einfluss auf die Auswahl der Informationen haben, zu denen Abhän-

gigkeiten erstellt werden. Die Modellierung der Businesslogik führt ebenfalls zu einer hohen Kopplung zwischen Modellierung und Black-Box-Knoten.

- Die Modellierung von Abhängigkeiten führt zu einer **Scheingenauigkeit** des IFM-Visualizers. Dem Anwender sollte bewusst sein, dass der Visualizer an Black-Box-Knoten nur die Modellierung wiedergibt, die möglicherweise vom realen Verhalten des CPS abweicht.

Die genannten Anforderungen und Einschränkungen führen dazu, dass das Dependency Modeling nicht immer anwendbar ist. Ist es jedoch anwendbar, kann es die durchgehende Verfolgung von Abhängigkeiten durch den IFM-Visualizer ermöglichen. Das Dependency Modeling sollte dabei möglichst sparsam eingesetzt werden – dies wird durch alternative Methoden bei Sensoren und Aktoren ermöglicht.

3.4 IFM-Blockchain-Konzept für vertrauenslose Netzwerke

Bei der Konzeption des IFM wurde davon ausgegangen, dass alle an der Informationsbeschaffung beteiligten Akteure gutmütig sind und den Prozess nicht stören oder manipulieren wollen. In heterogenen, von mehreren Parteien betriebenen Netzwerken, ist davon auszugehen, dass dies in der Realität nicht immer der Fall ist. Böswillige Akteure, die absichtlich fehlerhafte Daten im Netzwerk verbreiten, wollen möglicherweise ihre Aktivitäten verschleiern und die dem IFM zur Verfügung gestellten Abhängigkeitsinformationen manipulieren. Verschiedene Parteien sind hierbei unterschiedliche Interessensgruppen, beispielsweise Firmen. In CPS mit vielen verschiedenen Parteien können auch bösartige Akteure Parteien bilden, die durch bösartiges Verhalten eigene Interessen vertreten.

In Abschnitt 4.1.1 ist ein Beispiel für ein CPS dargestellt, das auch Smart-Home- und Smart-City-Komponenten anhand von gemessenen Sensordaten steuert. In solchen Netzwerken gibt es Möglichkeiten für Knoten, verfälschte oder manipulierte Daten zu versenden. Durch diese Weitergabe von fehlerhaften Daten lassen sich so beispielsweise Ampelschaltungen manipulieren. Ebenfalls könnten im gezeigten

Beispiel durch die Verbreitung von manipulierten Daten Fenster geöffnet werden. Hierdurch lassen sich physische Angriffe auf die Teilnehmer des CPS realisieren.

Daher besteht im Grunde kein vollständiges Vertrauen zwischen den verschiedenen Parteien im Netzwerk, so dass eine einzige gemeinsame IFM-Instanz in diesem Fall nicht erwünscht sein kann. Einzelne Parteien wollen ihre eigenen IFM-Instanzen für ihre Knoten betreiben, denen sie vertrauen können. Nutzer benötigen ein hohes Maß an Vertrauen in die Arbeit dieser Systeme, um sie nutzen zu können [93].

Im Folgenden wird gezeigt, wie sich das IFM-Protokoll für ein im Protokollentwurf nicht bedachtes Szenario erweitern lässt. In diesem Kapitel wird der IFM+Blockchain (IFMBC)-Prototyp [94][40] beschrieben, der zeigen soll, wie das IFM-Protokoll in einem Zero-Trust-Netzwerk[41] eingesetzt werden kann, indem sich die einzelnen Akteure nicht vertrauen.

Übliche Eigenschaften, die für solche Systeme gelten sollten, sind:

- **Authentizität:** Daten müssen als unmanipuliertes Original nachweisbar sein.
- **Zurechenbarkeit:** Der Ersteller von Daten muss nachweisbar identifizierbar sein.
- **Nichtabstreitbarkeit:** Es darf keine Möglichkeit geben, die Verantwortung für geteilte Daten glaubhaft abzustreiten zu können.
- **Integrität/Konsistenz:** Es darf keine Möglichkeit geben, einmal abgelegte Daten, über die Konsens besteht, zu verändern.

[40] Dieser Beitrag wurde veröffentlicht in:
S. Gries u. a. „Using Blockchain Technology to Ensure Trustful Information Flow Monitoring in CPS“. In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C 2018)*. 2018, S. 35–38. DOI: 10.1109/ICSA-C.2018.00014

[41] Zero-Trust-Modelle sind Sicherheitskonzepte, bei denen Systeme niemandem innerhalb und/oder außerhalb des eigenen Netzwerkes vertrauen. Im Kontext dieser Arbeit bezieht sich dies insbesondere auf Knoten im CPS, die von anderen als der eigenen Partei betrieben werden.

Angewandt auf den in diesem Kapitel dargestellten Sachverhalt lassen sich aus diesen Eigenschaften die folgenden Hauptanforderungen für den IFMBC-Prototyp ableiten:

- Es muss gespeichert werden, wer welche Daten weitergegeben hat. Später kann festgestellt werden, an welcher Stelle Informationen verfälscht wurden.
- Die Parteien müssen einander nicht vertrauen. Wenn eine Partei fehlerhafte oder manipulierte Daten weitergibt, muss dies dokumentiert und persistiert werden.
- Gespeicherte Informationen sollten die Unleugbarkeit unterstützen. Jeder muss so die Verantwortung für die Korrektheit seiner geteilten Daten übernehmen.
- Sollte ein Knoten manipulierte Daten verschickt haben, muss dessen Verantwortung, auch für abhängige Knoten, zweifelsfrei nachweisbar sein.

Dies führt zu einigen technischen Anforderungen:

- Abhängigkeitsinformationen sollten unveränderbar werden, sobald sie von einem Spy zur Speicherung freigegeben werden.
- Gespeicherte Informationen sollten überprüfbar sein.
- Es sollte keine zentralen Instanzen geben, denen man vertrauen muss.

Das Ziel des Ansatzes ist es dementsprechend nicht zu verhindern, dass Knoten falsche Informationen weitergeben oder falsche Abhängigkeiten benennen, sondern das alle geteilten Daten nichtabstreitbar persistiert werden, um im Fehler- oder Manipulationsfall Verantwortung und Quelle identifizieren und nachweisen zu können.

3.4.1 Konzept

3.4.1.1 Vertrauensvolles Information Flow Monitoring

Die Blockchain-Technologie ermöglicht die Erfüllung einiger der oben genannten Anforderungen. In diesem Abschnitt wird ein Konzept und Prototyp vorgestellt, wie eine öffentliche Blockchain, wie z. B. Ethereum, verwendet werden kann, um Abhängigkeitsinformationen zwischen gesendeten Informationen zu speichern. Hierbei werden auch Hashes der Nutzlast, und bei Bedarf die Nutzlast selbst, mit abgelegt. Da das IFMBC-Konzept also sowohl Nutzlast als auch IFM-Metadaten mit einschließt, wird für beides gemeinsam im Folgenden nur noch der generische Begriff der Daten verwendet.

Im IFM-Protokoll ist es die Aufgabe der Spies, Abhängigkeitsinformationen zur Persistierung an den Master zu senden. Der Master sammelt und speichert diese Abhängigkeitsinformationen. Jeder Teilnehmer muss dieser Instanz vertrauen, dass keine gespeicherten Informationen verändert oder gelöscht werden. Gespeicherte Informationen sollten daher unveränderbar sein. Der Grund dafür ist, dass die Verantwortung für fehlerhafte Daten später von keinem Teilnehmer in der Lieferkette abgestritten werden kann und dass niemand die aufgezeichneten Informationen über Abhängigkeiten oder die Nutzdaten manipulieren kann. Der Wunsch nach einer verteilten, vertrauenswürdigen Infrastruktur zur Speicherung dieser Daten beinhaltet das Ersetzen der zentralen IFM-Master-Instanz als Speichermedium durch eine dezentrale Plattform. Die Master-Instanz(en) sollte(n) die an einem bestimmten Ort gespeicherten Daten nur noch visualisieren, aber nicht mehr selbst speichern.

Im Folgenden wird die Idee der verteilten Datenspeicherung von Abhängigkeitsinformationen mit Hilfe der Blockchain-Technologie vorgestellt. Das Konzept wird anhand der Kommunikation zwischen zwei beteiligten Knoten, Alice und Bob, veranschaulicht.

1. Alice, die zu einer bestimmten Firma gehört, möchte Daten mit Bob, der Mitglied einer anderen Firma ist, austauschen. Alice und Bob vertrauen einander nicht, da es sich um unterschiedliche Parteien handelt. Daher werden die zu übertragenden Daten von Alice digital signiert und dann von Alice an Bob übertragen.

2. Mit der erhaltenen Signatur kann Bob nun für die Daten sowohl nachweisen, dass diese Daten von Alice stammen, als auch, dass die Daten nicht verändert wurden.

3. Bob signiert die Daten ebenfalls und sendet die Signatur an Alice zurück. Alice ist nun in der Lage zu beweisen, dass die Daten korrekt an Bob übertragen wurden.

4. Alice und Bob wissen nun, dass die Dateiübertragung erfolgreich war. Beide sind in der Lage, dies zu beweisen. Beide können auch beweisen, welche Daten übertragen worden sind.

5. Immer wenn Alice Daten sendet, sammeln sich weitere Signaturen (die eigene und die von Bob erhaltene) und Abhängigkeitsinformationen über das IFM-Protokoll an, die von Alice aufbewahrt werden müssen. Bob muss ebenfalls beide Signaturen aufbewahren. Daher speichert entweder Alice oder Bob diese Daten periodisch in der Blockchain.

6. Die Blockchain kann die Signaturen durch einen Smart Contract automatisch überprüfen.

7. Wenn die Signaturen gültig sind, akzeptiert die Blockchain die Daten und beide Knoten können die lokal gespeicherten Daten löschen.

Durch das vorgestellte Verfahren wird sichergestellt, dass nur Informationen in der Blockchain gespeichert werden, die einen Konsens zwischen Sender und Empfänger bilden. Wenn der sendende Spy andere als die mitgeteilten Abhängigkeitsinformationen auf der Blockchain speichert, können diese nicht als korrekt nachgewiesen werden, da dem Sender die Signatur des Empfängers fehlt. Darüber hinaus würde die Blockchain die Richtigkeit der Daten nicht bestätigen, da nachgewiesen werden kann, dass die Signaturen nicht mit den Daten übereinstimmen.

3.4.1.2 Manipulationssichere Speicherung von Daten

Das Speichern großer Datenmengen auf der Blockchain ist aufgrund der hohen Speicherkosten [95] in der Regel nicht wirtschaftlich. Die Grundidee einer Blockchain besteht darin, dass jeder Knoten eine vollständige Kopie der Chain lokal

speichert und darauf aufbauend neue Transaktionen und Blöcke validieren kann. Seit Beginn der Bitcoin-Blockchain gab es Überlegungen und Ansätze, historische Daten zu beschneiden, obwohl diese Ansätze in den meisten Blockchains [96, 97] noch nicht vollständig implementiert sind. Was jedoch bleibt ist die vollständige Verteilung und Duplizierung der Daten, die nicht skaliert, wenn kontinuierlich große Datenmengen gespeichert werden müssen.

Eine bessere Möglichkeit, Daten vertrauensvoll zu speichern, besteht also nicht darin, die Daten selbst in der Blockchain zu speichern, sondern einen kürzeren Fingerabdruck der Daten zu verwenden, d.h. einen durch eine Hash-Funktion bestimmten Wert. Auf diese Weise können die Daten außerhalb der Blockchain gespeichert werden und es kann trotzdem jederzeit überprüft werden, ob die gespeicherten Daten mit den Daten übereinstimmen, zu denen der Eintrag in der Blockchain erstellt wurde. Dies setzt voraus, dass der Hash-Wert kryptographisch sicher ist und keine Umkehrfunktion bekannt ist, was bei den heutigen Standardmethoden der Fall ist. Das bedeutet, dass die ursprünglichen Daten nun an einem anderen Ort gespeichert werden müssen, weil die Informationen in der Chain nicht mehr ausreichen, um die Informationen wiederherzustellen.

Hierfür ist jede Speichermöglichkeit, auf die alle Beteiligten zugreifen können, potenziell geeignet. Gerade für den Fall, dass nicht jeder Teilnehmer alles speichern möchte – was ein großer Teil der Motivation ist, die Daten nicht auf der Blockchain selbst zu speichern – kann dies schnell zu einer zusätzlichen Einführung von Zentralisierung führen. Der speicherverwaltende Knoten könnte Daten löschen und damit eine weitere Nutzung verhindern, da die Blockchain nur noch der Verifizierung der Daten dient, sie aber nicht mehr selbst speichert. Zur dezentralen Speicherung der Daten gibt es jedoch andere Technologien, die die Aufgabe bewältigen können. Technisch gesehen garantieren diese keine 100%ige Verfügbarkeit, machen es aber nahezu unmöglich, Daten zu verlieren, indem sie diese auf eine Teilmenge von Knoten aus unabhängigen Einheiten verteilen. Ein beliebtes Beispiel ist InterPlanetary File System (IPFS), das hauptsächlich auf einer verteilten Hash-Tabelle basiert. Für die Adressierung selbst wird der Hash der Datei verwendet, d.h. genau der Teil, der auf der Blockchain gespeichert werden würde. Dies erlaubt es, eine Datei direkt über ihren Inhalt zu adressieren und führt dazu, dass kein zusätzliches Adress-System benötigt wird. In der verteilten Hash-Tabelle wird ein Mapping auf die speichernden Knoten durchgeführt, was potenziell beliebig vielen Knoten erlaubt, Kopien der Daten zu speichern, indem

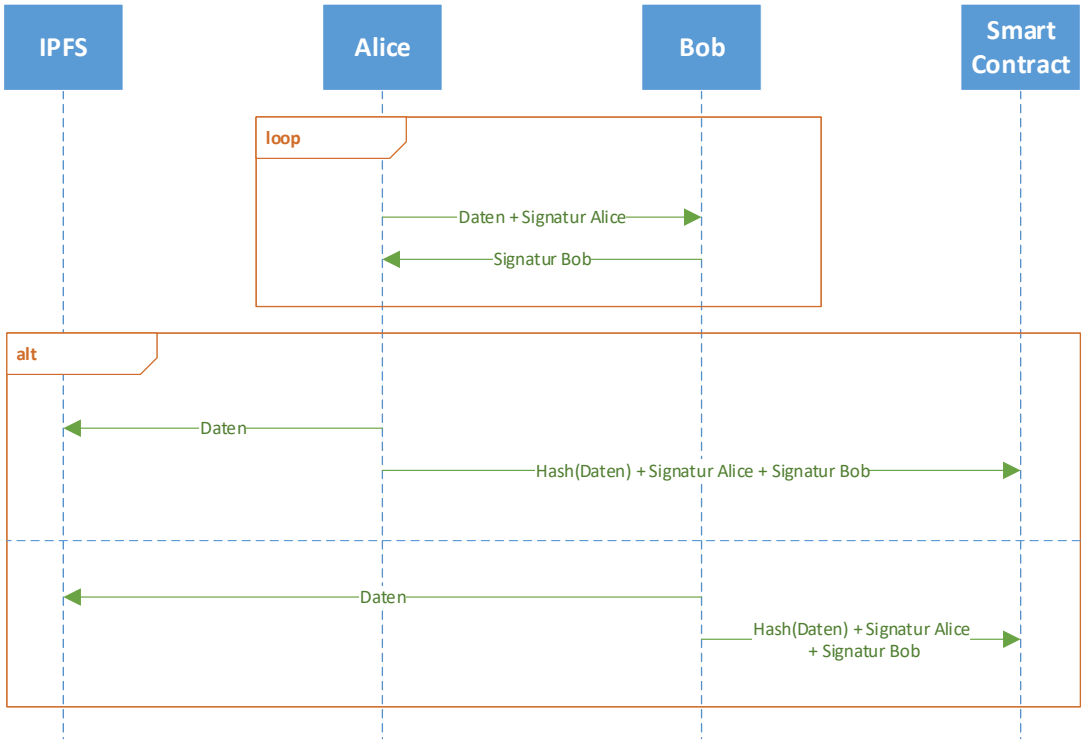


Abbildung 40: IFM-Blockchain-Speicherungsprozess (in Anlehnung an [94])

sie diese unter Verwendung des Hash der Datei als Schlüssel registrieren. Wenn diese Knoten voneinander unabhängig sind, ist es für den Einzelnen schwierig, Daten vollständig aus dem Netzwerk zu löschen, da er andere Knoten nicht verändern kann. Auf diese Weise können Daten mit relativ hoher Verfügbarkeit außerhalb der Blockchain gespeichert werden, ohne sie auf jedem Knoten des CPS zu replizieren.

Ausgehend von dieser Trennung zwischen dezentraler Datenspeicherung und dem durch die Blockchain erzwungenen Konsens über die Daten ist das technische Konzept für den hier vorgestellten Ansatz in Abbildung 40 dargestellt. Es wird davon ausgegangen, dass jedem Mitglied des Konsortiums, das das IFM-Netzwerk betreibt, eine feste Adresse und damit auch ein festes kryptographisches Schlüsselpaar zugeordnet werden kann. Jeder kommunizierte Datensatz muss mit diesem signiert werden. Bob kann also für alle Daten, die er von Alice erhält, eindeutig beweisen, dass sie von Alice übermittelt wurden. Wir gehen davon aus, dass diese Nachprüfbarkeit für den Empfänger von besonderem Interesse ist, da er darauf aufbauend Aktionen durchführt und sich mit den Folgen von fehlerhaften und manipulierten Daten auseinandersetzen muss. Zur Absicherung sollte jedoch

```
1  contract IFM {
2    ...
3    mapping(bytes32 => bool) internal addressStore;
4    ...
5    function verify(bytes32 sigA, bytes32 sigB, bytes32 storageAddress)
         public {
6      if(check(sigA, storageAddress) && check(sigB, storageAddress)){
7        addressStore[storageAddress] = true;
8      }
9    }
10   ...
11 }
```

Abbildung 41: Smart Contract [94]

vorgesehen werden, dass Bob durch eine Signatur bestätigt, dass er die Daten erhalten hat, so dass Alice auch die korrekte Übermittlung der Daten nachweisen kann. Auf diese Weise werden auf beiden Seiten eine Menge Daten und Signaturen gesammelt. Wenn nun einer der Kommunikationsteilnehmer Informationen auf der Blockchain speichern will, um ein öffentliches Vertrauen zwischen den Parteien zu schaffen, reichen diese Informationen aus, ohne dass die andere Partei dies verhindern kann.

Signaturen signieren normalerweise den Hash einer Datei. Durch die Verwendung des gleichen Hash-Verfahrens, wie es das zur Adressierung verwendete Speichersystem verwendet, z. B. die *self-describing multihash method* des IPFS, können die Daten in einem solchen Speichersystem leicht gespeichert werden und die Signatur beweist automatisch, dass die Daten unter der Speicheradresse tatsächlich übertragen wurden. Nachdem die Daten z. B. im IPFS gespeichert wurden, kann die entsprechende Partei, wie in Abbildung 41 dargestellt, einen Smart Contract oder genauer gesagt Executable Distributed Code Contract (EDCC) aufrufen, der die IPFS-Adresse und die beiden Signaturen entgegennimmt. Ein Architekturüberblick über die verschiedenen an diesem Prozess beteiligten Komponenten ist in Abbildung 42 dargestellt.

Auf diese Weise kann das Netzwerk überprüfen, ob die Daten im IPFS tatsächlich zwischen beiden Parteien kommuniziert wurden, ohne die Daten selbst kennen zu müssen, und dann einen entsprechenden Eintrag in der Blockchain erstellen. Danach sind alle wichtigen Informationen unveränderbar gespeichert und können

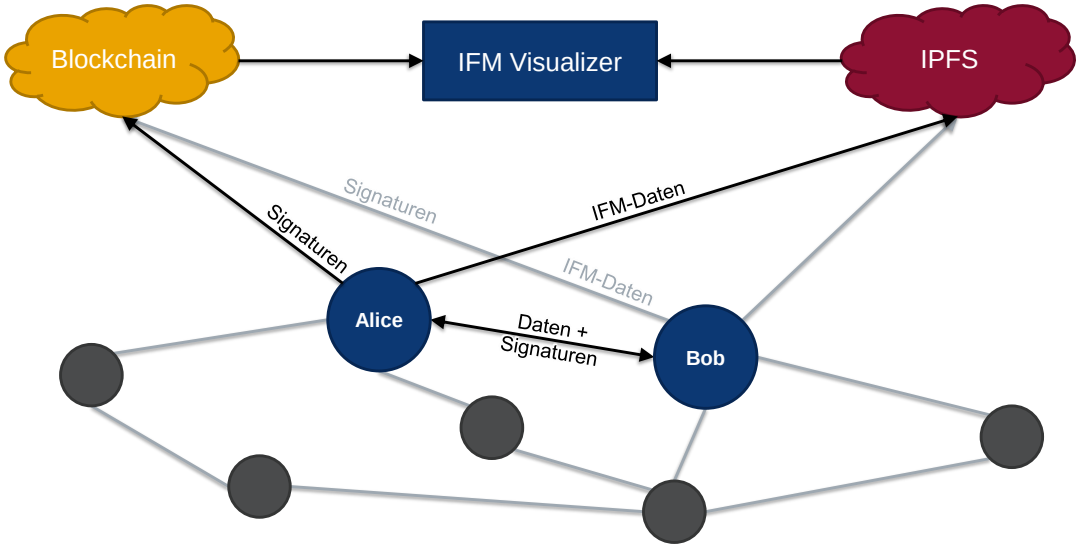


Abbildung 42: IFM-Netzwerk mit Blockchain- und IPFS-Speicherung sowie Visualizer (in Anlehnung an [94])

später überprüft werden, wenn z. B. ein Fehler auftritt. Da jede Codeausführung einen bestimmten Geldbetrag kostet, sollte eine Interaktion mit einem eingesetzten EDCC nur bei Bedarf erfolgen. Indem jeder Partei die Möglichkeit gegeben wird, zunächst einen Satz von übertragenen Daten zu sammeln und später eine gebündelte Transaktion durchzuführen, bietet dieser Ansatz eine flexible Lösung zur Minimierung der Transaktionskosten. So kann jeder Teilnehmer vor der Übergabe der Daten ein eigenes Intervall wählen und es entsprechend an die individuellen Bedürfnisse des Vertrauens anpassen.

3.4.2 Implementierung

Im Rahmen eines Projektes wurde das IFMBC Konzept prototypisch implementiert, um die Machbarkeit zu überprüfen. Hierzu wurde das AQ-CPS (vgl. Abschnitt 4.1) genutzt und um entsprechende Komponenten erweitert. In dieser ersten Implementierung wurde auf IPFS verzichtet und stattdessen eine private Blockchain genutzt, da öffentliche Blockchains wie Etherum durch Transaktionskosten bei Experimenten hohe Kosten verursachen. Durch die vollständig entfallenen Transaktionskosten ist in dieser ersten Implementierung hierdurch IPFS nicht notwendig, um ein Proof-of-Concept zu erstellen.

Um die im vorherigen Kapitel beschriebenen Funktionalitäten zu gewährleisten wurden folgende Komponenten in das AQ-CPS eingefügt:

- **Geth-Knoten**, die für jede Partei einen eigenen Knoten der Blockchain bereitstellen.

  Geth ist eine offizielle Implementierung der Ethereum Blockchain [98]. Eine Menge an Geth-Knoten bildet dabei die Blockchain. Um Daten in der Blockchain zu speichern wird eine Verbindung zu einem Geth-Knoten aufgebaut, der die Daten dann in der Blockchain ablegt. Die Gesamtheit der Geth-Knoten ersetzt somit den IFM-Master, der als Datenbank der Abhängigkeitsinformationen fungierte. Jede Partei im Netzwerk betreibt hierzu einen eigenen Geth-Knoten, dem sie vertrauen. Über diesen Geth-Knoten können die einzelnen Sensoren, Aktoren und Aggregatoren des AQ-CPS Abhängigkeitsinformationen in der Blockchain ablegen.

- **IFMBC-Interface** in den Knoten (Sensoren, Aggregatoren, Aktoren) des CPS, um die Kommunikation mit der Blockchain zu realisieren.

  Bisher kommunizieren die Spy-Knoten direkt mit dem IFM-Master, um Abhängigkeitsinformationen zu speichern. Da dieser durch die Blockchain ersetzt wurde, benötigen die Spies eine Komponente zur Kommunikation mit der Blockchain. Das IFMBC-Interface existiert hierzu zusätzlich zum IFM-Interface in jedem Spy, um die Kommunikation mit den Geth-Knoten der eigenen Partei abzuwickeln.

- **IFM-Visualizer for BC** der Abhängigkeitsinformationen aus der Blockchain abrufen und visualisieren kann. Er ersetzt den zuvor gezeigten IFM-Visualizer for Mongo.

Im Zusammenspiel realisieren die beschriebenen Komponenten die Funktionalität aus Abschnitt 3.4.1. Es handelt sich hierbei um einen lauffähigen Prototyp, der die Erweiterbarkeit des IFM-Konzepts um Blockchain-Komponenten belegt. Hierdurch wird das Sammeln von Abhängigkeitsinformationen zwischen mehreren Parteien, die sich nicht vertrauen, ermöglicht.

3.4.3 Fazit

Durch die Kombination von IFM und Blockchain-Technologie wird es möglich, Abhängigkeitsinformationen auch in vertrauenslosen CPS-Netzwerken zu sammeln.

Grundsätzlich führt die Kombination aus IFM und Blockchain allerdings nicht dazu, dass Knoten keine fehlerhaften Angaben über Abhängigkeiten machen können. Diese Angaben werden jedoch unabänderbar persistiert und können so jederzeit zum Beweis der Verantwortung herangezogen werden. Als manipuliert identifizierte geteilte Informationen können durch diese Information zweifelsfrei ihrem Ersteller zugeordnet werden. So kann kein Knoten abstreiten, bestimmte später als fehlerhaft identifizierte Informationen erstellt und geteilt zu haben. Die Verantwortung für manipulierte und/oder fehlerhafte Daten ist somit nachweisbar.

3.5 Fazit

Das vorstehende Kapitel 3 besteht aus vier zentralen Bausteinen: Der Einführung des Information Flow Monitor (IFM), dem Platzierungsproblem der Spy-Knoten, dem Information Dependency Modeling und dem IFM-Blockchain-Konzept.

Der IFM, vorgestellt in Abschnitt 3.1, wurde als zentrales Lösungselement dieser Arbeit präsentiert. Er ermöglicht die durchgehende Erfassung von Abhängigkeiten über alle durch Spy-Knoten überwachten Pfade des CPS hinweg. Durch den IFM-Visualizer können komplexe Abhängigkeitsstrukturen zentral visualisiert werden. Hierbei werden sowohl ausgetauschte Nachrichten als auch beteiligte Knoten in der Abhängigkeitshierarchie identifiziert. Je nach Ausprägung des Netzwerkes können im Visualizer auch Nutzdaten dargestellt werden, die teils für die Identifizierung der Fehlerquelle notwendig sind. Die Identifizierung der Quelle eines Fehlers erlaubt somit erst dessen Behebung.

Zur Anwendbarkeit des IFM wurde die Lösung zur Platzierung von Spy-Knoten in CPS-Netzen in Abschnitt 3.2 vorgestellt. Die darin ermittelten Heuristiken helfen bei der Platzierung von Spy-Knoten in CPS, um hohe Pfadabdeckungen zu erreichen. Die vorgestellten Methoden wurden auf verschiedene Netzklassen

angewandt. Die Ergebnisse geben Orientierung bei der Anwendung der Heuristiken auf andere Netze, in die der IFM implementiert werden soll.

Das Information Dependency Modeling in Abschnitt 3.3 ermöglicht den Einsatz des IFM-Konzepts in Netzwerken, in denen COTS-Komponenten oder andere Knoten eingesetzt werden, deren Software nicht IFM-konform ist und auch nicht IFM-konform gemacht werden kann. Das Information Dependency Modeling soll hierbei vor und nach dem Black-Box-Knoten aufgezeichnete Abhängigkeitsinformationen verbinden.

In Abschnitt 3.4 wurde als letzter Beitrag dieses Kapitels die Kombination aus IFM und Blockchain eingeführt. Die Kombination der beiden Technologien unterstützt den Einsatz des IFM in vertrauenslosen Netzwerkumgebungen.

Die in diesem Kapitel vorgestellten Konzepte und Methoden werden in Kapitel 4 evaluiert.

4 Evaluation

In diesem Kapitel: Evaluation des IFM-Konzepts mit Hilfe eines echtweltlichen Szenarios. Performanz Evaluierung des Ansatzes sowie Diskussion der Auswirkungen der reduzierten Performanz auf CPS.

Mit Hilfe des IFM werden Abhängigkeiten in CPS aufgezeichnet, um diese anschließend nachvollziehen zu können. Im folgenden Kapitel werden die ursprünglichen Anforderungen (vgl. Abschnitt 3.1.1) anhand der erstellten Lösung evaluiert. Hierzu werden zwei Evaluationsschritte durchgeführt:

1. **Überprüfung der Anwendbarkeit in einem echtweltlichen Szenario**
   Es wird gezeigt, dass die mit Hilfe des IFM-Protokolls aufgezeichneten Abhängigkeitsinformationen, die im IFM-Master gesammelt werden, zum Nachvollziehen von Fehlerkaskaden bis hin zur ursprünglichen Fehlerquelle genutzt werden können. Die Evaluation dieses Teils (s. Abschnitt 4.1) geschieht im Kontext eines realen Anwendungsszenarios für CPS zur Messung der Luftqualität und der Steuerung von Smart-City- und Smart-Home-Komponenten.

2. **Überprüfung der Performanz**
   Das IFM-Protokoll realisiert Funktionalität im CPS und muss dafür zusätzliche Daten im Netzwerk austauschen. Diese Daten benötigen Zeit bei der Erzeugung und Bandbreite bei der Übertragung. Daher entsteht durch den Einsatz des IFM-Protokolls zusätzlicher Bedarf an Ressourcen im Netzwerk. Um in der Praxis bei Einsatz des IFM abwägen zu können, ob diese zusätzlichen Ressourcen zur Verfügung stehen, wurde untersucht (s. Abschnitt 4.2), wie groß der zusätzliche Bedarf ist und welchen Einfluss das IFM-Protokoll auf die Performanz des Netzwerks hat.

4.1 AirQuality-Anwendungsfall

Im Jahr 2015 kam es in Deutschland zum in den Medien so genannten *Abgasskandal*, bei dem mehrere deutsche und internationale Autobauer beschuldigt wurden, dass ihre Fahrzeuge im Betrieb deutlich mehr gesundheitsschädliche Stoffe emittieren als angegeben. Infolgedessen entstand in der Öffentlichkeit eine Debatte, inwiefern Fahrbeschränkungen, Fahrverbote oder Nachrüstung von Fahrzeugen die Situation verbessern können. Im Rahmen der Aufklärung wurden auch an Messstationen verschiedener Innenstädte teils erhebliche Überschreitungen der Grenzwerte an Kohlenstoffdioxid oder Feinstaub gemessen [99].

Luftverschmutzung wird von der World Health Organization (WHO) wie folgt definiert:

> „Luftverschmutzung ist die Verunreinigung der Innen- oder Außenumgebung durch chemische, physikalische oder biologische Stoffe, die die natürlichen Eigenschaften der Atmosphäre verändern." – World Health Organization [100]

Die schlechte Qualität der Umgebungsluft ist ein typisches Problem von urbanen und durch Industrie geprägten Regionen. Hauptverursacher sind hierbei der Straßenverkehr, die Industrie sowie die Massentierhaltung in Kombination mit industrieller Landwirtschaft. 400.000 EU-Bürger sterben jedes Jahr vorzeitig (Schätzung Europäische Union (EU) [101], unterstützt von der United Nations Economic Commission for Europe (UNECE) [102]) an den Folgen der Luftverschmutzung. Grundsätzlich hat sich die Luftqualität seit den 1990er Jahren verbessert [103, 104]. Bestimmte Verschmutzungen wie Feinstaub (PM 2.5), Ozon oder Stickstoffdioxid bleiben in ihrer Belastung aber problematisch und beschädigen Gesundheit und Umwelt [102].

In Deutschland ist die Situation ähnlich. Im Vergleich der EU-Länder stößt Deutschland am meisten Kohlenstoffdioxid, Kohlenstoffmonoxid, Stickoxide und Feinstaub (PM 2.5 und PM 10) aus [105]. In urbanen Regionen ist vor allem der Verkehr, insbesondere Dieselfahrzeuge, sowie Heizungsanlagen für den Feinstaub verantwortlich [102]. In Deutschland liegt die Verantwortung für Maßnahmen zur Luftreinhaltung bei den Kommunen. Diese suchen nach Möglichkeiten, die

genaue Belastung innerhalb der Städte, insbesondere nahe der Wohnbebauung, zu messen.

In Nordrhein-Westfalen (NRW) betreibt das Landesamt für Natur, Umwelt und Verbraucherschutz Nordrhein-Westfalen (LANUV) im Auftrag der Kommunen derzeit 162 aktive Messstationen. Darunter befinden sich sowohl aktive Messcontainer, die über Sensorik verschiedene Umweltparameter erfassen, als auch Passivsammler, die periodisch manuell ausgewertet werden [106]. Somit kommt auf etwa 210 km² oder 111.000 Einwohner eine Messstation. Dies liegt insbesondere in den hohen Kosten dieser Messstationen begründet. Ein einzelner Messcontainer kostet etwa 150.000 €, weswegen immer mehr Passivsammler eingesetzt werden, die allerdings nur Monatsmittelwerte liefern [107]. Studien [108, 109] kommen jedoch zu dem Schluss, dass sich Messwerte innerhalb kürzester Distanzen und kurzer Zeit stark unterscheiden können, sodass nur ein eng vermaschtes Sensornetz Aufschluss über die tatsächliche Luftqualität in Städten geben kann.

Solche Sensornetze sind auch CPS, die über viele verteilte Sensoren verfügen, die dezentral und autonom Daten erfassen und über Netzwerke teilen. So wird die Luftqualität über geographische Distanzen hinweg erfassbar und auswertbar. Ein solches Netz wurde im Rahmen dieser Arbeit konzeptioniert, implementiert und simuliert, um daran die Evaluation der Anwendbarkeit des IFM-Konzepts vorzunehmen.

Im Folgenden wird dieses um Aktoren erweiterte Sensornetz genutzt, das sich am oben skizzierten Anwendungsfall orientiert [110][42]. Der so genannte AirQuality-Prototyp ist eine Implementierung des CPS, welches zusätzlich zu den Sensoren, die Umweltparameter wie Feinstaub oder Kohlenmonoxid in der Luft messen, auch Aktoren enthält, die direkt auf die veränderte Luftqualität reagieren können. Der Prototyp wurde im Rahmen dieser Arbeit konzipiert und implementiert [110]. Während die Topologie und Kommunikationsfähigkeiten des Prototyps sehr realistisch aufgebaut wurden, ist die Businesslogik der einzelnen Komponenten

[42] Dieser Beitrag wurde veröffentlicht in:
S. Gries u. a. „Developing a Convenient and Fast to Deploy Simulation Environment for Cyber-Physical Systems“. In: *Proceedings - 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS 2018)*. 2018, S. 1551–1552. ISBN: 978-1-5386-6871-9. DOI: 10.1109/ICDCS.2018.00166

im Vergleich zu einer nicht-prototypischen Implementierung einfach und naiv gehalten. Das CPS ist also so komplex wie nötig, aber so einfach wie möglich.

Die Evaluation des IFM-Konzepts kann nicht ohne ein solches reales Anwendungsszenario durchgeführt werden, da Domänenwissen bei der Nutzung des IFM-Visualizers notwendig ist. Der IFM ist ein Debugging-Werkzeug, dass Abhängigkeiten aufzeichnet und mit dem Visualizer darstellt. Der Nutzer des Visualizers muss sich aber im dargestellten Abhängigkeitsbaum orientieren können und anhand der dargestellten Nutzdaten entscheiden, welcher Pfad im Baum fehlerbehaftet ist. Dies geschieht mit Hilfe von Domänenwissen, welches dabei hilft zu entscheiden, ob ein Wert im vorliegenden Szenario plausibel ist.

4.1.1 AirQuality-Anwendungsfallbeispiel

Grundsätzlich gibt es bei sogenannten Smart Cities mehrere Parteien, die unterschiedliche Daten benötigen, um eigene Interessen umzusetzen. Das sind verschiedene Privatpersonen, aber auch Behörden oder Institutionen, die beispielsweise Infrastruktur zur Verfügung stellen. Das Konzept der Smart City sieht vor, dass diese unterschiedlichen Parteien kooperieren, um Daten auszutauschen, die der jeweils andere für seinen Anwendungsfall benötigt.

In unserem Szenario entsteht hierbei ein Sensor-Aktor-Netz, also ein CPS, in dem diese verschiedenen Parteien Sensoren, Aktoren und Aggregatoren betreiben [110]. Fokus ist in unserem Szenario die Erfassung der Luftqualität durch Sensorik sowie deren Auswertung durch Aggregatoren. Zusätzlich kann direkt auf veränderte Luftqualität mit Hilfe von Aktoren reagiert werden.

In Abbildung 43 ist eine schematische Darstellung eines Ausschnitts einer Smart City dargestellt. Hierbei sind drei Parteien (Blau, Rot, Gelb) abgebildet, die jeweils unterschiedliche Anforderungen haben.

- **Partei Blau**
  Der Benutzer (Privatperson) legt großen Wert auf frische Luft, wenn er nach draußen gehen möchte. Daher möchte der Benutzer gewarnt werden, wenn die aktuelle Luftqualität einen gewissen Schwellenwert erreicht.

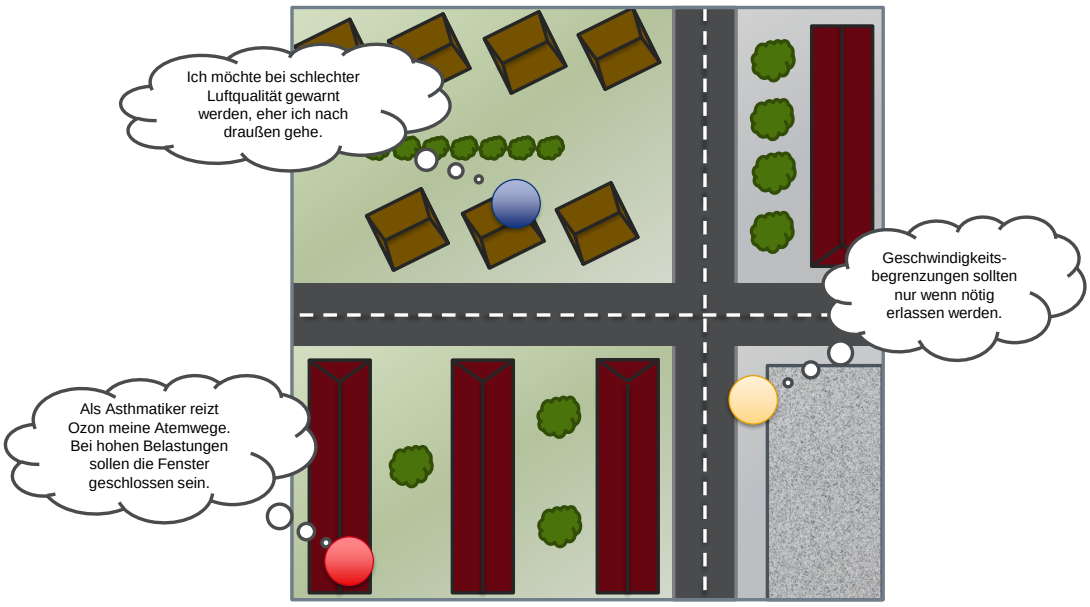


Abbildung 43: Kartenausschnitt einer Smart City mit AirQuality Anwendungsfall und Anforderungen bestimmter Teilnehmer in Netzwerk

Als Metrik hierfür wurde der Air Quality Health Index (AQHI) gewählt [111]. Dieser Index ist eine allgemeine Vereinfachung verschiedener Messwerte (Ozon, Feinstaub 2.5, Stickstoffdioxid) auf einen Gesamtindex, der sich zwischen 1 und 10+ bewegt. Ein Wert von 1 signalisiert hierbei ein sehr geringes Gesundheitsrisiko, während Werte über 10 ein sehr hohes Gesundheitsrisiko darstellen. Ab einem Wert von 7 wird der AQHI mit einem hohen Risiko bewertet. Daher wurde in diesem Beispiel der Wert 7 als Schwellenwert für eine Warnung gewählt.

- **Partei Rot**
  Der Benutzer (Privatperson) hat Asthma. Daher treten bei ihm durch hohe Ozonbelastungen Atemnot oder Asthmaanfälle häufiger auf. Wenn im Freien hohe Ozonwerte gemessen werden, sollen daher in der Wohnung die Fenster automatisch geschlossen werden.

  In Deutschland gibt es laut Umweltbundesamt bei Ozon einen Informationsschwellenwert von 180 $\mu g/m^3$ im 1-Stunden-Mittelwert [112]. Daher wird in diesem Beispiel auch dieser Schwellenwert zum Schließen der Fenster angenommen.

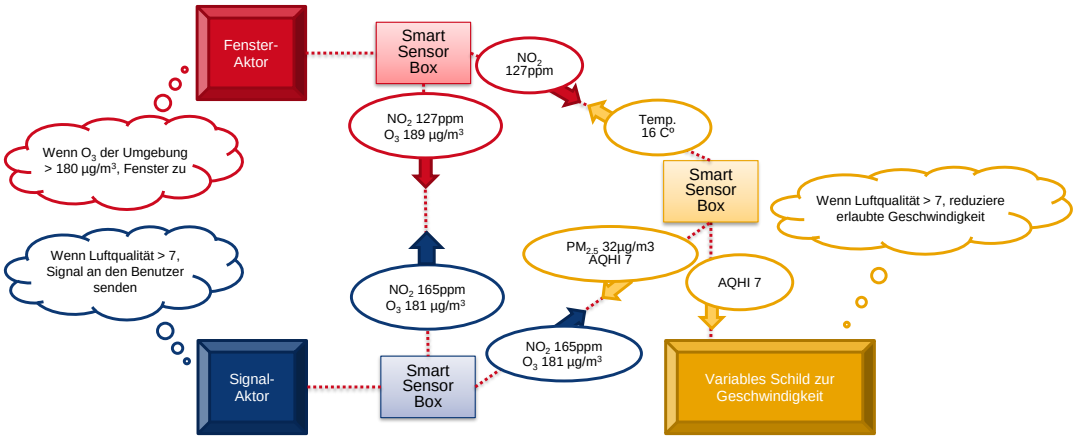


Abbildung 44: Sicht auf die einzelnen Komponenten und die Vernetzung des AirQuality Anwendungsfalls

- **Partei Gelb**
  Die Kommune ist dafür zuständig, die Bevölkerung vor einer zu schlechten Luftqualität zu schützen. Um insbesondere Anwohner von Durchgangsstraßen vor hohen Emissionswerten zu schützen, gilt die Reduktion der erlaubten Höchstgeschwindigkeit als bewährtes Mittel. Statt die Geschwindigkeit pauschal zu drosseln, soll die Begrenzung nur angewandt werden, wenn es notwendig ist.

  Ab einem AQHI von 7 soll daher die Geschwindigkeitsbegrenzung herabgesetzt werden. Ein Aktor in Form einer digitalen Geschwindigkeitsanzeige soll hierzu eingesetzt werden.

Damit eine Partei ihre Ziele erreichen kann, werden auch Daten anderer Parteien benötigt. Wie bereits in Abschnitt 4.1 beschrieben, sind einzelne Messwerte einer Station oft nicht sinnvoll nutzbar. Es werden Werte mehrerer Standorte benötigt. Zusätzlich handelt es sich bei manchen Werten, z. B. dem AQHI, nicht um Messwerte, sondern um Aggregate verschiedener Messwerte. Diese Messwerte können aus verschiedenen Quellen zusammengetragen werden, um einen aggregierten Wert zu berechnen. In diesem Beispiel sind die Parteien mit je einer Sensor Box ausgestattet, die wie folgt konfiguriert ist (vgl. Abbildung 44):

- **Partei Blau**

  Eigene Sensoren: Die blaue Partei unterhält eine Sensor Box mit einem Ozon- und einem Stickstoffdioxidsensor.

Benötigte Messwerte: Um den benötigten AQHI zu berechnen, wird zusätzlich auch die Feinstaubbelastung der Luft benötigt. Des Weiteren können Messwerte anderer Stationen die Genauigkeit der Messung verbessern.

Verarbeitung der Messwerte: Die blaue Partei kann zusätzlich zu den eigenen Sensoren auch die gemessenen Ozon- und Stickstoffdioxidwerte der roten Partei sowie den Feinstaub- und AQHI-Wert der gelben Partei nutzen. So kann sowohl ein fremder AQHI genutzt, sowie ein eigener berechnet werden. Auf der Grundlage dieser Werte kann entschieden werden, ob der Benutzer über eine schlechte Luftqualität informiert wird.

- **Partei Rot**

  Eigene Sensoren: Die rote Partei unterhält eine Sensor Box mit einem Ozon- und einem Stickstoffdioxidsensor.

  Benötigte Messwerte: Es wird an dieser Stelle nur ein Ozonwert der Luft benötigt. Punktuelle Messungen sind jedoch nicht repräsentativ für die Umgebung. Daher ist es notwendig, so viele Ozonwerte verschiedener Messstationen wie möglich zu sammeln und zu einem gemeinsamen, repräsentativen Wert zu aggregieren.

  Verarbeitung der Messwerte: Die rote Partei kann zusätzlich zu den eigenen Sensoren auch die gemessenen Ozonwerte der blauen Partei nutzen. Hierdurch können zumindest zwei Messwerte, die mit räumlicher Distanz gemessen wurden, genutzt werden, um einen repräsentativen Wert der Umwelt zu berechnen. Dieser wird anschließend genutzt, um zu entscheiden, ob das Fenster geöffnet oder geschlossen wird.

- **Partei Gelb**

  Eigene Sensoren: Die gelbe Partei unterhält eine Sensor Box mit einem Feinstaub- und einem Temperatursensor.

  Benötigte Messwerte: Um den AQHI berechnen zu können, werden zusätzlich zum eigenen Feinstaubsensor auch Messwerte bzgl. Stickstoffdioxidkonzentration und Ozonkonzentration der Umgebung benötigt. Beide können von der roten und blauen Partei ergänzt werden, sodass alle Werte aus mehreren Quellen von verschiedenen Standorten gleichzeitig bezogen werden können.

Verarbeitung der Messwerte: Die gelbe Partei kann aus den bezogenen Messwerten der roten und blauen Partei sowie den eigenen Messwerten den AQHI berechnen und zur Schaltung der digitalen Geschwindigkeitsbegrenzung nutzen.

Durch die Verknüpfung der verschiedenen Datenquellen (Sensoren), Berechnungen (Aggregatoren) und Aktoren entsteht so ein CPS, in dem Daten ausgetauscht werden. Zusätzlich entstehen jedoch auch Abhängigkeiten zwischen den Parteien, da einzelne Services nicht mehr korrekt ausgeführt werden können, wenn fehlerhafte Daten von anderen bereitgestellt werden.

4.1.2 AirQuality-Prototyp-Architektur

Um die Evaluation anhand des beschriebenen Anwendungsfalls zu ermöglichen, wurde zur Umsetzung des CPS ein hybrides Modell gewählt. Ziel war es, die komplette Netzwerkkommunikation real ablaufen zu lassen und nicht nur zu simulieren. Die einzelnen Knoten im CPS wiederum sollten mit virtualisierter Hardware betrieben werden, sodass es keinen Unterschied mehr macht, ob die Software der Knoten auf dedizierter oder virtueller Hardware ausgeführt wird. Diese Architektur hebt den Ansatz dieses Prototypen von reiner Simulationssoftware, in der keine echte Netzwerkkommunikation stattfindet und Knotensoftware nicht getrennt voneinander ausgeführt wird, ab.

Da jeder Knoten auf eigener virtualisierter Hardware betrieben wird, kann – wie in einem CPS üblich – ausschließlich über das Netzwerk mit anderen Knoten kommuniziert werden. Dies führt auch dazu, dass in das Netzwerk durch seine bestehende Architektur auch Knoten auf eigener Hardware ganz ohne Virtualisierung eingefügt werden könnten. Es ist dem einzelnen Knoten also nicht bekannt, ob der andere Knoten, mit dem kommuniziert wird, auf echter oder virtualisierter Hardware ausgerollt wurde, da dies von außen betrachtet keinen Unterschied macht.

Knotentypen im Anwendungsfall

Jeder Knoten im CPS dieses AirQuality-Anwendungsfalls (im Folgenden AQ-CPS genannt) erfüllt eine bestimmte Funktion. Im Netz sind Sensoren, Aktoren und

Aggregatoren verschiedener Ausprägungen. Unter den Sensoren messen einige beispielsweise Feinstaub oder die Ozonbelastung. Für unseren Anwendungsfall wurde ein Repertoire an verschiedenen Sensoren, Aktoren und Aggregatoren zusammengestellt, die im AQ-CPS zusammenarbeiten können.

Folgende Knotentypen wurden definiert:

Sensoren

- **Luftdrucksensor**: *sensor-airPressure*
  Misst den Luftdruck in *bar* und gibt alle 10 Sekunden einen Messwert aus.
  Ausgabe: *sensor-airPressure*

- **Lichtschranke**: *sensor-lightBarrier*
  Gibt alle 5 Sekunden einen Messwert aus, der angibt, ob die Lichtschranke unterbrochen ist oder nicht. Über diese Stichproben kann im Stadtverkehr beispielsweise ermittelt werden, ob viele oder wenige Fahrzeuge auf der Straße fahren.
  Ausgabe: *sensor-lightBarrier*

- **Stickstoffdioxidsensor**: *sensor-no2*
  Misst alle 10 Sekunden den Stickstoffdioxidgehalt der Luft in *ppm* und gibt den Messwert aus.
  Ausgabe: *sensor-no2*

- **Ozonsensor**: *sensor-o3*
  Misst alle 10 Sekunden den Ozongehalt der Luft in *ppb* und gibt den Messwert aus.
  Ausgabe: *sensor-o3*

- **Feinstaubsensor $PM_{2,5}$**: *sensor-pm25*
  Misst jede Sekunde den Feinstaubgehalt der Luft mit Partikeln mit einem maximalen Durchmesser von 2,5 Mikrometern in g/m^3 und gibt den Messwert aus.
  Ausgabe: *sensor-pm25*

- **Feinstaubsensor PM_{10}**: *sensor-pm10*
  Misst jede Sekunde den Feinstaubgehalt der Luft mit Partikeln mit einem maximalen Durchmesser von 10 Mikrometern in g/m^3 und gibt den Messwert aus.
  Ausgabe: *sensor-pm10*

- **Temperatursensor**: *sensor-temp*
  Gibt alle 10 Sekunden die aktuelle Temperatur in $°C$ aus.
  Ausgabe: *sensor-temp*

Aggregatoren

- **Durchschnittstemperatur**: *aggregator-averageTemperature*
  Errechnet die durchschnittliche Temperatur über eine Minute und gibt im gleichen Rhythmus den berechneten Wert aus.
  Eingabe: *sensor-temp*, Ausgabe: *aggregator-avgTemp*

- **Stichprobenzähler für eine Lichtschranke**:
  aggregator-lightBarrierCounter
  Aggregiert 30 Sekunden lang die Ausgaben der Lichtschranke und gibt im gleichen Rhythmus deren Summe aus.
  Eingabe: *sensor-lightBarrier*,
  Ausgabe: *aggregator-lightBarrierCounter*

- **Durchschnittliche Feinstaubbelastung**:
  aggregator-particleMatter25MinuteAverage
  Berechnet über den Zeitraum von einer Minute die durchschnittliche Feinstaubbelastung der Luft und gibt im gleichen Rhythmus den berechneten Wert aus.
  Eingabe: *sensor-pm25*, Ausgabe: *aggregator-avgPm25*

- **Luftqualitätsindex**: *aggregator-airQualityIndex*
  Berechnet jede Minute einen Luftqualitätsindex aus den vorliegenden Werten des Stickstoffdioxids, Feinstaubs und Ozons. Hierbei handelt es sich um einen Index, der an den zuvor genannten AQHI angelehnt ist.
  Eingabe: *sensor-no2*, *sensor-pm25*, *sensor-o3*, Ausgabe: *aggregator-aqhi*

- **Berechnung der Geschwindigkeitsbegrenzung**: *aggregator-trafficSpeedDetermination*
  Berechnet jede Sekunde mit Hilfe der Aggregatorwerte der durchschnittlichen Feinstaubbelastung, des Luftqualitätsindexes, der gezählten Fahrzeuge und der aktuellen Temperatur sowie Vorgaben der Betreiber eine zulässige Höchstgeschwindigkeit für den Verkehr.
  Eingabe: *sensor-temp*, *aggregator-avgPm25*, *aggregator-aqhi*, *aggregator-lightBarrierCounter*,
  Ausgabe: *aggregator-trafficSpeedLimit*

- **Berechnung des Fensterzustands**: *aggregator-windowState*
  Berechnet jede Minute, ob ein Fenster geöffnet oder geschlossen werden sollte. Hierzu wird sowohl die durchschnittliche Feinstaubbelastung und der Ozonwert als auch die durchschnittliche Außentemperatur sowie festgelegte Schwellenwerte des Betreibers des Aggregators berücksichtigt.
  Eingabe: *sensor-o3*, *aggregator-avgPm25*, *aggregator-avgTemp*,
  Ausgabe: *aggregator-windowState*

- **Berechnung der Ampelsteuerung**:
  aggregator-trafficLightControl
  Berechnet alle 10 Sekunden, welchen Zustand eine Ampel annehmen soll (1 = Grün, 2 = Rot). Dies ist insbesondere von der erlaubten Höchstgeschwindigkeit sowie dem Verkehrsaufkommen abhängig. Bei hohem Verkehrsaufkommen und/oder niedrigen erlaubten Geschwindigkeiten können so Grünphasen beispielsweise verlängert werden, um einen besseren Verkehrsfluss sicherzustellen.
  Eingabe: *aggregator-trafficSpeedLimit*, *sensor-lightBarrier*, Ausgabe: *aggregator-trafficLightControl*

Aktoren

- **Fenster**: *actuator-smartHomeWindow*
  Repräsentiert ein Fenster, das automatisch geöffnet oder geschlossen werden kann.
  Eingabe: *aggregator-windowState*

- **Smart Display**: *actuator-smartDisplay*
  Repräsentiert ein Smart Home Display, dass den Benutzer mit Informationen versorgen kann.
  Eingabe: *aggregator-aqhi*, *aggregator-avgTemp*, *aggregator-avgPm25*, *sensor-pm10*, *sensor-o3*, *sensor-airPressure*

- **Variables Geschwindigkeitsschild**: *actuator-trafficSpeedSign*
  Repräsentiert ein Schild zur variablen Anzeige der zulässigen Höchstgeschwindigkeit.
  Eingabe: *aggregator-trafficSpeedLimit*

- **Ampel**: *actuator-trafficLight*
  Repräsentiert eine Ampel.
  Eingabe: *aggregator-trafficLightControl*

Die genannten IDs der Ein- und Ausgabe einzelner Knoten geben an, welche Knoten miteinander verbunden werden können, um Daten auszutauschen. Ein Stickstoffdioxidsensor, der Daten vom Typ *sensor-no2* als Ausgabe erzeugt, kann so mit einem Aggregator zur Berechnung des Luftqualitätsindex verbunden werden, der genau diese Messwerte als Eingabe benötigt. Für jeden Knotentyp wurde so eine technische Definition erstellt, die neben den IDs für Ein- und Ausgaben auch weitere Informationen enthalten, wie beispielsweise den Ausgaberhythmus von Werten. Eine exemplarische Knotentypdefinition für den Aggregator zur Berechnung der Geschwindigkeitsbegrenzung (*aggregator-trafficSpeedDetermination*) ist in Abbildung 45 dargestellt. Hierin sind sowohl die oben beschriebenen Ein- und Ausgaben definiert (Z. 15 f.), als auch weitere Werte, die das Verhalten des Knotens definieren. Hierzu ist in Zeile 5-7 definiert, in welchem zeitlichen Rhythmus und in welcher Einheit Informationen erzeugt werden. In Zeile 10-12 ist definiert, welchen Wertebereich die Daten dabei annehmen können. Zusätzlich ist ein typischer exemplarischer Wert im Wertebereich angegeben.

```
{
  "nodeType": "aggregator",
  "aggregatorType":"aggregator-trafficSpeedDetermination",
  "meta": {
    "measurementUnit": "km/h",
    "updateRate" : 1,
    "updateRateUnit" : "m"
  },
  "data" : {
    "value" : 30,
    "minValue": 10,
    "maxValue": 70
  },
  "interest" : {
    "input" : ["sensor-temp/+/value", "aggregator-avgPm25/+/value",
        "aggregator-aqhi/+/value", "aggregator-lightBarrierCounter/+/value"],
    "output": "aggregator-trafficSpeedLimit"
  }
}
```

Abbildung 45: Knotentypdefinition *aggregator-trafficSpeedDetermination*

Generierung des Netzwerkes

Die Knotendefinitionen werden genutzt, um daraus semantisch korrekte Netzwerke des Anwendungsfalls zu generieren. Hierzu wurde ein Werkzeug entwickelt, das die Knoten so in einem Netzwerk zusammenfügt, dass Ein- und Ausgaben zusammenpassen. In Abbildung 46 ist die Benutzeroberfläche des Werkzeugs, dem Network Generator, dargestellt. Hierbei handelt es sich nicht um ein Simulationswerkzeug, sondern um eine Software, die die Topologie eines CPS erstellt. Der Generator erlaubt sowohl die Konfiguration der Größe des Netzwerkes über die Festlegung der Anzahl der Sensoren, Aggregatoren und Aktoren, als auch den Grad der Vermaschung. Eine Edge Probability von 1 würde dazu führen, dass 100% aller möglichen Verknüpfungen von Ein- und Ausgaben der Knoten auch verknüpft werden. Die Verknüpfung erfolgt hierbei zufällig, aber semantisch korrekt bzgl. der zuvor beschriebenen kompatiblen Ein- und Ausgaben der jeweiligen Knotentypdefinitionen. Hierbei kann ein Knoten auch mehrere Eingaben des gleichen Typs erhalten, er sei denn, es handelt sich um einen Aktor. Jeder Aktor kann nur von einem Aggregator gesteuert werden. Die Ausgaben eines Knotens wiederum können von mehreren anderen Knoten verwendet werden. Aus dieser Art der Generierung der Vernetzung resultiert auch, dass nicht alle Ausgaben aller Knoten genutzt werden und nicht immer alle theoretisch möglichen Eingaben auch eingegeben werden.

Über den Link zu einem GIT-Repository kann ebenfalls konfiguriert werden, welche Software auf den einzelnen Knoten ausgeführt werden soll.

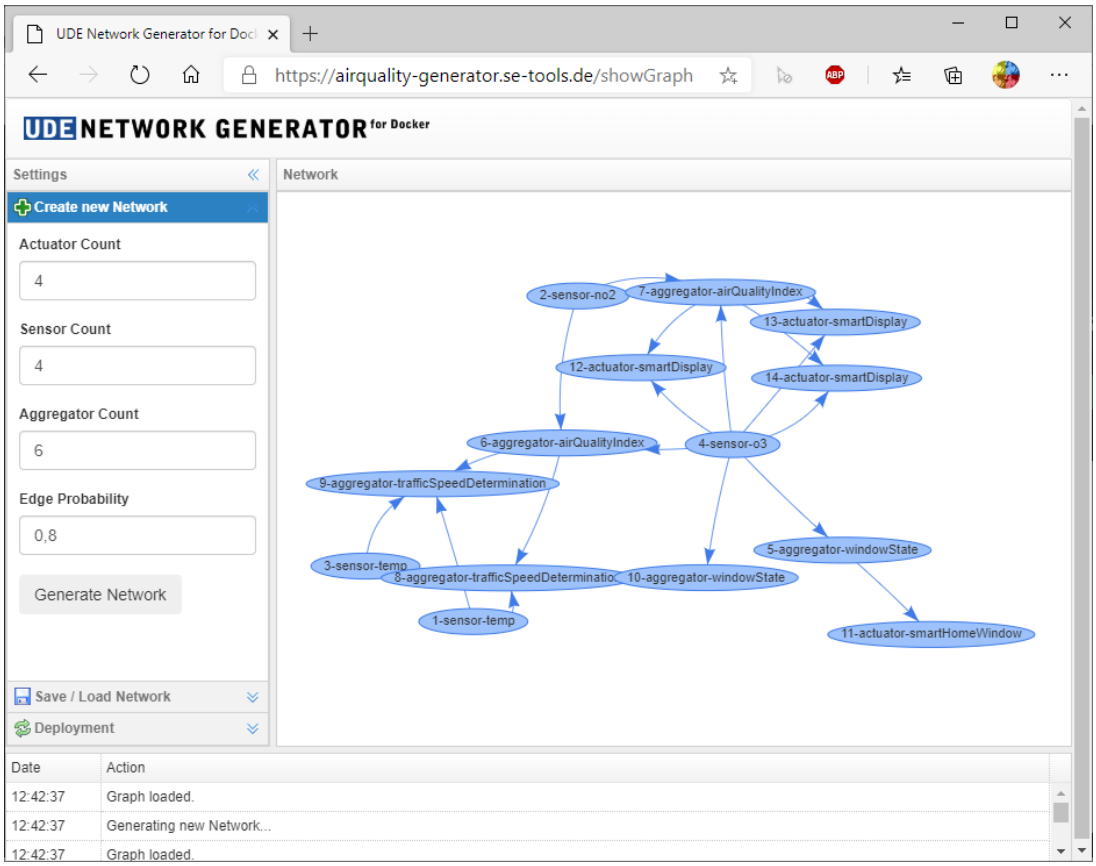


Abbildung 46: Benutzeroberfläche des Network Generator

Ausrollen des Netzwerkes

Um die Knoten aus dem Network Generator automatisch ausrollen zu können, wurde Docker zur Virtualisierung der einzelnen Knoten gewählt. Im Folgenden werden die in Abbildung 47 gezeigte Architektur und die darin enthaltenen Komponenten beschrieben.

Per Knopfdruck kann das generierte Netzwerk ausgerollt werden. Hierbei kommuniziert der Network Generator über die Docker Bridge mit der Rancher-Application

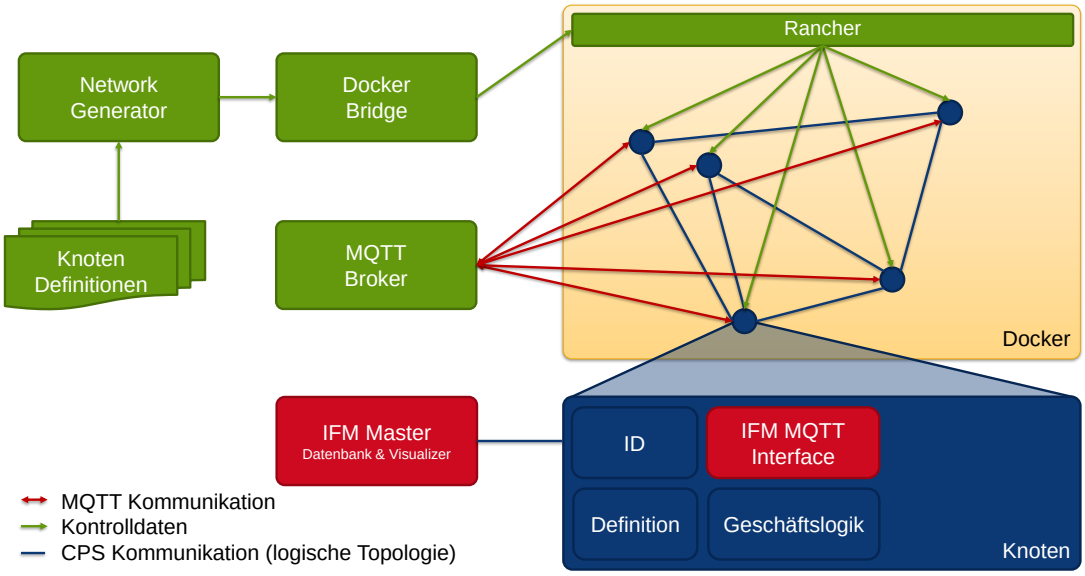


Abbildung 47: Technische Architektur des AirQuality Anwendungsfalls

Programming Interface (API)[43] (vgl. Abbildung 47). Rancher [113] ist ein Verwaltungswerkzeug für Docker, das es erlaubt, über eine REST-API Docker Container auszurollen. Die Docker Bridge stellt hierbei einen Adapter zwischen Network Generator und Rancher bereit. Soll ein Netzwerk ausgerollt werden, führt dies dazu, dass mit Hilfe der Docker Bridge und Rancher auf dem Docker Server für jeden Knoten ein Container erzeugt und ausgeführt wird (blaue Kreise).

Der Network Generator nutzt die Knotentypdefinition, um daraus individuelle Knotenkonfigurationen für jeden Knoten zu erzeugen. Diese Knotenkonfiguration enthält insbesondere auch die Verletzungsinformationen für den jeweiligen Knoten. Eine exemplarische Knotenkonfiguration von Knoten 7 (vgl. Abbildung 46) ist in Abbildung 48 dargestellt. Für jeden Knoten wird eine individuelle Knotenkonfiguration beim Ausrollen des Netzwerkes vom Network Generator über die Docker Bridge an Rancher gesendet. Jeder Knoten erhält so die Konfiguration als Umgebungsvariable und hat Zugriff darauf.

[43] Ein Application Programming Interface (API), ist eine Programmierschnittstelle, über die Software von anderen Programmen gesteuert werden kann. Ebenfalls können über APIs Daten ausgetauscht werden.

```
{
  "id":7,
  "type":"aggregator",
  "template":{
    "nodeType":"aggregator",
    "aggregatorType":"aggregator-airQualityIndex",
    "meta":{
      "measurementUnit":"AQHI",
      "updateRate":1,
      "updateRateUnit":"m"
    },
    "data":{
      "value":3,
      "minValue":1,
      "maxValue":12
    },
    "interest":{
      "input":[
        "sensor-no2/+/value",
        "sensor-pm25/+/value",
        "sensor-o3/+/value"
      ],
      "output":"aggregator-aqhi"
    }
  },
  "label":"aggregator-airQualityIndex",
  "location":"here",
  "pub":[
    "aggregator-aqhi/7/value"
  ],
  "sub":[
    "sensor-no2/2/value",
    "sensor-o3/4/value"
  ],
  "data":{
    "stack":"1st53",
    "giturl":"ssh://git@git.uni-due.de/se-foerderprojekte/cps/airquality/virtualnode.git",
    "gitbranch":"ifm"
  }
}
```

Abbildung 48: Knotenkonfiguration für Knoten 7

Ausführen des Netzwerkes

Die Gesamtheit aller Knotenkonfigurationen ergibt zur Laufzeit der Knoten die im Network Generator erzeugte Topologie. Die Knotenkonfigurationen werden genutzt, um die Software der Knoten zu konfigurieren. Alle Knoten führen die gleiche Software aus, verhalten sich aber wegen ihrer individuellen Konfiguration komplett anders. Durch die Konfiguration wird ein Verhalten vorgegeben, das die Software der Knoten umsetzt.

Da die Kommunikation in diesem CPS über MQTT erfolgt, wird die Vernetzung über die Angabe der abonnierten Kanäle (s. Abschnitt 3.1.5.1) realisiert (vgl.

Z. 28-34). Im vorliegenden Beispiel erhält der Knoten NO_2-Werte von Knoten 2 sowie O_3-Werte von Knoten 4.

Hierbei wird über das sub-Array (Z. 31-34) festgelegt, welche eingehenden Informationen gesammelt werden. Selbst veröffentlicht der Knoten ausgehende Informationen auf dem Kanal *aggregator-aqhi/7/value*. Die Knoten 12, 13 und 14 werden diesen Kanal abonnieren, damit hier die entsprechende Vernetzung sichergestellt ist (vgl. Abbildung 46).

Die Knoten kommunizieren ausschließlich über den MQTT-Broker miteinander. Bei Betrachtung der logischen Kommunikation zwischen zwei Knoten kann dieser jedoch ignoriert werden, da er nur für den Nachrichtentransport zuständig ist. Hieraus ergibt sich die logische Topologie (vgl. Abbildung 47).

Zusätzlich zu den CPS-Komponenten wurden die notwendigen IFM-Komponenten integriert (vgl. Abbildung 47, rot markiert). Jeder Knoten beinhaltet ein IFM-MQTT-Interface, welches die Kommunikation mit anderen Knoten durchführt. Dieses Interface stellt hierbei sicher, dass das IFM-Protokoll eingehalten wird und dass die Abhängigkeitsinformationen erfasst werden. Werden einzelne Knoten über eine Umgebungsvariable als Spy definiert, bauen diese zusätzlich eine Verbindung zum Master auf, um die entsprechend aufgezeichneten Abhängigkeitsinformationen zu übermitteln. Der Master ist hierbei eine eigene Instanz im Netzwerk und beinhaltet eine Datenbank, um die Abhängigkeitsinformationen zu speichern, sowie den Visualizer, um diese Abhängigkeiten visualisieren zu können.

4.1.3 IFM-Evaluierung

Dieser Teil der Evaluierung mit Hilfe des AQ-CPS konzentriert sich auf das IFM-Protokoll. Fokus ist die Funktionsweise des Protokolls in einem echtweltlichen Szenario, die resultierenden aufgezeichneten Daten sowie die Nachverfolgung eines Fehlers zur Quelle. Hierzu wird das zuvor beschriebene Szenario genutzt.

4.1.3.1 Konzept

Beim AirQuality-Anwendungsfall handelt es sich um ein realistisches CPS anhand dessen das IFM-Protokoll auf Funktion evaluiert werden soll. Hierzu soll mit Hilfe des IFM beobachtetes fehlerhaftes Verhalten zur Quelle zurückverfolgt werden.

Im ersten Schritt wird hierzu ein ausreichend großes Netz mit Hilfe des Network Generators erstellt, welches manuell durch die Größe und Verzweigung nicht mehr nachvollziehbar ist.

Im zweiten Schritt wird das Netzwerk mit Spy-Knoten angereichert. Hierzu wird das in Abschnitt 3.2 beschriebene Verfahren genutzt. Mit Hilfe der für dieses Netzwerk passenden Heuristik kann so ein Spy Set ermittelt werden. Die Spy-Funktionalität der einzelnen Knoten kann über eine Docker Environment-Variable eingeschaltet werden.

Im dritten Schritt kann das Netzwerk ausgerollt und hochgefahren werden. Die Knoten beginnen dann zu kommunizieren und tauschen Daten aus. Hierdurch entstehen Abhängigkeiten zwischen den ausgetauschten Informationen.

Im vierten Schritt wird ein Fehlerfall simuliert, dessen Quelle innerhalb eines Experiments durch einen Probanden ermittelt werden soll. Hierzu wird vom Leiter des Experiments ein Sensor zufällig ausgewählt, der fehlerhafte Daten erstellen soll. Hierzu wird dem betroffenen Sensor ein Multiplikator für die Sensorwerte übermittelt, sodass zu hohe Sensorwerte emittiert werden. Anschließend ermittelt der Leiter des Experiments einen von diesem Sensor abhängigen Aktor, der die fehlerhaften Daten mittelbar erhält und somit ein fehlerhaftes Verhalten ausführt. In der realen Welt könnte an diesem Aktor fehlerhaftes Verhalten beobachtet werden.

Im letzten Schritt sollen nun von einem Probanden die gesammelten Daten im IFM-Master genutzt werden, um vom betroffenen Aktor aus die Quelle der fehlerhaften Daten zu ermitteln. Dazu wird dem Probanden mitgeteilt, an welchem Aktor ein fehlerhaftes Verhalten beobachtet wurde. Gelingt es dem Probanden mit Hilfe von Domänenwissen und der IFM-Werkzeuge den fehlerhaften Sensor zu ermitteln, hat das IFM-Protokoll seinen Zweck erfüllt.

Innerhalb des Experiments hat der Proband keine Kenntnis darüber, welcher Sensor mit einem Multiplikator versehen wurde. Seine Aufgabe ist es, diesen zu ermitteln. Das Experiment evaluiert dementsprechend, ob die vorliegenden aufgezeichneten Abhängigkeitsinformationen in Kombination mit Domänenwissen genutzt werden können, um die Fehlerquelle zu identifizieren. Da die Dauer, die zur Ermittlung der Fehlerquelle benötigt wird, stark vom Anwendungsfall, insbesondere der Domäne, der Netzgröße und der Vernetzung abhängig ist, wird hier nur die Effektivität des IFM-Ansatzes evaluiert, nicht aber seine Effizienz.

4.1.3.2 Durchführung

Zu Beginn wurde ein Netzwerk mit 100 Knoten generiert. Dieses besteht aus 30 Sensoren, 40 Aggregatoren und 30 Aktoren. Diese Aufteilung entspricht einer Verteilung, die in diesem Anwendungsfall zu einer sinnvollen Vernetzung der jeweiligen Komponenten führt. Eine andere Verteilung führt dazu, dass sehr viele mögliche Ein- und Ausgaben der Knoten ungenutzt bleiben.

Je mehr Aggregatoren im Netzwerk enthalten sind, desto unübersichtlicher ist die Vernetzung und desto komplexer sind die resultierenden Abhängigkeitsstrukturen. Die Spies können bei der Generierung gemäß des in Abschnitt 3.2 beschriebenen Verfahrens platziert werden. Das resultierende zufällig generierte Netzwerk ist in Abbildung 49 dargestellt. Wie in der Abbildung zu sehen ist, ist bereits ein Netzwerk mit 100 Knoten unübersichtlich und eine manuelle Nachvollziehbarkeit deutlich erschwert. Das resultierende Netzwerk ist also so komplex, dass eine manuelle Nachverfolgung von Abhängigkeiten, kaum möglich ist.

Das Netzwerk wurde anschließend über Rancher in Docker ausgerollt. Die Konfiguration der Container stellen hierbei sicher, dass sich das Netzwerk in einem Normalzustand befindet. Für diesen Anwendungsfall bedeutet dies, dass es wenig Verkehr gibt und die Luftqualität gut ist. Über die Konsole der einzelnen Knoten konnte beobachtet werden, dass die Knoten Informationen untereinander austauschen. Betrachtet man beispielsweise Aggregator 47, der den AQHI berechnet, kann über die Konsole beobachtet werden, welche Informationen der Knoten von anderen erhält und welche er selbst versendet. Die Konsole ist in Abbildung 50 dargestellt. Hierin ist sowohl zu erkennen, dass der Aggregator von verschiedenen Knoten O_3-, PM_{25}- und NO_2-Messwerte erhält (vgl. großer roter Kasten in der

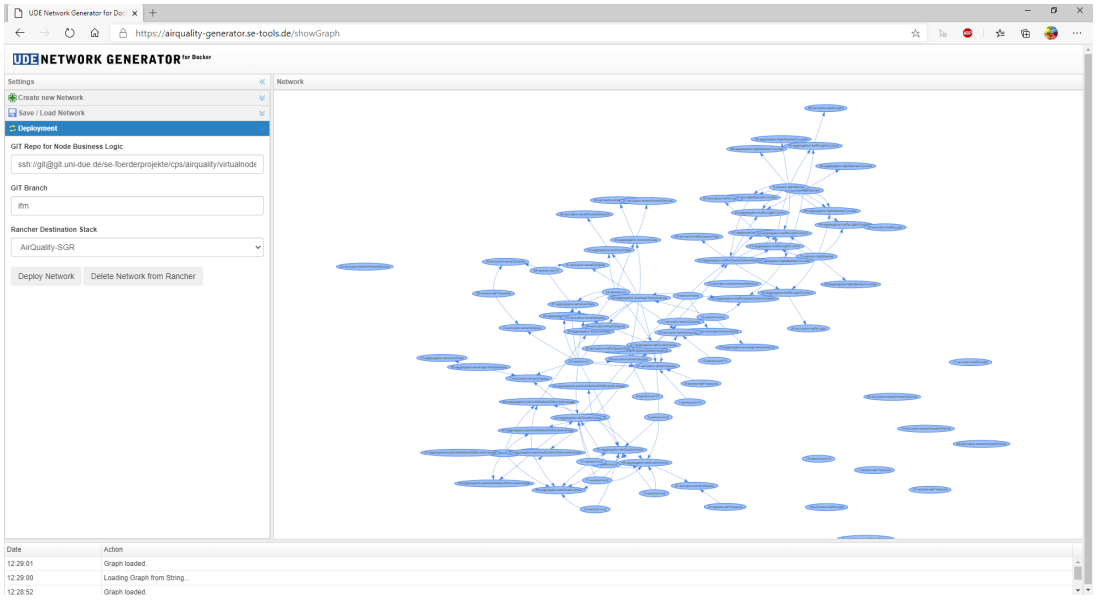


Abbildung 49: Netzwerk zur Evaluation des IFM-Protokolls

Abbildung), als auch, dass der Knoten selbst anschließend den AQHI berechnet und emittiert (vgl. kleiner roter Kasten in der Abbildung). Ebenfalls zu erkennen ist, dass die Kommunikation über das IFM-Protokoll erfolgt, da das entsprechende JSON-Schema (vgl. Abbildung 9) eingehalten wird.

Nachdem das Netzwerk nun hochgefahren ist und seine Arbeit aufgenommen hat, soll der Fehlerfall simuliert werden. Hierzu wurde vom Leiter des Experiments ein Sensor zufällig gewählt, der manipuliert wurde. In diesem Experiment war dies Sensor 29 (ein $PM_{2,5}$-Sensor). Der Sensor wurde mit einem Multiplikator versehen, sodass dieser stark erhöhte Werte ausgab.

Der Leiter des Experiments ermittelte aus der für ihn bekannten Topologie und den Logs der einzelnen Knoten einen zu Sensor 29 abhängigen Knoten. In diesem Fall war dies Aktor 98 (eine Ampel).

Dem Probanden wird nun mitgeteilt, dass Aktor 98 fehlerhaft agiert und unplausible Schaltzeiten der Ampelphasen aufweist. Wäre Aktor 98 eine echtweltliche Ampel, wäre das fehlerhafte Verhalten in der realen Welt sichtbar. Dem Probanden wird ebenfalls mitgeteilt, dass die Ampel selbst fehlerfrei ist und somit fehlerhafte Daten erhalten muss.

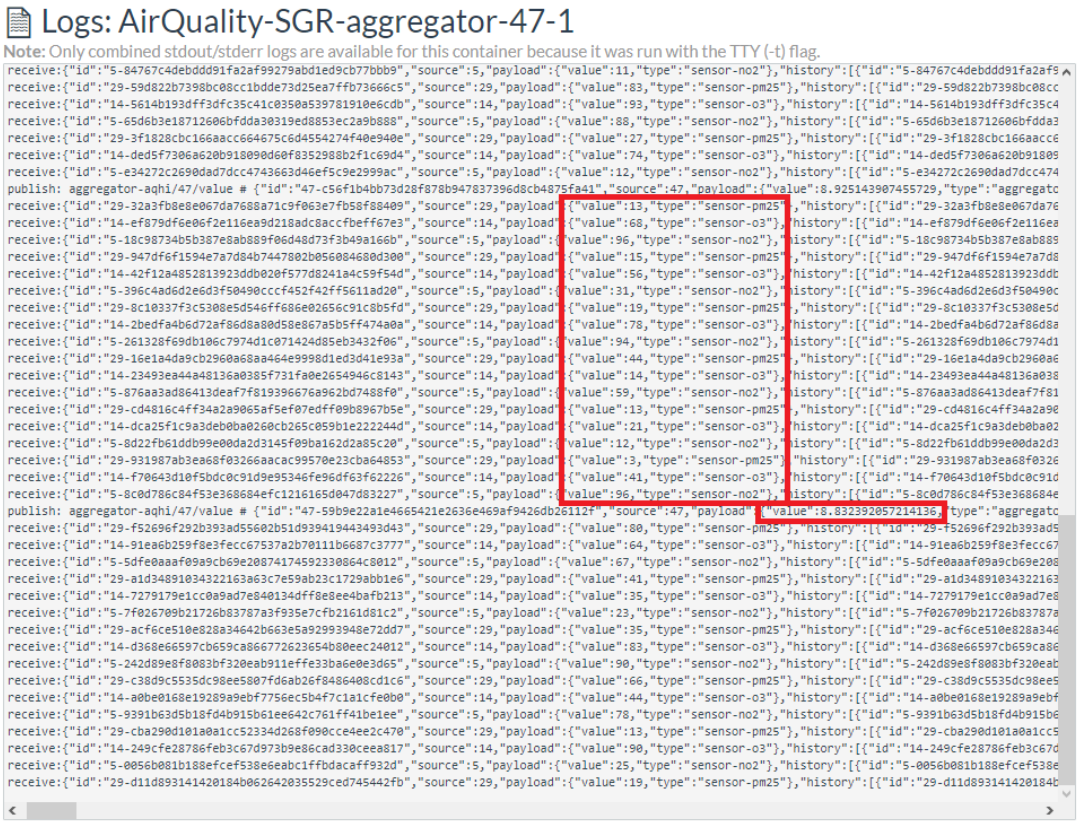


Abbildung 50: Konsole der Aggregators 47

Vorgehen des Probanden

Mit Hilfe des IFM-Visualizers soll der Proband nun herausfinden, wie es zur Fehlfunktion des Aktors kam. Um dies zu analysieren, betrachtet der Proband die Logs des betroffenen Aktors und wählt eine beliebige empfangene Nachricht aus. Diese Nachricht wurde genutzt, um die Abhängigkeiten aufzudecken. Nachricht *44-3585eba1b548d9da79a5e748a7b16933ef147a08* wurde in diesem Fall ausgewählt. Der Nachricht kann man durch die ID schon ansehen, dass sie von Knoten 44 erzeugt wurde. Knoten 98 wird also von Knoten 44 mit Steuerungsdaten versorgt. Der Proband nutzte dann den IFM-Visualizer, um die Abhängigkeiten der Nachricht aufzudecken.

Nach Eingabe der ID der zu untersuchenden Nachricht in den IFM-Visualizer wurde die dem IFM-Master bekannte geladen und angezeigt (vgl. Abbildung 51). Hierbei wurde die Information mit der eingegebenen ID als Wurzel des Abhängigkeitsbaums angezeigt. Der Baum ist so noch recht groß und unübersichtlich, sodass es dem Probanden nicht möglich war, direkt die Fehlerquelle zu identifizieren.

Abbildung 51: Visualizer mit Nachricht 44-3585eba1...

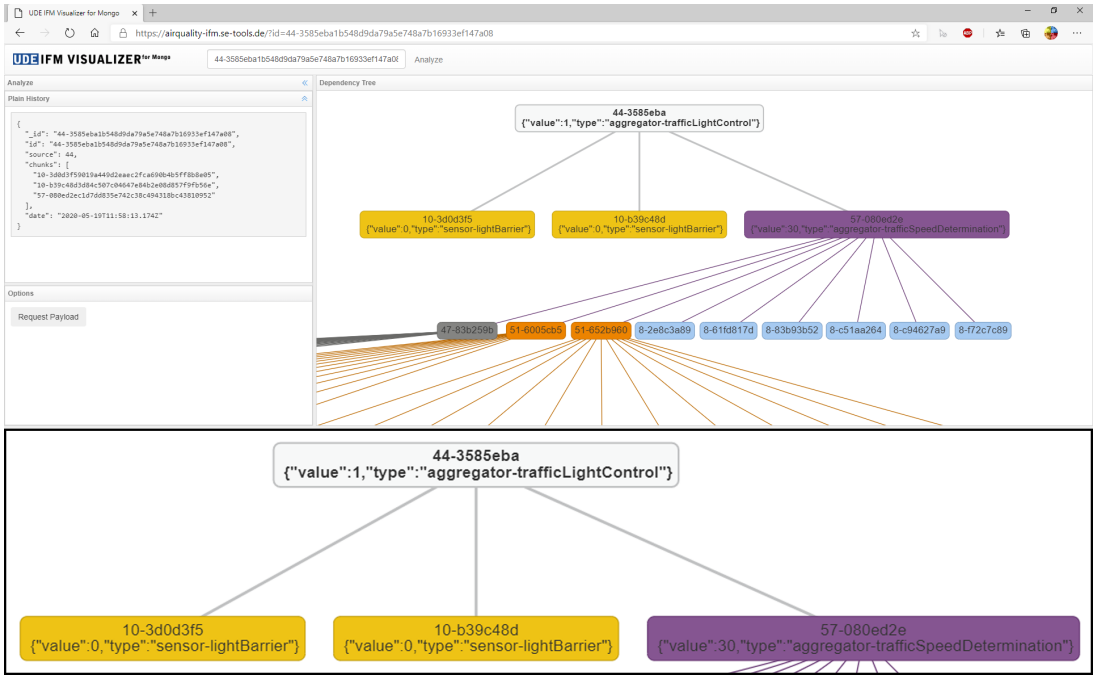


Abbildung 52: Visualizer Nachricht Paket 44-3585eba1..., (Unten: Ausschnittvergrößerung Ebene 1 und 2)

Im Folgenden werden Ausschnittvergrößerungen genutzt, um die Entscheidungsfindung des Probanden zu illustrieren. Der Visualizer kann nun genutzt werden, um die Quelle der fehlerhaften Information aufzudecken. Hierzu wird in die Ansicht gezoomed und mit der Analyse an der Wurzel des Baumes begonnen[44].

Die Aufgabe des Probanden besteht nun darin, den gezeigten Baum von oben nach unten durchzugehen, um die Fehlerquelle zu identifizieren. Hierzu muss der Proband bei jedem Schritt in eine tieferliegende Ebene entscheiden, welche Verzweigung für den Fehler verantwortlich ist. Die Entscheidungen des Probanden sind hierzu im Folgenden dokumentiert.

[44] Je nach Szenario protokolliert das IFM-Protokoll auch die Payload der ausgetauschten Daten. In diesem Beispiel können auch mit Hilfe der *Request Payload* Option im linken Menü einzelne noch aktive Knoten direkt aufgefordert werden, den Payload der entsprechenden Nachricht zu senden. Da der Payload im IFM-Protokoll nicht immer enthalten ist, kann er aber zu Analysezwecken nachträglich durch den Visualizer vom Knoten angefordert werden, falls die Knoten erreichbar sind. Daher sind in diesem Szenario auch die Nutzdaten visualisiert (vgl. Abbildung 52).

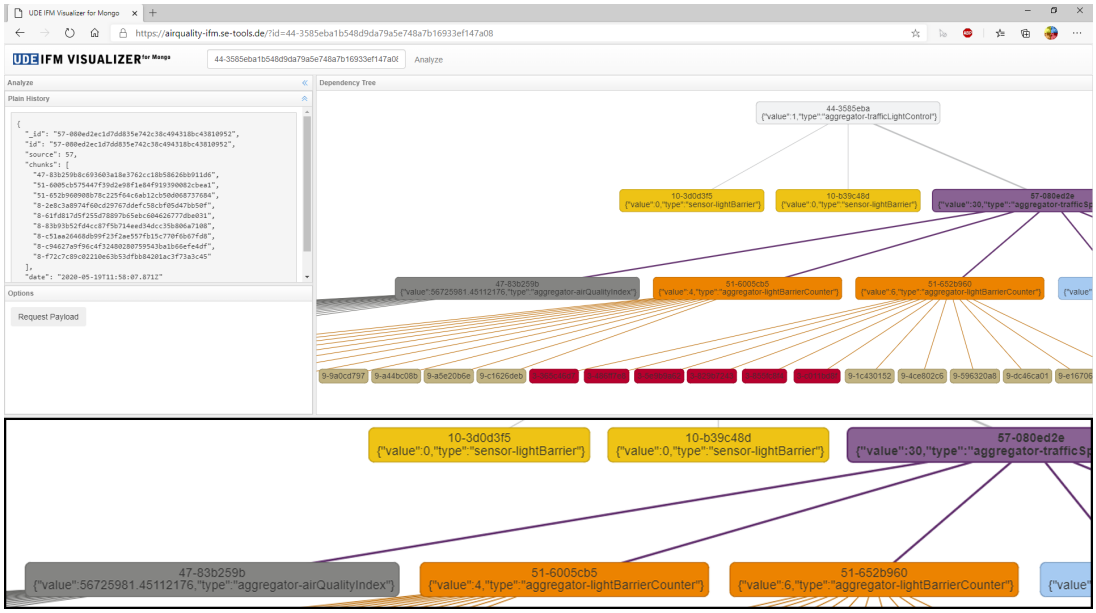


Abbildung 53: Visualizer mit Nachricht 44-3585eba1..., Teilbaum 57-080ed2e... (Unten: Ausschnittvergrößerung Ebene 2 und 3)

Übergang Ebene 1 zu Ebene 2

Mit Hilfe der Visualisierung (vgl. Abbildung 52) kann analysiert werden, welcher Teilbaum für die Fehlfunktion verantwortlich ist. In diesem Fall identifizierte der Proband eine Information von Knoten 57 (Violett). Bei Knoten 57 handelt es sich um einen Aggregator zur Berechnung der zulässigen Höchstgeschwindigkeit, der für die Straße der Ampel zuständig ist. Da dem Probanden durch sein Domänenwissen bekannt ist, dass wenig Verkehr und eine gute Luftqualität herrscht, erscheint die niedrige, erlaubte Höchstgeschwindigkeit von 30 km/h unplausibel. Diese erklärt auch die stark veränderten Schaltzeiten der Ampel.

Übergang Ebene 2 zu Ebene 3

Unklar ist jedoch, ob die Fehlfunktion nun bei Knoten 57 liegt, oder ob dieser bereits fehlerhafte Informationen erhalten hat. Hierzu betrachtete der Proband die nächste Ebene und schaute sich die Abhängigkeiten der Information von Knoten 57 an (vgl. Abbildung 53). Der Lichtschrankenzähler Knoten 57 (Orange) hat zwei Werte 4 und 6 gemeldet und die Temperatursensoren (Hellblau) lieferten ebenfalls plausible Werte. Dem Probanden fällt jedoch auf, dass Knoten 47 (Grau) zur Berechnung des Luftqualitätsindex einen AQHI von 56725981 bestimmt. Dies ist

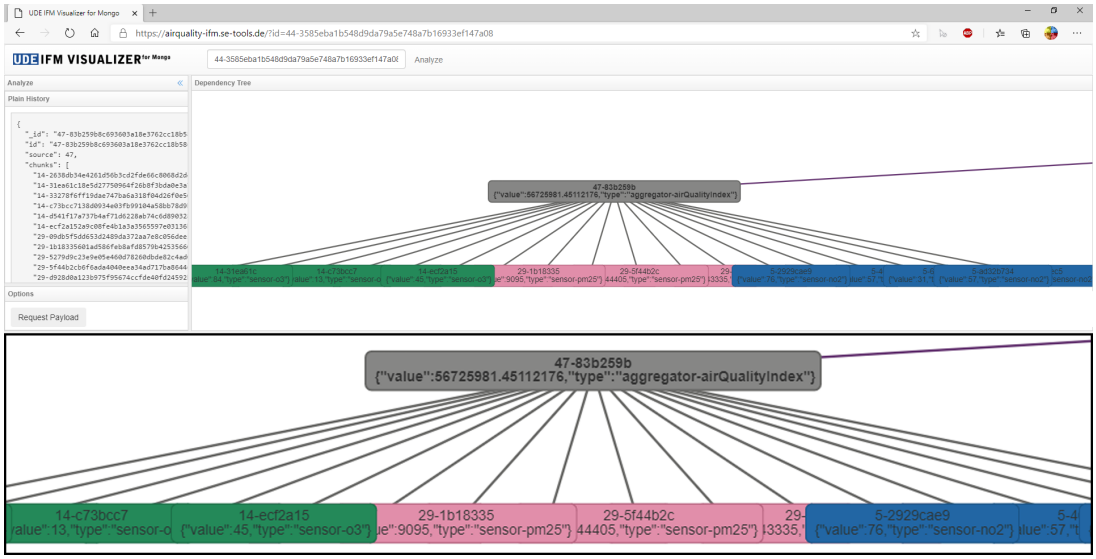


Abbildung 54: Visualizer mit Nachricht 44-3585eba1..., Teilbaum 47-83b259b... (Unten: Ausschnittvergrößerung, Ebene 3 und 4)

zwar ein zulässiger Wert, da die Skala nach oben nicht begrenzt ist, jedoch handelt es sich bereits bei Werten oberhalb von 12 um extrem schlechte Luftqualitätswerte. Daher wurde dieser Wert vom Probanden als fehlerhaft eingestuft.

Übergang Ebene 3 zu Ebene 4

Es wurden daher die Abhängigkeiten der von Knoten 47 erzeugten Nachricht (Grau, vgl. Abbildung 54) betrachtet. Diese Information hat Abhängigkeiten zu Nachrichten von Knoten 14 (Grün), 29 (Rosa) und 5 (Dunkelblau). Auffällig waren die extrem hohen $PM_{2,5}$-Werte von Knoten 29. Da diese keine weiteren Abhängigkeiten hatten, handelte es sich hier um von Knoten 29 erzeugte Daten. Daher war dieser Knoten ebenfalls verantwortlich für die bei der Ampel angelangten falschen Steuerdaten. Knoten 29 wurde daher vom Probanden als fehlerhaft identifiziert und kann repariert oder aus dem Netz entfernt werden.

4.1.3.3 Fazit

Die Evaluation des IFM-Protokolls hat gezeigt, dass es möglich ist, in einem großen verzweigten CPS (s. Abbildung 49) Abhängigkeiten über mehrere Knoten und Hierarchieebenen hinweg nachzuverfolgen. Gleichzeitig ist es gelungen, mit

Hilfe der Baumdarstellung der Abhängigkeiten im IFM-Visualizer fehlerhafte Knoten zu identifizieren. Der Visualizer ist somit ein Debugging-Werkzeug, das ausgetauschte Daten, die im IFM-Master gesammelt wurden, auswertbar macht. Damit dies möglich ist, benötigt der Benutzer des IFM-Visualizers allerdings Domänenwissen, damit fehlerhafte Werte im dargestellten Abhängigkeitsbaum auch erkannt werden können. Ohne dieses Domänenwissen kann der Pfad von der Wurzel des Knotens bis zur Quelle des Fehlers nur mit hohem Aufwand identifiziert werden.

4.1.4 Information-Dependency-Modeling-Fallstudie

Das Dependency Modeling soll es ermöglichen, einzelne Knoten im Netzwerk, die sich nicht an das IFM-Protokoll halten, zu überbrücken. Ohne das Modeling würden an dieser Stelle Abhängigkeitsinformationen verloren gehen, da Black-Box-Knoten die mitgeteilten Abhängigkeiten ohne Protokollunterstützung nicht weitergeben können.

Im Folgenden wird eine kurze Fallstudie beschrieben, die in einem kleinen CPS durchgeführt wurde. Die Fallstudie zeigt wie mit Hilfe des Dependency Modelings in einem echten CPS Black-Box-Knoten durch Modellierung überbrückt werden können.

4.1.4.1 Versuchsaufbau

Mit Hilfe des bereits zuvor gezeigten Network Generators wird ein kleines CPS bestehend aus 9 Knoten generiert. Es gibt 3 Sensoren (Knoten 1-3), 3 Aggregatoren (Knoten 4-6) und 3 Aktoren (7-9). Das Netzwerk ist in Abbildung 55 dargestellt. Das Netzwerk wurde anschließend in der bekannten Docker-Umgebung ausrollt. Die Sensoren, Aggregatoren und Aktoren begannen wie gehabt mit der Generierung von Daten.

In dieser Fallstudie ist Knoten 6 ein Black-Box-Knoten. Knoten 7 wird von Knoten 6 gesteuert. Knoten 7 erhält jedoch keine Abhängigkeitsinformationen von 6, da dieser sich nicht an das IFM-Protokoll hält und die für die ausgehenden Informationen genutzten Eingaben nicht bekannt gibt. In dieser Fallstudie soll

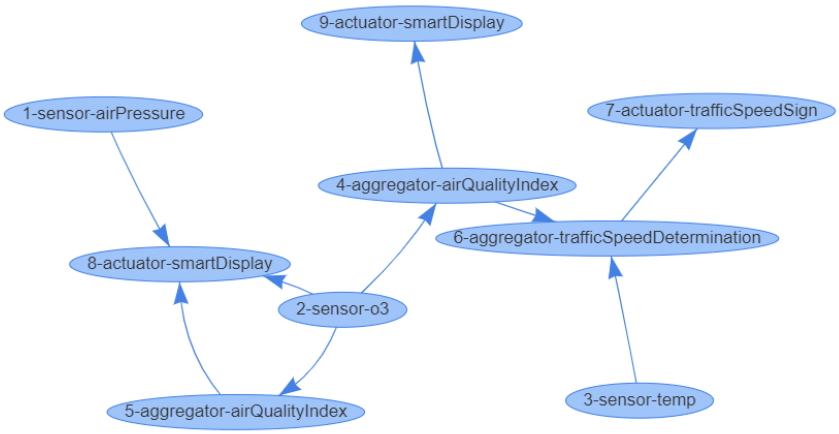


Abbildung 55: Beispiel-CPS mit Black-Box-Knoten 6

dennoch ein vollständiger Abhängigkeitsbaum – auch über Knoten 6 hinweg – erzeugt werden. Daher werden die Knoten 3 und 4 als Spy definiert, da diese Informationen an den Black-Box-Knoten senden. Auf diese Weise kennt der IFM-Master alle von 3 und 4 erzeugten Nachrichten. Das sind die Eingaben in die Black-Box. Durch das IFM-Protokoll an anderen Stellen im Netzwerk wird sichergestellt, dass abhängige Informationen dieser Nachrichten bekannt sind.

Knoten 7 erhält jedoch nur das Ergebnis von Knoten 6. In diesem Fall sind für die von Knoten 7 empfangenen Nachrichten keine Abhängigkeitsinformationen verfügbar, da der IFM-Master die Beziehung zwischen den von 6 ausgegebenen Daten und den Eingaben in 6 von Knoten 3 und 4 nicht kennt. Diese Beziehung wird nun modelliert. Um dies tun zu können, muss der Modellierer Kenntnis über die Geschäftslogik zur Generierung der Ausgabe von 6 haben. In dieser Studie wird davon ausgegangen, dass Knoten 6 die letzten 4 empfangenen Nachrichten verwendet, um seinen eigenen Output für 7 zu erzeugen. Diese Beziehung wird in Abbildung 56 beschrieben (vgl. Abschnitt 3.3).

```
1  {
2    find: "history",
3    filter:{"date":{$lte:givenPacket.date},
4      $or:[{"source":3}, {"source":4}]},
5    sort: {"date": -1},
6    limit: 4
7  }
```

Abbildung 56: MongoDB Query

Auf diese Weise werden alle Vorbereitungen für das Netzwerk getroffen, um das Dependency Modeling zu testen. Es wurde eine Topologie definiert, die Abhängigkeiten zwischen den Komponenten im Netzwerk erzeugt. Zusätzlich wurde eine zentrale Komponente (Knoten 6) als Black-Box-Knoten definiert. Dieser Knoten unterstützt das IFM-Protokoll nicht und teilt nicht mit, welche Daten er als Eingabe verwendet, um eine Ausgabe zu erzeugen. Es wurden alle liefernden und empfangenden Knoten dieses Black-Box-Knotens als Spy definiert, damit Ein- und Ausgaben überwacht werden können. Als letztes wurden die Abhängigkeiten mit Hilfe des Dependency Modeling als Verbindung zwischen Ein- und Ausgängen definiert.

4.1.4.2 Durchführung

Nach dem Start der einzelnen Docker Container, die jeweils einen Knoten repräsentieren, beginnen die Sensoren mit der Ausgabe von Sensordaten. Es konnte beobachtet werden, dass die Knoten 3 und 4 Abhängigkeitsinformationen an den Master liefern. Knoten 7 wurde ebenfalls als Spy definiert und stellt daher Informationen über empfangene Informationen bereit, kennt jedoch nicht deren Abhängigkeiten, da die Informationen vom Black-Box-Knoten nicht geliefert wurden.

Um die Abhängigkeiten aufzudecken, wurde der IFM-Visualizer genutzt, dem die gezeigte Modellierung (vgl. Abbildung 56) bekannt ist. Als Eingabe wurde eine von Knoten 7 empfangene Information (siehe Abbildung 57, oberster Knoten im Baum) genutzt. Der IFM-Visualizer nutzt die modellierten Abhängigkeiten und führt die aufgezeichneten Daten mit den modellierten Abhängigkeiten zusammen. Das Ergebnis ist ein Abhängigkeitsbaum, der sich über den Black-Box-Knoten hinweg erstreckt. An der Spitze steht eine von Knoten 7 empfangene Nachricht, die von Knoten 6 erzeugt wurde. Durch die Modellierung stehen 4 Nachrichten auf der nächsten Ebene.

In Abbildung 57 ist der Abhängigkeitsbaum dargestellt. Die Verbindung der obersten Nachricht mit der darunterliegenden mittleren Ebene wurde modelliert. Die Verbindung der mittleren mit der untersten Schicht wurde aufgezeichnet.

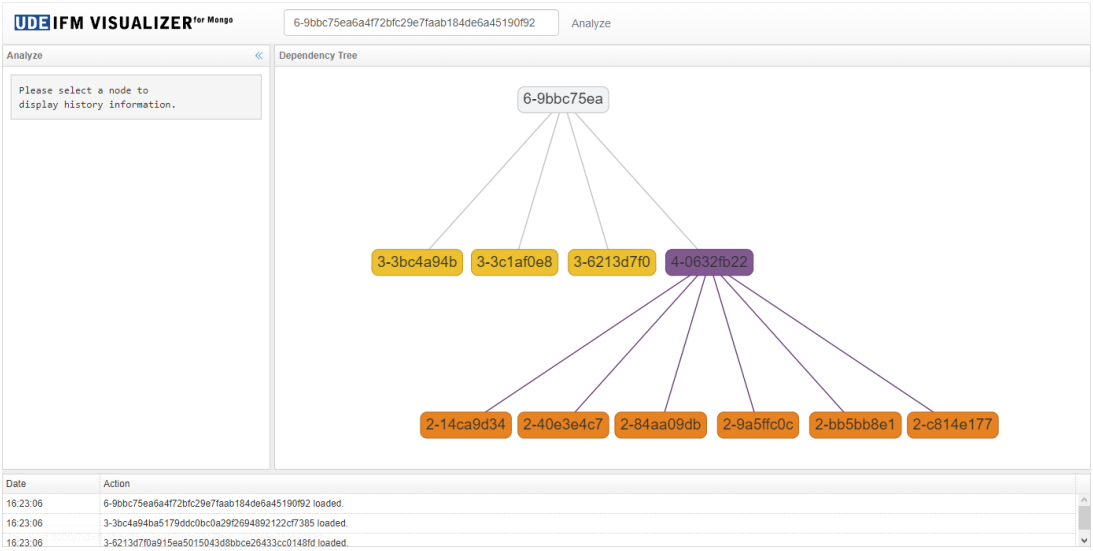


Abbildung 57: IFM-Visualizer für Beispiel-CPS

Auf der unteren Ebene stehen die aufgezeichneten Abhängigkeiten dieser Nachrichten. Es entsteht also ein kontinuierlicher Abhängigkeitsbaum, der unten mit den Daten von Sensor 1 beginnt und mit einem durch Knoten 6 erzeugten Datenfragment endet. Es handelt sich um einen zusammengeführten Baum aus aufgezeichneten und modellierten Abhängigkeiten.

4.1.4.3 Fazit

Das Ziel des Dependency Modeling ist es, Abhängigkeiten auch dann zu überwachen, wenn das Netzwerk Black-Box-Knoten enthält. Die Fallstudie hat gezeigt, dass es das Dependency Modeling möglich macht, Knoten in diesen Ansatz einzubeziehen, die das IFM-Protokoll nicht unterstützen, solange Informationen oder Annahmen über die Geschäftslogik der Black-Box-Knoten für die Modellierung zur Verfügung stehen. Damit kann die Abdeckung des Netzwerkes durch den IFM erhöht werden.

4.2 Performanz-Evaluierung

Da das IFM-Protokoll zusätzliche Daten erzeugt und über das Netzwerk versendet, nimmt die Gesamtperformanz des CPS durch den Einsatz des IFM ab. Um einen

performanten Einsatz des IFM zu gewährleisten, muss bekannt sein, wie viel zusätzlicher Ressourcenbedarf entsteht. Zusätzlich kann durch diese Einschätzung auch besser entschieden werden, ob der Einsatz des IFM zweckdienlich ist oder ob die Kosten des zusätzlichen Ressourcenbedarfs den Nutzen übersteigen. Im Folgenden wird hierzu die Performanz und der Ressourcenbedarf des IFM untersucht. [114][45]

4.2.1 Versuchsaufbau

Für dieses Experiment wird die bereits beschriebene Docker-Umgebung mit virtualisierten Knoten verwendet. Dies hat den Vorteil, dass jeder Knoten beobachtet werden kann und verschiedene Leistungsmetriken gemessen werden können.

Das Ziel des Experiments ist die Messung der folgenden Werte:

- **Nachrichtengröße:** Durch die Verwendung des IFM-Protokolls werden zusätzliche Informationen über Abhängigkeiten innerhalb des Netzwerks gesendet. Dadurch wird die Größe der einzelnen Nachrichten erhöht. Es soll gemessen werden, wie sich die Nachrichtengröße verschiedener Knotentypen wie Sensoren oder Aggregatoren verändert.

- **Bearbeitungszeit:** Beim Senden von Nachrichten entsprechen Sensoren und Aggregatoren dem IFM-Protokoll. Aggregatoren geben auch an, welche empfangene Information zur Erstellung einer ausgehenden Information verwendet wurde. Dies erfordert zusätzliche Arbeit im Knoten, was zusätzliche Verarbeitungszeit erfordert. Es wird gemessen, um wie viel die Verarbeitungszeit einer zu sendenden Nachricht zunimmt, wenn das IFM-Protokoll verwendet wird.

- **Ausgangsleistung:** In der Regel versenden Sensoren und Aggregatoren Informationen mit einer definierten Rate und versenden nicht so viele

[45] Dieser Beitrag wurde veröffentlicht in:
Stefan Gries und Volker Gruhn. „Performance Evaluation of the Information Flow Monitor Protocol in Cyber-Physical Systems“. In: *Frontiers in Artificial Intelligence and Applications*. Hrsg. von Hamido Fujita, Ali Selamat und Sigeru Omatu. IOS Press, Sep. 2020. ISBN: 978-1-64368-114-6. DOI: 10.3233/FAIA200582

Nachrichten, wie technisch möglich ist. Die Leistung eines einzelnen Knotens kann jedoch auch an der maximalen Verarbeitungsmenge von Nachrichten gemessen werden. Dazu soll ermittelt werden, wie viele Nachrichten weniger ein Knoten in einem definierten Zeitrahmen versenden kann, wenn das IFM-Protokoll verwendet wird.

Für die Messung der Nachrichtengröße werden mehrere Netzwerke verwendet, je Netzklasse (vgl. Tabelle 2, Seite 136) eines. Für die Messung der Bearbeitungszeit und Ausgangsleitung wird ein Netzwerk mit 4 Sensoren, 4 Aggregatoren und 4 Aktoren verwendet. Das Netzwerk ist in Abbildung 58 dargestellt. An dieser Stelle wurde nur ein recht kleines Netzwerk verwendet, damit jedem Knoten des Netzwerkes im Experiment ein eigener Core der Virtualisierungsumgebung zugewiesen werden konnte. Hierdurch ist sichergestellt, dass jeder Knoten eine definierte Leistung erhält, die unabhängig vom Leistungsbedarf anderer Knoten zur Verfügung steht. Da dies für die Messung der Nachrichtengröße keine Relevanz hat, können dort größere Netzwerke verwendet werden.

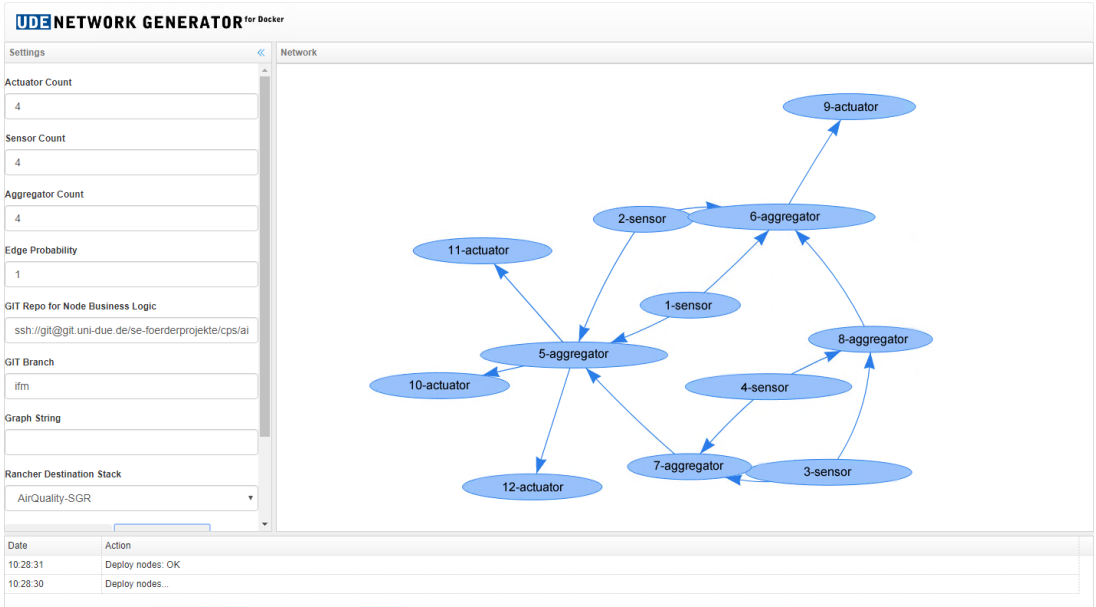


Abbildung 58: Netzwerk zur Evaluation der Bearbeitungszeit und Ausgangsleistung

Durchgeführt wurde das Experiment auf einem vServer einer VMware-Umgebung. Dem vServer standen hierbei 12 Cores und 8 GB Arbeitsspeicher zur Verfügung. Bei der Messung der Nachrichtengröße konnten sich die auf dem vServer ausgeführten Container frei an den Ressourcen bedienen. Bei der Messung der

Ausgangsleistung und Bearbeitungszeit erhielt jeder Sensor und Aggregator einen eigenen Core und 0,5 GB Arbeitsspeicher. Auch wenn die Arbeitsspeichermenge gering erscheint, wurde diese in keinem der Experimente ausgeschöpft, da auf den Knoten sehr leichtgewichtige Software ausgeführt wurde. Bei allen Experimenten wurde nach dem Hochfahren der Container eine Wartezeit von 10 Minuten vor den Messungen abgewartet, damit sich das Netz stabilisiert.

4.2.2 Durchführung

4.2.2.1 Nachrichtengröße

Sensoren und Aggregatoren in CPS versenden Nachrichten. Aktoren sind in der Regel Datensenken und konsumieren Daten. Da Aktoren keine eigenen Nachrichten senden, werden hier nur die Sensoren und Aggregatoren berücksichtigt.

In diesem Experiment wurde für die Netzklassen 10S10A, 30S30A und 40S40A jeweils ein Netzwerk generiert. Durch die in Abschnitt 4.1.2 beschriebene Vernetzung des generierten CPS gehörten die Netzwerke jeweils zu den Buckets mit den kürzesten Pfadlängen. Aus Tabelle 2 geht daher hervor, dass für alle generierten Netzwerke dieses Experiments die Konfiguration [[ROLES,3]] genutzt werden kann, da hier die entsprechenden durchschnittlichen Pfadlängen vorliegen.

Im Folgenden werden die Ergebnisse exemplarisch für das Netzwerk der Klasse 10S10A vorgestellt, bevor anschließend die Ergebnisse für die verschiedenen Klassen in einer Übersicht dargestellt werden. Für die Messung der Nachrichtengrößen wurden alle 100 Knoten gleichzeitig gestartet und die ausgetauschten Nachrichten für den Zeitraum von 10 Minuten beobachtet. Die Knoten kommunizieren über MQTT. Dieser Prozess wurde mit verschiedenen Spy-Knoten-Mengen durchgeführt.

Der erste Durchlauf wurde komplett ohne Spy-Knoten durchgeführt, sodass die Nachrichtengröße auf den Pfaden bis zum letzten Knoten ansteigen. IFM-Netzwerke ganz ohne Spy-Knoten können zwar keine Abhängigkeiten aufzeichnen, bzgl. ihrer Nachrichtengröße lässt sich jedoch so die obere Schranke für die Größe der Nachrichten in diesem Netzwerk ermitteln. Der Durchlauf mit 0 Spy-Knoten repräsentiert stellvertretend Netze mit extrem wenigen Spy-Knoten.

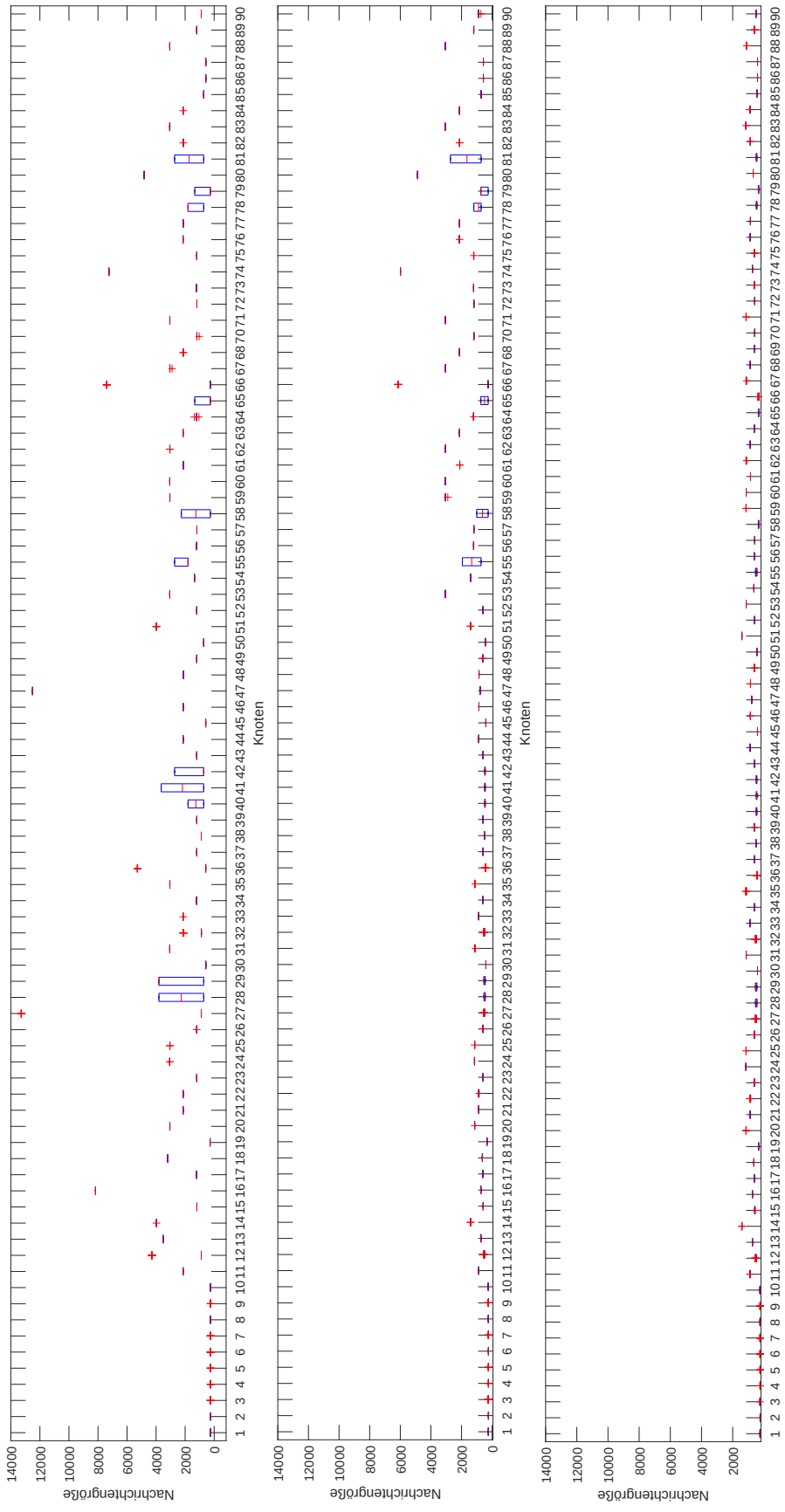


Abbildung 59: Box-Plot zur Größe des Overheads durch das IFM-Protokoll ohne Spies (oben), mit 42 Spies (mittig) und mit 80 Spies (unten) in einem Netzwerk der Klasse 10S10A

Der zweite Durchlauf wurde mit der für dieses Netz empfohlenen Menge an Spy-Knoten (vgl. Tabelle 2), 42 Stück, durchgeführt. Diese wurden gemäß der genannten Konfiguration im Netzwerk verteilt.

Der dritte Durchlauf wurde mit 80 Spy-Knoten durchgeführt, sodass jeder Aggregator ein Spy-Knoten war. Wenn jeder Aggregator ein Spy-Knoten ist, müssen Abhängigkeitsinformationen nicht lange mitgeführt werden, ehe sie zum Master gesendet werden. Dieser Durchlauf ist dementsprechend als untere Schranke für die Nachrichtengröße zu verstehen.

Alle ausgetauschten Nachrichten wurden in Bezug auf ihre Größe gemessen. Die Ergebnisse der drei Durchläufe sind in Abbildung 59 dargestellt. Knoten 1-10 sind hierbei die Sensoren, während Knoten 11-90 Aggregatoren sind. Die Aktoren sind nicht dargestellt, da diese keine Nachrichten emittieren. Dargestellt ist jeweils der Overhead[46] der Nachricht durch das IFM-Protokoll je Knoten als Box-Plot.

Der durchschnittliche Overhead für die Sensoren beträgt 250 Byte. Dies ist in allen drei Durchläufen gleich, da sich die Durchläufe nur durch die Anzahl ihrer Spies unterscheiden. Dies hat keinen Einfluss auf die Sensoren, die selbst keine Spy-Knoten sind und immer am Anfang der Pfade liegen. Im ersten Durchlauf, ganz ohne Spy-Knoten, ergibt sich über alle Aggregatoren hinweg ein durchschnittlicher Overhead von 1758 Byte. Durch den Einsatz von 42 Spy-Knoten (Knoten 11-52) im zweiten Durchlauf lässt sich dieser Wert auf 1034 Byte reduzieren. Beim Einsatz von 80 Spy-Knoten im dritten Durchlauf sinkt der durchschnittliche Overhead der Aggregatoren auf 613 Byte. Je mehr Spy-Knoten im Netzwerk eingesetzt werden, desto geringer ist der Overhead durch das IFM-Protokoll, da Abhängigkeitsinformationen zügiger an den Master übergeben werden können und nicht über viele Knoten hinweg mittransportiert werden müssen. Der Unterschied des Overheads je Knotentyp resultiert aus der Tatsache, dass Nachrichten von Sensoren selbst keine Abhängigkeiten haben, die durch das Protokoll mitgeteilt werden müssen. Daher fallen Nachrichten von Aggregatoren tendenziell größer aus.

[46] Der Overhead durch das IFM-Protokoll ist definiert als Differenz aus Gesamtnachrichtengröße und Nutzlast.

In den Beispielnetzen erzeugen sowohl die Sensoren als auch die Aggregatoren einen Integer oder Float als Messwert oder Ausgabeinformation. Daraus ergibt sich eine sehr geringe Nutzlast, da nicht viele Informationen übertragen werden müssen. Die durchschnittliche Größe einer übertragenen Nachricht ohne Verwendung des IFM-Protokolls beträgt in unserem Beispiel nur 15 Byte. Daher wäre hier der Anteil des Overheads am Gesamtvolumen sehr groß. Der Overhead des IFM-Protokolls ist jedoch unabhängig zur Größe der Nutzlast[47]. Für große Nachrichten sinkt daher der Anteil des Overheads.

Die Ergebnisse für alle untersuchten Netzklassen sind in Tabelle 3 dargestellt. Hierbei werden einige Zusammenhänge deutlich, die unabhängig von der Netzklasse gelten:

- Je mehr Spy-Knoten eingesetzt werden, desto geringer ist der durchschnittliche Overhead. Dies liegt darin begründet, dass Abhängigkeitsinformationen schneller an den Master abgegeben werden können und nicht mittransportiert werden müssen.
- Je mehr Sensoren die Aggregatoren mit Daten versorgen, desto größer ist der Overhead ausgehender Nachrichten der Aggregatoren. Der Grund hierfür ist, dass die Nachrichten der Aggregatoren dann auch mehr Abhängigkeiten zu eingegangenen Nachrichten haben, die im Protokoll benannt werden.
- Aggregatoren erzeugen einen größeren Overhead als Sensoren, da Sensoren keine Abhängigkeiten haben.

4.2.2.2 Bearbeitungszeit

Wenn ein Knoten neue Nachrichten generiert, ist zusätzliche Zeit erforderlich, um die Metadaten für das IFM-Protokoll zu generieren. Zusätzliche Zeit wird für Aggregatoren benötigt, da alle Abhängigkeiten der zu sendenden Nachricht gesammelt und aufgelistet werden müssen. In diesem Teil des Experiments wurde die Zeit gemessen, die ein Knoten benötigt, um eine Nachricht zu erstellen und

[47] Dies gilt nur, wenn das Protokoll nicht auch die Nutzlast im Master sammelt.

Netzklasse	Spy-Knoten-Anzahl	Durchschn. Overhead Sensoren	Durchschn. Overhead Aggregatoren	Durchschn. Overhead Spy-Knoten	Durchschn. Overhead alle Knoten	Durchschn. Overhead Netzklasse
10S10A	0	250	1758	-	1235	829
	42	250	1034	604	763	
	80	250	613	613	488	
30S30A	0	251	2886	-	1074	743
	40	251	767	767	411	
40S40A	0	252	5414	-	787	575
	20	252	1326	1326	363	

Tabelle 3: Überblick über die Nachrichtengrößen in Byte für die verschiedenen Netzwerkklassen

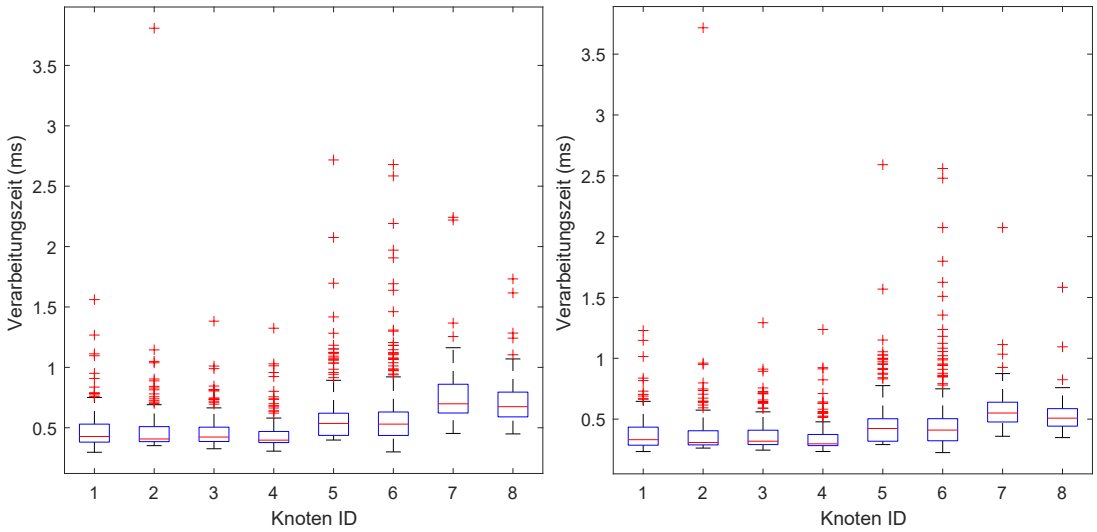


Abbildung 60: Box-Plots zur Knoten-Verarbeitungszeit mit (links) und ohne (rechts) IFM-Protokoll

zu senden. Gemessen wurde die Zeit zwischen dem Befehl, dass die Nachricht versendet werden soll und dem Zeitpunkt, an dem die Nachricht gesendet wurde.

Die durchschnittliche Zeit für die Verarbeitung einer zu sendenden Nachricht beträgt in unserem Beispiel 0,522 Millisekunden, wenn das IFM-Protokoll verwendet wird. Ohne Verwendung des IFM-Protokolls beträgt die durchschnittliche Verarbeitungszeit 0,408 Millisekunden. Die Auswirkungen der Verwendung des IFM-Protokolls in den einzelnen Knoten sind in Abbildung 60 und Tabelle 4 dargestellt.

Die Unterschiede der Boxplots sind nur schwer erkennbar, da die durchschnittliche Verarbeitungsdauer durch den Verzicht auf das IFM-Protokoll nur um 21,839 % abnimmt (von 0,522 ms auf 0,408 ms). Die prozentuale Abnahme der Verarbeitungszeit betrifft Sensoren (21,522 %) und Aggregatoren (22,045 %) ähnlich, während die absoluten Werte erkennen lassen, dass Aggregatoren (0,138 ms) stärker als Sensoren (0,099 ms) von der Abnahme der Verarbeitungszeit betroffen sind.

4.2.2.3 Ausgangsleistung

Zusätzlich wurde gemessen, wie viele Nachrichten ein Knoten mit und ohne Verwendung des IFM-Protokolls senden kann. Dazu werden die Knoten so konfi-

Knoten	Typ	ohne IFM	mit IFM	ohne IFM	mit IFM	ohne IFM	mit IFM
1	Sensor	0,375	0,472	0,361	0,460	0,408	0,522
2		0,366	0,468				
3		0,363	0,463				
4		0,339	0,438				
5	Aggregator	0,457	0,585	0,488	0,626		
6		0,484	0,610				
7		0,602	0,799				
8		0,552	0,744				

Tabelle 4: Überblick über die durchschnittlichen Verarbeitungszeiten in Millisekunden für die verschiedenen Knoten des Netzwerkes

guriert, dass sie unmittelbar nach dem Senden einer Nachricht mit der nächsten Nachricht beginnen. Über einen Zeitraum von zwei Minuten wurde gemessen, wie viele Nachrichten gesendet werden können. Die Ergebnisse werden in Tabelle 5 gezeigt. Je nach Typ und Aufgabe ergibt sich für die einzelnen Knoten eine Leistungszunahme zwischen 0,35 % und 10,50 %, wenn auf das IFM-Protokoll verzichtet wird. Für alle Knoten ergibt sich eine durchschnittliche Leistungszunahme von 4,93 %.

Während des Experiments konnte beobachtet werden, dass die einzelnen Knoten die zur Verfügung stehenden Ressourcen der CPU ausschöpften. Die genutzte Netzwerkbandbreite lag im Mittel bei 0,510 MB/s. Daher ist davon auszugehen, dass in diesem Experiment die CPU-Leistung der limitierende Faktor war. Zu bedenken ist allerdings, dass die Ausführung der einzelnen Knoten innerhalb von Containern auf demselben vServer zu kurzen Latenzen und hohen Bandbreiten führt. Während die Bandbreite kaum ausgeschöpft wurde, spielt die Latenz eine wesentliche Rolle für den Kommunikationsaufbau. Daher ist davon auszugehen, dass in Netzwerken mit hohen Latenzen die Ausgangsleistung im Wesentlichen hierdurch beeinflusst wird. Dieses Experiment fokussierte allerdings die Leistung der Knoten selbst, da das Problem der Latenzen unabhängig vom IFM-Konzept jegliche Netzwerkkommunikation betrifft und durch den IFM nicht beeinflusst wird.

Knoten	1	2	3	4	5	6	7	8	Total
mit IFM	42644	46043	46185	45701	44777	45551	44556	45588	361045
ohne IFM	43463	48692	48983	48898	47058	45710	49235	47748	379787
Zunahme	1.92%	5.75%	6.06%	7.00%	5.10%	0.35%	10.50%	4.74%	4.93%

Tabelle 5: Maximale Ausgangsleistung (gemessen in Anzahl der Nachrichten) der einzelnen Knoten in einem 2-Minuten Zeitfenster mit und ohne IFM-Protokoll sowie Leistungszunahme durch Abschalten des IFM-Protokolls (gemessen in Prozent)

4.2.3 Fazit

Das IFM-Protokoll wirkt sich sowohl auf die Nachrichtengröße als auch auf die Ausgangsleistung und Bearbeitungszeit aus.

Der absolute Overhead des Protokolls ist unabhängig von der Größe der übertragenen Daten. Bei sehr kleinen zu übertragenden Nutzdaten nimmt der Protokoll-Overhead daher einen großen Anteil ein. Wenn die zu sendenden Nutzdaten jedoch größer als die IFM-Metadaten werden, spielt der Overhead des IFM-Protokolls eine geringere Rolle. Des Weiteren lässt sich die Größe der transportierten Abhängigkeitsinformationen absenken, indem weitere Spy-Knoten dem Netzwerk hinzugefügt werden. So sammeln sich die Abhängigkeitsinformationen in geringerem Umfang an ehe sie zum IFM-Master gesendet werden und müssen so über weniger Knoten hinweg mittransportiert werden.

Ähnlich verhält es sich mit der Verarbeitungszeit. Auch hier ist ein Anstieg zu sehen, der unabhängig von der Datengröße ist, aber nur mit der Anzahl der abhängigen Informationen skaliert. Dadurch ist die Verarbeitungszeit sehr stark vom Anwendungsfall abhängig. Lange Abhängigkeitsketten erhöhen die Verarbeitungszeit. Die Verarbeitungszeit und Nachrichtengröße beeinflussen auch die Output-Performanz, die leicht abnimmt.

Zusammenfassend erfordert der Einsatz des IFM zusätzliche Ressourcen im Netzwerk, die beim Entwurf des CPS eingeplant werden sollten. Kosten und Nutzen des IFM müssen einzelfallspezifisch gegeneinander abgewogen werden. Kann die durch den Einsatz des IFM reduzierte Performanz nicht dauerhaft im Netzwerk akzeptiert werden, sollte der IFM nur bei Bedarf eingesetzt werden.

4.3 Fazit

Die Evaluation des IFM-Konzepts und der dazugehörigen Werkzeuge teilte sich in diesem Kapitel in zwei Abschnitte: Die Evaluation der Funktionsweise des IFM mit Hilfe des AirQuality-Anwendungsfalls sowie die Performanz-Evaluierung.

Der AirQuality-Anwendungsfall in Abschnitt 4.1 wurde hierbei genutzt, um die generelle Funktionsweise des IFM in einem realen Szenario zu evaluieren. Hierbei wurde gezeigt, dass der IFM in der Lage ist, Abhängigkeiten knotenübergreifend aufzuzeichnen, diese Abhängigkeiten mit Hilfe geeigneter Werkzeuge zu visualisieren und Fehlerquellen durch Aufbereitung der Abhängigkeiten auffindbar zu machen. In diesem Anwendungsbeispiel wurde die Funktionsweise des Information Dependency Modeling ebenfalls demonstriert.

Der zweite Teil der Evaluierung in Abschnitt 4.2 beschäftigte sich mit der Messung der Performanz der IFM-Implementierung. Hierbei wurde gemessen, wie stark der Einsatz des IFM die Performanz des Gesamtnetzwerkes reduziert. Durch die Ergebnisse der Evaluierung wird es möglich anwendungsfallspezifisch abzuschätzen, ob genügend Ressourcen zur Verfügung stehen oder ob weitere Maßnahmen zur Sicherstellung der Performanz ergriffen werden müssen.

5 Fazit und Ausblick

In diesem Kapitel: Erläuterung des Beitrags der Dissertation durch Darstellung der enthaltenen Bausteine. Diskussion und kritische Reflektion der Arbeit und Ausblick auf weitere Arbeiten, die auf dem IFM-Konzept aufbauen können.

5.1 Beitrag dieser Dissertation

Fokus dieser Arbeit war die Erstellung eines Konzepts und Werkzeugs zum Umgang mit dem Problem der Cascading Data Corruption in CPS. Als zentralen Baustein zum Umgang mit diesem Problem stellt die Arbeit das IFM-Konzept und das gleichnamige Werkzeug vor.

Diese Arbeit betrachtete dazu zunächst zentrale Aspekte aus der Problemdomäne:

- Erörterung der Motivation des Problems und der Problemstellung selbst mit dem Fokus auf CPS, Cascading Data Corruption und Abhängigkeiten in verteilten Netzwerken (vgl. Abschnitt 1.1, Abschnitt 1.2) sowie die Darstellung eines Beispiels, in dem Cascading Data Corruption auftritt (vgl. Abschnitt 1.3)
- Darstellung der Vision, wie ein Lösungsansatz des Problems dieser Arbeit aussehen könnte und Erläuterung des Vorgehens, wie der Lösungsansatz in ein Konzept und Werkzeug überführt werden kann (vgl. Abschnitt 1.4, Abschnitt 1.5, Abschnitt 1.6)

Bevor der Kernbeitrag der Dissertation – der Information Flow Monitor (IFM) – vorgestellt werden konnte, wurden zunächst jedoch einige Grundlagen und verwandte Ansätze erläutert, um das Verständnis zu erleichtern:

- Darstellung des Begriffs der Cyber-Physical Systems (CPS) und deren Eigenschaften, um anschließend zu einer eigenen CPS-Definition zu gelangen (vgl. Abschnitt 2.1), sowie Definition weiterer zentraler Begriffe wie Sensor, Aggregator und Aktor (vgl. Abschnitt 2.1.3)

- Analyse des Problems der Abhängigkeiten in CPS und detaillierte Beschreibung des daraus resultierenden Problems des Cascading Data Corruption (vgl. Abschnitt 2.2)

- Darstellung von verwandten Arbeiten zu den Themen CPS, Abhängigkeiten in verteilten Systemen sowie Herausforderungen im Software Engineering (vgl. Kapitel 2) und weitere verwandte Arbeiten, die in Bezug auf den Lösungsansatz des IFM eine Rolle spielen (vgl. Abschnitt 2.3)

Die zuvor genannten Beiträge der Dissertation beleuchten und diskutieren das Problem der Cascading Data Corruption in CPS. Zentraler Beitrag der Dissertation ist das IFM-Konzept und Werkzeug:

- Definition der Anforderung an den IFM sowie der daraus abgeleiteten Architektur und des Protokolls inkl. Anwendungsbeispiel und Darstellung der Visualisierungsmöglichkeiten der Abhängigkeiten im Beispiel-CPS (vgl. Abschnitt 3.1). Zentraler Beitrag ist das IFM-Konzept, das die Nachverfolgung von beobachteten Fehlern zu ihrer Quelle erlaubt.

- Best Practices zur Umsetzung und Implementierung des IFM-Konzeptes in der Praxis (vgl. Abschnitt 3.1.7) und Diskussion weiterer Fragestellungen, die sich beim Praxiseinsatz stellen (vgl. Abschnitt 3.1.8).

- Beschreibung und Analyse des Problems der effizienten Platzierung von Spy-Knoten sowie Vorstellung des evolutionären Algorithmus, der die Platzierung vereinfacht (vgl. Abschnitt 3.2). Beitrag sind Heuristiken, die die effiziente Platzierung von Spy-Knoten ermöglichen.

- Einführung des Information Dependency Modeling zum Umgang mit Black-Box-Knoten, deren Software nicht IFM-konform ist, in CPS-Netzwerken mit IFM (vgl. Abschnitt 3.3). Durch das Dependency Modeling wird die Einbettung von diesen Black-Box-Knoten in IFM-Netzwerke erst möglich.

- Konzept, wie der IFM in Umgebungen eingesetzt werden kann, in denen zwischen den einzelnen Knoten im Netzwerk kein Vertrauen herrscht (vgl. Abschnitt 3.4). Die Kombination aus Blockchain-Technologie und IFM erlaubt die manipulationssichere Speicherung von Abhängigkeitsinformationen und Nutzlasten.

Nachdem der zentrale Baustein der Arbeit, der IFM, vorgestellt wurde, entstand die Evaluation der Konzepte, Werkzeuge und Methoden, die diese Dissertation einführte:

- Evaluation der Funktionsweise des IFM sowie des Dependency Modelings im Rahmen eines CPS-Anwendungsfalls (vgl. Abschnitt 4.1). Dieser Evaluationsschritt zeigt, wie sich der IFM in einem echtweltlichen Szenario anwenden lässt und wie die IFM-Werkzeuge zur Nachverfolgung von Abhängigkeiten eingesetzt werden können.

- Messung und Bewertung der Performance des IFM-Werkzeugs (vgl. Abschnitt 4.2), insbesondere des Overheads, der durch Nutzung des IFM entsteht. Fokus war hierbei die Messung der Ausgangsleistung, der Verarbeitungszeit und des Overheads der Nachrichten bzgl. Nachrichtengröße.

Die Beiträge dieser Arbeit werden im Folgenden diskutiert (vgl. Abschnitt 5.2). Anschließend werden die Optionen für weiterführende und ergänzende Arbeiten beschrieben (vgl. Abschnitt 5.3).

5.2 Diskussion und Einschränkungen

Der im Rahmen dieser Arbeit entwickelte Ansatz des IFM schafft durch die Ermöglichung der Nachverfolgung von Informationsabhängigkeiten in CPS-Netzwerken einen Mehrwert, da Fehler, die im Netzwerk oder in der realen Welt an beliebiger Stelle beobachtet werden, auch zu ihrer ursprünglichen Quelle zurückverfolgt werden können. Diese Abhängigkeiten zwischen Ursache und beobachteter Wirkung können durchgehend durch das CPS verfolgt, visualisiert und durch Domänenexperten ausgewertet werden.

Die von den Spy-Knoten erfassten und vom IFM-Master gesammelten Abhängigkeitsinformationen werden durch Abhängigkeitsbäume im IFM-Visualizer verbildlicht. Hierdurch wird es dem Anwender möglich, die große Menge an Abhängigkeitsinformationen zu beherrschen und auszuwerten. Eine automatische Auswertung ist auch hier nicht möglich. Dies liegt jedoch im Konzept des IFM begründet: Der Fokus des IFM sind die nicht automatisch behebbaren Fehler. Diese Konzeptentscheidung beruht auf der These, dass durch beispielsweise Plausibilitätsprüfungen automatisch auffindbare Fehler auch automatisch unterbindbar sein sollten. Dementsprechend können der Master und Visualizer dem Anwender auch nur dabei helfen, Fehler zurückzuverfolgen, aber die Fehlerquelle selbst nicht automatisch identifizieren. Die Unterstützung des Anwenders durch Bereitstellung von Werkzeugen ist aber zentraler Bestandteil des IFM-Konzepts. Der IFM ist also insbesondere ein Debugging-Werkzeug, das von Domänenexperten genutzt werden kann, um Fehlerquellen zu identifizieren. Ohne Domänenwissen kann in einem Abhängigkeitsbaum oft nicht entschieden werden, welche Verzweigung zur Fehlerquelle führt.

Eine weitere Einschränkung ergibt sich durch den Einsatz eines eigenen Protokolls. Das System wird nicht nur von außen beobachtet, sondern die Knoten im Netzwerk müssen sich selbst am Sammeln der Informationen beteiligen. Dies führt dazu, dass im Bestfall auch alle Knoten das IFM-Protokoll unterstützen sollten. In Abschnitt 3.3 wurde daher das Information Dependency Modeling eingeführt, das einzelne Knoten im Netzwerk erlaubt, die das IFM-Protokoll nicht unterstützen müssen. Von zentraler Bedeutung ist jedoch, dass die große Mehrheit der Knoten das Protokoll unterstützt und das Dependency Modeling nur vereinzelt zum Einsatz kommt. Anderenfalls entwickelt sich der IFM zu einem reinen Modellierungswerkzeug, der mit dem realen Zustand des Netzwerkes nichts mehr zu tun hat. Information Dependency Modeling soll nur zur Verknüpfung aufgezeichneter Abhängigkeiten eingesetzt werden, so dass Teilbäume in der Abhängigkeitsstruktur zusammengefügt werden können. Für Netzwerke, die überwiegend aus Black-Box-Knoten bestehen, kommt der IFM nicht in Betracht. Des Weiteren lässt sich das Information Dependency Modeling auch nur für bestimmte Knoten anwenden. Für diese muss zum einen die präzise Funktionsweise der Businesslogik bekannt sein, um diese nachzubilden. Zum anderen müssen die entstehenden Abhängigkeiten auch durch die vorliegenden Informationen modellierbar sein und sich nicht anhand der Nutzlasten entscheiden. Sollte der

IFM in emergente Netze übertragen werden, wäre der Dependency Modeling Ansatz grundsätzlich nicht anwendbar, da sich die Netze und auch die modellierten Abhängigkeiten zu schnell verändern.

Das eigene Protokoll führt ebenfalls dazu, dass das zusätzliche Sammeln von Daten Performanzeinbußen mit sich bringt. Dies betrifft sowohl die Nachrichtengröße als auch Verarbeitungszeit und Ausgangsleistung der Knoten. Die Nachrichtengröße kann jedoch durch den Einsatz von mehr Spy-Knoten reduziert werden. Die Evaluation in Abschnitt 4.2 hilft die benötigten Ressourcen abzuschätzen. Für die Auswertung von Verarbeitungszeit und Ausgangsleistung sind weitere Arbeiten notwendig, die hier aus Kapazitätsgründen der Testumgebung nicht durchgeführt wurden. Die Auswirkungen auf Echtzeitsysteme wurden noch nicht untersucht.

Das IFM-Protokoll muss von möglichst vielen Knoten im Netz, im Bestfall von allen Knoten, unterstützt werden. Hierzu muss das IFM-Interface (vgl. Abschnitt 3.1.7) in alle Knoten integriert werden. Des Weiteren muss die Businesslogik der Knoten Auskunft darüber geben, welche Daten verwendet wurden, um ausgehende Nachrichten zu erzeugen. Diese Funktion muss der Businesslogik hinzugefügt werden und erfordert eine Anpassung der Knotensoftware.

Für die genannten Einschränkungen konnten innerhalb dieser Arbeit Lösungsansätze präsentiert werden, die die Einschränkung abmildern. Verschiedene Anwendungsbereiche (Echtzeitsysteme, Emergente Netze, Netze mit vielen COTS-Komponenten, etc.) erfordern jedoch weitere Arbeiten, um den IFM für diese zu optimieren.

5.3 Ausblick auf weitere Arbeiten

Im Rahmen dieser Arbeit wurde der IFM mit einer zentralen Master-Instanz vorgestellt, die Abhängigkeitsinformationen von Spy-Knoten entgegennimmt und mit Hilfe des IFM-Visualizers verbildlichen kann. Für große Netzwerke kann so eine Master-Instanz zum Nadelöhr werden. Grundsätzlich spricht jedoch nichts dagegen, in CPS mehrere Master-Instanzen einzusetzen. Dies ist beispielsweise im IFM-Blockchain-Konzept (vgl. Abschnitt 3.4) bereits der Fall, da die Blockchain und IPFS verteiltes Speichern unterstützen. Grundsätzlich ließe sich ein verteiltes Speichern von Abhängigkeitsinformationen jedoch auch außerhalb der Blockchain

realisieren. Verschiedene Visualizer-Instanzen können dann bei Bedarf die benötigten Informationen bei den jeweils zuständigen Master-Instanzen abfragen. Ein spezifisches Konzept zur Umsetzung kann Fokus zukünftiger Arbeiten sein.

In Bezug auf den Visualizer sind ebenfalls weitere Arbeiten denkbar. Es ist vorstellbar, dass in zukünftigen Versionen des Visualizers der Anwender durch weitere Funktionen noch mehr dabei unterstützt wird, die Fehlerquelle zu identifizieren. Maschinelles Lernen (ML) könnte hierbei Techniken zur Analyse und Annotation der Daten bieten. Dabei könnte nicht nur der aktuelle Abhängigkeitsbaum genutzt werden, sondern auch historische Daten zu bereits identifizierten Fehlern. Auf solchen Daten können ML-Algorithmen arbeiten, um Vorschläge oder Hinweise bei der Suche der Fehlerquelle zu geben. Ebenfalls könnte der ML-Algorithmus auf historischen Daten lernen, um Ausreißer in den Daten zu identifizieren. Durch ML könnte so womöglich eine Unterstützung oder (Teil-)Automatisierung der Fehlerrückverfolgung realisiert werden.

Da, wie bereits in Abschnitt 5.2 genannt, die Unterstützung des IFM-Protokolls von möglichst vielen Knoten im Netzwerk von zentraler Bedeutung ist, kann hier die Bereitstellung von Bibliotheken zur Implementierung die Einstiegshürden in das IFM-Konzept senken. Hierbei sollten Bibliotheken für relevante Programmiersprachen als auch für verschiedene Kommunikationsprotokolle zur Auswahl stehen.

Eine weitere mögliche Erweiterung des IFM-Konzepts ist für emergente CPS denkbar. In seiner aktuellen Ausprägung ist der IFM zwar auch in emergenten Netzen einsetzbar, kann dort allerdings nur für kurze Zeit seine Arbeit effizient verrichten. Dies liegt insbesondere daran, dass sich emergente Netze zur Laufzeit restrukturieren, und das IFM-Konzept diesen Strukturänderungen nicht Schritt hält. Platzierte Spy-Knoten könnten also binnen kürzester Zeit durch Änderungen der Topologie deutlich schlechter platziert sein als zum Platzierungszeitpunkt. Um den IFM also in emergenten Netzen einsetzbar zu machen, wird dementsprechend eine Methode benötigt, die die Spy-Knoten automatisiert anhand der topologischen Änderungen des Netzes neu platziert und organisiert. Problematisch hierbei ist ebenfalls, dass emergente Netze oft nicht vollständig bekannt sind. Dies erschwert den Einsatz der Guidelines, deren Anwendung teilweise Wissen über die Umgebung der betrachteten Knoten benötigt. Um den IFM also in emergenten Netzen einsetzen zu können, müssen nicht nur Methoden zur automatischen Re-

strukturierung der Spy-Knoten erstellt werden, sondern diese auch mit Methoden der Topology Discovery kombiniert werden (vgl. Abschnitt 2.3.3).

Als letzte große mögliche Erweiterung des IFM-Konzepts wäre die Betrachtung der Verallgemeinerung der Anwendbarkeit auf nicht-CPS-Netze denkbar. In der hier vorliegenden Arbeit liegt der Fokus ausschließlich auf CPS. Dies betrifft sowohl das Konzept als auch die Platzierung von Spy-Knoten und die Evaluation. Konzeptuell sind vor allem Vernetzungseigenschaften betroffen, die sich auf fast alle Teile der Arbeit auswirken. Im aktuellen Zustand nutzt der IFM Eigenschaften, die sich aus der Präsenz von Sensoren, Aggregatoren und Aktoren ergeben. Diese wären in anderen Netzwerken nicht vorhanden, die jedoch andere Vernetzungseigenschaften und Besonderheiten aufweisen können. Um das Konzept auf andere Systemtypen übertragen zu können, müsste die Arbeit beginnend beim Protokolldesign an die neuen Gegebenheiten angepasst werden.

5.4 Fazit

Im Rahmen dieser Arbeit entstand der Information Flow Monitor (IFM) als zentraler Beitrag. Bestehend aus IFM-Konzept, IFM-Protokoll, Konzepten zur Platzierung von Spy-Knoten und Werkzeugen wie dem IFM-Visualizer bringt der IFM umfassende Handlungsempfehlungen und -möglichkeiten mit sich, um Abhängigkeiten in CPS zu erkennen und zurückzuverfolgen. Beobachtete Fehler können mit Hilfe der in dieser Arbeit vorgestellten Ansätze zu ihrer ursprünglichen Fehlerquelle zurückverfolgt werden, um die Fehler dort zu beheben.

Danksagung

Im Laufe der vergangenen Jahre haben mich viele Menschen, ganz egal ob Freunde, Kollegen oder Familie, bei meinem Promotionsvorhaben unterstützt. Sei es durch gemeinsame Veröffentlichungen, Zeit, Diskussionen, Feedback oder Lektorat – Sie alle haben dazu beigetragen, dass die Arbeit zum Ziel gekommen ist. Ich danke euch sehr! Besonders hervorheben möchte ich die Unterstützung von:

Rabea Aschenbruck, Florian Blum, Matthias Book, Anke Gries, Norbert Gries, Volker Gruhn, Marc Hesenius, Philipp Hückinghaus, Ole Meyer, Angela Müller, Julius Ollesch.

Literatur

[1] E. A. Lee und S. A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. 2. Auflage. Cambridge, Massachuetts: MIT Press, 2017. ISBN: 978-0-262-53381-2.

[2] V. Gunes u. a. „A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems“. In: *KSII Transactions on Internet and Information Systems* 8.12 (2014), S. 4242–4268. ISSN: 19767277. DOI: 10.3837/tiis.2014.12.001.

[3] S. Kowalewski, B. Rumpe und A. Stollenwerk. „Cyber-Physical Systems - Eine Herausforderung Für Die Automatisierungstechnik?“ In: *VDI Berichte 2012* Proc. Automation 2012 (2012), S. 113–116. arXiv: 1409.0385.

[4] E. Geisberger und M. Broy, Hrsg. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*. 1. Auflage. Acatech STUDIE. Berlin Heidelberg: Springer-Verlag, 2012. ISBN: 978-3-642-29098-5.

[5] acatech, Hrsg. *Cyber-Physical Systems: Innovationsmotor Für Mobilität, Gesundheit, Energie Und Produktion*. 1. Auflage. Bd. 11. Acatech BEZIEHT POSITION. Berlin Heidelberg: Springer-Verlag, 2011. ISBN: 978-3-642-27566-1. DOI: 10.1007/978-3-642-27567-8.

[6] M. Gatzke. *CPS.HUB NRW*. URL: https://cps-hub-nrw.de/ (besucht am 22. 06. 2020).

[7] R. Drath und A. Horch. „Industrie 4.0: Hit or Hype?“ In: *IEEE Industrial Electronics Magazine* 8.2 (2014), S. 56–58. ISSN: 1932-4529. DOI: 10.1109/MIE.2014.2312079.

[8] J. Wan u. a. „Software-Defined Industrial Internet of Things in the Context of Industry 4.0“. In: *IEEE Sensors Journal* 16.20 (2016), S. 7373–7380. ISSN: 1530-437X. DOI: 10.1109/JSEN.2016.2565621.

[9] J. Shi u. a. „A Survey of Cyber-Physical Systems“. In: *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*. 2011, S. 1–6. DOI: 10.1109/WCSP.2011.6096958.

[10] M. Broy und A. Schmidt. „Challenges in Engineering Cyber-Physical Systems“. In: *Computer* 47.2 (2014), S. 70–72. ISSN: 0018-9162.

[11] W. Wolf. „Cyber-Physical Systems“. In: *Computer* 42.3 (2009), S. 88–89. ISSN: 0018-9162. DOI: 10.1109/MC.2009.81.

[12] R. Rajkumar u. a. „Cyber-Physical Systems: The Next Computing Revolution“. In: *47th Design Automation Conference, DAC 2010*. DAC '10. New York, NY, USA: ACM, 2010, S. 731–736. ISBN: 978-1-4503-0002-5. DOI: 10.1145/1837274.1837461.

[13] T. Sanislav und L. Miclea. „Cyber-Physical Systems - Concept, Challenges and Research Areas“. In: *Journal of Control Engineering and Applied Informatics* 14.2 (2012), S. 28–33. ISSN: 1454-8658.

[14] E. A. Lee. „CPS Foundations“. In: *Proceedings of the 47th Design Automation Conference, DAC 2010*. Hrsg. von S. S. Sapatnekar. Anaheim, California: ACM, 2010, S. 737–742. DOI: 10.1145/1837274.1837462.

[15] FJ. Wu, YF. Kao und YC. Tseng. „From Wireless Sensor Networks towards Cyber Physical Systems“. In: *Pervasive and Mobile Computing* 7.4 (2011), S. 397–413. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2011.03.003.

[16] M. Hesenius u. a. *Kerntechnologien Für Cyber Physical Systems*. URL: https://www.cps-hub-nrw.de/knowledgebase/publikation/3476-kerntechnologien-fuer-cyber-physical-systems (besucht am 28. 09. 2020).

[17] E. A. Lee. „Cyber Physical Systems: Design Challenges“. In: *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2008)*. Orlando, Florida: IEEE Computer Society, 2008, S. 363–369. DOI: 10.1109/ISORC.2008.25.

[18] KJ. Park, R. Zheng und X. Liu. „Cyber-Physical Systems: Milestones and Research Challenges“. In: *Computer Communications* 36.1 (2012), S. 1–7. ISSN: 01403664. DOI: 10.1016/j.comcom.2012.09.006.

[19] International Organization for Standardization. *ISO/IEC 25010:2011*. 2011.

[20] M. Broy und Deutsche Akademie der Technikwissenschaften, Hrsg. *Cyber-Physical Systems: Innovation Durch Software-Intensive Eingebettete Systeme*. Acatech DISKUTIERT. Berlin: Springer, 2010. ISBN: 978-3-642-14498-1 978-3-642-14901-6. DOI: 10.1007/978-3-642-14901-6.

[21] J. A. Stankovic. „Research Directions for the Internet of Things“. In: *IEEE Internet of Things Journal* 1.1 (2014), S. 3–9. ISSN: 2327-4662. DOI: 10.1109/JIOT.2014.2312291.

[22] J. Lin u. a. „A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications“. In: *IEEE Internet of Things Journal* 4.5 (2017), S. 1125–1142. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2683200.

[23] BH. Schlingloff. „Cyber-Physical Systems Engineering“. In: *Engineering Trustworthy Software Systems*. Hrsg. von Z. Liu und Z. Zhang. Lecture Notes in Computer Science 9506. Springer International Publishing, 2016, S. 256–289. ISBN: 978-3-319-29627-2.

[24] International Organization for Standardization. *ISO/IEC/IEEE 15288:2015*. 2015.

[25] INCOSE. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. 4. Auflage. Hoboken, New Jersey: Wiley, 2015. ISBN: 978-1-118-99940-0.

[26] S. C. Suh u. a., Hrsg. *Applied Cyber-Physical Systems*. 2. Auflage. New York: Springer-Verlag, 2014. ISBN: 978-1-4614-7335-0. DOI: 10.1007/978-1-4614-7336-7.

[27] E. W. U. Küppers. *Eine Transdisziplinäre Einführung In Die Welt Der Kybernetik: Grundlagen, Modelle, Theorien Und Praxisbeispiele*. Springer Vieweg, 2019. ISBN: 978-3-658-23724-0. DOI: 10.1007/978-3-658-23725-7.

[28] H. Czichos. *Mechatronik: Grundlagen Und Anwendungen Technischer Systeme*. 3. Auflage. Springer Vieweg, 2015. ISBN: 978-3-658-09950-3. DOI: 10.1007/978-3-658-09950-3.

[29] A. Ceccarelli u. a. „Basic Concepts on Systems of Systems“. In: *Cyber-Physical Systems of Systems: Foundations – A Conceptual Model and Some Derivations: The AMADEOS Legacy*. Hrsg. von A. Bondavalli, S. Bouchenak und H. Kopetz. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, S. 1–39. ISBN: 978-3-319-47590-5. DOI: 10.1007/978-3-319-47590-5_1.

[30] S. C. Mukhopadhyay und N. K. Suryadevara. „Internet of Things: Challenges and Opportunities“. In: *Internet of Things: Challenges and Opportunities*. Hrsg. von S. C. Mukhopadhyay. Smart Sensors, Measurement and Instrumentation. Cham: Springer International Publishing, 2014, S. 1–17. ISBN: 978-3-319-04223-7. DOI: 10.1007/978-3-319-04223-7_1.

[31] R. Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015. ISBN: 978-0-262-02911-7.

[32] C. Janiesch. *Cyber-Physische Systeme*. URL: https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/informationssysteme/Sektorspezifische-Anwendungssysteme/cyber-physische-systeme (besucht am 13. 07. 2020).

[33] M. Broy. „Engineering Cyber-Physical Systems: Challenges and Foundations“. In: *Complex Systems Design & Management, Proceedings of the Third International Conference on Complex Systems Design & Management CSD&M 2012*. Hrsg. von M. Aiguier u. a. Berlin, Heidelberg: Springer, 2013, S. 1–13. ISBN: 978-3-642-34404-6. DOI: 10.1007/978-3-642-34404-6_1.

[34] F. Xia u. a. „Network QoS Management in Cyber-Physical Systems“. In: *The 2008 International Conference on Embedded Software and Systems Symposia*. 2008, S. 302–307. DOI: 10.1109/ICESS.Symposia.2008.84.

[35] L. Monostori. „Cyber-Physical Systems“. In: *CIRP Encyclopedia of Production Engineering*. Hrsg. von S. Chatti und T. Tolio. Berlin, Heidelberg: Springer, 2018, S. 1–8. ISBN: 978-3-642-35950-7. DOI: 10.1007/978-3-642-35950-7_16790-1.

[36] K. Reif. *Bosch Autoelektrik Und Autoelektronik - Bordnetze, Sensoren Und Elektronische Systeme*. Wiesbaden: Vieweg+Teubner, 2011. ISBN: 978-3-8348-1274-2 978-3-8348-9902-6. DOI: 10.1007/978-3-8348-9902-6.

[37] J. Fitzgerald u. a. „Cyber-Physical Systems Design: Formal Foundations, Methods and Integrated Tool Chains“. In: *Proceedings - Third FME Workshop on Formal Methods in Software Engineering (Formalise 2015)*. Florence, Italy, 2015, S. 40–46. DOI: 10.1109/FormaliSE.2015.14.

[38] J. H. Kazmi u. a. „A Flexible Smart Grid Co-Simulation Environment for Cyber-Physical Interdependence Analysis“. In: *2016 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 2016, S. 1–6. DOI: 10.1109/MSCPES.2016.7480226.

[39] T. DeMarco. *Structured Analysis and System Specification*. Prentice-Hall Software Series. Englewood Cliffs, N.J: Prentice-Hall, 1979. ISBN: 978-0-13-854380-8.

[40] S. M. McMenamin und J. F. Palmer. *Essential Systems Analysis*. Yourdon Press, 1984. ISBN: 978-0-917072-30-7.

[41] S. M. Rinaldi, J. P. Peerenboom und T. K. Kelly. „Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencies“. In: *IEEE Control Systems* 21.6 (2001), S. 11–25. ISSN: 1066-033X. DOI: 10.1109/37.969131.

[42] M. Ouyang. „Review on Modeling and Simulation of Interdependent Critical Infrastructure Systems“. In: *Reliability Engineering & System Safety* 121 (2014), S. 43–60. ISSN: 0951-8320. DOI: 10.1016/j.ress.2013.06.040.

[43] R. Zimmerman. „Social Implications of Infrastructure Network Interactions“. In: *Journal of Urban Technology* 8.3 (2001), S. 97–119. ISSN: 1063-0732. DOI: 10.1080/106307301753430764.

[44] P. Zhang und S. Peeta. „A Generalized Modeling Framework to Analyze Interdependencies among Infrastructure Systems“. In: *Transportation Research Part B: Methodological* 45.3 (2011), S. 553–579. ISSN: 0191-2615. DOI: 10.1016/j.trb.2010.10.001.

[45] D. D. Dudenhoeffer, M. R. Permann und M. Manic. „CIMS: A Framework for Infrastructure Interdependency Modeling and Analysis“. In: *Proceedings of the 2006 Winter Simulation Conference*. IEEE, 2006, S. 478–485. DOI: 10.1109/WSC.2006.323119.

[46] E. E. Lee, J. E. Mitchell und W. A. Wallace. „Restoration of Services in Interdependent Infrastructure Systems: A Network Flows Approach“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (2007), S. 1303–1317. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.905859.

[47] W. A. Wallace u. a. „Managing Disruptions to Critical Interdependent Infrastructures in the Context of the 2001 World Trade Center Attack“. In: *Beyond September 11th: An Account of Post-Disaster Research* (2003), S. 165–198.

[48] L. Miclea und T. Sanislav. „About Dependability in Cyber-Physical Systems“. In: *2011 9th East-West Design & Test Symposium (EWDTS 2011)*. 2011, S. 17–21. DOI: 10.1109/EWDTS.2011.6116428.

[49] K. Marashi, S. S. Sarvestani und A. R. Hurson. „Quantification and Analysis of Interdependency in Cyber-Physical Systems“. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. 2016, S. 149–154. DOI: 10.1109/DSN-W.2016.47.

[50] D. E. Culler und H. Mulder. „Smart Sensors to Network the World“. In: *Scientific American* 290.6 (2004), S. 84–91. ISSN: 0036-8733. DOI: 10.1038/scientificamerican0604-84.

[51] S. Gries, M. Hesenius und V. Gruhn. „Cascading Data Corruption: About Dependencies in Cyber-Physical Systems: Poster“. In: *DEBS 2017 - Proceedings of the 11th ACM International Conference on Distributed Event-Based Systems*. 2017, S. 345–346. DOI: 10.1145/3093742.3095092.

[52] S. Gries, M. Hesenius und V. Gruhn. „Tracing Cascading Data Corruption in CPS with the Information Flow Monitor“. In: *New Trends in Intelligent Software Methodologies, Tools and Techniques - Proceedings of the 16th International Conference, SoMeT_17*. Hrsg. von H. Fujita, A. Selamat und S. Omatu. Bd. 297. Frontiers in Artificial Intelligence and Applications. Kitakyushu, Japan: IOS Press, 2017, S. 399–408. ISBN: 978-1-61499-799-3. DOI: 10.3233/978-1-61499-800-6-399.

[53] S. Helmis und R. Hollmann. „Informationen, Daten Und Wissen - Ein Definitionsversuch“. In: *Webbasierte Datenintegration: Ansätze Zur Messung Und Sicherung Der Informationsqualität in Heterogenen Datenbeständen*

Unter Verwendung Eines Vollständig Webbasierten Werkzeuges. Wiesbaden: Vieweg+Teubner, 2009, S. 95–102. ISBN: 978-3-8348-9280-5. DOI: 10.1007/978-3-8348-9280-5_9.

[54] K. Jendrian und C. Weinmann. „Daten Und Informationen“. In: *Datenschutz und Datensicherheit - DuD* 34.2 (2010), S. 108–108. ISSN: 1862-2607. DOI: 10.1007/s11623-010-0046-y.

[55] Inc. Cisco Systems. *Introduction to Cisco IOS® NetFlow*. 2012. URL: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.pdf (besucht am 19. 01. 2017).

[56] S. Poznyakoff. *GNU Cflow*. URL: https://www.gnu.org/software/cflow/ (besucht am 09. 01. 2017).

[57] Z. Su u. a. „CeMon: A Cost-Effective Flow Monitoring System in Software Defined Networks“. In: *Computer Networks* 92 (2015), S. 101–115. ISSN: 13891286. DOI: 10.1016/j.comnet.2015.09.018.

[58] V. Sekar u. a. „cSamp: A System for Network-Wide Flow Monitoring“. In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*. Hrsg. von J. Crowcroft und M. Dahlin. Bd. 8. USENIX Association, 2008, S. 233–246.

[59] E. Noak, S. Jovalekic und H. Grochowski. „Protokollanalyse Und Informationsflussverfolgung Zur Fehlerdiagnose in Verteilten Echtzeitsystemen“. In: *Software-Intensive Verteilte Echtzeitsysteme*. Springer, Berlin, Heidelberg, 2009, S. 129–138. DOI: 10.1007/978-3-642-04783-1_16.

[60] H. Mukhtar u. a. „Autonomous Network Topology Discovery of Large Multi-Subnet Networks Using Lightweight Probing“. In: *2008 IEEE Network Operations and Management Symposium Workshops - NOMS 08*. 2008, S. 351–356. DOI: 10.1109/NOMSW.2007.54.

[61] R. A. Alhanani und J. Abouchabaka. „An Overview of Different Techniques and Algorithms for Network Topology Discovery“. In: *2014 Second World Conference on Complex Systems (WCCS)*. 2014, S. 530–535. DOI: 10.1109/ICoCS.2014.7061004.

[62] W. Zhangchao u. a. „An Algorithm and Implementation of Network Topology Discovery Based on SNMP“. In: *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*. 2016, S. 283–286. DOI: 10.1109/CCI.2016.7778926.

[63] B. Li u. a. „A Survey Of Network Flow Applications“. In: *Journal of Network and Computer Applications* 36.2 (2013), S. 567–581. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2012.12.020.

[64] Z. Liu u. a. „One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon“. In: *SIGCOMM '16: Proceedings of the 2016 ACM SIGCOMM Conference*. SIGCOMM '16. New York, NY, USA: ACM, 2016, S. 101–114. ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2934906.

[65] V. Sekar, M. K. Reiter und H. Zhang. „Revisiting the Case for a Minimalist Approach for Network Flow Monitoring“. In: *IMC '10: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. New York, NY, USA: ACM, 2010, S. 328–341. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879186.

[66] S. Gries, M. Hesenius und V. Gruhn. „Tracking Information Flow in Cyber-Physical Systems“. In: *Proceedings - 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)*. 2017, S. 2589–2590. DOI: 10.1109/ICDCS.2017.116.

[67] Z. Zhang u. a. „Decentralized Cyber-Physical Systems: A Paradigm for Cloud-Based Smart Factory of Industry 4.0“. In: *Cybersecurity for Industry 4.0: Analysis for Design and Manufacturing*. Hrsg. von L. Thames und D. Schaefer. Springer Series in Advanced Manufacturing. Cham: Springer International Publishing, 2017, S. 127–171. ISBN: 978-3-319-50660-9. DOI: 10.1007/978-3-319-50660-9_6.

[68] W3C. *XML Protocol Activity*. 2001. URL: https://www.w3.org/2000/xp/ (besucht am 11. 02. 2019).

[69] W3C. *XML Protocol Working Group Charter*. 2004. URL: https://www.w3.org/2004/02/XML-Protocol-Charter (besucht am 11. 02. 2019).

[70] A. Banks u. a. *MQTT Version 5.0 - OASIS Standard*. Hrsg. von R. Coppen. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[71] R. Chaâri u. a. „Cyber-Physical Systems Clouds: A Survey“. In: *Computer Networks* 108 (2016), S. 260–278. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2016.08.017.

[72] L. Richardson und S. Ruby. *Web-Services Mit REST*. O'Reilly Germany, 2007. ISBN: 978-3-89721-727-0.

[73] S. Tilkov u. a. *REST Und HTTP: Entwicklung Und Integration Nach Dem Architekturstil Des Web*. dpunkt.verlag, 2015. ISBN: 978-3-86491-644-1.

[74] Inc. MongoDB. *Introduction to MongoDB - MongoDB Manual*. URL: https://docs.mongodb.com/manual/introduction (besucht am 07. 11. 2019).

[75] K. L. Kelly. „Twenty-Two Colors of Maximum Contrast“. In: *Color Engineering* 3.26 (1965), S. 26–27.

[76] Clay Mathematics Institute. *P vs NP Problem*. URL: http://www.claymath.org/millennium-problems/p-vs-np-problem (besucht am 21. 02. 2020).

[77] S. G. Krantz und H. R. Parks. „The P/NP Problem“. In: *A Mathematical Odyssey: Journey from the Real to the Complex*. Hrsg. von S. G. Krantz und H. R. Parks. Boston, MA: Springer US, 2014, S. 217–254. ISBN: 978-1-4614-8939-9. DOI: 10.1007/978-1-4614-8939-9_10.

[78] R. M. Karp. „Reducibility Among Combinatorial Problems“. In: *Complexity of Computer Computations*. Hrsg. von R. E. Miller und J. W. Thatcher. The IBM Research Symposia Series. Yorktown Heights, New York: Plenum Press, New York, USA, 1972, S. 85–103. ISBN: 978-0-306-30707-2. DOI: 10.1007/978-1-4684-2001-2_9.

[79] J. Liu u. a. „Evaluating the Importance of Nodes in Complex Networks“. In: *Physica A: Statistical Mechanics and its Applications* 452 (2016), S. 209–219. ISSN: 0378-4371. DOI: 10.1016/j.physa.2016.02.049.

[80] T. Opsahl, F. Agneessens und J. Skvoretz. „Node Centrality in Weighted Networks: Generalizing Degree and Shortest Paths“. In: *Social Networks* 32.3 (2010), S. 245–251. ISSN: 03788733. DOI: 10.1016/j.socnet.2010.03.006.

[81] L. Lü u. a. „Vital Nodes Identification in Complex Networks“. In: *Physics Reports*. Vital Nodes Identification in Complex Networks 650 (2016), S. 1–63. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2016.06.007.

[82] L. C. Freeman. „Centrality in Social Networks Conceptual Clarification“. In: *Social Networks* 1.3 (1978), S. 215–239. ISSN: 03788733. DOI: 10.1016/0378-8733(78)90021-7.

[83] L. Zou, M. Lu und Z. Xiong. „A Distributed Algorithm for the Dead End Problem of Location Based Routing in Sensor Networks“. In: *IEEE Transactions on Vehicular Technology* 54.4 (2005), S. 1509–1522. ISSN: 0018-9545. DOI: 10.1109/TVT.2005.851327.

[84] H. Mahyar u. a. „Identifying Central Nodes for Information Flow in Social Networks Using Compressive Sensing“. In: *Social Network Analysis and Mining* 8.1 (2018), S. 33. ISSN: 1869-5450, 1869-5469. DOI: 10.1007/s13278-018-0506-1.

[85] D. Kempe, J. Kleinberg und É. Tardos. „Maximizing the Spread of Influence through a Social Network“. In: *KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: ACM, 2003, S. 137–146. ISBN: 978-1-58113-737-8. DOI: 10.1145/956750.956769.

[86] Habiba u. a. „Finding Spread Blockers in Dynamic Networks“. In: *Advances in Social Network Mining and Analysis*. Hrsg. von L. Giles u. a. Bd. 5498. Lecture Notes in Computer Science. Las Vegas, NV, USA: Springer Berlin Heidelberg, 2008, S. 55–76. ISBN: 978-3-642-14929-0. DOI: 10.1007/978-3-642-14929-0_4.

[87] C. Budak, D. Agrawal und A. El Abbadi. „Limiting the Spread of Misinformation in Social Networks“. In: *WWW '11: Proceedings of the 20th International Conference on World Wide Web*. WWW '11. New York, NY, USA: ACM, 2011, S. 665–674. ISBN: 978-1-4503-0632-4. DOI: 10.1145/1963405.1963499.

[88] Norbert Henze. *Stochastik Für Einsteiger*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. ISBN: 978-3-658-22043-3 978-3-658-22044-0. DOI: 10.1007/978-3-658-22044-0. URL: http://link.springer.com/10.1007/978-3-658-22044-0 (besucht am 27. 11. 2020).

[89] S. Gries, M. Hesenius und V. Gruhn. „Embedding Non-Compliant Nodes into the Information Flow Monitor by Dependency Modeling“. In: *Proceedings - 2018 IEEE 38th International Conference on Distributed*

Computing Systems (ICDCS 2018). 2018, S. 1541–1542. DOI: 10.1109/ICDCS.2018.00163.

[90] S. Gries, J. Ollesch und V. Gruhn. „Modeling Semantic Dependencies to Allow Flow Monitoring in Networks with Black-Box Nodes“. In: *Proceedings of the 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS'19)*. SEsCPS '19. IEEE Press, 2019, S. 14–17. DOI: 10.1109/SEsCPS.2019.00010.

[91] J. Sztipanovits u. a. „Toward a Science of Cyber-Physical System Integration“. In: *Proceedings of the IEEE* 100.1 (2012), S. 29–44. ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2161529.

[92] Inc. MongoDB. *SQL to MongoDB Mapping Chart — MongoDB Manual*. URL: https://docs.mongodb.com/manual/reference/sql-comparison (besucht am 13. 11. 2019).

[93] L. Sha u. a. „Cyber-Physical Systems: A New Frontier“. In: *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (Sutc 2008)*. 2008, S. 1–9. DOI: 10.1109/SUTC.2008.85.

[94] S. Gries u. a. „Using Blockchain Technology to Ensure Trustful Information Flow Monitoring in CPS“. In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C 2018)*. 2018, S. 35–38. DOI: 10.1109/ICSA-C.2018.00014.

[95] J. Eberhardt und S. Tai. „On or Off the Blockchain? Insights on Off-Chaining Computation and Data“. In: *Service-Oriented and Cloud Computing*. Hrsg. von F. De Paoli, S. Schulte und E. Broch Johnsen. Bd. 10465. Cham: Springer International Publishing, 2017, S. 3–15. ISBN: 978-3-319-67261-8 978-3-319-67262-5. DOI: 10.1007/978-3-319-67262-5_1.

[96] M. Swan. *Blockchain: Blueprint for a New Economy*. 1. Auflage. O'Reilly Media, Inc., 2015. ISBN: 978-1-4919-2049-7.

[97] A. M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. 1. Auflage. Sebastopol CA: O'Reilly, 2015. ISBN: 978-1-4493-7404-4 978-1-4919-0260-8.

[98] The go-ethereum Authors. *Go Ethereum*. URL: https://geth.ethereum.org/ (besucht am 09. 06. 2020).

[99] Umwelt Bundesamt. *Luftqualität 2015 - Vorläufige Auswertung*. 2016. URL: https://www.umweltbundesamt.de/publikationen/luftqualitaet-2015.

[100] World Health Organization. *Ambient Air Quality - Public Health, Environmental and Social Determinants of Health (PHE)*. URL: http://www.who.int/phe/health_topics/outdoorair/en/ (besucht am 07. 04. 2020).

[101] The European Council. *Air Quality: Agreement on Stricter Limits for Pollutant Emissions*. URL: http://www.consilium.europa.eu/en/press/press-releases/2016/06/30/air-quality/ (besucht am 07. 04. 2020).

[102] R. Maas u. a. *Towards Cleaner Air - Scientific Assessment Report 2016*. 2016. URL: http://www.unece.org/fileadmin/DAM/env/lrtap/ExecutiveBody/35th_session/CLRTAP_Scientific_Assessment_Report_-_Final_20-5-2016.pdf (besucht am 07. 04. 2020).

[103] M. Tista u. a. *European Union Emission Inventory Report 1990-2014 under the UNECE Convention on Long-Range Transboundary Air Pollution (LRTAP)*. Luxembourg: Publications Office, 2016. ISBN: 978-92-9213-748-9.

[104] European Environment Agency. *European Union Emission Inventory Report 1990-2016 under the UNECE Convention on Long-Range Transboundary Air Pollution (LRTAP)*. 2018.

[105] Eurostat. *Eurostat - Air Emissions Accounts by NACE Rev. 2 Activity*. URL: http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=env_ac_ainah_r2&lang=en (besucht am 07. 04. 2020).

[106] Landesamt für Natur, Umwelt und Verbraucherschutz Nordrhein-Westfalen. *Messorte Der Luftqualitätsüberwachung In NRW*. URL: https://www.lanuv.nrw.de/umwelt/luft/immissionen/messorte-und-werte (besucht am 07. 04. 2020).

[107] Landesamt für Natur, Umwelt und Verbraucherschutz Nordrhein-Westfalen. *Ergänzende Stickstoffdioxid-Messungen in Bielefeld*. URL: https://www.lanuv.nrw.de/landesamt/veroeffentlichungen/pressemitteilungen/details/1347-ergaenzende-stickstoffdioxid-messungen-in-bielefeld (besucht am 07. 04. 2020).

[108] J. Duyzer u. a. „Representativeness of Air Quality Monitoring Networks“. In: *Atmospheric Environment* 104 (2015), S. 88–101. ISSN: 1352-2310. DOI: 10.1016/j.atmosenv.2014.12.067.

[109] B. Paas u. a. „Small-Scale Variability of Particulate Matter and Perception of Air Quality in an Inner-City Recreational Area in Aachen, Germany“. In: *Meteorologische Zeitschrift* 25.3 (2016), S. 305–317. ISSN: 0941-2948. DOI: 10.1127/metz/2016/0704.

[110] S. Gries u. a. „Developing a Convenient and Fast to Deploy Simulation Environment for Cyber-Physical Systems“. In: *Proceedings - 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS 2018)*. 2018, S. 1551–1552. ISBN: 978-1-5386-6871-9. DOI: 10.1109/ICDCS.2018.00166.

[111] Ministry of the environment, conservation and parks. *What Is the Air Quality Health Index?* URL: http://www.airqualityontario.com/science/aqhi_description.php (besucht am 15. 04. 2020).

[112] Umwelt Bundesamt. *Ozon-Belastung*. Text. 2013. URL: https://www.umweltbundesamt.de/daten/luft/ozon-belastung (besucht am 15. 04. 2020).

[113] Rancher Labs. *Rancher Documentation - Rancher 1.6 Docs*. URL: https://rancher.com/docs/rancher/v1.6/en/ (besucht am 27. 05. 2020).

[114] Stefan Gries und Volker Gruhn. „Performance Evaluation of the Information Flow Monitor Protocol in Cyber-Physical Systems“. In: *Frontiers in Artificial Intelligence and Applications*. Hrsg. von Hamido Fujita, Ali Selamat und Sigeru Omatu. IOS Press, Sep. 2020. ISBN: 978-1-64368-114-6. DOI: 10.3233/FAIA200582.

Abkürzungsverzeichnis

acatech	Deutsche Akademie der Technikwissenschaften
API	Application Programming Interface
AQHI	Air Quality Health Index
COTS	Commercial off-the-shelf
CPS	Cyber-Physical System
DQL	Data Query Language
EDCC	Executable Distributed Code Contract
EU	Europäische Union
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	Identifikator
IFM	Information Flow Monitor
IFMBC	IFM+Blockchain
IKT	Informations- und Kommunikationstechnik
INCOSE	International Council of Systems Engineering
IoT	Internet of Things

IP	Internet Protocol
IPFS	InterPlanetary File System
IT	Informationstechnik
ITS	Intelligent Transportation System
JSON	JavaScript Object Notation
LANUV	Landesamt für Natur, Umwelt und Verbraucherschutz Nordrhein-Westfalen
M2M	Machine-to-Machine
MAC	Media Access Control
ML	Maschinelles Lernen
MQTT	Message Queuing Telemetry Transport
NRW	Nordrhein-Westfalen
NoSQL	Not only Structured Query Language
NSF	National Science Foundation
PCAST	President's Council of Advisors on Science and Technology
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SoS	System of Systems
SQL	Structured Query Language
UI	User Interface

UNECE United Nations Economic Commission for Europe

WHO World Health Organization

WLAN Wireless Local Area Network

XML Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis