Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Verbesserung von Robustheit und Ausfallsicherheit bei der Datenverarbeitung in IoT-Umgebungen

Tim Schubert

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Daniel Del Gaudio, M.Sc.
Beginn am:	8. November 2019
Beendet am:	3. Juli 2020

Kurzfassung

Das Internet der Dinge (Internet of Things) ist ein Sammelbegriff für Technologien und beschreibt die Entwicklung und zunehmende Vernetzung von „intelligenten“ Gegenständen. Eine Vielzahl heterogener Geräte wird über standardisierte Internetprotokolle verbunden, um zusammen verschiedenste Aufgaben, wie zum Beispiel die Automatisierung alltäglicher Aktivitäten, zu übernehmen. Daten von Sensoren werden von anderen Geräten ausgewertet und verarbeitet, um entsprechende Aktoren zu steuern. IoT-Anwendungen stellen Anforderungen an IoT-Umgebungen, wie zum Beispiel die Zuverlässigkeit der Nachrichtenübertragung oder die Toleranz gegenüber Fehlern in Messungen von Sensoren. Durch die Einschränkung der Ressourcen von den Geräten erhöht sich das Risiko auf Fehler oder Ausfälle, die die Funktionalität des Systems beeinträchtigen und die Anforderungsspezifikationen verletzen. Solche Fehlerszenarien können anhand verschiedener Merkmale in lokale Soft- oder Hardwarefehler und Netzwerkfehler unterteilt werden. Die Einteilung in Kategorien dient unter anderem dazu, passende Methoden zur Fehlererkennung zu finden. Diese kann lokal, zentral, durch eine Monitoring-Komponente, oder dezentral, durch andere IoT-Geräte durchgeführt werden. Für die verschiedenen Fehlerkategorien müssen außerdem jeweils andere Konzepte zur Behandlung der Fehler gefunden werden. Auch diese kann lokal, auf den IoT-Geräten, oder über einen Fernzugriff von einer zentralen oder dezentralen Komponente durchgeführt werden. Für die Erkennung und Behandlung der Fehler wurden sowohl bereits vorhandene Konzepte verwendet und angepasst, als auch eigene entwickelt. Ziel jeder dieser Methoden ist das Aufrechterhalten aller Funktionalitäten von unterschiedlichen IoT-Anwendungen und das Einhalten derer Anforderungen. Der zusätzliche Aufwand durch Überwachung und Vermeidung von Fehlern wurde anschließend diskutiert und durch eine Implementierung eines Prototyps auf Machbarkeit geprüft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

API Application Programming Interface. 11, 55

CEP Complex Event Processing. 11, 28

EWMA Exponentially Weighted Moving Average. 11, 41

HTTP Hypertext Transfer Protocol. 11, 54

IoT Internet of Things. 3, 5, 7, 9, 11, 15, 16, 17, 18, 19, 22, 23, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 54, 56, 57, 58, 59, 61, 62, 63

JSON JavaScript Object Notation. 11, 50, 53

M2M Machine-to-Machine. 11, 19, 20, 61

MBP Multi-purpose Binding and Provisioning Platform. 11, 28, 40, 48, 49, 50, 51, 53, 54, 55, 56, 62

MQTT Message Queuing Telemetry Transport. 5, 7, 9, 11, 18, 19, 20, 21, 22, 30, 50, 51, 53, 56, 61

OSI Open Systems Interconnection. 11, 20

QoS Quality-of-Service. 5, 9, 11, 21, 23, 24, 30, 46, 61

REST Representational State Transfer. 11, 53, 54, 55

SaaS Software-as-a-Service. 11, 23, 24

SLA Service Level Agreement. 11, 23, 24

SMA Simple Moving Average. 11, 44

SSH Secure Shell. 11, 28, 40

TCP Transmission Control Protocol. 11, 21

VM Virtual Machine. 11, 24, 51

ZODB Zope Object Database. 6, 11, 50, 51, 55, 56

1 Einleitung

Weltweit steigt die Anzahl der Geräte, die mit dem Internet verbunden sind, stark an[1]. Einen bedeutenden Anteil davon bilden die Geräte des Internets der Dinge (Internet of Things, IoT). „Dinge“ im IoT werden auch IoT-Geräte genannt und bilden zusammen ein Netzwerk. IoT-Geräte sind häufig leistungsschwache Mikro-Computer, die mit Sensoren und Aktoren ausgestattet sind [VF13]. Kommuniziert wird meist, im Gegensatz zur heute noch üblichen Machine-To-Human Kommunikation, direkt von Gerät zu Gerät [LN10a].

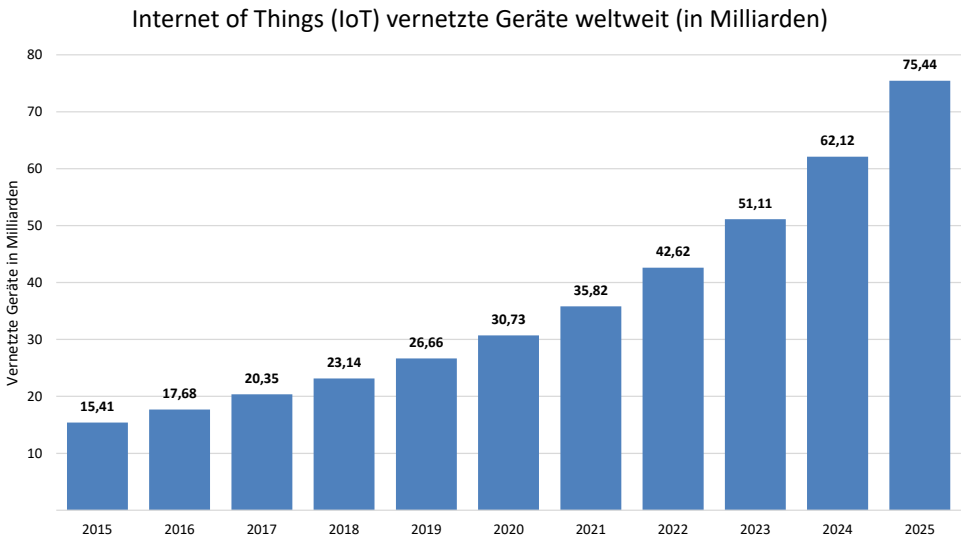

Abbildung 1.1: Vernetzte Geräte weltweit von 2015 bis 2025 [MF16]

In Abbildung 1.1 wird der Verlauf der Anzahl vernetzter Geräte bis zum Jahr 2025 prognostiziert. Während zwischen den Jahren 2015 und 2016 nur 2,27 Milliarden neue Geräte hinzukamen und die absolute Anzahl auf 17,68 Milliarden stieg, wird der Anstieg zwischen 2024 und 2025 auf rund 13,32 Milliarden neue IoT-Geräte geschätzt. Aufgrund der steigenden Anzahl an internetfähigen

[1] https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#6a558beb292d

Geräten wurden bereits mehrere neue Standards im Bereich der Internettechnologie definiert. So zum Beispiel auch der IPv6-Standard, welcher benötigt wird, um die vielen Geräte mit eindeutigen Adressen zu versehen [DRC98]. Durch die 128-bit Adresse besteht nun die Möglichkeit über 630 Sextillionen Geräte eindeutig zu identifizieren.
Der Wachstum von Netzwerken, in denen jedes Gerät mit jedem kommunizieren kann, wird durch solche Technologien unterstützt. Sowohl in der Industrie (*Smart Factories*) als auch in privaten Haushalten (*Smart Homes*) findet das IoT immer mehr Verwendung. So steigt die prognostizierte Anzahl der Smart Homes, also die Verwendung von IoT-Geräten für Hausautomatisierungen, von aktuell nur 175 Millionen Haushalte auf 375 Millionen Haushalte im Jahr 2024[2]. Ein ähnliches Wachstum ist in der Verwendung von verknüpften IoT-Geräten in der Industrie bemerkbar. Allein in der EU steigt die Anzahl der aktiven Verbindungen zwischen IoT-Geräten von 3,8 Millionen im Jahr 2019 auf prognostizierte 18,6 Millionen im Jahr 2025 [Dig19].
Um die großen Datenmengen von vielen Geräten zu verarbeiten, reicht eine einzige zentrale Einheit nicht mehr aus [SCZ+16]. Die zentrale Datenverarbeitung hat den Vorteil der einfacheren Handhabung und des Managements der Daten, sowie der Vereinheitlichung von Sicherheitsmechanismen. Auch Kosten können durch die günstige vertikale Skalierung der Hardware eines leistungsstärkeren Computers gegenüber vielen kleinen mittelmäßig leistungsfähigen Rechnern eingespart werden. Durch das starke Wachstum der Netzwerke und der Nachfrage nach mehr Automatisierung würden die benötigten Bandbreiten und Rechenleistungen einer zentralen Maschine allerdings zu hoch sein, um alle Auswertungen und Steuerungen dort vorzunehmen. Ein neuer Weg ist daher die dezentrale Datenverarbeitung, auch *Edge Computing* [SCZ+16] genannt. Die gesammelten physikalischen Messwerte von Sensoren oder andere Anwendungsdaten werden beispielsweise durch *Stream Processing* [FHPM18] bereits auf den IoT-Geräten am Rand des gesamten Netzwerkes verarbeitet. So können Verzögerungen bis zur Reaktion eines Aktors minimiert werden und die im gesamten Netz benötigte Rechenleistung auf kleinere Geräte aufgeteilt werden [CWH13].
Mit der Dezentralisierung und der steigenden Größe, sowie der Komplexität in einem Netzwerk, steigt automatisch auch die Wahrscheinlichkeit eines Fehlers oder Ausfalls einer Netzwerkfunktion beziehungsweise Verbindung zwischen zwei Geräten [SDSB19]. Hinzu kommt die meist kabellose Übertragung von Daten, die aktuell nicht so leistungsfähig und zuverlässig wie eine kabelgebundene Datenübertragung ist [FP05]. Es kann zu fehlerhaften oder fehlenden Daten in Übertragungen kommen oder, im Falle von mobilen Geräten, kann die Verbindung komplett durch das Verlassen des Signalbereichs abbrechen. Doch nicht nur im Netzwerk erhöht sich das Ausfallrisiko mit der steigenden Anzahl der Geräte. Auch die IoT-Geräte, die meist nur eine geringe Rechenleistung und begrenzte Energie durch den Betrieb per Batterie zur Verfügung haben, weisen eine höhere Ausfallwahrscheinlichkeit auf als in der Größe vergleichbare kabelgebundene Rechnernetze in Rechenzentren. Speicher und Prozessoren fallen, meist aus Gründen des Preises und des Leistungsverbrauches, kleiner aus als in stärkeren zentralen Computern [VWB+16]. All die Vorteile eines IoT-Gerätes, wie die Mobilität, der geringe Energieverbrauch und die geringe Baugröße, tragen zur Erhöhung eines Ausfallrisikos der Geräte und des Netzwerkes bei.
Um Ausfälle und Fehler zu minimieren, müssen also Techniken und Methoden entwickelt werden, die sowohl die Vorteile eines IoT-Gerätes bewahren, als auch Robustheit und Ausfallsicherheit verbessern. Die vorliegende Arbeit soll für Geräteausfälle, Sensor-, sowie Anwendungsfehler und Netzwerkabbrüche Methoden zur Erkennung und Vermeidung aufzeigen. Dabei werden sowohl

[2] https://www.statista.com/statistics/887613/number-of-smart-homes-in-the-smart-home-market-worldwide

zentrale als auch dezentrale Konzepte vorgestellt und anschließend bewertet. Das Ziel ist eine automatische Fehlerbehebung mit möglichst geringem Mehraufwand in IoT-Umgebungen mit unterschiedlichen Anforderungen zu realisieren.

1.1 Motivation

Als Anwendungsbereich in dem die Robustheit und Ausfallsicherheit eines IoT-Netzwerkes verbessert werden soll, wird ein Smart-Home-Szenario beschrieben.
Eine Anwendung in der Heimautomatisierung soll Feuer im Haus erkennen und melden. Dafür werden die Temperatursensoren T1.1, T1.2, T2, T3, T4.1 und T4.2 an verschiedenen Stellen im Haus verbaut, mindestens jedoch ein Sensor pro Zimmer. Jeder Sensor ist batteriebetrieben und muss mindestens alle 15 Minuten die aktuelle Temperatur melden. Um Energie zu sparen sendet in den Räumen mit zwei Sensoren jeweils nur ein Sensorgerät. Die anderen Geräte (T1.2 und T4.2) sind im Standby zwar mit dem Netzwerk verbunden, messen allerdings weder Temperatur noch senden sie Nachrichten an andere Komponenten. Zusätzlich zu den Temperaturmessungen wird die Luft permanent auf eine erhöhte CO_2-Konzentration überwacht. Auch diese Meldung muss mindestens jede Viertelstunde erfolgen. Um Kosten zu sparen ist nur einzelner CO_2-Sensor an einer zentralen Stelle im Haus, beispielsweise im Flur, verbaut. Eine beispielhafte Skizze ist in Abbildung 1.2 dargestellt.

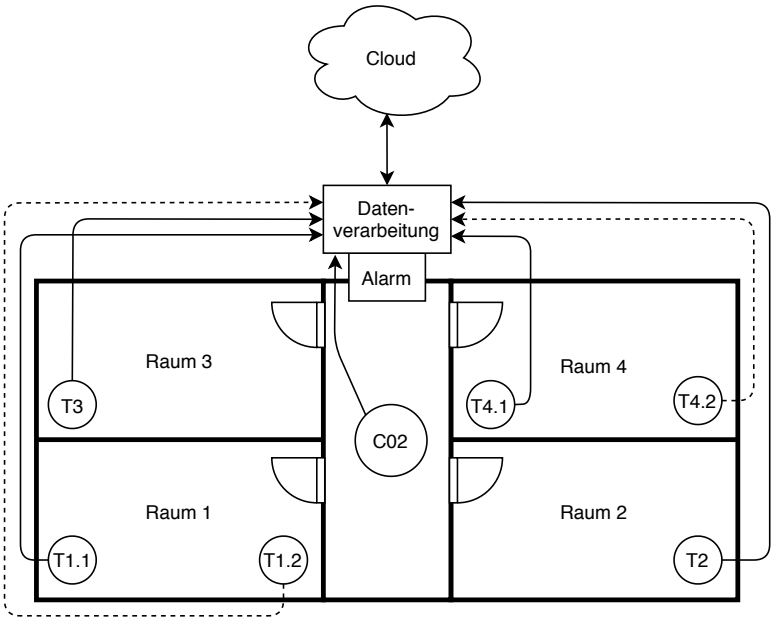

Abbildung 1.2: Brandmeldeanlage im Smart Home als Anwendungsbeispiel

Die Sensoren spannen ein IoT-Netzwerk über einen häufig genutzten Funkstandard, wie zum Beispiel W-LAN [16], auf. Die gemessenen Sensordaten werden vorab dezentral auf jedem Gerät nach gewissen Grenzwerten, wie beispielsweise negative Temperaturen, gefiltert. Über das Netzwerk werden die Messwerte an die zentrale Einheit übertragen und dort weiterverarbeitet. Dieser zentrale

Computer gibt bei einem Feuerverdacht eine Warnung an die Bewohner per akustischem und visuellem Signal ab. Zusätzlich wird eine Benachrichtigung über die Cloud verschickt.
Brandmeldeeinheiten müssen Feuer zuverlässig erkennen und auch bei Sensorausfällen noch robust genug sein, um Fehler der Sensoren zu kompensieren. Fehlerhafte Sensordaten sollten nicht zu einem Fehlalarm führen oder die Auslösung eines Alarms verhindern. Ebenso muss die Funktionalität des Systems bei einem Ausfall eines Sensors oder Gerätes weiterhin gegeben sein. Zu den möglichen Ausfällen gehören neben Hardwarefehlern von Sensorgeräten auch der Absturz von laufender Software auf den Geräten. Außerdem kann die Spannung der Batterien nicht mehr ausreichend sein, um die Geräte mit genügend Strom zu versorgen, was zu einem Geräteausfall führt. Die Sensormesswerte könnten durch Hardware- oder Übertragungsfehler verfälscht werden. Ein weiteres Problem sind Störungen des Funksignals durch Rauschen oder Überlagerungen, welche zu schlechten Verbindungen zwischen den Geräten oder zu einem Verbindungsabbruch führen können. In diesem Szenario soll die Brandmeldeanlage vollkommen eigenständig arbeiten und die höchstmögliche Robustheit gegenüber Fehlern und Ausfällen von Geräten oder des Netzwerkes aufweisen. Bei Problemen soll der Nutzer zusätzlich über die Cloud benachrichtigt werden. Eine Interaktion einer Person mit dem System soll aber nur im Notfall notwendig sein, wenn die ursprünglich spezifizierte Funktionalität nicht mehr gegeben ist.

1.2 Struktur

Ziel dieser Arbeit ist, vorhandene Literatur und Techniken zu recherchieren und strukturieren, sowie eigene Konzepte zur Fehlererkennung und -behandlung in IoT-Netzwerken zu entwickeln.
Zu Beginn werden die Grundlagen des Internet of Things eingeführt, die zum Verständnis der später vorgestellten Konzepte und Methoden notwendig sind. Enthalten sind außerdem das in der späteren Implementierung verwendete Kommunikationsprotokoll MQTT (Abschnitt 2.2) und eine Einführung in Monitoring bzw. Cloud-Monitoring (Abschnitt 2.3). In Kapitel 3 werden verwandte Arbeiten vorgestellt und Unterschiede zu dieser Arbeit aufgezeigt, Kapitel 4 beinhaltet den Hauptteil. Am Anfang des Kapitels werden ausgewählte Anforderungen von IoT-Anwendungen definiert und beschrieben. Eine Auswahl an Fehlerszenarien in einem IoT-Netzwerk wird in Abschnitt 4.2 kategorisiert und analysiert, um entsprechende Methoden zur Fehlererkennung und Fehlerbehandlung auszuarbeiten. Dabei wird vorhandene Literatur strukturiert und mit den definierten Fehlerkategorien verknüpft. Mögliche Vorgehensweisen zur Erkennung von Fehlern werden in Abschnitt 4.3 vorgestellt. Abschnitt 4.4 beschreibt Konzepte zur Fehlerbehandlung und -vermeidung. Diese Methoden zur Fehlererkennung und -behandlung enthalten sowohl strukturierte und angepasste State of the Art Techniken, als auch selbst entwickelte Konzepte. Um die genannten Methoden umzusetzen, wird eine generische Architektur eines IoT-Netzwerkes in Abschnitt 4.5 beschrieben, welche in der Lage ist, die beschriebenen Fehler zu erkennen und zu behandeln. Ein implementierter Prototyp in der Programmiersprache Python zeigt die Umsetzbarkeit dieser Konzepte und soll hierfür eine praktische Anwendung darstellen. Beschrieben wird die Implementierung in Kapitel 5. Es folgt eine kurze Evaluation der Konzepte in Kapitel 6, die die Umsetzbarkeit diskutiert und Schwierigkeiten bei der Erkennung und Behandlung von Fehlern in IoT-Umgebungen aufzeigt. Den Abschluss dieser Arbeit bildet eine Zusammenfassung der Ergebnisse (Kapitel 7) und einen Ausblick auf weitere Möglichkeiten für die Verbesserung der Ausfallsicherheit und Robustheit in IoT-Netzwerken.

2 Grundlagen

Dieses Kapitel enthält die, für das Verständnis der in Abschnitt 4.4 vorgestellten Konzepte, notwendigen Grundlagen. Diese Grundlagen beinhalten die Einführung in das Thema Internet of Things, sowie Edge Computing (Abschnitt 2.1). Außerdem werden die in der Implementierung verwendeten Techniken, wie das Nachrichtenprotokoll MQTT (Abschnitt 2.2) und Monitoringkonzepte (Abschnitt 2.3) vorgestellt.

2.1 Internet of Things (IoT)

Der Begriff Internet of Things definiert eine Technologie von vernetzten Systemen, in denen die Hauptkommunikation nicht nur die Human-to-Human bzw. Human-to-Machine Kommunikation ist, sondern auch die Machine-to-Machine Kommunikation [LN10b]. Anwendungen lassen sich dadurch in starker Weise automatisieren und vernetzte Geräte können Aufgaben eigenständig, ohne menschlichen Eingriff lösen.
Jedes Gerät bekommt eine eindeutige Identität zugewiesen und kann so mit anderen Geräten kommunizieren oder Befehle entgegennehmen. Durch den ungehinderten Austausch von Daten und der Fähigkeit Daten zu verarbeiten, spricht man bei diesen Geräten meist von Smart Devices. Solche Geräte sind in den meisten Fällen auch mit dem Internet verbunden und bilden mit Services und der Cloud eine Schnittstelle für den Benutzer, um Befehle einzuspeisen oder Daten abzufragen. IoT-Geräte können mit Sensoren und Aktoren ausgestattet sein [VF13]. Sensoren erfassen, meist in einem bestimmten Intervall, physikalische Daten aus ihrer Umgebung, wie zum Beispiel Temperatur, Bewegung oder den aktuellen Standort [DP11]. Die gemessenen Daten werden in dem Netzwerk entweder dezentral verarbeitet oder an eine zentrale Komponente weitergeleitet und entsprechend die Aktoren gesteuert.

Internet of Things findet in der Industrie Verwendung, aber auch in privaten Haushalten Verwendung [LDB04]. Die sogenannte Industrie 4.0 wird mittlerweile von der Verwendung von IoT-Geräten geprägt [Sin17]. Meist werden diese für die Automatisierung und Erfassung von Daten an Produktionsmaschinen, welche mittels Echtzeitauswertung beispielsweise der Geräteüberwachung dienen, verwendet. Außerdem kann der Mitarbeiter durch IoT besser in den Produktionsprozess eingebunden werden, zum Beispiel, dass ihm nach dem Scannen eines Bauteils auf einem Tablet verschiedene Spezifikationen und Arbeitsschritte angezeigt werden. Im Allgemeinen beschleunigt die Vernetzung der Geräte die Produktion und die Reaktionen auf fehlerhafte Produktionsabläufe.

Smart Devices im privaten Gebrauch sind Teil des Smart Homes. Verschiedenste Geräte im Haushalt, zum Beispiel Lampen, die Heizungsteuerung oder das Garagentor, können durch die Kommunikation untereinander automatisiert handeln und auf Werte von angebundenen und vernetzten Sensoren reagieren. Zusätzlich lassen sich alltägliche Geräte intelligent automatisieren und per Internet vom Anwender steuern.

Dank der Steigerung in Leistung und Kompaktheit von Mikroprozessoren wird es immer einfacher Geräte intelligent zu machen, das heißt, programmierbare Anwendungen auszuführen, Daten auszuwerten und eine Verbindung mit dem Internet herzustellen. Durch drahtlose Kommunikationstechniken und leistungsstärkere Akkus werden Geräte zudem unabhängiger von kabelgebundenen Netzwerken und Stromversorgungen und können somit einfach an verschiedensten Orten eingesetzt werden.

2.1.1 Edge Computing

Edge Computing ist eine Erweiterung des Cloud Computing und beschreibt die Verlagerung der Rechenleistungen weg von einer zentralen Komponente, wie zum Beispiel einem Rechenzentrum, an den äußeren Rand des Netzwerks [SCZ+16]. Mittels Edge Computing sollen vor allem Probleme, wie die hohe Latenz und die hohen Kosten für einen Netzzugang mit hoher Bandbreite adressiert werden. Da die Daten überwiegend am Rand des Netzwerks, zum Beispiel durch Sensoren. produziert werden, ist es deutlich effizienter auch dort die Daten zu verarbeiten [SCZ+16].
Die Verarbeitung der Daten kann somit lokal, ohne deren Übertragung und Verarbeitung in einer zentralen Komponente erfolgen. Die Steigerung der Leistungsfähigkeit wird durch das *proof-of-concept* System von Yi et al. deutlich: In der Anwendung für Gesichtserkennung wurde durch die Dezentralisierung der Berechnungen die Reaktionszeit von 900 ms auf 169 ms gesenkt [YHQL15]. Durch die Nutzung von *Cloudlets* - kleine Rechenzentren verteilt am Rand des Netzwerks, konnte außerdem der Energiebedarf einer Anwendung für tragbare kognitive Unterstützung um 30-40 Prozent gesenkt werden [HCH+14].

2.2 Message Queuing Telemetry Transport (MQTT)

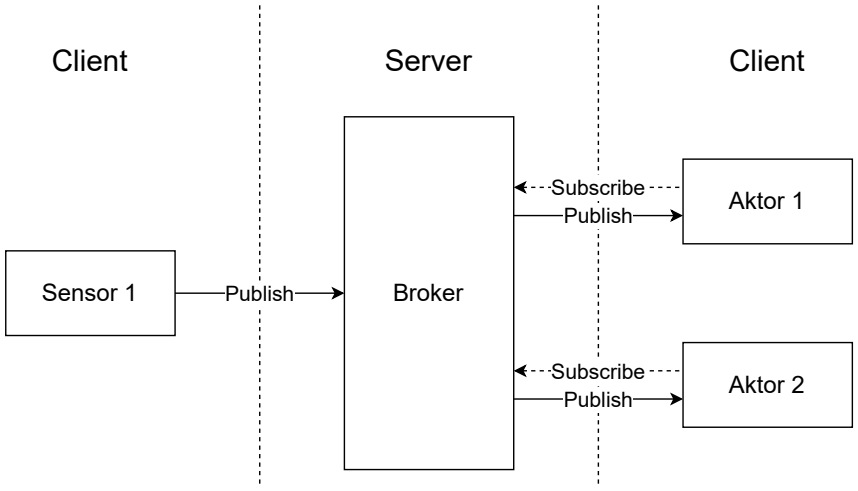

Abbildung 2.1: MQTT Client-Server Umgebung mit einem Sensor und zwei Aktoren

Message Queuing Telemetry Transport ist ein Nachrichtenprotokoll auf der Anwendungsschicht des OSI-Schichtmodells für die Machine-to-Machine Kommunikation [X2094]. Es handelt sich um ein Publish/Subscribe-Verfahren, bei dem die Übertragung von Telemetriedaten in Form von Nachrich-

ten immer zwischen Client und Server stattfindet. Zwingend benötigt wird eine TCP-Verbindung, da MQTT verbindungsorientiert arbeitet [ABBG19].
Wie in Abbildung 2.1 veranschaulicht, können Geräte dem sogenannten *Broker* nach dem Verbindungsaufbau *Publish*-Nachrichten senden. Diese enthalten einen Header mit fester Länge, in dem unter anderem Quality-of-Service Werte definiert sein können, sowie einen Header mit variabler Länge und die zu übertragenden Daten. Andere Geräte abonnieren sogenannte *Topics*, um die darin veröffentlichten Nachrichten vom Broker geliefert zu bekommen.

2.2.1 Quality-of-Service (QoS)

Quality-of-Service bezeichnet die Güte der Kommunikation aus Sicht des Anwenders. Formal werden in QoS Anforderungen an das Verhalten des Systems definiert (Tabelle 2.1). In MQTT werden vier verschiedene QoS definiert.

QoS Wert	Bit 2	Bit 1	Beschreibung
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	reserved

Tabelle 2.1: QoS Definition in MQTT

At most once delivery
Die zum Broker veröffentlichte Nachricht wird zum Broker gesendet, dort an alle verbundenen *Subscriber* einmalig weitergeleitet und sofort danach jegliche Referenzen dazu gelöscht. Die Nachricht wurde also „gesendet und vergessen“. Sowohl der Absender als auch der Broker können keinerlei Aussage treffen, ob die Nachricht bei allen *Subscribern* dieses *Topics* angekommen ist.

At least once delivery
Hierbei wird dem Sender versichert, dass die Nachricht mindestens einmal bei allen *Subscribern* des *Topics* angekommen ist. Der *Publisher* und der Broker behalten diesmal die Nachricht nach dem Senden im Speicher. Erst wenn der Broker an alle *Subscriber* des jeweiligen *Topics* die Nachricht weiterleiten kann, löscht er diese und sendet eine Bestätigung des *Publish* (PUBACK) an den ursprünglichen Sender zurück. Der Absender kann nun sicher sein, dass die Nachricht bei allen anderen Geräten angekommen ist.

Exactly once delivery
Die Auslieferung der Nachricht wird hier gewährleistet, in dem der Broker nach der Weiterleitung der Nachricht an alle *Subscriber* zunächst eine Bestätigung vom Erhalt der Nachricht (PUBREC) sendet. Sobald der Absender daraufhin die Nachricht PUBREL an den Broker sendet, löscht dieser die Nachricht und sendet die Beendung der gewährleistenden Übertragung (PUBCOMP) wieder zurück. Die bis dahin gespeicherte Nachricht beim Absender wird daraufhin gelöscht.

2.2.2 Topics

Im Header mit variabler Länge befindet sich das *Topic*, welches die Nachricht hierarchisch einstuft, zum Beispiel *stockwerk1/raum2/gerät1*. Die verschiedenen Hierarchieebenen werden durch ein Slash „/“ getrennt.
Andere Clients, die *Subscriber*, können ein *Topic* abonnieren und bekommen anschließend bei einer Veröffentlichung einer neuen Nachricht in diesem *Topic* diese ausgeliefert. Durch die *Wildcard #* werden alle *Topics* ab der jeweiligen Hierarchieebene abonniert. Mit einem + wird eine Ebene als *Wildcard* gesetzt.

2.2.3 MQTT-S

Eine Erweiterung und Anpassung von dem MQTT-Protokoll an leistungsschwache IoT-Netzwerke ist MQTT-S [HTS08]. Dabei wurde vor allem darauf geachtet, dass MQTT-S auf Geräten mit niedriger Rechenleistung und geringer Energieaufnahme, wie zum Beispiel Sensoren, lauffähig ist. Das Protokoll wurde außerdem für die Übertragung in Netzwerken mit wenig Bandbreite und hoher Dynamik optimiert.

MQTT-S-Gateways wurden als ein neuer Gerätetyp definiert, welcher als Schnittstelle zwischen den IoT-Netzwerken und dem MQTT Broker im ursprünglichen Netzwerk dienen soll. Ein solches Gateway kann, muss aber nicht im Broker direkt integriert sein. Es kommuniziert zu den MQTT-S-Clients per MQTT-S und zum Broker per MQTT.
Dabei wird zwischen *transparent* Gateways und *aggregating* Gateways unterschieden. *Aggregating* Gateways halten nur eine MQTT-Verbindung zum Broker offen, im Gegensatz zu den *transparent* Gateways, welche pro offener MQTT-S-Verbindung eine Verbindung zum Broker halten.

Um die in IoT-Netzwerken meist geringe maximale Größe einer Nachricht nicht zu stark zu verkleinern, verwendet MQTT-S, nicht wie MQTT die Übertragung von lesbaren Strings für Topics, sondern eine TopicID. Diese wird dem Client vom MQTT-S-Gateway mitgeteilt, sobald dieser ein Topic abonniert. Um große Nachrichten zusätzlich zu vermeiden, wird die Nachricht vom Verbindungsaufbau eines *Clients* zum Gateway in drei Teile, der ClientID, dem *Topic* und der eigentlichen Nachricht aufgeteilt.

2.3 Monitoring

Als Monitoring[1] bezeichnet man die Überwachung eines Vorgangs, Prozesses oder Gerätes. Verschiedene Parameter werden gesammelt, ausgewertet und zum Beispiel bei Überschreitung eines gewissen Grenzwertes registriert und ein Alarm ausgegeben. Das Ziel von Monitoring ist, den fehlerfreien Ablauf eines Prozesses sicherzustellen und bei Abweichung der vorher definierten Ziele eine Warnung auszugeben.

In der Informationstechnologie können solche vordefinierten Ziele zum Beispiel Service Level Agreements (SLAs) sein. SLAs definieren mittels Festlegung von QoS eine bestimmte Qualität eines Services zwischen Nutzer und Dienstleister [PRS09].
Ein gutes Monitoring zeichnet die frühzeitige Erkennung von Ursachen für eine Verletzung solcher SLAs aus.

2.3.1 Cloud Monitoring

Um verschiedene Methoden des Cloud Monitorings im späteren Verlauf dieser Arbeit auf das IoT zu übertragen, werden zunächst im folgenden Abschnitt die Grundlagen vorgestellt.
Durch den Anstieg der Nutzung von Software-as-a-Service (SaaS), also Software und Dienstleistungen, die von externen IT-Dienstleistern in der Cloud betrieben werden, steigt auch der Bedarf diese Infrastruktur zu überwachen. Der Cloud-Anbieter stellt bei SaaS die Software, inklusive der darunter liegenden Hardware, bereit. Kunden nehmen diese als Dienstleistung über das Internet wahr. Der große Vorteil liegt in der schnellen Inbetriebnahme und hohen Flexibilität für den Nutzer, sowie die Abgabe der Wartungspflichten für Hardware und Software an den Dienstleister. Dadurch liegt diese Verantwortung beim Betreiber, welcher in den oben erwähnten SLAs dem Konsumenten eine gewissen Qualität seines Produktes zugesichert hat und diese mittels Cloud-Monitoring überwachen muss.

Eine große Herausforderung für das Monitoring ist die Erfassung der Gesamtheit von komplexen Systemen. Der Zustand muss nicht nur alle Komponenten beachten, sondern auch die Realität sehr genau abbilden. Vor allem die gewachsene Virtualisierung und die damit gewonnene Unabhängigkeit von Hardware erschwert das Monitoring deutlich. So können die virtuellen Ressourcen zu jeder Zeit zwischen verschiedenen physikalischen Systemen wechseln [ABDP13].

Nach Kaufman und Potter kann eine Cloud in sieben, für das Cloud Monitoring relevante, Schichten aufgeteilt werden [Spr11a; Spr11b]:

- *Facility Layer:* Diese Schicht beinhaltet die Überwachung auf physikalischer Ebene, sowie den physikalischen Zugriff auf Hardware. Dieser Facility Layer wird in den folgenden Konzepten (Kapitel 4) nicht behandelt.
- *Network Layer:* Die Netzwerkebene beschreibt die Verbindung zwischen Nutzer und Cloud-Anbieter. In unserem Anwendungsszenario stellt diese Schicht die Verbindung zwischen den einzelnen IoT-Geräten dar (Abschnitt 4.2.5, 4.2.6).

[1] https://wirtschaftslexikon.gabler.de/definition/ueberwachung-48200/version-271458

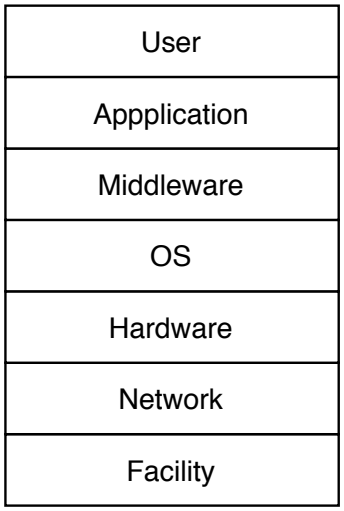

Abbildung 2.2: Cloud Monitoring in Schichten nach Kaufman und Potter [Spr11a; Spr11b]

- *Hardware Layer:* Der Hardware Layer beschreibt die physikalischen Systeme und deren Zustand. Hier findet das Monitoring von den verschiedenen Hardware-Metriken, wie Speichernutzung, Prozessorlast, Temperatur und Energieversorgung statt (Abschnitt 4.2.1, 4.2.2).
- *OS:* Das Betriebssystem hat vollen Zugriff auf alle virtuellen Maschinen und somit auch auf die Vitalität dieser virtuellen Systemen. Es gilt die reibungslose Arbeit des Host-OS zu überwachen, da nur so VMs und Anwendungen ausgeführt werden können (Abschnitt 4.2.4).
- *Middleware Layer:* Middleware beschreibt einen sehr großen Bereich im Cloud Computing. Darunter fällt unter anderem das Virtualisierungs-Management, die Datenkonvertierung oder auch das Ausführen von Sicherheitsfunktionen zwischen unterschiedlichen Plattformen. Im Folgenden werden keine Konzepte zur Überwachung oder Behandlung von Fehlern in dieser Schicht behandelt.
- *Application Layer:* Die Anwendungs-Schicht ist für Software-as-a-Service eine sehr wichtige Komponente aber zugleich auch durch die Vielzahl an Codes und Features am anfälligsten für Fehler. Die Software muss die zuvor abgeschlossenen SLAs erfüllen und in den zugesicherten Quality-of-Services auch überwacht werden (Abschnitt 4.2.3, 4.2.2).
- *User:* Die letzte Schicht bildet der User. Hier ist ein nur schwer überwachbares großes Fehlerpotential durch die Interaktion des Nutzers vorhanden. Monitoring und Fehlerbehandlung von dieser Schicht wird in der vorliegenden Arbeit nicht behandelt.

2.3.2 Monitoringverfahren

Die meisten Monitoringverfahren basieren auf der Überwachung von verschiedenen Metriken der Komponenten. Hardware und Geräte können zum Beispiel durch Vitalparameter, wie zum Beispiel CPU Auslastung, Temperaturen und Energieversorgung überwacht werden [ABDP13].

Ein weitere Möglichkeit zur Überwachung von der Funktion eines Systems ist das Durchführen von Tests. Solche Tests können in zwei Kategorien aufgeteilt werden [ABDP13]:

- *Computation-based:* Generische Tests können zum Beispiel die Anzahl der möglichen Requests an einen Server pro Zeiteinheit oder die Reaktionszeit des Systems messen. Die benötigte Zeit für die Ausführung einer Funktion des Systems kann dadurch herausgefunden werden und so bei Überschreitung eines Grenzwertes eine Warnung ausgegeben werden.
- *Network-based:* Durch das Verschicken von Testpaketen kann in Netzwerken zum Beispiel die Round-Trip Zeit, die Paket- beziehungsweise Datenverluste und die verfügbare Bandbreite gemessen werden.

Das Überwachen der Metriken und das Testen kann dabei zentral von einer Monitoring-Komponente oder dezentral von den Geräten eines IoT-Netzes durchgeführt werden.
In dieser Arbeit werden Konzepte, sowohl mit zentralem, als auch dezentralem Monitoring vorgestellt.

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt und Unterschiede zu dieser Arbeit aufgezeigt.

Seeger et al. beschreiben ein Konzept um dynamische, dezentrale IoT-Systeme robuster zu machen [SDSB19]. Dazu gehört die automatische Erfassung von neuen Geräten und deren Eigenschaften, genannt *offerings*. Diese sind in funktionale Eigenschaften, wie Inputs und Outputs, sowie nicht-funktionale Informationen aufgeteilt. Letzteres kann beispielsweise der Standort oder andere administrative Merkmale sein. Um IoT-Anwendungen automatisch auf Geräte zu verteilen, werden sogenannte *Recipe Runtime Configurations* erstellt. Diese dienen der Verknüpfung von Programminstanzen mit den jeweils benötigten Ressourcen. Dieses Konzept wird in dieser Arbeit verwendet, um einen Abgleich von zu Beginn definierten Konfigurationen der Geräte durchzuführen. Weiter beschreiben Seeger et al. einen Mechanismus zur Erkennung und Vorhersage von Geräteausfällen und der Verschlechterung von Netzwerkverbindungen bis zum Abbruch. Das in dieser Arbeit vorgestellte Konzept erweitert Seegers Arbeit, sodass zusätzlich die Erkennung von Sensorfehlern, sowie Sensordatenfehlern möglich ist. Ebenfalls können mit dem im Folgenden vorgestellten Konzept Fehler und Abstürze von IoT-Anwendungen erkannt und behandelt werden. Die Überwachung der Geräte erfolgt durch die Berechnung einer Funktion, die die Ausfallwahrscheinlichkeit eines einzelnen Gerätes angibt. Die Rechengrundlage bieten empfangene *Heartbeats*, die wahlweise auch eine Angabe von verbleibenden Ressourcen, wie zum Beispiel den Batteriestatus, enthalten können. Zusätzlich beschreiben Seeger et al. eine Möglichkeit zur Berechnung der Stabilität von Netzwerkverbindungen. Die definierten Berechnungsvorschriften werden teilweise für diese Arbeit verwendet, um die Wahrscheinlichkeit eines Geräteausfalls zu bestimmen.

Um dezentrale IoT-Netzwerke robuster und toleranter gegen Fehler zu machen, beschreiben Penn et al. in „Decentralized Fault Tolerance Mechanism for Intelligent IoT/M2M Middleware“ ein weiteres Konzept [SSH+14]. Durch das Überwachen des Vorgängerknotens bildet sich ein Monitoring Ring, sodass zwar alle Ausfälle detektiert werden, jedes Gerät aber jeweils nur ein anderes überwachen muss. Die Erkennung eines Ausfalls kann dadurch mit einer geringeren Latenz erfolgen, als wenn das Monitoring zentral oder durch eine Anwendung in der Cloud übernommen wird. Penn et al. beschreibt nur die Erkennung eines kompletten Geräteausfalls auf eine dezentrale Weise. Fehlerbehandlungen sehen nur den Wiederaufbau des Monitoring-Rings vor und die Eingliederung von Ersatzgeräten in diesen. Um die Toleranz gegenüber Fehlern zu erhöhen, muss allerdings auch die Funktionalität zusätzlich zur Überwachung nach einem Ausfall wieder hergestellt werden. Mechanismen dafür und für die Erkennung von weiteren Fehlerkategorien, wie beispielsweise Netzwerkfehler, werden in dieser Arbeit beschrieben.

Chakraborty et al. zeigen in „Fall-curve: A novel primitive for IoT Fault Detection and Isolation“ eine Methode zur Erkennung und Behandlung von Sensordatenfehlern in IoT-Umgebungen [CNC+18a]. Dabei wird hauptsächlich die Erkennung von Fehlern auf der Hardwareebene des Sensors beschrieben. Laut Chakraborty et al. ist es schwer, Datenfehler von echten Daten zu unterscheiden und durch die verschiedenen Voraussetzungen trifft das auch auf IoT-Umgebungen

zu. Um einen Sensor als defekt zu klassifizieren, messen Chakraborty et al. Spannungsverläufe nach Abschalten eines Sensors. So sollen fehlerhafte Hardwarekomponenten, wie zum Beispiel Kondensatoren in Sensoren erkannt werden und Datenfehler daraus ableitbar sein. Im Unterschied zu Chakrabortys Herangehensweise, soll diese Arbeit Methoden vorstellen, wie Sensordatenfehler und Sensorausfälle durch Überwachungsanwendungen und Datenfilter erkannt werden können und zusätzlich, sofern möglich, vor einer Einschränkung in der Funktionalität behandelt werden.

Um Anwendungen automatisch auf Geräte in IoT-Netzwerken zu verteilen und zu verknüpfen, beschreiben Franco da Silva et. al die Software Multi-purpose Binding and Provisioning Platform (MBP)[1]. Mittels SSH-Zugriff, erlaubt MBP das Management einzelner Geräte inklusive deren Sensoren und Aktoren [FHM19; FHPM18]. Ein IoT-Gerät kann registriert werden und mittels sogenannten *extraction operators* mit Sensortypen verknüpft werden. Diese *extraction operators* sind Anwendungen zur Erfassung von Sensordaten, und können allerdings für beliebige Skripte, die auf dem Gerät laufen sollen, verwendet werden. MBP wird in dieser Arbeit verwendet, um Anwendungen automatisch auf IoT-Geräte zu verteilen und den Anwendungsstatus abzufragen. In *An Approach for CEP Query Shipping to Support Distributed IoT Environments* beschreibt Franco da Silva die automatische Verteilung mithilfe von MBP von CEP-Queries auf IoT-Geräte [FHPM18]. Dabei werden die technischen Voraussetzungen zum Ausführen einer solchen Query mit einem Modell beschrieben. Dieses Modell kann für Anwendungen und Sensorapplikationen erweitert werden, um diese ebenfalls automatisch auf Geräte zu verteilen. So können beliebige Anwendungen auf entfernten IoT-Geräten automatisch geladen und gestartet werden. Eine nützliche Option stellt das bereitgestellte passive Monitoring dar, welches es erlaubt durch die Schnittstelle von MBP den Status der Anwendung abzufragen.

In „Idea: A System for Efficient Failure Management in Smart IoT Environments“ [KKSS16] beschreiben Kodeswaran et al. ein System, um Fehler in IoT-Netzwerken mit vielen Sensoren zu erkennen. Dafür werden Modelle und Abhängigkeiten zwischen Sensoraktivitäten im Regelbetrieb erstellt. Beispielsweise ist eine Erhöhung der Temperatur auf allen Sensoren im Raum, wenn auch unterschiedlich stark messbar und nicht nur auf einem. Zusätzlich verbinden weitere Regeln Sensorereignisse mit zeitlichen Abläufen, wie zum Beispiel die Aktivierung einer Kaffeemaschine mit einer bestimmten Uhrzeit morgens am Tag. Es entstehen Regeln, die mit einer gewissen Wahrscheinlichkeiten gelten, sofern bei den Sensoren oder im IoT-Netzwerk keine Fehler vorliegen. Alle Sensordaten werden zur Laufzeit nach diesen Regeln ausgewertet, wodurch bestimmt werden kann, ob sich eine Menge von Sensoren normal verhält oder ein Problem vorliegt und ein Alarm ausgeben werden muss. Im Gegensatz zu dieser Arbeit beschreiben Kodeswaran et al. keine anderen Möglichkeiten zur Fehlerbehandlung außer die Wartung der Sensoren durch entsprechendes Personal. Die automatische Fehlerbehandlung beziehungsweise Fehlervermeidung stellt allerdings einen wichtigen Faktor dar, um die Robustheit in IoT-Netzwerken zu erhöhen.

Teil dieser Arbeit ist es, verwandte Arbeiten zu strukturieren und durch die Kategorisierung (Abschnitt 4.2) miteinander in Verhältnis zu setzen.

[1] https://github.com/IPVS-AS/MBP/blob/master/README.md

4 Verbesserung der Robustheit und Ausfallsicherheit in IoT-Umgebungen

Mit ansteigender Komplexität in IoT-Umgebungen steigt auch die Wahrscheinlichkeit für Fehler [SDSB19]. Um trotzdem Robustheit und Ausfallsicherheit zu gewährleisten, müssen diese Fehler erkannt und behandelt werden. Ein Fehler wird im Federal Standard 1037C [TD96] definiert als (a) ein zufälliger Zustand, der dazu führt, dass eine Funktionseinheit ihre erforderliche Funktion nicht mehr erfüllt, (b) ein Defekt, der eine reproduzierbare Fehlfunktion verursacht oder (c) ein unbeabsichtigter oder teilweiser Kurzschluss zwischen zwei erregten Leitern oder einem erregtem Leiter und Masse. Das Gerät kann dabei nur temporär oder permanent in einem solchen Fehlerzustand sein. In dieser Arbeit werden Fehler als eine Funktion oder Prozess, welcher nicht das erwartete Ergebnis liefert oder das gewünschte Verhalten aufzeigt, bezeichnet. Dazu zählt außerdem der komplette Ausfall einer Funktion oder eines gesamten Systems. Diese Fehler finden dabei zur Laufzeit des Gerätes statt und sind keine Resultate aus falschen Annahmen oder Modellen.

Um Leistungsanforderungen, wie kleine Latenzen, größere Flexibilität und niedrigere Kosten zu erreichen, wird die Datenverarbeitung im Cloud Computing und in IoT-Netzwerken immer mehr dezentralisiert [DH19]. Wenn Daten nahe an der Quelle verarbeitet werden, ergibt sich eine bessere Datensparsamkeit und Kosteneffizienz. Durch die Auslagerung der Rechenressourcen auf mehrere Komponenten steigt jedoch automatisch auch das Risiko eines Ausfalls oder Fehlers im gesamten Netzwerk [RRN+18]. Übertragungen von Daten finden meist drahtlos statt und variieren dadurch stärker in Geschwindigkeit und Zuverlässigkeit als kabelgebundene Netzwerke [RRN+18]. Durch die dadurch resultierende Möglichkeit zur Positionsänderungen von IoT-Geräten, kann zudem die Netzwerktopologie zu jedem Zeitpunkt stark variieren [ASSC02]. Um alle Geräte miteinander zu vernetzen, werden mehrere Kommunikationswege und Verbindungen benötigt. Die Erhöhung der Anzahl von Verbindungen, sowie die Schwankungen in der Geschwindigkeit und Zuverlässigkeit, lassen die Fehlerrate im Netzwerk zusätzlich steigen. Nicht nur der komplette Geräteausfall stellt eine Herausforderung dar, sondern auch Fehler in Anwendungen oder im Betriebssystem müssen erkannt und behandelt werden, um eine Verbesserung der Robustheit in IoT-Umgebungen sicher zu stellen. In IoT-Netzwerken sind außerdem die Sensoren zu beachten, die fehlerhafte Werte liefern oder komplett ausfallen können. In kritischen Anwendungsfällen stellt dies einen schwerwiegenden Fehler dar.

4.1 Anforderungen in IoT-Netzwerken

Für Anwendungen in IoT-Netzwerken müssen Anforderungen definiert werden, um Fehler richtig klassifizieren zu können und die Prioritäten in der Fehlererkennung festzulegen. Abhängig von den Anforderungen müssen Erkennung und Behandlung von Fehlern unterschiedlich leistungsfähig sein. Dabei reicht es zum Beispiel für eine private Wetterstation, wenn ein Fehler am Temperatursensor

zunächst nur gemeldet wird. Bei einem Sensorausfall bei einer Temperaturüberwachung in einer Produktionsmaschine dagegen, sollten so schnell wie möglich Sofort-Maßnahmen, wie zum Beispiel das Abschalten der Maschine, eingeleitet werden.
Während in manchen Szenarien ein zeitweiser Verlust von nicht übertragenen Sensordaten toleriert werden kann, muss in anderen Umgebungen, wie zum Beispiel bei Anwendungen in der Überwachungstechnik, eine vollständige Übertragung gewährleistet werden [HN16]. Durch das Annotieren und korrekte Verarbeiten von Nachrichten mit Quality-of-Service-Anforderungen können solche Voraussetzungen an die Übertragung umgesetzt werden. Bei MQTT kann, wie bereits in Abschnitt 2.2.1 beschrieben, mit der *At-least once delivery* oder der *Exactly once delivery* eine solche Voraussetzung definiert werden. Auch Anwendungen, die in Echtzeit agieren, stellen besondere Anforderungen. Nicht nur die Übertragung der Daten muss ohne Verlust gewährleistet sein, sondern auch die Verarbeitung muss in Echtzeit ohne Fehler ablaufen können. Um dies zu garantieren, muss beispielsweise auch die Fehlererkennung, sowie die Fehlerbehandlung in Echtzeit ablaufen, sodass mögliche Gefahren im weiteren Anwendungsprozess verhindert werden und frühzeitig eingeschritten werden kann. In manchen Anwendungen müssen verloren gegangene Nachrichten möglichst schnell nachgereicht werden, wohingegen in anderen wiederum keine Notwendigkeit besteht diese Nachrichten erneut zu übertragen, da es sich inzwischen nicht mehr um relevante Daten handelt.
Eine weitere Anforderung, die von der Anwendung abhängt, ist die Toleranz der Abweichung eines gemessenen Wertes. Bei einer geringen Toleranzgrenze müssen vor allem Datenfehler (Abschnitt 4.2.2) genauer erkannt werden, um Probleme in der IoT-Umgebung zu vermeiden.

In dieser Arbeit werden folgende Anforderungen definiert: Tabelle 4.1

Anforderung	**Beschreibung**	**Beispiel**
At-least once delivery	Der Empfänger muss mindestens einmal die Daten empfangen können. Es muss also eine verlustfreie Übertragung gewährleistet sein.	Benachrichtigung über ein Ereignis im Smart Home
exactly once delivery	Übertragene Daten werden in jedem Fall nur einmal an den Empfänger gesendet und von diesem empfangen. Diese Übertragung muss bestätigt sein.	Industrielle Produktion mit situationsbedingten Entscheidungen
kein Datenverlust	Gemessene oder verarbeitete Daten dürfen nicht verloren gehen.	Überwachung von Vitalparametern in Produktionsanwendungen
kein Datenverlust für x Minuten	Daten können in einem Intervall von x Minuten verloren gehen. Mindestens nach diesem Zeitabschnitt müssen Daten wieder gespeichert werden können.	Aufzeichnung von Statistiken
geringe Fehlertoleranz in Daten	Sensorwerte oder andere Daten dürfen nur eine sehr geringe Abweichung vom realen beziehungsweise erwarteten Wert haben.	Überwachung von kritischen Infrastrukturen
hohe Fehlertoleranz von einzelnen Daten	Einzelne Datenpunkte können eine große Abweichung aufweisen, ohne die Funktionalität der Anwendung einzuschränken.	Mittelwertberechnung über viele unkritische Daten
Echtzeit	Die Anwendungsdaten sind nur in einem beschränkten Zeitfenster relevant. Um die Funktionalität der Anwendung zu gewährleisten, muss die Übertragung in einer gewissen Zeit stattgefunden haben. Ein Datenverlust führt hier ebenfalls zu einem Fehler in der Anwendung.	Industrielle Produktion mit situationsbedingten Entscheidungen

Tabelle 4.1: Anforderungen in IoT-Umgebungen

4.2 Kategorisierung und Analyse von Fehler bei IoT-Geräten

Fehler bei IoT-Geräten oder in IoT-Netzwerken können, wie in Abbildung 4.1 dargestellt, in lokale Fehler und Netzwerkfehler unterteilt werden. Ein Netzwerkfehler tritt auf, sobald eine Eigenschaft einer funktionierenden Verbindung zwischen zwei oder mehreren Geräten nicht mehr erfüllt ist. Solche Eigenschaften sind zum Beispiel die Zuverlässigkeit der Verbindung und der Datenübertragung,

sowie akzeptable Werte bei Performanztests, welche beispielsweise Latenz und Round-Trip-Time messen [ABDP13]. Untergliedert werden diese Kategorien in instabile oder langsame Netzwerke (4.2.5), sowie den kompletten Verbindungsabbruch (4.2.6), wenn es nicht mehr möglich ist, Daten zu übertragen.
Lokale Fehler sind solche, welche unabhängig von der Netzwerkverbindung auf dem jeweiligen Gerät auftreten können. Der Defekt betrifft nur lokale Komponenten, wie zum Beispiel die Software oder Hardware dieses Gerätes. Gerätefehler (4.2.1) stellen Ausfälle der Hardware oder der Energieversorgung dar und sorgen für einen kompletten Geräteausfall. Ein weiterer Fehler, welcher in die Kategorie des Hardwarefehlers fällt, ist der Sensorausfall oder der Sensordatenfehler (4.2.2). Hierbei liefert der Sensor keine oder vom erwarteten Wert stark abweichende Sensordaten. Als Softwarefehler wird der Ausfall einer Anwendung (4.2.3) bezeichnet, welche zum Beispiel Daten verarbeitet oder Sensorwerte weiterleitet, sowie der Betriebssystemfehler (4.2.4), bei dem das Gerät vollständig abstürzt.

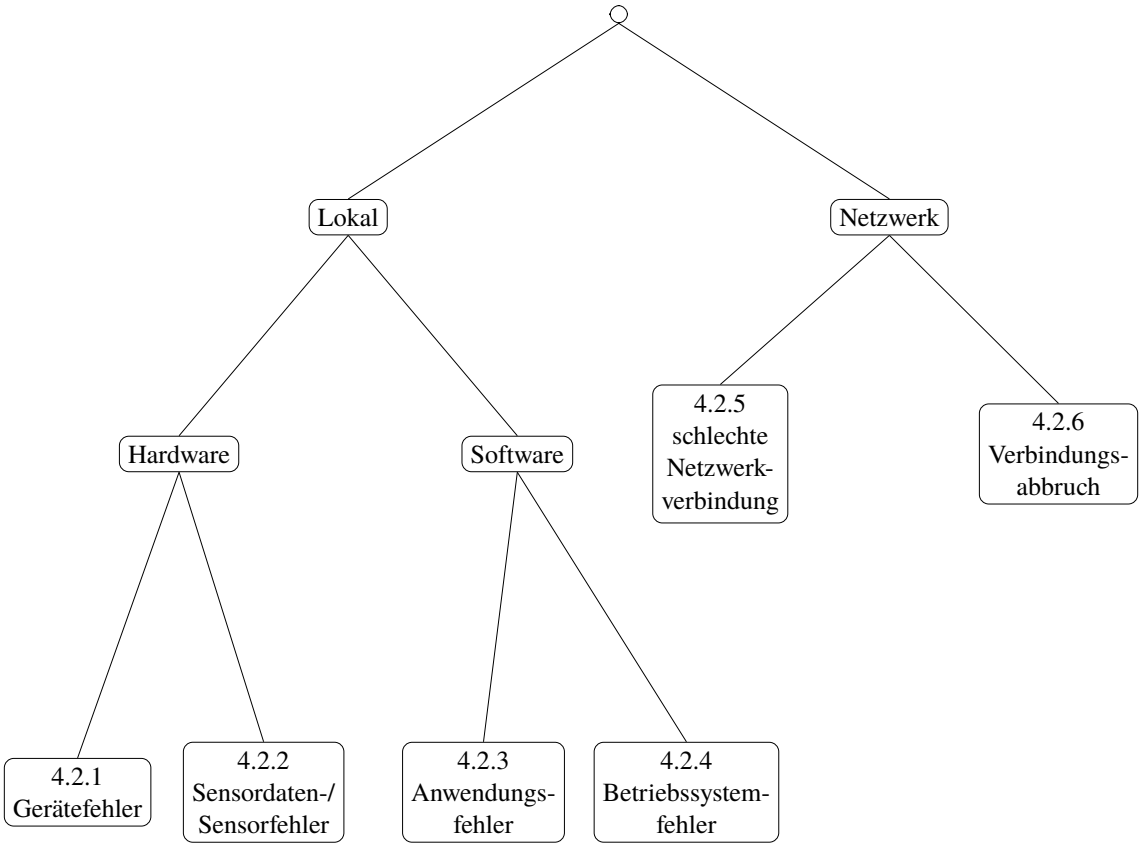

Abbildung 4.1: Fehlerkategorien in IoT-Umgebungen

Die Tabellen 4.2 und 4.4 geben einen Überblick über die Merkmale eines Fehlerzustands auf einem IoT-Gerät und werden in den einzelnen Abschnitten (4.2.1 - 4.2.6) durch den jeweiligen Fehler genauer beschrieben.

	Gerätefehler (4.2.1)	Sensorausfall/ Sensordatenfehler (4.2.2)	Anwendungsfehler (4.2.3)	Betriebssystem-fehler (4.2.4)
Gerät arbeitet nicht mehr	✓	×	×	✓
Gerät arbeitet nicht mehr korrekt	✓	✓	✓	✓
Gerät liefert keine validen Daten mehr	✓	✓	✓	✓
Gerät nicht mehr erreichbar	✓	×	×	✓
Speicher/Zustand geht verloren	✓	×	×	✓
Speicher/Zustand ist fehlerhaft	✓	✓	✓	✓
Nachrichten gehen verloren	✓	×	×	✓

✓ trifft zu × trifft nicht zu

Tabelle 4.2: Merkmale der Fehlerkategorien (Teil 1)

4.2.1 Gerätefehler

Gerätefehler fallen, wie in Abbildung 4.1 veranschaulicht, in die Kategorie der lokalen Hardwarefehler. Diese Fehlerkategorie beinhaltet beispielsweise den Verlust der Energieversorgung oder den Ausfall des IoT-Gerätes. Speziell in IoT-Umgebungen sind viele der verbundenen Geräte batteriebetrieben [RRN+18]. Damit sind diese zwar gegen einen Stromausfall abgesichert, haben aber nur eine begrenzte Menge an Energie zur Verfügung und sind somit für einen Gerätefehler aufgrund von Verlust der Stromversorgung anfälliger. In diese Fehlerkategorie fällt auch das manuelle Ausschalten und das Entfernen des Gerätes aus dem Netzwerk.
Bei einem Gerätefehler arbeitet das Gerät nicht mehr und kann auch keine Befehle mehr entgegen nehmen. Die Kommunikation mit dem fehlerhaften Gerät kann nicht mehr stattfinden. Daten auf nicht-persistenten Speichern sind verloren und können nicht wiederhergestellt werden. Durch das Fehlen von Daten befindet sich der Speicher also in fehlerhaftem Zustand. Alle Anforderungen aus Tabelle 4.1 können auf diesem Gerät nicht mehr erfüllt werden.

4.2.2 Sensorausfall und Sensordatenfehler

Ein weiterer lokaler Fehler, der auf einem Gerät mit Sensoren auftreten kann, ist der Sensorfehler. Gründe hierfür kann ein elektrotechnischer Fehler in der Hardware des Sensors sein. Bei einem Ausfall liefert der angeschlossene Sensor keine validen Daten mehr und speichert eventuell auch fehlerhafte Daten ab. Sensordatenfehler werden in dieser Arbeit anhand die Kategorisierung von Chakraborty et al. in Tabelle 4.3 unterteilt und beschrieben [CNC+18b].

Short	Ein Datenpunkt der signifikant von dem erwarteten Wert oder Zeitpunkt abweicht
Spikes	Eine Änderungsrate über einen kürzeren oder längeren Zeitabschnitt, die deutlich größer ist als erwartet. Dabei muss sich die Änderung nicht wieder normalisieren
Stuck-at	Eine Datenserie, die über einen langen Zeitraum nicht mehr oder nur noch sehr gering variiert
Noise	Daten, die über einen gewissen Zeitabschnitt unerwartet sehr stark variieren
Calibration	Sensordaten haben eine gleichbleibende Verschiebung gegenüber des wahren Wertes

Tabelle 4.3: Unterteilung von Sensordatenfehlern nach Chakraborty et al. [CNC+18b]

Um Sensordatenfehler korrekt zu definieren, muss außerdem immer der Anwendungsfall betrachtet werden. Die Beschreibungen in Tabelle 4.3 sind immer in Hinblick auf die zu erwarteten Daten der jeweiligen Anwendung zu sehen. Von I. Lee und K. Lee wird zum Beispiel die Anwendung von IoT in der Überprüfung von Kühlketten genannt [LL15]. In diesem Beispiel wird die Temperatur von Gütern mittels Sensoren in einem IoT-Netzwerk überwacht. Ein *Short* und *Spike* muss dort beispielsweise sehr genau erkannt werden, wohingegen es den Fehler *Stuck-at* praktisch nicht gibt, da erwartet wird, dass der gemessene Wert gleich bleibt.
Sowohl bei dem Ausfall des Sensors als auch bei einem Sensordatenfehler ist zu jeder Zeit das Gerät sowie die Anwendung erreichbar und der Nachrichtenaustausch kann ohne Einschränkungen stattfinden.

4.2.3 Anwendungsfehler

Ein Anwendungsfehler bedeutet meist ein Absturz der Software, welche Daten verarbeitet, erzeugt oder weiterleitet. Da diese Anwendung essentiell für den einwandfreien Ablauf ist, kann die fehlerfreie Ausführung der Aufgabe dieses IoT-Gerätes nicht mehr gewährleistet werden.
Infolgedessen kann die Übertragung der Daten Lücken aufweisen oder komplett unterbrochen sein. Sollte es sich hierbei um ein Gerät handeln, welches Daten verarbeitet, kann dieses außerdem Zustandsfehler, die durch fehlende Daten entstanden sind, aufweisen. Die Erreichbarkeit ist trotzdem zu jedem Zeitpunkt gegeben und eine Kommunikation mit anderen Knotenpunkten im Netzwerk kann wie bisher stattfinden.

4.2.4 Betriebssystemfehler

Ein Fehler im Betriebssystem ähnelt den Merkmalen von dem Gerätefehler (Abschnitt 4.2.1), ist allerdings softwareseitig. Das Gerät ist ab dem Absturz nicht mehr erreichbar und kann keine Daten mehr korrekt verarbeiten oder weiterleiten. Daten auf transienten Speichern sind nicht mehr abrufbar und gehen verloren. Durch die fehlende Verwaltung der Prozessorzeit und des Speicherplatzes von Anwendungen, können sämtliche Programme nicht mehr ausgeführt werden [TB15]. Da die

Netzwerkschnittstelle auch vom funktionierenden und fehlerfreien Betriebssystem abhängig ist, kann die Verbindung mit anderen Geräten nicht gehalten werden und die Kommunikation ist nicht mehr möglich.

	schlechte Netzwerk-verbindung (4.2.1)	Verbindungsabbruch (4.2.1)
Gerät arbeitet nicht mehr	×	×
Gerät arbeitet nicht mehr korrekt	✓	✓
Gerät liefert keine validen Daten mehr	-	✓
Gerät nicht mehr erreichbar	-	✓
Speicher/Zustand geht verloren	×	×
Speicher/Zustand ist fehlerhaft	×	×
Nachrichten gehen verloren	-	✓

✓ trifft zu - trifft teilweise zu × trifft nicht zu

Tabelle 4.4: Merkmale der Fehlerkategorien (Teil 2)

4.2.5 Schlechte Netzwerkverbindung

Schlechte Netzwerkverbindungen zeichnen sich durch die Veränderung von Netzwerkparametern, wie zum Beispiel Paketverluste und hohe Antwortzeiten aus. Da unterschiedliche Topologien in Netzwerken unterschiedliche Werte in diesen Eigenschaften aufweisen, muss grundsätzlich ein Vergleich zum, im Normalfall zu erwartenden Wert gemacht werden. Auch hier hängt die Definition einer schlechten Netzwerkverbindung stark vom Anwendungsfall ab und kann nicht pauschal definiert werden. In kritischen Anwendungsfällen ist der Grenzwert eines Netzwerkparameters deutlich geringer als in manchen IoT-Netzwerken mit vielen Sensoren, deren Messwerte nur in großen Zeitintervallen übertragen werden.
Lokale Prozesse, die nicht auf eine Kommunikation mit anderen Geräten angewiesen sind, werden durch die schlechte Netzwerkverbindung allerdings nicht gestört und Messwerte können beispielsweise wie bisher von Sensoren weiterhin ausgelesen werden.

4.2.6 Verbindungsabbruch

Der Verbindungsabbruch eines Gerätes bedeutet, dass es weder durch eine zentrale Komponente, noch durch andere Geräte im Netzwerk erreichbar ist. Da sich viele Internet of Things (IoT)-Geräte in kabellosen Netzen befinden, kann dieser Fehler oft auftreten. Der Empfang von Geräten kann durch Abschirmungen oder Interferenzen mit dem Funksignal gestört werden. Auch das Entfernen aus dem Sendebereich ist bei mobilen Geräte, zum Beispiel autonom agierende und fahrende Roboter oder Autos möglich. Zwischen dem Gerät und anderen Knotenpunkten kann nicht mehr kommuniziert werden und der Datenaustausch wird unterbrochen. Die Anwendung auf dem Gerät ist allerdings weiterhin ohne Internetverbindung ausführbar, sofern diese nicht unbedingt benötigt wird.

4.3 Fehlererkennung

Im folgenden Abschnitt werden Konzepte und Möglichkeiten zur Fehlererkennung der in Abschnitt 4.2 beschriebenen Fehlerkategorien aufgezeigt. Dabei werden verschiedene Maßnahmen vorgestellt, durch die diese erkannt werden können. Es wird zwischen der lokalen Fehlererkennung auf dem jeweiligen Gerät, der Erkennung des Fehlers durch eine zentrale Monitoring-Komponente und der Erkennung durch gegenseitiges Monitoring der IoT-Geräte untereinander unterschieden.
Die in Tabelle 4.5 dargestellte Übersicht zur Fehlererkennung wird in den folgenden Abschnitten näher erläutert. Wenn Fehler in einer Kategorie zwar erkannt werden können, jedoch nicht verlässlich dieser Kategorie zugeordnet werden, da Fehler aus anderen Kategorien auf die gleiche Art ebenfalls erkannt werden, wird dies als bedingt mögliche Fehlererkennung bezeichnet. In Abschnitt 4.3.1 werden die verschiedenen Komponenten eines IoT-Netzwerkes, auf denen die Fehlererkennung ausgeführt werden kann, vorgestellt und ab Abschnitt 4.4.1 folgt die Beschreibung der Erkennung der einzelnen Fehlerkategorien.

	Fehlererkennung auf dem Gerät	Fehlererkennung durch zentrale Monitoring-Komponente	Fehlererkennung durch andere IoT-Geräte
Gerätefehler	×	-	-
Sensorausfall	✓	×	×
Sensordatenfehler	✓	×	✓
Anwendungsfehler	✓	×	✓
Betriebssystem-fehler	×	-	-
schlechte Netzwerk-verbindung	✓	✓	✓
Verbindungsabbruch	✓	-	-

✓ Fehlererkennung mit Klassifizierung möglich - Fehlererkennung ohne Klassifizierung möglich × Fehlererkennung nicht möglich

Tabelle 4.5: Fehlererkennung durch verschiedene Komponenten einer IoT-Umgebung

4.3.1 Fehlererkennung durch verschiedene Komponenten einer IoT-Umgebung

Im Folgenden werden drei verschiedene Arten zur Fehlererkennung auf IoT-Geräten vorgestellt: (4.3.1.1) Lokale Fehlererkennung auf dem Gerät, (4.3.1.2) Fehlererkennung durch zentrales Monitoring und (4.3.1.3) Fehlererkennung durch andere IoT-Geräte.

4.3.1.1 Lokale Fehlererkennung auf dem Gerät

Um Fehler lokal zu erkennen, wird eine Monitoring Anwendung benötigt, die Zugriff auf Vitalparameter der Hardware und des Betriebssystems hat. Außerdem benötigt die Anwendung Zugriff auf die Netzwerkschnittstelle, um Netzwerkfehler zu erkennen. Datenfehler und Sensorfehler können nur erkannt werden, sofern die Überwachungsanwendung auch Zugriff auf Anwendungsdaten, wie zum Beispiel Sensormesswerte hat. Diese Anwendung wird nachfolgend als lokaler Agent bezeichnet und unterscheidet sich damit vom Monitoring Agent einer über das Netzwerk verbundenen Monitoring-Komponente.

4.3.1.2 Fehlererkennung durch zentrales Monitoring

IoT-Geräte können auch von einer zentralen Monitoring-Komponente überwacht werden. Die Abbildung 4.2 stellt beispielhaft ein IoT-Netzwerk mit einer zentralen Komponente zur Fehlererkennung dar. Zentrale Überwachungssysteme können Netzwerkfehler, wie zum Beispiel einen Verbindungsabbruch zu dem jeweiligen Gerät sowie eine schlechte Verbindung erkennen. Die Klassifizierung von lokalen Fehlern kann allerdings nur mittels Monitoring Agenten auf den Geräten korrekt durchgeführt werden [SWWM10]. Diese Anwendungen übermitteln eine Meldung an die zentrale Komponente, um dort entsprechende Maßnahmen zu Fehlerbehandlung einzuleiten. In dieser Arbeit wird eine Erkennung beziehungsweise Klassifizierung durch einen Monitoring Agenten allerdings als lokale Fehlererkennung bezeichnet und wird nicht von der zentralen Monitoring-Komponente durchgeführt. Daten-, Sensor- und Anwendungsfehler können nicht erkannt werden, da die Möglichkeit des direkten Zugriffs auf die Anwendungsdaten fehlt.

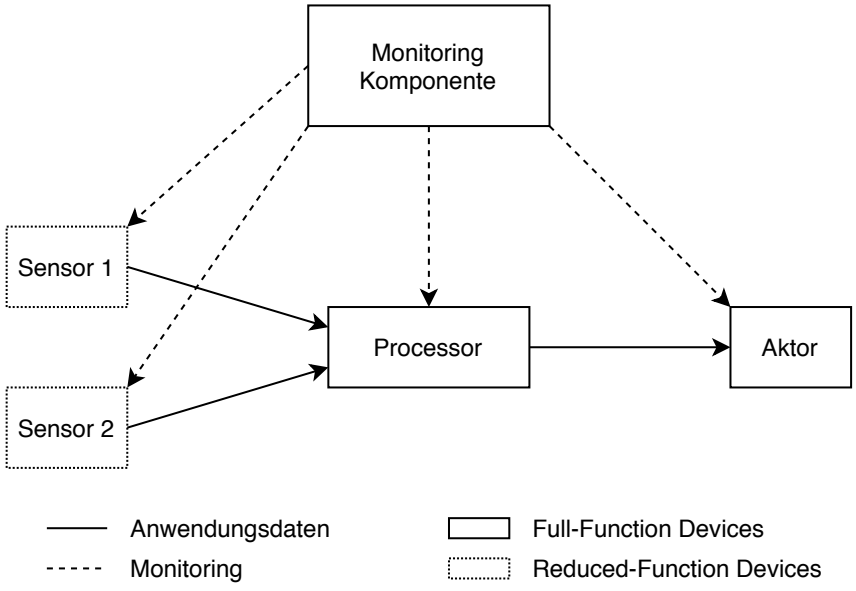

Abbildung 4.2: Zentrale Fehlererkennung durch Monitoring-Komponente

4.3.1.3 Fehlererkennung durch andere IoT-Geräte

Dezentrales Fehlermonitoring kann durch die gegenseitige Überwachung mehrerer IoT-Geräte umgesetzt werden. Durch eine dezentrale Überwachung entfällt der *Single-Point of Failure* einer zentralen Monitoring-Komponente, da die Aufgabe auf mehrere Geräte verteilt wird. Ein Beispiel hierfür stellt die Abbildung 4.3 dar. Die Geräte eines IoT-Netzwerkes müssen hierfür in *Full-Function Devices* und *Reduced-Function Devices* unterteilt werden [DD07]. *Full-Function Devices* sind alle, meist mit Strom versorgten Geräte, die eine verhältnismäßig hohe Rechenleistung und eine vollständige Netzwerkschnittstelle haben. Unter *Reduced-Function Devices* versteht man Geräte, die meist nur einfache Aufgaben erfüllen, wie zum Beispiel ein Sensor der in einem bestimmten Zeitintervall gemessene Daten versendet. Diese Geräte haben meist kein Betriebssystem, sondern sind nur mit dem jeweiligen auszuführendem Programm geflasht. Sie sind in vielen Fällen batteriebetrieben und somit auch sehr leistungsschwach. *Full-Function Devices* überwachen dabei alle *Reduced-Function Devices* und sich gegenseitig, wobei *Reduced-Function Devices* keine Monitoring Aufgabe übernehmen können. Um die maximal mögliche Redundanz und Robustheit zu erreichen, können alle Geräte alle anderen überwachen [SDSB19]. Dies beeinflusst allerdings die übrige Rechenkapazität negativ, sodass das Monitoring der IoT-Geräte auch untereinander aufgeteilt werden kann, wobei dann keine Redundanz mehr besteht.

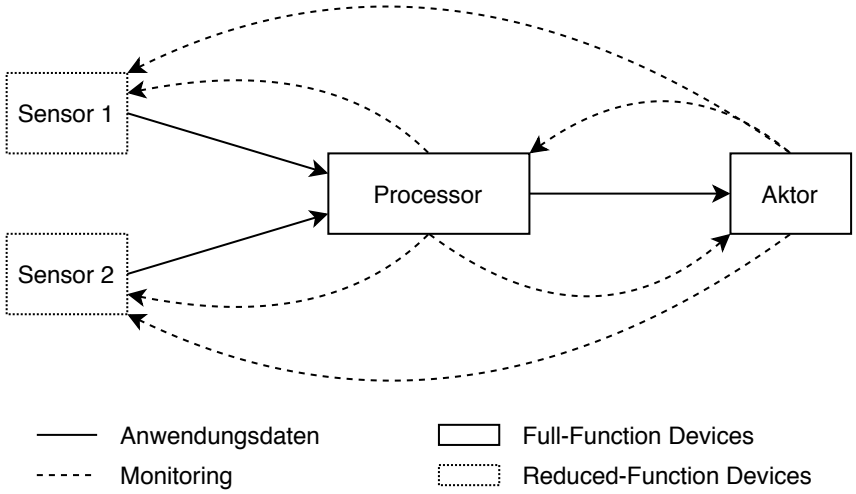

Abbildung 4.3: Dezentrale Fehlererkennung durch IoT-Geräte

Eine weitere Möglichkeit zur dezentralen Überwachung von IoT-Geräten wurde von Su et al. in „Decentralized Fault Tolerance Mechanism for Intelligent IoT/M2M Middleware“ beschrieben [SSH+14]. Um den Overhead des Monitorings für jedes Gerät minimal zu halten, überwacht jeder Knotenpunkt im Netz den Vorgängerknoten (siehe Abbildung 4.4). Der letzte Knoten schließt den Ring durch die Überwachung des ersten Gerätes. Dieses Vorgehen setzt allerdings voraus, dass es sich bei allen Geräten in der Kette um *Full-Function Devices* handelt, die genug Ressourcen haben, um die Aufgabe des Monitorings zu übernehmen. Ein Geräte- oder Betriebssystemfehler kann nun an alle Geräte effizient propagiert werden und entsprechende Maßnahmen zur Fehlerbehandlung eingeleitet werden. Die Geräte kommunizieren nach einem Ausfall, um erneut ein konsistentes Gesamtbild zu erhalten. Das gegebenenfalls dazugekommene Ersatzgerät ist dann ein Teil der Monitoring-Kette.

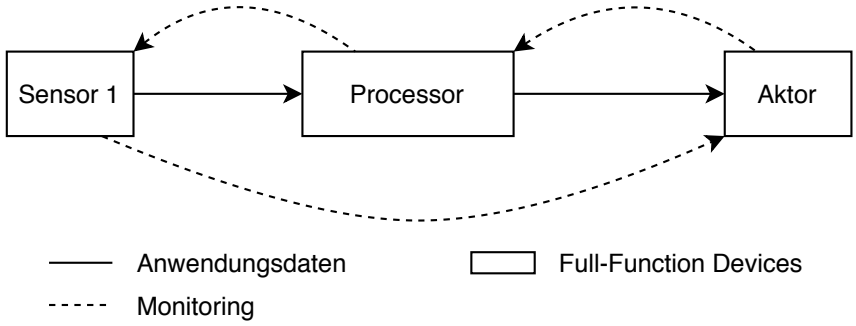

Abbildung 4.4: Dezentrale Fehlererkennung durch IoT-Geräte

4.3.2 Gerätefehler

Sollte das Gerät durch einen Gerätefehler ausfallen, kann der in Abschnitt 4.3.1.1 beschriebene Monitor Agent nicht mehr laufen und erkennt somit auch keine Fehler mehr. Ein Gerätefehler kann also nicht lokal, sondern nur durch eine zentrale Monitoring-Komponente oder durch andere IoT-Geräte festgestellt werden. Die Klassifizierung des Fehlers als hardwarebedingter Gerätefehler ist aber nur schwer möglich, da das Gerät aus Sicht einer anderen Komponente gleichermaßen wie bei dem Absturz des Betriebssystems oder eines Verbindungsabbruchs nicht mehr erreichbar ist und auf keinerlei Anfragen antwortet. Eine Möglichkeit einen Totalausfall eines IoT-Gerätes festzustellen, stellt das von Seeger et al. vorgestellte Verfahren dar [SDSB19]. Dabei laufen auf den Geräten Monitoring Agenten, welche zu einem konstanten Zeitintervall sogenannte *Heartbeats* an eine zentrale Stelle oder an andere Geräte schicken. Durch die Berechnung von Durchschnitt und Varianz der zeitlichen Abstände von *Heartbeat*-Nachrichten $t_0 < t_1 < \ldots < t_n$, wird effizient die Wahrscheinlichkeit für den Ausfall des Gerätes berechnet. Seeger et al. verwenden zur Berechnung des Durchschnitts die Summe der Abstände der bisher eingetroffenen Nachrichten ρ geteilt durch die Anzahl dieser n:

$$\mu(t) = \frac{\rho}{n} \tag{4.1}$$

Die Varianz wird wie folgt berechnet:

$$\sigma(t)^2 = \left(\frac{\kappa}{n} - \left(\frac{\rho}{n}\right)^2\right)\left(\frac{n}{n-1}\right) \tag{4.2}$$

Wobei ρ und κ nach einer bestimmten Anzahl von Nachrichten $\omega_{\max}$ zurückgesetzt werden. Um dabei die Wirkung auf die Qualität der Abschätzung zu minimieren, werden die Abschätzungen mit den alten Werten durchgeführt, bis eine gewisse Anzahl $\omega_{\min}$ an *Heartbeats* erreicht ist. Als Wert, der zunehmend auf einen Ausfall eines Gerätes hindeutet, verwenden Seeger et al. folgende Berechnungsvorschrift:

$$s(t) = -\log_{10}\left(\frac{\sigma(t)^2}{\sigma(t)^2 + \left(t - t_{n-1} - \mu(t)\right)^2}\right) \tag{4.3}$$

Durch weitere Approximation des Verlaufes einer Größe, wie zum Beispiel dem Batteriestand, können darüber hinaus zukünftig auftretende Fehler frühzeitig erkannt werden und dadurch auch beispielsweise zwischen einem Verbindungsabbruch und einem Gerätefehler unterschieden werden. Seeger et al. verwenden für diese Abschätzung Lagrange-Polynome.

$$r(t) = \sum_{j=0}^{3} \left[\rho_{t_{n-j-1}} \prod_{i=0, i\neq j}^{3} \left(\frac{t - t_{n-i-1}}{t_{n-j-1} - t_{n-i-1}} \right) \right] \tag{4.4}$$

wobei $\rho \in [0, 1]$ mit jeder *Heartbeat*-Nachricht übertragen wird und die normierte Anzahl an übrigen Ressourcen dieses Gerätes angibt. Dabei kann der aktuelle Batteriestand, aber auch die CPU-Auslastung in die Berechnung von ρ auf dem Gerät mit einfließen.

4.3.3 Sensorausfall und Sensordatenfehler

Ein Sensorausfall ist nur für Geräte erkennbar, die das Ausbleiben von Daten bemerken können. Da eine zentrale Monitoring-Komponente oder beliebige andere IoT-Geräte keinen Zugriff auf die Sensordaten in dem IoT-Netzwerk haben, kann dort kein Sensorfehler erkannt werden. Der Ausfall oder Fehler in den Sensordaten kann also nur lokal oder durch Geräte, die diese Daten verarbeiten oder weiterleiten, erkannt werden.

4.3.3.1 Lokale Fehlererkennung auf dem Gerät

Zur Feststellung eines Sensorfehlers kann die Überwachung der Kontinuität der Messwerte dienen. Wird der zuletzt gemessene Wert nicht durch eine erneute Messung erneuert, kann davon ausgegangen werden, dass ein Defekt im Sensor vorliegt und dieser keine neuen Werte ermittelt. Dabei ist das Intervall, in welchem der Sensor einen neuen Datenpunkt misst, zu beachten, um das tatsächliche Ausbleiben eines solchen Wertes festzustellen. Ähnlich wie bei der Erkennung eines Gerätefehlers (Abschnitt 4.3.2), spielt dabei vor allem der Durchschnitt und die Varianz der Größe der Zeitintervalle eine Rolle und kann zur Berechnung eines Wahrscheinlichkeitswertes für einen Ausfall dienen [SDSB19]. Die Formel 4.3 kann übernommen werden, allerdings wird hier nicht der zeitliche Abstand der *Heartbeat*-Nachrichten, sondern der Abstand der eigentlichen Sensorwerte verwendet.
Um einmalige Datenpunkte (*Short*) als Fehler zu identifizieren, können neue gemessen Daten mit dem bisherigen Verlauf der Werte abgeglichen werden [CNC+18b]. Dieser Abgleich bietet auch die Möglichkeit *Spikes* und *Stuck-at* Datenfehler zu erkennen. Bei einer starken Abweichung vom bisherigen Mittelwert liegt mit hoher Wahrscheinlichkeit ein Datenfehler vor. Bei stark schwankenden Datenreihen kann diese Aussage nicht pauschal angenommen werden.
Die Erkennung und Überprüfung von Korrelationen in Sensornetzwerken kann auch hilfreich sein. Steigt zum Beispiel der gemessene Wert eines Sensors stark an, ist es mit hoher Wahrscheinlichkeit kein Fehler, wenn der gleiche Messwert auch bei einem anderen Sensor in räumlicher Nähe ansteigt. Durch die Erkennung und Beachtung von Korrelationen zwischen Sensoren können vor allem *Spikes*, *Stuck-at* und *Calibration* Fehler erkannt werden [CNC+18b]. Eine Möglichkeit, alle Datenfehler zu erkennen, ist die Erhöhung der Redundanz von Sensoren. Durch den Abgleich eines anderen Sensors, der den gleichen Messwert aufzeichnet, sind fehlerhafte Datenpunkte schnell erkennbar.

4.3.3.2 Fehlererkennung durch andere IoT-Geräte

Handelt es sich um ein IoT-Gerät, welches Sensordaten verarbeitet oder weiterleitet, kann dieses ebenfalls eine Prüfung auf die Kontinuität und Plausibilität der erhaltenden Messwerte durchführen [SDSB19]. Sollten keine weiteren Sensorwerte empfangen werden, muss beispielsweise mittels Netzwerk-Ping die Verbindung zum Sender überprüft werden, um einen Verbindungsabbruch auszuschließen. Wenn zusätzlich die Schnittstelle der Anwendung weiterhin erreichbar ist, kann ein Anwendungsfehler ausgeschlossen werden und es muss ein Sensorausfall vorliegen. Um ein Sensordatenfehler zu erkennen, kann ein solches Gerät durch die im vorigen Abschnitt 4.3.3.1 beschriebenen Techniken in den übertragenen Anwendungsdaten einen Datenfehler feststellen.

4.3.4 Anwendungsfehler

Ein Anwendungsfehler kann ohne zusätzlichen Monitoring Agenten auf dem Gerät von einer zentralen Komponente nicht erkannt werden. Die Monitoring Instanz hat meist keinen Zugriff auf die Anwendungen eines Gerätes und kann diese somit nicht auf einen Ausfall oder Fehler überprüfen. Sollte es sich um eine Anwendung mit offener Schnittstelle handeln, beispielsweise um SSH, kann von der zentralen Komponente diese Schnittstelle überprüft werden. Zusätzlich können über eine solche Netzwerkschnittstelle auch andere Anwendungen überwacht werden. Eine fertige Lösung bietet die MBP, welche den Status verschiedener Anwendungen auf dem IoT-Gerät abfragen kann. Ein Problem hierbei stellt allerdings die Skalierbarkeit einer solchen Überprüfung dar, da die Monitoring-Komponente sich auf jedem Gerät einzeln einloggen müsste, um einen Fehler in der Anwendung zu erkennen.

4.3.4.1 Lokale Fehlererkennung auf dem Gerät

Auf eine ähnliche Art der Erkennung eines Hardwarefehlers im Sensor, kann auch die Anwendung, die den Sensor ausliest, überwacht werden. Diese Fehlererkennung funktioniert allerdings nur bei Anwendungen auf IoT-Geräten, welche Messdaten ermitteln und weiterleiten.
Um auch Anwendungen für die Datenverarbeitung oder Ausführung einer Aktion auf einem Aktor zu überwachen, müssen Daten dieser Programme erfasst werden. Das kann zum Beispiel die CPU-Nutzung, Speichernutzung oder momentane Netzwerkauslastung der jeweiligen Anwendung sein [ABDP13]. Durch das Auswerten solcher Messwerte, kann zum Beispiel ein Absturz oder auch die Leistungsfähigkeit des Programms festgestellt werden.

4.3.4.2 Fehlererkennung durch andere IoT-Geräte

Je nach Anwendungsfall sind auch Fehler in der Anwendung für andere Geräte erkennbar. Wenn beispielsweise Daten über die Schnittstelle einer Anwendung für andere Geräte abrufbar sind, kann dort ein Anwendungsfehler auch von diesen festgestellt werden. Sollte die Schnittstelle nicht mehr verfügbar sein oder eine Fehlermeldung zurückgeben, das Gerät selbst aber noch erreichbar ist, liegt ein Anwendungsfehler vor. Handelt es sich dagegen um eine Anwendung, die in einem bestimmten Intervall Daten an ein anderes Gerät sendet, kann das Ausbleiben dieser Daten ein Hinweis auf ein Anwendungsfehler sein. Voraussetzung dafür ist, dass das Gerät über das Netzwerk noch erreichbar ist, um einen Verbindungsabbruch auszuschließen.

4.3.5 Betriebssystemfehler

Ähnlich wie bei Gerätefehlern kann auch hier die Erkennung nur durch andere IoT-Geräte oder eine zentrale Monitoring-Komponente erfolgreich sein. Bei einem Absturz des Betriebssystems sind keine Anwendungen mehr lauffähig, also auch nicht eine Anwendung zur eigenen Überwachung. Andere Geräte erkennen einen Betriebssystemfehler an der Unerreichbarkeit des Gerätes. Eine Möglichkeit, die Erreichbarkeit eines Gerätes zu überprüfen, wurde bereits in Abschnitt 4.3.2 vorgestellt und kann hier ebenso angewandt werden. Die Unterscheidung zu einem Gerätefehler oder einem Verbindungsabbruch ist allerdings nicht möglich, sodass ein Fehler im Betriebssystem lediglich erkannt, aber nicht klassifiziert werden kann.

4.3.6 Schlechte Netzwerkverbindung

Mithilfe von *Network-based* Tests, wie bereits in Abschnitt 2.3.2 beschrieben, kann die Verbindungsqualität eines Netzwerkes überprüft werden [ABDP13]. Durch das Versenden von Testpaketen können Parameter wie Latenzzeit, Paketverluste und die Round-Trip-Zeit gemessen werden. Bei ausreichender Abweichung der Testergebnisse mit Referenzwerten aus dem Netzwerk ohne Fehler, liegt eine schlechte Netzwerkverbindung vor. Durch die Berechnung des Exponentially Weighted Moving Average (EWMA) kann zusätzlich die Situation nicht nur statisch erfasst werden, sondern auch die Veränderung über die Zeit [SDSB19].

$$p(t_n) = \alpha\kappa + (1 - \alpha)p(t_n - 1) \tag{4.5}$$

wobei κ ein durch die gemessenen Netzwerkparameter berechneter Wert ist. Mit größerem α haben neue Werte ein größeres Gewicht und alte werden weniger gewichtet. Der EWMA ist ein Verfahren zur Glättung von Zufallsschwankungen, welches neue und alte Daten berücksichtigt, wobei letztere eine abnehmende Gewichtung erhalten [Hol04]. Vorteile dieser Berechnung sind zum einen die effiziente und einfache Berechnung und die Notwendigkeit von nur wenigen Daten. Der Grenzwert für die Definition und Erkennung einer schlechten Verbindung ist abhängig vom Anwendungsfall. Das Monitoring der Netzwerkverbindung kann von jeder Komponente im Netzwerk gleichermaßen durchgeführt werden.

4.3.7 Verbindungsabbruch

Ein Verbindungsabbruch kann beispielsweise über die Abfrage der Erreichbarkeit des Gerätes mittels Netzwerk-Ping durchgeführt werden. Sofern eine Abfrage an nur eine andere Komponente durchgeführt werden kann, muss unterschieden werden, ob der Verbindungsabbruch am eigenen Gerät oder bei der anderen Komponente vorliegt. Daher muss das Gerät, zu dem die Verbindung geprüft wird, basierend auf dem Anwendungsfall korrekt ausgewählt werden. Schickt das Gerät Daten nur an ein einziges weiteres, muss es nur diese Verbindung auf Bestehen überprüfen. Sollte das IoT-Gerät allerdings auch mit dem Router zum Internet oder einer anderen zentralen Komponente, beispielsweise einer Monitoring-Komponente verbunden sein, müssen auch diese Verbindungen getestet werden, um einen Verbindungsabbruch korrekt festzustellen. Um zwischen einem Verbindungsabbruch und einem Geräte- oder Betriebssystemfehler zu unterscheiden, kann der berechnete EWMA dienen, da dieser in der Lage ist, vergangene Störungen im Netzwerk zu vergessen und neue, sobald sie einmal aufgetreten sind, zu lernen [SDSB19].

4.4 Konzepte zur Fehlerbehandlung

Für die Verbesserung der Robustheit und Ausfallsicherheit in IoT-Umgebungen wird zusätzlich zur Fehlererkennung die Fehlerbehandlung benötigt. Tabelle 4.6 gibt einen Überblick über die verschiedenen Fehlerkategorien und deren Behandlung lokal auf dem Gerät durch eine zentrale Monitoring-Komponente oder durch andere IoT-Geräte im Netzwerk.
In den folgenden Abschnitten 4.4.1 - 4.4.5 werden die Konzepte zur Fehlerbehandlung der einzelnen Fehlerkategorien genauer beschrieben.

	Fehlerbehandlung auf dem Gerät	Fehlerbehandlung durch zentrale Monitoring-Komponente	Fehlerbehandlung durch andere IoT-Geräte
Gerätefehler/ Betriebssystem-fehler	×	✓	-
Sensorausfall	-	✓	-
Sensordatenfehler	✓	×	✓
Anwendungsfehler	✓	✓	-
schlechte Netzwerk-verbindung	✓	×	×
Verbindungsabbruch	✓	×	×

✓ Fehlerbehandlung möglich - Fehlerbehandlung bedingt möglich × Fehlerbehandlung nicht möglich

Tabelle 4.6: Fehlerbehandlung durch verschiedene Komponenten einer IoT-Umgebung

4.4.1 Gerätefehler und Betriebssystemfehler

Bei einem Gerätefehler oder Absturz des Betriebssystem ist eine lokale Fehlerbehandlung auf dem Gerät nicht möglich, da das Gerät nicht lauffähig ist und somit auch keine Maßnahmen zur Lösung des Fehlers starten kann. Sollte es sich lediglich um einen Softwarefehler handeln, kann das Gerät einen automatischen Neustart einleiten. Nach einer *Kernel Panic*, also einem Betriebssystemfehler, kann zum Beispiel in Linux durch eine vom Kernel bereitgestellte Methode der Neustart automatisch erfolgen [RF09].

4.4.1.1 Fehlerbehandlung durch zentrale Monitoring-Komponente

Sollte das ausgefallene Gerät nach einem automatischen Neustart wieder erreichbar sein, kann eine zentrale Komponente mittels automatischer Softwareverteilung die bisherige auf dem Gerät laufende Anwendung erneut ausrollen und starten [FHPM18]. Diese Möglichkeit funktioniert nur, wenn das Gerät selbstständig startet und sich mit dem Netzwerk verbindet. Eine Alternative stellt

die automatische Verteilung der Software an ein redundantes Ersatzgerätes dar. Um diese Fehlerbehandlung auszuführen, müssen allerdings folgende Voraussetzungen erfüllt sein: Es muss ein Modell definiert werden, bei dem zu jeder Anwendung die Hardware-Voraussetzungen definiert sind. In der Arbeit von Franco da Silva et al. wird dies anhand CEP-Queries gezeigt. Für jedes Query-Modell werden benötigte Ressourcen, wie zum Beispiel Rechenleistung und Speicherplatz festgelegt [FHPM18].
Dieses Konzept lässt sich erweitern durch zwei Meta-Modelle einer allgemeinen IoT-Anwendung: Ein Model der Datenverarbeitung und Geschäftslogik und die Beschreibung der IoT-Umgebung mit unter anderem Art der Geräte, verfügbaren Sensoren, Netzwerkverbindungen und räumliche Positionen der Geräte [DRH20]. Letzteres spielt vor allem in IoT-Netzwerken mit vielen Sensoren eine Rolle, um redundant messende Sensoren zu identifizieren. Das Modell der Anwendung wird durch eine Menge an Operationen O verbunden durch Kanten $E = \{e = (i_i, o_j) | o_i, o_j \in O\}$, sowie eine Menge an Voraussetzungen R. Eine Funktion $req : O \rightarrow \mathcal{P}(R)$ bildet Operationen auf die jeweilige Voraussetzung ab. IoT-Netzwerke werden von Del Gaudio et. al als Tupel $E = (D, L, C, cap, w)$ definiert, wobei D die Geräte, $L = \{l = \{d_1, d_j\} | d_i, d_j \in D\}$ die Netzwerkverbindungen sind und die Funktion $cap : D \rightarrow \mathcal{P}(C)$ Geräte auf Fähigkeiten abbildet. w modelliert die Gewichtung einer Verbindung, also die Kosten einer Übertragung. Die Hardware Spezifikationen von allen IoT-Geräten und Topologien kann manuell oder mittels *network crawlers* automatisch generiert werden [FHPM18]. Binz et al. beschreiben hierfür einen Mechanismus [BBKL13]. Treffen die modellierten Voraussetzungen R der Software auf die Fähigkeiten C eines anderen IoT-Gerätes zu, lässt sich dort die Anwendung durch eine zentrale Komponente laden und starten.
In diesem Fall reicht es aus, dem Nutzer eine Benachrichtigung auszugeben, da die Funktion des ausgefallenen Gerätes zwar von einem anderen übernommen wurde, allerdings trotzdem nach dem Grund des Fehlers gesucht und gegebenenfalls das defekte Gerät ausgetauscht werden muss. Eine kritische Warnung an den Nutzer muss nur dann gesendet werden, wenn das defekte Gerät nicht mehr automatisch neu startet, also nach einer gewissen Zeit immer noch nicht erreichbar ist. Auch wenn es kein anderes Gerät in dem IoT-Netzwerk gibt, welches die gleiche Funktionalität besitzt oder die Anwendung auf keinem anderen Gerät ausführbar ist, muss eine entsprechende Warnung ausgegeben werden, dass eine sofortige manuelle Behandlung des Fehlers notwendig ist.

4.4.1.2 Fehlerbehandlung durch anderes IoT-Gerät

Die Fehlerbehandlung auf einem anderen IoT-Gerät kann entweder durch die Aufforderung eines weiteren Gerätes, beispielsweise der zentralen Monitoring-Komponente, oder durch die eigene Fehlererkennung gestartet werden. Dabei sollte dieses Gerät die gleiche oder mehr Funktionalität, wie das ausgefallene besitzen. Wie bereits im vorigen Abschnitt (4.4.1.1) erwähnt kann die Modellierung von Software und deren benötigte Voraussetzungen an die Hardware eines Gerätes, die automatische Verteilung an gleichwertige Geräte im Fehlerfall ermöglichen. Hardwareeigenschaften und Topologien werden durch *network crawlers* oder manuell spezifiziert [FHPM18]. Diese modellierten Eigenschaften bilden eine Menge C. Es ist nun möglich vor dem Start der Anwendung oder zur Laufzeit in der IoT-Umgebung für jedes Gerät die Mengenbeziehung zu den Spezifikationen der anderen Geräte herzustellen. Wenn $C_j \subseteq C_i$ kann Gerät i für das Gerät j im Falle eines Ausfalls einspringen und die Anwendung ist somit auch auf Gerät i lauffähig. Mindestens alle Fähigkeiten C_j besitzt Gerät i ebenfalls und kann so alle Aufgaben übernehmen. Die Ausführung dieser Anwendung kann bereits vorbereitet werden und wird erst laufen, sobald Gerät j ausfällt und Gerät i dafür einspringt.

Daraufhin erfolgt eine Nachricht an alle anderen IoT-Geräte, um redundante Ausführungen von weiteren IoT-Geräten zu vermeiden. Zusätzlich wird eine Warnung an den Nutzer ausgegeben, um das defekte Gerät auszutauschen beziehungsweise wieder funktionstüchtig zu machen.

4.4.2 Sensorausfall und Sensordatenfehler

Ein Sensorausfall kann nicht von dem gleichen IoT-Gerät behoben werden. Es ist lediglich möglich eine Nachricht an eine zentrale Komponente oder an andere Geräte zu schicken, um über den Ausfall zu informieren. Die Behandlung wird dann durch die zentrale Komponente (Abschnitt 4.4.2.2) oder durch andere IoT-Geräte ausgeführt (Abschnitt 4.4.2.3). Sensordatenfehler können nicht von der zentralen Komponente behandelt werden, sondern nur lokal und von Geräten, welche die fehlerhaften Anwendungsdaten verarbeiten.

4.4.2.1 Lokale Fehlerbehandlung durch Gerät

Sensordatenfehler können durch die Filterung der fehlerhaften Werte behandelt werden. Hier ist allerdings wichtig, wie schon in Abschnitt 4.3.3.1 beschrieben, die Anforderungsspezifikation der Anwendung zu kennen und keine Daten zu filtern, die keine Fehler waren (*False Positives*), oder Fehler aufgrund einer zu grobgranularen Filterung nicht zu erkennen (*False Negatives*). Erkannte *Shorts* werden gefiltert und durch „leere“ Werte ersetzt. Das Auslassen der Nachricht ist nicht möglich, da sonst die zeitlichen Abstände der gesendeten Sensorwerte gestört werden und eine falsche Fehlererkennung des Sensorausfalls in der Monitoring-Komponente ausgelöst werden kann. Dabei müssen die spezifizierten Anforderungen (Tabelle 4.1) der Anwendung berücksichtigt werden, um die Aggressivität der Filterung anzupassen. *Spikes* können meist erst nach einer gewissen Menge an fehlerhaften Datenwerten erkannt und somit auch erst dann behandelt werden [CNC+18a]. Eine Behandlung in Echtzeit ist somit also nur schwer möglich. Ebenso können *Stuck-at* Fehler erst zu einem späteren Zeitpunkt behandelt werden. Eine unmittelbare Reaktion und Behandlung ist nicht gewährleistet. *Noise* kann lokal auf dem IoT-Gerät, beispielsweise mit dem gleitenden Mittelwert (*Simple Moving Average*) geglättet werden. Redhyka et al. beschreiben in ihrer Arbeit die Filterung und Glättung von Daten eines Lage- und Beschleunigungssensors, um damit eine Plattform zu stabilisieren [RSS15].

$$SMA = \frac{x_k + x_{k-1} + \ldots + x_{k-(n-1))}}{n} \tag{4.6}$$

$$SMA_{jetzt} = SMA_{vorher} + \frac{x_{k-n}}{n} - \frac{x_k}{n} \tag{4.7}$$

Dabei gibt n in der Formel 4.6 und 4.7 zur Berechnung des Simple Moving Average (SMA) die Ordnung an. Durch die Erhöhung von n wird der Durchschnitt aus mehreren Datenpunkten berechnet und der Verlauf damit stärker geglättet. In der vorliegenden Arbeit wird davon ausgegangen, dass *Calibration* Fehler manuell vor der Laufzeit überprüft und entsprechend behandelt werden.

4.4.2.2 Fehlerbehandlung durch zentrale Monitoring-Komponente

Sollte ein Sensorausfall auf einem Gerät bekannt sein, kann die zentrale Monitoring-Komponente, wenn möglich, die Aufgabe der Datenerfassung auf ein anderes gleichwertiges Gerät übertragen und die Anwendung dort durch eine automatische Softwareverteilung, starten [FHPM18]. Dies ist nur möglich, wenn andere Geräte gleichwertige Sensoren besitzen, um dieselben Daten zu messen. Wie bereits in Abschnitt 4.4.1.1 beschrieben, können *network crawler* [BBKL13] eingesetzt werden, um Netzwerktopologie und Hardwareeigenschaften der Geräte in einem Netzwerk zu ermitteln und zentral zu speichern. Somit ist der zentralen Komponente bekannt welche Sensoren an welchem Gerät verfügbar sind. Zusätzlich muss die räumliche Position der Geräte bekannt sein, um Messwerte nur von Geräten an der gleichen Stelle im Raum zu erfassen. Vor der Ausführung der Anwendung kann ein Modell erstellt werden, welches die räumliche Redundanz der Geräte darstellt und welche Aufgabe im Falle eines Sensorfehlers auf welches andere Gerät übertragen werden kann.
Im Falle des Übertragens der Funktion auf ein anderes Gerät muss der Nutzer benachrichtigt werden, um den defekten Sensor auszutauschen. Sollte es kein passendes Gerät, das alle Voraussetzungen erfüllt, in der räumlichen Nähe geben, muss eine Warnung ausgegeben werden, da die bisherige Funktionalität nun nicht mehr gegeben ist.

4.4.2.3 Fehlerbehandlung durch anderes IoT-Gerät

Ähnlich wie in Abschnitt 4.4.1.1 beschrieben, kann die Spezifikation von Sensoren als Menge S anderen Geräten mitgeteilt werden. Ein Gerät i kann nun abgleichen, ob $S_j \subseteq S_i$ für jedes andere Gerät j erfüllt ist. Beim Hinzufügen des Gerätes in das IoT-Netzwerk muss außerdem eine genaue räumliche Position, sofern das für die Anwendung eine Rolle spielt, im Gerät vermerkt werden. Durch diese beiden Eigenschaften kann ein Gerät i im Falle des Sensorausfalls die Funktion des anderen Gerätes übernehmen. Eine Warnung an den Nutzer muss zwingend ausgegeben werden, wenn kein Gerät den Sensorausfall kompensieren kann.
Wenn Daten, die Datenfehler (siehe Abschnitt 4.2.2) enthalten, von anderen Geräten empfangen werden, können die gleichen, wie in Abschnitt 4.4.2.1 beschriebenen Mechanismen zur Datenfilterung angewandt werden. Ebenso müssen die Anforderungen von den jeweiligen Anwendungen beachtet werden, beispielsweise ob geringe Abweichungen vom tatsächlichen Wert toleriert sind oder nicht.

4.4.3 Anwendungsfehler

Anwendungsfehler können lokal, zentral und dezentral behandelt werden. Im Folgenden werden für jede Methode entsprechende Konzepte vorgestellt.

4.4.3.1 Lokale Fehlerbehandlung durch Gerät

Durch eine lokale Monitoring-Anwendung können Abstürze von laufenden Anwendungen erkannt werden [ABDP13]. Nach der Erkennung eines Absturzes kann sofort ein Neustart der Anwendung veranlasst werden, um die Funktionalität schnellstmöglich wieder zu gewährleisten. Es reicht

lediglich einen Log-Eintrag bei einem automatisieren Neustart zu verfassen. Wenn der Start der Anwendung aufgrund von fehlerhaften oder manipulierten Daten im Programmcode nicht mehr möglich ist, sollte die Fehlerbehandlung von einer zentralen Komponente erfolgen.

4.4.3.2 Fehlerbehandlung durch zentrale Monitoring-Komponente

Beim Auftreten eines Anwendungsfehlers wird zunächst die Anwendung neu aufgesetzt und dann gestartet. Hierfür eignet sich ein, wie in Abschnitt 4.4.1.1 schon erwähntes, Netzwerkwerkzeug zur automatischen Verteilung von Software [FHPM18]. Dieses lädt die zur Ausführung benötigten Quellcode-Dateien erneut auf das Gerät und startet die Anwendung über die Netzwerkschnittstelle des Gerätes.

4.4.3.3 Fehlerbehandlung durch anderes IoT-Gerät

Die Daten, die an die ausgefallene Schnittstelle eines anderen Gerätes gesendet werden sollen, können solange zwischengespeichert werden, bis die Anwendung wieder verfügbar ist. Sobald der Ausfall erkannt wurde, werden alle weiteren empfangenen oder aufgezeichneten Daten in eine lokale Datenbank gespeichert. Eine Benachrichtigung an den Nutzer wird erstellt, um über den Ausfall der Schnittstelle des anderen Gerätes und über die begonnene Aufzeichnung der Daten zu informieren. Sobald die Anwendungsschnittstelle wieder erreichbar ist und Daten erneut übertragen werden können, sendet das Gerät alle zwischengespeicherten Daten mit entsprechender Zeitmarkierung an das andere Gerät. Bei Mangel an Speicherressourcen werden ältere Daten gelöscht und es wird eine Warnung über den Verlust von Daten ausgegeben.

4.4.4 Schlechte Netzwerkverbindung

Die Behandlung von einer schlechten Netzwerkverbindung ist nur lokal auf dem Gerät, welches Daten übermitteln soll, möglich. Eine zentrale Behandlung kann dabei lediglich eine Warnung über den schlechten Zustand der Netzwerkverbindung zu einem Geräte an den Nutzer umfassen. Sollten Echtzeitanwendungen auf eine nicht verzögerte Nachrichtenübertragung angewiesen sein und keine alten Nachrichten mehr akzeptieren, ist keine Behandlung notwendig, da zu spät ankommende Daten nicht mehr relevant sind und somit nicht erneut übertragen werden müssen. Wenn Daten allerdings trotzdem vollständig und zuverlässig ankommen sollen, ist es wichtig, dass die gesamten Daten aufgezeichnet werden und durch die Verwendung von QoSs auf jeden Fall bei den anderen Geräten ankommen (*Exactly once delivery*). Hier können, sobald eine schlechte Netzwerkverbindung erkannt wurde, die aufgenommenen oder verarbeiteten Daten lokal auf dem Gerät zunächst zwischengespeichert werden. Erst, wenn eine Bestätigung für die erfolgreiche Übertragung empfangen wurde, können die Daten gelöscht werden. Sollte diese Bestätigung ausbleiben, müssen die Daten erneut übertragen werden. Um die ursprüngliche Reihenfolge von den Daten wiederherzustellen, sollten inkrementierende Sequenznummern an jedes Datenpaket angefügt werden. Dieses Verfahren lässt sich mit bereits in Übertagungsprotokollen implementierten Techniken leicht umsetzten. Wenn die Anwendung keine Anforderung bezüglich der Kontinuität der eintreffenden Daten stellt, beziehungsweise wenn auch größere Datenlücken kein Problem darstellen, muss die oben beschriebene Fehlerbehandlung nicht durchgeführt werden, um Speicherplatz auf dem Gerät zu sparen.

4.4.5 Verbindungsabbruch

Ähnlich wie bei der schlechten Netzwerkverbindung ist die Behandlung von Verbindungsabbrüchen nur auf Geräten, die Daten übermitteln, möglich. Andere Geräte im Netzwerk, sowie eine zentrale Monitoring-Komponente kann lediglich den Nutzer über einen Verbindungsabbruch warnen. Sollte die Verbindung abbrechen, werden zunächst alle weiteren Daten lokal in eine schlanke Datenbank gespeichert. Dies wird solange gemacht, bis eine stabile Netzwerkverbindung wieder aufgebaut ist. Die gespeicherten Daten werden dann zusammengefasst an das andere Gerät übertragen. Um die Daten richtig zu verarbeiten, müssen die Empfänger in der Lage sein, die Datenpunkte zu extrahieren. Durch das Hinzufügen eines Zeitstempels oder eine Sequenznummer an die Daten, wird die Einhaltung der Reihenfolge und die korrekte zeitliche Einordnung der Daten sichergestellt. Wenn Anforderungsszenarien Datenverluste für einen bestimmten Zeitraum erlauben, müssen solange keine Daten abgespeichert werden, um den benötigten Speicherplatz und die Größe der anschließenden Übertragung zu verringern.

4.5 Architektur

In Abbildung 4.5 wird eine generische Architektur dargestellt, um Fehler aus den vorigen beschriebenen Kategorien zu erkennen und wenn möglich zu vermeiden.

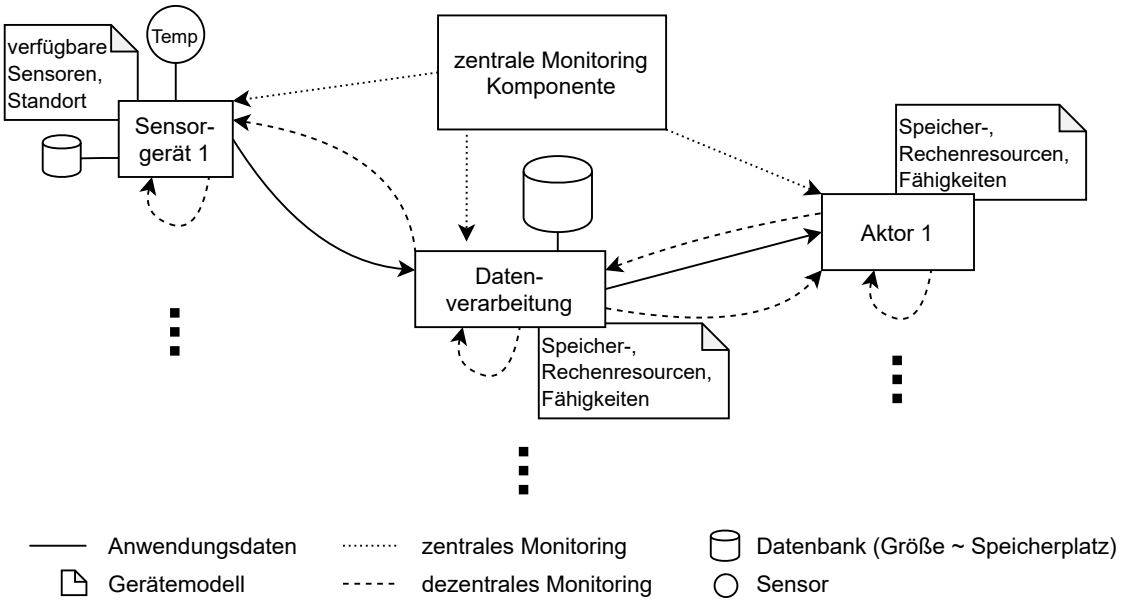

Abbildung 4.5: Architektur einer IoT-Umgebung

Diese IoT-Architektur stellt ein Beispiel dar, in welchem beliebig viele Sensoren Daten an andere Geräte zur Datenverarbeitung schicken. Diese werten die Daten aus und senden gegebenenfalls Nachrichten an eine beliebige Anzahl von unterschiedlichen Aktoren. Um hier die Robustheit und Ausfallsicherheit zu steigern, werden folgende Maßnahmen angewandt:

1. zentrale Monitoring-Komponente:
   Um Gerätefehler (Abschnitt 4.3.2), sowie Betriebssystemfehler (Abschnitt 4.3.5) und Netzwerkabbrüche (Abschnitt 4.3.6 und 4.3.7) auf den IoT-Geräten zu erkennen, werden Netzwerk- und Geräteparameter zentral erfasst und auf einen Ausfall überprüft. Dies umfasst die Auswertung der empfangenen *Heartbeats* von den Geräten und des Zustands des Netzwerkes.
   Eine weitere Funktion kann die automatische Verteilung von Software an Ersatzgeräte umfassen (Abschnitt 4.4.1.1). Hierfür kann beispielsweise die Multi-purpose Binding and Provisioning Platform eingesetzt werden.

2. dezentrales Monitoring der Sensoren:
   Die datenverarbeitende Komponente überwacht die verbundenen Sensoren und kann so mittels Auswertung der Messwerte Sensorausfälle und Datenfehler erkennen (Abschnitt 4.3.3).

3. dezentrales Monitoring des Aktors:
   Ein Absturz der IoT-Anwendung auf dem Aktor oder dessen Unerreichbarkeit kann bereits dezentral auf IoT-Geräten festgestellt werden.

4. lokales Monitoring:
   Um Anwendungsfehler zu bemerken, überwacht jedes Gerät, sofern möglich, die eigenen laufenden Anwendungen (Abschnitt 4.3.4.1).

5. Modellierung von Hardwareeigenschaften und Fähigkeiten:
   Ein Gerät kann die Aufgabe eines anderen IoT-Gerätes nur übernehmen, wenn es die gleichen oder höhere Hardwarespezifikationen und Fähigkeiten des Ausgefallenen besitzt (Abschnitt 4.4.1.2). Gleiches gilt für Geräte mit Sensoren. Die Durchführung von den Messungen kann nur übernommen werden, wenn Sensortyp und Standort der Geräte übereinstimmt.

6. kleine Datenbank an Sensorgeräten:
   Um den Datenverlust bei einem Netzwerkausfall oder einer schlechten Verbindung zwischen Sensor und Datenverarbeitung zu verhindern, benötigt das Sensorgerät eine kleine Datenbank (Abschnitt 4.4.5). Daten können so zwischengespeichert und nach dem erneuten Netzwerkaufbau gesendet werden.

5 Implementierung

Um die in Kapitel 4 vorgestellten Konzepte zur Erkennung und Behandlung und die daraus resultierende Verbesserung von Robustheit und Ausfallsicherheit in IoT-Umgebungen als Gesamtkonzept zu evaluieren, wurde ein Prototyp implementiert. Aufgrund des begrenzten Umfangs dieser Arbeit wurde nur ein Teil des Motivations-Szenarios (Abbildung 1.2) und der vorgestellten Methoden zur Fehlererkennung und -behandlung umgesetzt.
In Abbildung 5.1 werden die drei verschiedene Komponenten des Prototyps dargestellt: (a) Datenverarbeitungskomponente, (b) Monitoring Komponente und (c) zwei Temperatursensoren. Die Erkennung von Geräte-, Betriebssystem- und Anwendungsfehlern, sowie Sensorausfällen findet auf der zentralen Monitoring Komponente mithilfe der MBP statt. Lediglich für die Überwachung von Gerätefehlern wurde zusätzlich das dezentrale Monitoring (Abschnitt 4.3.1.3) implementiert. Die Skripte lassen sich mit geringem Programmieraufwand auch auf die dezentrale Erkennung anderer Fehlerszenarien, wie zum Beispiel Sensorausfälle anwenden. Für die Erkennung von Anwendungsfehlern wurde nur das Konzept des zentralen Monitorings mittels MBP (Abschnitt 4.3.4) implementiert. Die MBP wird außerdem verwendet, um bei Geräte-, Betriebssystemfehlern oder Sensorausfällen Anwendungen auf ein anderes Gerät zu laden und dort zu starten. Die Datenverarbeitungskomponente überprüft empfangene Messwerte auf Sensordatenfehler und filtert *Spikes* und *Shorts*. Die vorgestellten Methoden, um *Calibration* oder *Noise* Datenfehler zu vermeiden (Abschnitt 4.4.2.1), wurden nicht implementiert. Ein Verbindungsabbruch wird lokal auf dem Gerät erkannt und dort auch durch die in Abschnitt 4.4.5 vorgestellte Zwischenspeicherung von Messdaten behandelt. Schlechte Netzwerkverbindungen werden in dem Prototyp nicht erkannt, da sich die Konzepte hierfür nur leicht von der Erkennung und Behandlung von Verbindungsabbrüchen unterscheiden.

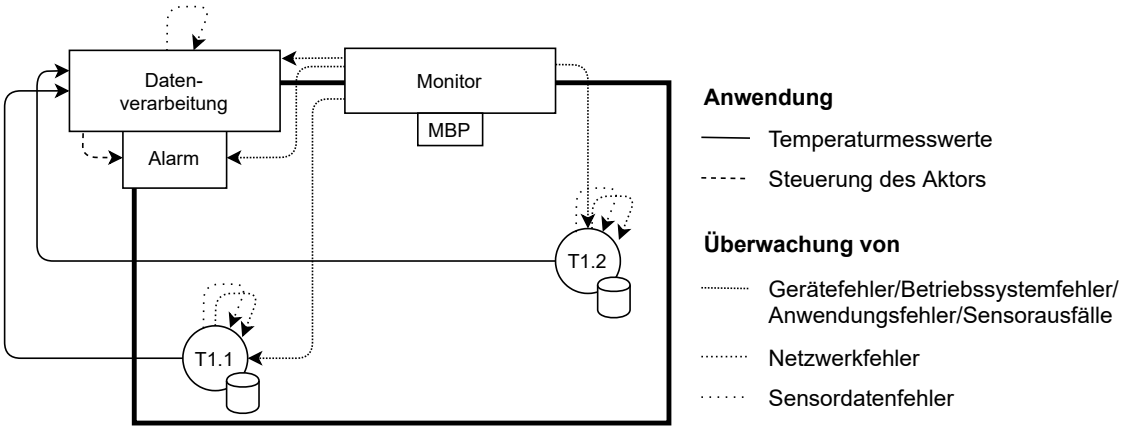

Abbildung 5.1: Prototyp zur Evaluierung des Gesamtkonzepts

In den folgenden Abschnitten wird die Implementierung vorgestellt und näher erläutert.

5.1 Verwendete Technologien

Für die Implementierung wurde die Programmiersprache Python[1] verwendet. Diese Script-Sprache bietet sich für die Programmierung von schlanken Anwendungen auf IoT-Geräten sehr gut an und wird von vielen verwendeten Schnittstellen und Technologien unterstützt. Im Folgenden werden die für die Implementierung verwendeten Anwendungen vorgestellt.

5.1.1 Mosquitto

Für die Kommunikation der IoT-Geräte wurde in dieser Arbeit das Nachrichtenprotokoll MQTT verwendet. Dieses Publish-/Subscribe System benötigt einen Broker, der als Server zur Verteilung von Nachrichten dient. Mosquitto ist ein bekannter *open-source* Broker der Eclipse Foundation [Lig17]. MQTT-Protokolle der Versionen 5.0, 3.1.1 und 3.1 werden unterstützt. Da die Implementierung des Konzeptes auf Geräten mit geringer Rechenleistung lauffähig sein soll, bietet sich Mosquitto und MQTT zur schlanken Kommunikation in IoT-Netzwerken an.

5.1.2 Zope Object Database (ZODB)

ZODB[2] ist eine Datenbank für die Programmiersprache Python. Objekte können ohne zusätzlichen Code in die Datenbank abgelegt und abgerufen werden. Die grundlegende Struktur ist ähnlich aufgebaut wie ein Python Dictionary, welches die Wurzel, auch *Root* genannt, bildet. Alle Objekte, die standardmäßig serialisierbar sind, wie zum Beispiel Listen oder Dictionaries, werden mit `transaction.commit()` persistent abgespeichert. Datei-Objekte oder Sockets können nicht von der Datenbank gespeichert werden.

5.2 Szenario

Das Motivations-Szenario aus Abschnitt 1.1 wurde auf Raspberry Pis, zum Teil ausgestattet mit Temperatursensoren, implementiert. Um den Rahmen dieser Arbeit einzuhalten, werden lediglich zwei Temperatursensoren, T1.1 und T1.2, sowie die zentrale Datenverarbeitung mit der Alarm-Komponente umgesetzt.
Die Skripte rufen zu Beginn eine in JSON formatierte Konfigurations-Datei (Listing 5.1) auf, um verschiedene Parameter, wie die Adresse des MQTT-Servers oder des MBP-Hosts einzulesen. Jedes Gerät erhält eine eindeutige ID, welche für die Anmeldung am MQTT-Broker und die Identifizierung des Gerätes benötigt wird. Außerdem kann durch diese Konfiguration auch das Ausgabeziel des Logs angegeben werden.

[1] https://www.python.org/about/
[2] http://www.zodb.org/en/latest/articles/ZODB1.html

```
{
  "device_id": "example_device",
  "host": "localhost",
  "mbp-host": "192.168.221.194",
  "logger": "console"
}
```

Listing 5.1: Konfigurationsdatei eines IoT-Gerätes

Temperatursensor
Um die Temperatur zu messen, wurde die Vorlage des Temperatur-Adapters aus dem MBP-Repository[3] verwendet. Die Temperatur-Sensoren senden per MQTT die gemessene Temperatur an einen Mosquitto-Broker auf einer VM. Zudem werden die Messdaten für einen gewissen Zeitabschnitt in eine ZODB zwischengespeichert.

Zentrale Datenverarbeitung
Die Datenverarbeitung findet auf einer virtuellen Maschine statt, die die empfangenen Temperaturen an ein Raspberry Pi mit 7-Segment Anzeige weiterleitet, um diese dort anzuzeigen. Bei zu hohen Temperaturen wird ein Alarm per MQTT an den Aktor gesendet.

Alarm
Der Aktor basiert auf dem Skript `buzzer_stub.py`, aus dem MBP-Repository. Bei einer Alarm-Nachricht von der zentralen Datenverarbeitung werden mehrere kurze Töne abgegeben.

5.3 Monitoring

Es werden zwei verschiedene Monitoring-Verfahren zur Überwachung von Gerätefehlern und Netzwerkabbrüchen verwendet. Zum einen die Überwachung mittels einer zentralen Komponente (Abschnitt 4.3.1.2), die jedes Gerät im IoT-Netzwerk überwacht, zum anderen die dezentrale Überwachungsmethode als Ring, wie in Abschnitt 4.3.1.3 beschrieben. Für das Monitoring von Geräte- und Betriebssystemfehler, sowie Sensorausfälle wird auf den Geräten ein Monitoring Agent ausgeführt.

5.3.1 Gerätefehler, Betriebssystemfehler

Die Geräte senden eine *Heartbeat*-Nachricht alle fünf Sekunden, um Gerätefehler und Betriebssystemfehler zu erkennen. Im Falle des zentralen Monitorings werden diese *Heartbeats* an genau eine Komponente geschickt. Beim Ring-Monitoring erhält immer der jeweilige Vorgänger-Knoten die Nachricht. Mittels dem vorgestellten Konzept von Seeger et al. können aus den Intervallen dieser Nachrichten eine Wahrscheinlichkeit eines Ausfalls berechnet werden (Listing 5.2). Diese Formel wird alle 200 Millisekunden neu berechnet und verwendet die Varianz und den Durchschnittswert der Zeitabstände zwischen den Intervallen als Parameter. Zusätzlich wird der aktuelle Zeitpunkt der Berechnung, sowie der Zeitstempel des letzten empfangenen *Heartbeats*, benötigt.

[3] https://github.com/IPVS-AS/MBP/tree/master/resources/operators/extraction

```
def calculateSuspiciousFunction(self):
  self.s = - math.log(self.variance /
        (self.variance + ((time.time() - self.lastRecieved) - self.mean)**2))
  return self.s
```

Listing 5.2: Funktion zur Berechnung der Ausfallswahrscheinlichkeit von Seeger et al [SDSB19]

Sofern noch ein *Heartbeat* empfangen wird, gibt der Wert der Suspicious-Funktion die Wahrscheinlichkeit an, dass das Gerät noch korrekt läuft. Eine Approximation von verbleibenden Ressourcen, wie Seeger et al. mit der Gleichung 4.4 beschrieben haben, wird nicht übertragen. Steigt die Zeit zwischen den *Heartbeats* des Gerätes stark an oder ist sie sehr unregelmäßig, erhöht sich der Funktionswert, der ein Maß für die Ausfallwahrscheinlichkeit ist. Wächst der Wert auf über 0,9 ist das Gerät oder der *Heartbeat*-Agent mit einer Wahrscheinlichkeit von mindestens 90% ausgefallen.
Im Falle des dezentralen Monitorings wird diese Berechnung von dem jeweiligen Vorgänger-Knotenpunkt übernommen. Jedes Gerät führt dabei die Abfrage unabhängig und nicht zwingend zum gleichen Zeitpunkt aus.

5.3.2 Sensorausfälle

Durch die gleiche Methode wie im obigen Abschnitt 5.3.1 beschrieben, können die Sensoren auf einen Ausfall überwacht werden. Die Monitoring-Agenten übermitteln bei einem neu gemessen Datenwert ein Sensor-*Heartbeat* an die zentrale Monitoring Komponente. Die Intervalle zwischen den empfangenen Sensorwerten lassen ebenfalls durch die Berechnung der *Suspicious-Function* von Seeger et al. Rückschlüsse auf die Wahrscheinlichkeit eines Defektes des Sensors zu [SDSB19].

5.3.3 Sensordatenfehler

Die Erkennung von Sensordatenfehlern erfolgt in dieser Implementierung durch die manuelle Klassifizierung von Messwerten als Fehler aufgrund der Spezifikation der Anwendung. Da sich alle Temperatursensoren im Înneren eines bewohnten Gebäudes befinden, können alle Werte unter 15°C pauschal als *Spike* oder *Short* Fehler bezeichnet werden. Ein Ausreißer in die oberen Temperaturbereiche kann aufgrund der niedrigen Messrate nicht als Fehler klassifiziert werden. Dadurch besteht die Gefahr einen echten Wert fälschlicherweise als Fehler zu bezeichnen, was im Falle einer Brandmeldeanlage ein schwerwiegendes Problem darstellen würde. Dadurch, dass die Temperatur im Normalfall nur gering in der Abhängigkeit von der Außentemperatur schwankt, werden *Stuck-at* Fehler erst nach mehreren gemessenen Datenpunkten erkannt. Für den Prototypen werden Messreihen ab exakt zehn gleichen Messpunkten als ein solcher Stuck-at Fehler definiert. *Noise* und *Calibration* Fehler werden, wie bereits erläutert, aufgrund der Vereinfachung dieses Prototyps vernachlässigt und nicht beachtet.

5.3.4 Anwendungsfehler

Die MBP bietet eine REST Schnittstelle[4] über die der Status von, durch die Plattform selbst ausgerollten und gestarteten Anwendungen überwacht werden kann. Diese Abfrage erfolgt alle fünf Sekunden durch die zentrale Monitoring-Komponente. Sollte eine Anwendung nicht mehr reagieren oder abgestürzt sein, wird bis zu fünfmal automatisch ein Neustart der Anwendung durch MBP ausgeführt (Listing 5.3).

```
responseState = requests.get(
  'http://' + mbp_host + ':8888/deploy/master/api/sensors/state/' + client.mbp_sensor_id)
```

Listing 5.3: Abfrage und Start der Anwendung via MBP

5.3.5 Verbindungsabbrüche

Verbindungsabbrüche und schlechte Netzwerkverbindungen werden in dem implementierten Prototyp zusammengefasst und nicht unterschieden. Es gibt viele bereits vorhandene Techniken, um eine fehlende Verbindung zu einem anderen Gerät festzustellen. In dieser Implementierung wurde der *Ping*, als einfacher Test auf Verfügbarkeit eines anderen Gerätes verwendet. Sendet dieses keine Antwort zurück, wird ein Verbindungsabbruch erkannt. Wenn die IP-Adresse des Gerätes nicht bekannt ist, wird der *Ping* und die Antwort über das Kommunikationsprotokoll MQTT versendet. Das Ausführen des *Pings* erfolgt durch das Gerät, welches Daten sendet, zu dem, welches die Daten empfangen soll. Ist dieses nicht bekannt, wird die Verbindung zu einer zentrale Komponente, wie beispielsweise dem MQTT-Broker, geprüft werden.

5.4 Fehlerbehandlung

Der implementierte Prototyp beinhaltet die Behandlung von Geräteausfällen durch eine zentrale Komponente (Abschnitt 4.4.1.1), sowie den automatischen Neustart von Anwendungen durch die MBP (Abschnitt 4.4.3.2). Außerdem wurde die Zwischenspeicherung von Daten während eines Verbindungsabbruches (Abschnitt 4.4.5) implementiert.

5.4.1 Gerätemodell

Um Ausfälle von Geräten und Sensoren behandeln zu können, wurde zunächst gemäß der Definition in Abschnitt 4.4.1.1 und 4.4.1.2 für jedes Gerät ein Modell definiert. Eine JSON-Datei beschreibt dabei alle Funktionalitäten des Gerätes, wie beispielsweise die Art der Sensoren, den Ort, die Art der Stromversorgung usw. (Listing 5.4).

[4] https://github.com/IPVS-AS/MBP/wiki/API-Reference

```
"capabilities": {
  "sensors": ["temperature"],
  "power": true,
  "memory": 4,
  "location": "room1"
}
```

Listing 5.4: Inhalt der initialen Nachricht beim Verbindungsaufbau

5.4.2 Kompatibilitätsprüfung

Jedes Gerät versendet beim Eintritt in das Netzwerk das eigene Gerätemodell (Abschnitt 5.4.1). Alle bereits vorhandene Geräte können anschließend einen Abgleich mit den eigenen modellierten Fähigkeiten machen. Durch die Kompatibilitätsprüfung ist jedem Gerät bekannt, ob es im Falle des Ausfalls eines anderen Gerätes die Funktionalitäten übernehmen kann. Im implementierten Prototyp wurde aufgrund des deutlich erhöhten Aufwands diese Kompatibilitätsprüfung lediglich von der zentralen Komponente durchgeführt. Mithilfe der Funktion `isCompatible` in Listing 5.5 wird eine Tabelle erstellt, in der jedem verbundenen Gerät die jeweils möglichen Fallback-Geräte zugeordnet werden.

```
def isCompatible(model1, model2):
  for sensor in model1.sensors:
    if sensor not in model2.sensors:
      return False
  if model1.memory > model2.memory:
    return False
  if model1.power and not model2.power:
    return False
  if model1.location != model2.location:
    return False
  return True
```

Listing 5.5: Kompatibilitätsprüfung mit einem anderen IoT-Gerät

5.4.3 Neustart der Anwendungen

Im Falle eines Anwendungsausfalls wird die Anwendung mit Hilfe der MBP REST-Schnittstelle neu gestartet. Um bei korrupten oder gelöschten Anwendungsdaten immer noch eine einwandfreie Ausführung zu gewährleisten, wird die Anwendung zuerst erneut auf das Gerät geladen und anschließend gestartet. Der entsprechende Code-Abschnitt mit dem HTTP-Request an die MBP wird in Listing 5.6 abgebildet. Implementiert wurde hier das in Abschnitt 4.4.3.2 vorgestellte Konzept.

```
responseDeploy = requests.post(
  'http://' + mbp_host + ':8888/deploy/master/api/deploy/sensor/' + client.mbp_sensor_id)
responseStart = requests.post(
  'http://' + mbp_host + ':8888/deploy/master/api/start/sensor/' + client.mbp_sensor_id)
```

Listing 5.6: Start einer Anwendung mit der REST-API von MBP

5.4.4 Übertragung von Funktionen beim Ausfall

Sensor- und Geräteausfälle benötigen meist zur Aufrechterhaltung der Funktionalität einen Gerätewechsel. Das bedeutet, dass die Funktionalität eines defekten Gerätes auf ein anderes übertragen wird (Abschnitt 4.4.1.1 und 4.4.2.2). Durch den Kompatibilitätscheck in Listing 5.5 ist der zentralen Komponente bereits bekannt, in welchem Fall eine solche Übertragung der Anwendung möglich ist. MBP bietet die Möglichkeit Geräte und Adapter über die REST-Schnittstele neu miteinander zu verknüpfen. In Listing 5.7 wird das Gerät (ID: `device_id`) mit dem Adapter (ID: `adapter_id`) verknüpft. Anschließend kann per REST-API die Verteilung und der Start der Anwendung, wie in vorherigen Abschnitt 5.4.3 beschrieben, durchgeführt werden.

```
body = {
  "name": "Temperature Sensor",
  "adapter": "mbp_host + ':8888/deploy/master/api/adapters/adapter_id",
  "device": "mbp_host + ':8888/deploy/master/api/devices/device_id",
}
responseDevice = requests.post(
  'http://' + mbp_host + ':8888/deploy/master/api/sensors', body)
```

Listing 5.7: Registrieren einer neuen Verknüpfung zwischen Gerät und Adapter

5.4.5 Zwischenspeicherung von Sensordaten

Bei schlechten Netzwerkverbindungen oder Verbindungsabbrüchen zwischen Geräten und deren Empfängern können weitere Fehler durch das Zwischenspeichern von ankommenden oder erzeugten Daten verhindert werden (Abschnitt 4.4.4 und 4.4.5). Für den implementierten Prototyp werden die vom Sensor gemessenen Daten in einer Zope Object Datenbank (ZODB) zwischengespeichert. Um die Reihenfolge beizubehalten, wird ein Zeitstempel an jeden Datenpunkt angefügt, bevor dieser abgespeichert wird. Sobald die Netzwerkverbindung zum Empfänger wieder aufgebaut wird, werden die Daten, als Block gekennzeichnet, versendet. Datenpakete, in denen die Messwerte als Block zusammengefasst sind, erfordern eine besondere Verarbeitung beim Empfänger. Die Datenpakete müssen in die einzelnen Datenpunkte aufgeteilt und erst danach durch die Anwendung weiterverarbeitet werden.

5.4.6 Evaluation der Implementierung

Der Prototyp läuft erwartungsgemäß auf Raspberry Pis und erfüllt die Aufgabe, Robustheit und Ausfallsicherheit in einem IoT-Netzwerk zu verbessern. Tests mit manuell provozierten Fehlern, wie zum Beispiel das Trennen der Netzwerkverbindung oder der Stromversorgung, rufen das gewünschte Verhalten hervor. Bei Netzwerkabbrüchen werden gemessene Daten zwischengespeichert und bei Wiederaufnahme der Verbindung als Block erneut versendet. Geräteausfälle und Sensorausfälle, simuliert durch das Abstecken der Sensoren, werden zuverlässig erkannt und durch die Verlagerung der Aufgabe auf ein anderes Gerät im IoT-Netzwerk behandelt, indem die Anwendung durch die MBP auf das andere IoT-Gerät geladen und gestartet wird, sodass das gesamte Anwendungsszenario wieder problemlos ausgeführt werden kann.

Wenn der Prototyp für andere IoT-Anwendungen verwendet werden soll, müssen hauptsächlich die Konfigurations-Dateien verändert werden. Statische Attribute, wie zum Beispiel die IP Adresse des MQTT-Hosts und des MBP-Hosts müssen angepasst werden. Außerdem muss jedes IoT-Gerät mit einer eindeutigen `device_id` und den entsprechenden *Capabilities* modelliert werden (Listing 5.4). Um die Fehlererkennung und -behandlung mit der MBP durchzuführen, muss die `mbp_device_id` und die `adapter_id` des auf dem Gerät ausgeführten Operators angegeben werden. Sofern Skripte für Sensoren verwendet werden sollen, müssen diese, um die gemessenen Daten in einer lokalen Datenbank (ZODB) zu speichern, erweitert werden. Für die Sensordatenfehlerbehandlung ist es erforderlich, anwendungsspezifische Grenzwerte und Filter auszuwählen und zu programmieren.

6 Evaluation

Im folgenden Abschnitt werden die verschiedenen in Kapitel 4 vorgestellten Methoden zur Fehlererkennung und -behandlung auf Vor- und Nachteile geprüft. Verschiedene Grenzen des Konzepts werden aufgezeigt sowie die mögliche Anwendung bei verschiedenen IoT-Anwendungen diskutiert.

6.1 Evaluation der Fehlerbehandlung

Die korrekte Fehlerdetektion gilt als Voraussetzung für eine gute Fehlerbehandlung und wird deshalb zu Beginn dieses Abschnittes bewertet. Geräteausfälle sind mit obigem Konzept (Abschnitt 4.3.2) gut detektierbar. In kleineren IoT-Netzwerken, wie beispielsweise das Motivations-Szenario in Abschnitt 1.1, ist dies durch eine zentrale Monitoring-Komponente (Abschnitt 4.3.1.2) möglich. Sollte die Netzwerkgröße steigen, macht eine dezentrale Geräteüberwachung mehr Sinn (Abschnitt 4.3.1.3). Diese skaliert durch die geringere Belastung des Monitoring-Overheads im Netzwerk deutlich besser, da die Anzahl an zu überwachenden Geräten weiterhin pro Monitoring-Komponente konstant bleibt.
Bei Benutzung einer zentralen Monitoring Komponente im IoT-Netz kann diese auf einem Edgeserver im lokalen Netzwerk eingesetzt werden. Dadurch werden die Varianzen in den Übertragungsgeschwindigkeiten, sowie die Latenzzeit bis zur Reaktion auf einen Ausfall im Gegensatz zu einer in der Cloud positionierten Komponente minimiert. Die Erkennung eines Gerätefehlers erfolgt spätestens, sobald die Differenz vom Intervall des aktuellen Zeitpunkts und der als letztes empfangene *Heartbeat* zum Mittelwert größer als $3 \cdot \sigma(t)$ ist. (Gleichung 6.3).

$$1 \leq -\log_{10}\left(\frac{\sigma(t)^2}{\sigma(t)^2 + \left(t - t_{n-1} - \mu(t)\right)^2}\right) \tag{6.1}$$

$$\iff 0 < \left(\frac{\sigma(t)^2}{\sigma(t)^2 + \left(t - t_{n-1} - \mu(t)\right)^2}\right) \leq \frac{1}{10} \tag{6.2}$$

$$\implies \text{wenn } t - t_{n-1} - \mu(t) > 3 \cdot \sigma(t) \text{ dann gilt } s(t) \geq 1 \tag{6.3}$$

Wie schon in Abschnitt 4.3.2 erwähnt, stellt dabei die Klassifizierung von Geräteausfällen, Betriebssystemfehlern und Netzwerkabbrüchen ein Problem dar. Abhilfe schafft, die von Seeger et al. vorgestellte Methode zum Versenden der Menge übriger Ressourcen ρ in einer Nachricht [SDSB19]. Diese Metrik muss passend zum Gerät gewählt werden. Es ergibt sich daraus trotz allem keine Garantie auf eine korrekte Klassifizierung eines Ausfalls oder einer Betriebssystemstörung. Beispielsweise kann der batteriebetriebene Sensor T1.1 aus Abbildung 1.2 die verbleibende Batterieleistung mit jeder *Heartbeat*-Nachricht senden. Dennoch kann es zu Situationen kommen, durch die ein Hardwaredefekt ohne Veränderung in ρ das Gerät zum Absturz bringt. Eine Lösung hierfür könnte die Entwicklung einer komplexen Funktion, die in Abhängigkeit mehrerer Geräteparameter die

Wahrscheinlichkeit eines bevorstehenden Ausfalls angibt, sein.
Die fehlerfreie Erkennung von Sensordatenfehlern ist nicht immer möglich. Um Messwerte eindeutig als falsch oder richtig zu klassifizieren, wird darüber hinaus genaues Wissen der Anwendungsanforderungen benötigt. So ist es bei einer Anwendung mit meist konstanten Messwerten einfacher ein *Spike* herauszufiltern, als bei Sensorwerten mit unterschiedlich starken Schwankungen. *Stuck-at* Fehler können hier wiederum sehr gut erkannt werden durch das plötzliche Auftreten ausschließlich konstanter Messwerte. Durch das Verwenden von kleinen Messintervallen können allgemein gute Ergebnisse erzeugt und Fehler von tatsächlichen Messdaten unterschieden werden. Im Beispiel der Brandmeldeanlage aus Abschnitt 1.1 ist anzunehmen, dass die Temperatur und die CO_2-Messwerte annähernd konstant sind oder sich nur langsam verändern. Die Erkennung eines *Stuck-at* Fehlers kann also nur nach einem längeren Zeitraum in dem normalerweise die Temperatur schwankt, erkannt werden, beispielsweise zwischen Tag und Nacht. *Spikes* können gut gefiltert werden, da die CO_2 Konzentration und Temperatur sich nicht plötzlich stark ändert, was im Gegensatz dazu beim Auftreten eines Sensordatenfehlers passieren kann.

Die Fehlerbehandlung von Gerätefehlern, Sensorausfällen und Betriebssystemabstürzen setzt eine vorhandene Redundanz von Geräten beziehungsweise Sensoren voraus. Geräteausfälle oder Betriebssystemfehler auf einem Gerät können nur durch Verlagerung der Aufgabe auf ein anderes Gerät zeitnah behoben werden. In Raum 1 und Raum 4 (Abbildung 1.2) ist Redundanz für die Temperatursensoren gegeben, sodass ein Ausfall kein unmittelbares Problem darstellt. Sollte diese nicht vorhanden sein, wie im Fall eines Ausfalls des CO_2-Sensors, ist meist ein manueller Eingriff von außen nötig, um die Funktionalität des Systems wieder herzustellen. Da Redundanzen im Normalbetrieb nur selten einen Mehrwert bringen, aber zusätzlich Anschaffungs- und Betriebskosten verursachen, werden in der Praxis nur in besonders kritischen Anwendungen redundante Geräte zu finden sein. Um die Qualität der Fehlerbehandlung zu bewerten, ist daher auch immer wichtig, die Anforderungsspezifikationen zu beachten. Bei geringen Anforderungen, wie zum Beispiel eine geringe Anzahl an validen Messungen von Sensordaten in einer großen Zeiteinheit oder eine große Toleranz für einen kurzzeitigen Datenverlust, erbringen die vorgestellten Konzepte ein gutes Ergebnis. Redundanzen sollten nicht nur bei den IoT-Geräten vorhanden sein, sondern auch bei Monitoring Komponenten oder ähnlichem. Sollte eine zentrale Monitoring Instanz ausfallen, muss ein zweites gleichwertiges Gerät zur Verfügung stehen, um ohne Unterbrechung die Funktionalität zu übernehmen. Genaue Analysen und Vergleiche von Ausfallzeiten oder Reaktionszeiten beziehungsweise Wiederherstellungszeiten für einen Fehler sind in dieser Arbeit allerdings nicht vorgesehen.
Wenn die Datenerfassung zeitkritisch ist, beziehungsweise mit so kleinen Latenzzeiten wie möglich erfolgen soll, stellt ein Verbindungsabbruch, trotz der Fehlerbehandlungen in den Unterkapiteln von Abschnitt 4.4 ein Problem dar. Die während der verbindungslosen Zeit gespeicherten Daten können zwar vollständig erneut gesendet werden, allerdings nur mit einer zeitlichen Verzögerung, die für harte Echtzeitanwendungen inakzeptabel sein könnte. Im Motivations-Szenario ist die nachträgliche Übertragung von Messdaten während eines Netzwerkabbruchs allerdings wichtig, um rückwirkend Gefahren zu erkennen. Sollte der Verbindungsabbruch zu lange andauern, ist auch hier die Funktionalität der Brandmeldeanlage nicht mehr gegeben.
Diese, in Abschnitt 4.4.5 vorgestellte Zwischenspeicherung funktioniert nur, solange genug Speicher auf dem IoT-Gerät vorhanden ist. Zudem setzt die Behandlung von Netzwerkfehlern voraus, dass nach einer möglichst kurzen Zeit wieder ein erfolgreicher Verbindungsaufbau erfolgt. Nur so können die zwischengespeicherten Daten überhaupt zur Datenverarbeitung gesendet werden.

Anwendungen mit Daten, die nicht oder nur unwesentlich zeitrelevant sind, profitieren allerdings von der vorgestellten Fehlerbehandlung, da dadurch keine Daten verloren gehen und die Messreihe nicht unterbrochen wird.

6.2 Evaluation des Overheads des Konzepts

Für diese Arbeit wurde keine genaue Messung der Performance-Einbußen durch den zusätzlichen Aufwand von Monitoring und Fehlerbehandlung gemacht. Es wird lediglich eine Schätzung auf den Einfluss in einem IoT-Netzwerk vorgenommen. Der implementierte Prototyp aus Kapitel 5 soll für die Evaluation des Mehraufwands der Konzepte dienen.

Ein performancebeeinflussender Faktor ist die zusätzlich für die Fehlererkennung benötigte Kommunikation zwischen den Geräten. Im Gegensatz zum zentralen Monitoring ist bei einer dezentralen Lösung die Netzwerk- und Rechenauslastung von zentralen Komponenten deutlich geringer . Nachrichten müssen kürzere Wegstrecken zurücklegen und werden verteilt auf verschiedenen Geräten verarbeitet. Die zusätzlichen Datenpakete führen allerdings in beiden Fällen zu einer erhöhten Netzwerkbelastung im Gegensatz zur normalen Anwendungskommunikation.
Beim dezentralen Monitoring wird eine höhere Rechenleistung der beteiligten IoT-Geräte benötigt. In IoT-Netzwerken mit vielen Sensoren ist dies häufig nicht realisierbar, da es sich bei den Sensoren in vielen Fällen nur um *Reduced-Function Devices* handelt, die zu wenig Ressourcen haben, um ein anderes Gerät zu überwachen. Wie bereits in Abschnitt 4.3.1.3 beschrieben, werden solche Geräte mit der Anwendungssoftware direkt geflasht und sind nicht in der Lage mehrere Programme nebenläufig auszuführen. In größeren IoT-Netzwerken, mit mehreren *Full-Function Devices*, kann aber durchaus eine dezentrale Überwachung und Fehlerbehandlung sinnvoll sein. Diese, meist mit einer externen Stromquelle verbundenen Geräte, sind mit leistungsfähigeren Komponenten, wie zum Beispiel Prozessor und Arbeitsspeicher ausgestattet.
Der in Kapitel 5 vorgestellte Code kann auf Einplatinen-Computern des Typs Raspberry Pi[1] der dritten Generation ausgeführt werden. Der lokale Agent läuft dabei parallel zur Anwendung des Motivations-Szenarios (Abschnitt 1.1). Der Mehraufwand durch das Monitoring und die Fehlerbehandlung bringt die Raspberry Pis dabei nicht an ihr Leistungslimit, sodass die Ausführung der eigentlichen Anwendung davon nicht beeinträchtigt wird. Die Übernahme der Aufgaben von anderen ausgefallenen Geräten, wie in Abschnitt 4.4.1.2 beschrieben, führt dennoch zu einer Auslastung freier Rechenressourcen. Dadurch stößt diese Implementierung auf relativ leistungsschwachen IoT-Geräten an ihre Leistungsgrenze, wenn viele Geräte in einem kurzen Zeitraum ausfallen.

[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

7 Zusammenfassung

Durch die steigende Anzahl weltweit vernetzter Geräte steigt der Bandbreiten- und Adressbedarf im sogenannten Internet of Things stark an. Die M2M-Kommunikation ermöglicht das Automatisieren von Prozessen und das gemeinsame Lösen von Aufgaben durch mehrere autonome IoT-Geräte [LN10b; VF13]. IoT findet nicht nur in der Industrie, beispielsweise zur Automatisierung in Produktion, sondern auch in privaten Haushalten durch Geräte zur Hausautomatisierung Verwendung [Dig19]. Durch die überwiegend kabellose Kommunikation und die Energieversorgung mit Hilfe von Batterien entstehen neue Möglichkeiten für die unabhängige Positionierung und Mobilität der Geräte.
IoT-Netzwerke mit vielen Sensoren sammeln eine große Anzahl an Daten. Um Kosten für hohe Bandbreiten und leistungsstarke Server zu sparen, werden die Datenströme meist schon am Rand des Netzwerks, der sogenannten *Edge*, ausgewertet [SCZ+16]. *Edge Computing* verringert zudem die Latenzzeit, da lange Wege durch große Netzwerke wegfallen [YHQL15]. Da die IoT-Geräte nur über eine geringe Leistungsfähigkeit verfügen und auf schlanke Anwendungen angewiesen sind, mussten neue Kommunikationsprotokolle mit geringem Overhead entwickelt werden. Ein oft eingesetztes und datensparsames Protokoll ist MQTT. Durch das *Publish/-Subscriber*-Prinzip können Nachrichten über bestimmte *Topics* an Geräte verteilt werden. Die Kommunikation läuft dabei über einen zentralen Server, dem Broker, der zusätzlich dem Absender verschiedene Quality-of-Services gewährleisten kann. Um die vielen Geräte in IoT-Netzwerken zu überwachen, wurden neue Monitoring-Konzepte entwickelt [ABDP13]. Diese Verfahren basieren meistens auf *computation-based* Tests, die Geräte nach den verbleibenden Ressourcen abfragen oder die Reaktionszeit auf eine Anfrage messen. Zusätzlich werden *network-based* Tests durchgeführt, um Latenzzeiten im Netzwerk zu messen oder Verbindungsabbrüche festzustellen.

Mit der ansteigenden Komplexität in IoT-Umgebungen durch die Vergrößerung oder die Dezentralisierung, erhöht sich die Wahrscheinlichkeit auf Fehler und Ausfälle von Geräten oder Verbindungen. Dabei können Funktionen einer Einheit oder des gesamten Netzwerkes nicht mehr sichergestellt werden. Während in manchen Szenarien ein zeitweiser Verlust von nicht übertragenen Daten toleriert werden kann, muss in anderen Anwendungsgebieten die Übertragung lückenlos und ohne Verzögerung gewährleistet sein. Um die Anforderungen der großen Menge an verschiedenen IoT-Anwendungen zusammenzufassen, wurde in dieser Arbeit eine Übersicht (Tabelle 4.1) mit ausgewählten Anforderungen erstellt und beschrieben. Unterschiedliche Anforderungen setzen verschiedene Genauigkeiten von Fehlererkennung und Fehlerbehandlung voraus, die eingehalten werden müssen.
In dieser Arbeit wurden Fehlerkategorien definiert, die helfen, gleiche Methoden zur Erkennung und Behandlung zu bündeln. Fehler und die damit einhergehende Einschränkungen in der Funktionalität können in Netzwerkfehler und lokale Fehler unterschieden werden. Letztere werden dabei in Geräte- und Sensorausfälle (Abschnitt 4.2.1), sowie Sensordatenfehler (Abschnitt 4.2.2) unterschieden. Diese Gruppe ordnet sich in die Kategorie der Hardwarefehler ein. Softwarefehler sind Abstürze und Fehler in der IoT-Anwendung (Abschnitt 4.2.3) oder im Betriebssystem (Abschnitt 4.2.4). Netzwerkfehler werden in Fehler aufgrund von schlechten Verbindungen (Abschnitt 4.2.5) oder

komplette Verbindungsabbrüche (Abschnitt 4.2.6) aufgeteilt.
Die Erkennung solcher Fehler kann lokal, auf einer zentralen Monitoring-Komponente, oder dezentral vorgenommen werden. Lokale Fehlerüberwachungen eignen sich vor allem bei Anwendungs- oder Netzwerkfehlern, da eine schnelle Reaktion gewährleistet werden kann und kein Fernzugriff auf das Gerät benötigt wird, beziehungsweise aufgrund des Fehlers gar nicht mehr möglich ist. Eine zentrale Monitoring-Komponente kann Geräteausfälle und Netzwerkparameter überwachen, hat allerdings keinen Zugriff auf Anwendungsdaten und erkennt somit ohne weiteres keine Datenfehler. Außerdem muss diese Komponente skalierbar sein und sollte nicht nur ein Gerät mit geringer Rechenleistung sein, da alle zusätzlich hinzukommenden Geräte im IoT-Netzwerk mehr Rechenaufwand bedeuten. Dieses Problem kann mit dezentralem Monitoring umgangen werden, indem jedes IoT-Geräte ein anderes überwacht, wie beispielsweise mit der von Su et al. beschriebenen Überwachung im Ring [SSH+14].
In dieser Arbeit wurden außerdem Konzepte zur Behandlung der genannten Fehlerkategorien vorgestellt. Die Fehlerbehandlung kann nicht nur lokal durch dasselbe Gerät durchgeführt werden, sondern auch durch eine zentrale oder dezentrale Komponente, die beispielsweise durch automatische Softwareverteilung Ersatzgeräte einsetzt. Die MBP ist eine hierfür gut geeignete Plattform und macht das automatische Verteilen und Starten beliebiger Skripte möglich [FHM19; FHPM18]. Um eine möglichst große Robustheit gegenüber den in Abschnitt 4.2 definierten Fehlern aufzuweisen, wird in Kapitel 4.5 eine Beispiel-Architektur eines kleinen IoT-Netzwerkes angegeben, welches in der Lage ist, die Fehler je nach Anforderungen der Anwendung bestmöglich zu erkennen und zu behandeln.

In Kapitel 5 wird die Implementierung von verschiedenen Monitoring- und Fehlerbehandlungsverfahren vorgestellt. Mit Hilfe der Skript-Sprache Python wird zusätzlich zum Motivations-Szenario aus Abschnitt 1.1, die Überwachung von Fehlern und deren Behandlung praktisch umgesetzt.
Diese Konzepte und Methoden werden im darauf folgenden Kapitel (Kapitel 6) auf die Effektivität in IoT-Netzwerken geprüft.
Inwieweit IoT-Umgebungen robuster gegen Ausfälle und Fehler gemacht werden können, hängt stark von den einzuhaltenden Anforderungen ab. Zwar können Maßnahmen, wie zum Beispiel erhöhte Redundanzen von Geräten und Sensoren dazu beitragen Ausfälle und Fehler zu kompensieren, trotzdem ist es nur schwer möglich, zum Beispiel Echtzeitanwendungen mit extrem hohen Anforderungen in IoT-Netzwerken mit Geräten mit vergleichsweiser geringen Rechenleistung oder mit kabelloser Datenübertragung, eine durchgängige zuverlässige Fehlervermeidung zuzusichern.

7.1 Ausblick

In Zukunft ist anzunehmen, dass die Verwendung von IoT zeitgleich mit der Anzahl an verbundenen Geräten in diesen Netzwerken weiter steigen wird. So ist die Prognose, dass bis zum Jahr 2025 über 75 Milliarden IoT-Geräte miteinander vernetzt sein werden, was wiederum die Wahrscheinlichkeit von Ausfällen in derartigen Netzwerken erhöht [MF16]. Durch den Fortschritt in der Rechenleistung von Mikrocomputern lassen sich allerdings komplexere Überwachungssysteme implementieren, die nicht nur aktuelle Fehler erkennen, sondern auch durch maschinelles Lernen Abhängigkeiten zwischen Sensoren und Aktoren herausfinden und somit Ausfälle und Fehler vorhersehen können [KKSS16; WVN15]. Um das in dieser Arbeit vorgestellte Konzept zu erweitern, könnten vor der Anwendungslaufzeit solche Abhängigkeiten auch manuell durch den Anwender beschrieben werden. Maßnahmen im Fehlerfall, wie manuelle Eingriffe durch einen Menschen oder automatische Softwareverteilungen auf andere Geräte, können damit ergriffen werden bevor der Fehler auftritt. Darüber hinaus ist die Fehlervorhersage eine weitere Möglichkeit, um die Robustheit in IoT-Umgebungen zu erhöhen und auch hohe Anforderungen von Anwendungen an die Ausfallsicherheit zu erfüllen. Durch verschiedene Muster in Geräte- und Netzwerkparametern können die Wahrscheinlichkeiten bevorstehender Ausfälle berechnet werden und präventiv Maßnahmen zur Fehlervermeidung eingeleitet werden [WVN15].

Danksagung

Recht herzlich möchte ich mich bei Prof. Dr.-Ing. habil. Bernhard Mitschang für die Möglichkeit meine Bachelorarbeit am Institut für Parallele und Verteilte Systeme durchführen zu können, bedanken.

Bei Daniel Del Gaudio möchte ich mich für die freundliche Betreuung und die motivierenden Worte bei der Erstellung meiner Bachelorarbeit bedanken. Vielen Dank für das Korrekturlesen und die wertvollen Anregungen bei der Gestaltung der Arbeit.

Ein großer Dank gilt auch dem Institut für Parallele und Verteilte Systeme für die Bereitstellung von drei Raspberry Pis mit diversen Sensoren und Aktoren zur Erprobung und Evaluierung meines Prototyps.

Literaturverzeichnis

[16] „IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications“. In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), S. 1–3534 (zitiert auf S. 17).

[ABBG19] E. by Andrew Banks, E. Briggs, K. Borgendale, R. Gupta. *MQTT Version 5.0*. März 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (zitiert auf S. 21).

[ABDP13] G. Aceto, A. Botta, W. de Donato, A. Pescapè. „Cloud monitoring: A survey“. In: *Computer Networks* 57.9 (2013), S. 2093–2115. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2013.04.001 (zitiert auf S. 23, 24, 31, 40, 41, 45, 61).

[ASSC02] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. „Wireless sensor networks: a survey“. In: *Computer Networks* 38.4 (2002), S. 393–422. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(01)00302-4 (zitiert auf S. 29).

[BBKL13] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. „Automated Discovery and Maintenance of Enterprise Topology Graphs“. In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. 2013, S. 126–134 (zitiert auf S. 43, 45).

[CNC+18a] T. Chakraborty, A. Nambi, R. Chandra, R. Sharma, M. Swaminathan, Z. Kapetanovic, J. Appavoo. „Fall-curve: A novel primitive for IoT Fault Detection and Isolation“. In: Nov. 2018, S. 95–107. DOI: 10.1145/3274783.3274853 (zitiert auf S. 27, 44).

[CNC+18b] T. Chakraborty, A. Nambi, R. Chandra, R. Sharma, M. Swaminathan, Z. Kapetanovic, J. Appavoo. „Fall-curve: A novel primitive for IoT Fault Detection and Isolation“. In: Nov. 2018, S. 95–107. DOI: 10.1145/3274783.3274853 (zitiert auf S. 32, 33, 39).

[CWH13] A. Chandra, J. Weissman, B. Heintz. „Decentralized Edge Clouds“. In: *IEEE Internet Computing* 17.5 (2013), S. 70–73 (zitiert auf S. 16).

[DD07] L. De Nardis, M. Di Benedetto. „Overview of the IEEE 802.15.4/4a standards for low data rate Wireless Personal Data Networks“. In: *2007 4th Workshop on Positioning, Navigation and Communication*. 2007, S. 285–289 (zitiert auf S. 37).

[DH19] D. Del Gaudio, P. Hirmer. „A lightweight messaging engine for decentralized data processing in the Internet of Things“. In: (Aug. 2019). DOI: 10.1007/s00450-019-00410-z (zitiert auf S. 29).

[Dig19] I. DigiWorld. *ETNO - Annual Economic Report 2019*. Jan. 2019 (zitiert auf S. 16, 61).

[DP11] W. Dargie, C. Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Jan. 2011. DOI: 10.1002/9780470666388 (zitiert auf S. 19).

[DRC98] S. Deering, N. R. Hinden, Cisco. *Internet Protocol, Version 6 (IPv6) Specification*. Dez. 1998 (zitiert auf S. 16).

[DRH20] D. Del Gaudio, M. Reichel, P. Hirmer. „A life cycle method for device management in dynamic IoT environments“. In: 2020 (zitiert auf S. 43).

[FHM19] A. C. Franco da Silva, P. Hirmer, B. Mitschang. „Model-Based Operator Placement for Data Processing in IoT Environments“. In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2019, S. 439–443 (zitiert auf S. 28, 62).

[FHPM18] A. C. Franco da Silva, P. Hirmer, R. K. Peres, B. Mitschang. „An Approach for CEP Query Shipping to Support Distributed IoT Environments“. In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2018, S. 247–252 (zitiert auf S. 16, 28, 42, 43, 45, 46, 62).

[FP05] E. Ferro, F. Potorti. „Bluetooth and Wi-Fi wireless protocols: a survey and a comparison“. In: *IEEE Wireless Communications* 12.1 (2005), S. 12–26 (zitiert auf S. 16).

[HCH+14] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, M. Satyanarayanan. „Towards Wearable Cognitive Assistance“. In: *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '14. Bretton Woods, New Hampshire, USA: Association for Computing Machinery, 2014, S. 68–81. ISBN: 9781450327930. DOI: 10.1145/2594368.2594383 (zitiert auf S. 20).

[HN16] D. Happ, T. M. N. Karowski. „Meeting IoT platform requirements with open pub/sub solutions“. In: *Ann. Telecommun. 72* (Juli 2016), S. 41–52. DOI: 10.1007/s12243-016-0537-4 (zitiert auf S. 30).

[Hol04] C. C. Holt. „Forecasting seasonals and trends by exponentially weighted moving averages“. In: *International Journal of Forecasting* 20.1 (2004), S. 5–10. ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2003.09.015 (zitiert auf S. 41).

[HTS08] U. Hunkeler, H. L. Truong, A. Stanford-Clark. „MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks“. In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*. Jan. 2008, S. 791–798. DOI: 10.1109/COMSWA.2008.4554519 (zitiert auf S. 22).

[KKSS16] P. A. Kodeswaran, R. Kokku, S. Sen, M. Srivatsa. „Idea: A System for Efficient Failure Management in Smart IoT Environments“. In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore: Association for Computing Machinery, 2016, S. 43–56. ISBN: 9781450342698. DOI: 10.1145/2906388.2906406 (zitiert auf S. 28, 63).

[LDB04] Li Jiang, Da-You Liu, Bo Yang. „Smart home research“. In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*. Bd. 2. Aug. 2004, 659–663 vol.2. DOI: 10.1109/ICMLC.2004.1382266 (zitiert auf S. 19).

[Lig17] R. Light. „Mosquitto: server and client implementation of the MQTT protocol“. In: *Journal of Open Source Software* 2.13 (2017), S. 265. DOI: 10.21105/joss.00265 (zitiert auf S. 50).

[LL15] I. Lee, K. Lee. „The Internet of Things (IoT): Applications, investments, and challenges for enterprises“. In: *Business Horizons* 58.4 (2015), S. 431–440. ISSN: 0007-6813. DOI: 10.1016/j.bushor.2015.03.008 (zitiert auf S. 33).

[LN10a] Lu Tan, Neng Wang. „Future internet: The Internet of Things“. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. Bd. 5. 2010, S. V5-376-V5–380 (zitiert auf S. 15).

[LN10b] Lu Tan, Neng Wang. „Future internet: The Internet of Things“. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. Bd. 5. Aug. 2010, S. V5-376-V5–380. DOI: 10.1109/ICACTE.2010.5579543 (zitiert auf S. 19, 61).

[MF16] I. Markit, Forbes. Nov. 2016. URL: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/%5C#6a558beb292d (zitiert auf S. 15, 63).

[PRS09] P. Patel, A. Ranabahu, A. Sheth. *Service Level Agreement in Cloud Computing*. 2009 (zitiert auf S. 23).

[RF09] R. van Riel, S. Feng. *Documentation for /prod/sys/kernel/**. 1998,1999,2009. URL: https://www.kernel.org/doc/Documentation/sysctl/kernel.txt (zitiert auf S. 42).

[RRN+18] R. Ranjan, O. Rana, S. Nepal, M. Yousif, P. James, Z. Wen, S. Barr, P. Watson, P. P. Jayaraman, D. Georgakopoulos, M. Villari, M. Fazio, S. Garg, R. Buyya, L. Wang, A. Y. Zomaya, S. Dustdar. „The Next Grand Challenges: Integrating the Internet of Things and Data Science“. In: *IEEE Cloud Computing* 5.3 (Mai 2018), S. 12–26. ISSN: 2372-2568. DOI: 10.1109/MCC.2018.032591612 (zitiert auf S. 29, 32).

[RSS15] G. G. Redhyka, D. Setiawan, D. Soetraprawata. „Embedded sensor fusion and moving-average filter for Inertial Measurement Unit (IMU) on the microcontroller-based stabilized platform“. In: *2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*. 2015, S. 72–77 (zitiert auf S. 44).

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge Computing: Vision and Challenges“. In: *IEEE Internet of Things Journal* 3.5 (Okt. 2016), S. 637–646. ISSN: 2372-2541. DOI: 10.1109/JIOT.2016.2579198 (zitiert auf S. 16, 20, 61).

[SDSB19] J. Seeger, R. A. Deshmukh, V. Sarafov, A. Bröring. „Dynamic IoT Choreographies“. In: *IEEE Pervasive Computing* 18.1 (Jan. 2019), S. 19–27. ISSN: 1558-2590. DOI: 10.1109/MPRV.2019.2907003 (zitiert auf S. 16, 27, 29, 37–41, 52, 57).

[Sin17] S. Singh. *Industrie 4.0 und das Industrial Internet of Things (IIoT) – eine Einordnung*. Aug. 2017. URL: https://www.industry-of-things.de/industrie-40-und-das-industrial-internet-of-things-iiot-eine-einordnung-a-629710/ (zitiert auf S. 19).

[Spr11a] J. Spring. „Monitoring Cloud Computing by Layer, Part 1“. In: *IEEE Security Privacy* 9.2 (März 2011), S. 66–68. ISSN: 1558-4046. DOI: 10.1109/MSP.2011.33 (zitiert auf S. 23, 24).

[Spr11b] J. Spring. „Monitoring Cloud Computing by Layer, Part 2“. In: *IEEE Security Privacy* 9.3 (Mai 2011), S. 52–55. ISSN: 1558-4046. DOI: 10.1109/MSP.2011.57 (zitiert auf S. 23, 24).

[SSH+14] P. H. Su, C. Shih, J. Y. Hsu, K. Lin, Y. Wang. „Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware“. In: *2014 IEEE World Forum on Internet of Things (WF-IoT)*. März 2014, S. 45–50. DOI: 10.1109/WF-IoT.2014.6803115 (zitiert auf S. 27, 37, 62).

[SWWM10] J. Shao, H. Wei, Q. Wang, H. Mei. „A Runtime Model Based Monitoring Approach for Cloud“. In: *2010 IEEE 3rd International Conference on Cloud Computing*. 2010, S. 313–320 (zitiert auf S. 36).

[TB15] A. S. Tanenbaum, H. Bos. *Modern operating systems*. Pearson, 2015, S. 3–5 (zitiert auf S. 33).

[TD96] N. C. S. Technology, S. Division. *Federal Standard 1037C*. General Services Administration Information Technology Service, Aug. 1996 (zitiert auf S. 29).

[VF13] O. Vermesan, P. Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River publishers, 2013 (zitiert auf S. 15, 19, 61).

[VWB+16] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, D. S. Nikolopoulos. „Challenges and Opportunities in Edge Computing“. In: *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. Nov. 2016, S. 20–26. DOI: 10.1109/SmartCloud.2016.18 (zitiert auf S. 16).

[WVN15] C. Wang, H. T. Vo, P. Ni. „An IoT Application for Fault Diagnosis and Prediction“. In: *2015 IEEE International Conference on Data Science and Data Intensive Systems*. 2015, S. 726–731 (zitiert auf S. 63).

[X2094] I.-T. X.200. *Information technology – Open Systems Interconnection – Basic Reference Model: The basic model*. Juli 1994 (zitiert auf S. 20).

[YHQL15] S. Yi, Z. Hao, Z. Qin, Q. Li. „Fog Computing: Platform and Applications“. In: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. Nov. 2015, S. 73–78. DOI: 10.1109/HotWeb.2015.22 (zitiert auf S. 20, 61).

Alle URLs wurden zuletzt am 01. 07. 2020 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift