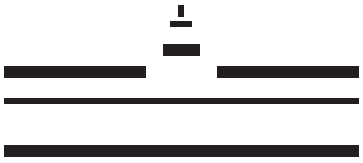
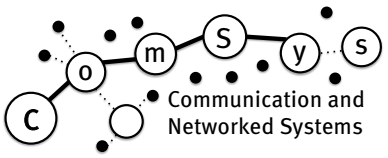


Westfälische
Wilhelms-Universität
Münster



Communication and Networked Systems

Masterarbeit

Migration autonomer Dienste im Internet der Dinge anhand topologischer Charakteristika

Simon Lansing

Betreuer: Prof. Dr. rer. nat. Mesut Güneş
Betreuender Assistent: MSc. Tim Löpmeier

Institut für Informatik, Universität Münster, Deutschland

08. Februar 2017

Zusammenfassung

Zusammenfassung

Das Internet der Dinge umfasst bereits heute eine Vielzahl heterogener Objekte. Aufgrund der geografischen Verteilung der Objekte, lässt sich das Internet der Dinge derzeit in viele einzelne Segmente unterteilen, zwischen denen die Kommunikation über Gateways und dem globalen Internet erfolgt. Alle von den Objekten ausgetauschten Informationen und getroffenen Entscheidungen erzeugen Unmengen an Daten, die über Dienste des Cloud Computings in einer entfernten Cloud-Infrastruktur ausgewertet werden. Durch die weite Verteilung und der immer weiter steigenden Anzahl heterogener Objekte im Internet der Dinge ist diese entfernte Cloud-Infrastruktur wegen hoher Latenzzeiten für Echtzeitdienste jedoch nicht geeignet. Die Lösung dieser Probleme ist das „Fog Computing“, das eine verteilte Ausführung von Diensten dezentral in der physischen Nähe der Objekte ermöglichen soll. Die physische Nähe ist dabei nicht definiert und erstreckt sich von den Gateways, über dedizierten Dienstobjekten bis hin zu allen anderen Objekten im Internet der Dinge.
Ziel dieser Masterarbeit ist die Entwicklung einer Plattform, die eine automatische Migration von Diensten in die physische Nähe der Objekte im Internet der Dinge umsetzt, um Latenzzeiten zu minimieren. Dazu werden zunächst verschiedene Eigenschaften möglicher Dienste analysiert, die innerhalb des Internets der Dinge für das Fog Computing angeboten werden können. Diese und weitere Untersuchungen zu bekannten topologischen Charakteristika bilden die Grundlage für die Bereitstellung der Dienste des Fog Computings. Die Implementierung der Plattform erfolgt anhand der Anforderungen zur OpenFog-Architektur des OpenFog-Konsortiums sowie weiterer Anforderungen für Middlewares im Internet der Dinge. Dabei wurde auf topologische Verfahren zur Aufnahme von Netzwerkdaten und Migration von Diensten aus früheren Untersuchungen zu Ad-Hoc-Netzwerken zurückgegriffen. Neu ist bei der Plattform gegenüber den Technologien der Middlewares die unterstützte Autonomie der bereitgestellten Dienste. Abschließend wird eine Evaluation durchgeführt, die durch den Vergleich verschiedener Positionen in einem segmentierten Internet der Dinge zeigt, dass Migrationen der Dienste innerhalb der Segmente die besten Latenzzeiten bei der Bereitstellung gegenüber Diensten an Gateways oder in einer Cloud-Infrastruktur bieten. Insgesamt kann so bestätigt werden, dass die Plattform für das Fog Computing durch die physische Nähe zwischen Diensten und Objekten die Latenzzeiten bei der Kommunikation verbessert. Zusätzlich kann mit den Ergebnissen dieser Masterarbeit gezeigt werden, dass die Bereitstellung der Dienste weitere Verbesserungen in der Latenzzeit bietet, wenn sie verteilt innerhalb einzelner Segmente des Internets der Dinge erfolgt. So wäre es in naher Zukunft bereits möglich allen Personen alle Objekte zu jeder Zeit zur Verfügung zu stellen.

Abstract

The todays Internet of Things already includes a multitude of heterogeneous objects. Due to the geographic distribution of the objects, the Internet of Things is currently divided into many individual segments, which communicate via gateways and the global Internet. All information exchanged and decisions taken by the objects generate vast amounts of data, which are evaluated via cloud computing services in a remote cloud infrastructure. In consequence of the wide distribution and ever-increasing number of heterogeneous objects in the Internet of Things, this remote cloud infrastructure is not suitable for real-time services because of high latencies. The solution to these problems is the “fog computing”, which is designed to enable a distributed execution of services decentrally in the physical proximity of the objects. The physical proximity is not defined and extends from the gateways, over dedicated service objects to all other objects in the Internet of Things.
The goal of this master’s thesis is the development of a platform that automatically migrates services to the physical proximity of objects in the Internet of Things to minimize latencies. For this purpose, various characteristics of possible services are analyzed, which can be offered for the fog computing within the Internet of Things. These and other studies on known topological characteristics form the basis for the provisioning of the services of fog computing. The platform is implemented according to the specifications of the OpenFog architecture of the OpenFog Consortium and other requirements for middlewares on the Internet of Things. In this case, topological procedures to record network data and to migrate services were used from previous investigations into ad-hoc networks. A new feature compared to the technologies of middlwares is the supported autonomy of the provided services of the platform. Finally, an evaluation is carried out by comparing different positions in a segmented Internet of Things. It shows that migrations of the services provided within the segments offer the best latencies compared to services on the gateway or in a cloud infrastructure. All in all, it can be confirmed that the platform for fog computing improves the latencies of communication by the physical proximity between services and objects. In addition, the results of this master’s thesis show that the provisioning of the services provides further improvements in the latency when dispersed within the individual segments of the Internet of Things. This indicates, that all objects could be available to all persons at any time in the future.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Akronyme

API Application Program Interface. 31, 39, 45, 46, 69

CPU Central Processing Unit. 45–47

ETT Erwartete Übertragungszeit (engl. Expected Transmission Time). 18

ETX Erwartete Übertragungsrate (engl. Expected Transmission Count). 17, 18, 20, 22–28, 40, 46, 50

IEEE Institute of Electrical and Electronics Engineers. 17, 19, 20, 22

IoT Internet der Dinge (engl. Internet of Things). 1–3, 5–13, 15–20, 24, 29, 31–36, 38–43, 45, 47, 48, 51, 61–67, 69–71

IPv4 Internet Protokoll Version 4. 23, 46

IPv6 Internet Protokoll Version 6. 23, 34

ITU Internationale Fernmeldeunion (engl. International Telecommunication Union). 1, 7

JDK Java Development Kit. 46

JRE Java-Laufzeitumgebung (engl. Java Runtime Environment). 47

MIOT Münster Internet of Things. 13, 15, 16, 18–20, 22–29, 39, 43–46, 69

MSS Maximale Segmentgröße (engl. Maximum Segment Size). 51, 55, 57, 58

MTU Maximale Übertragungseinheit (engl. Maximum Transmission Unit). 23, 51

NRO Network Resource Optimization. 45

QoS Quality of Service. 2, 35, 44

RAM Arbeitsspeicher (engl. Random-Access Memory). 46, 47

RFID Identifizierung mit Hilfe elektromagnetischer Wellen (engl. Radio-Frequency Identification). 1

RTT Paketumlaufzeit (engl. Round-Trip-Time). 10, 18, 20, 52, 55, 57–67, 70

SDN Software-Defined Networking. 13, 18, 20, 22, 24, 26, 28, 38, 43–48, 50, 52, 53, 55, 57, 60, 62, 63, 66–69

SOA Serviceorientierte Architektur. 6, 7, 10

STP Spannbaum-Protokoll (engl. Spanning Tree Protokoll). 47

TCP Transmission Control Protocol. 22, 39, 41, 42, 48, 51

TTL Time to live. 16

UDP User Datagram Protocol. 22, 39, 42, 49

VLAN Virtual Local Area Network. 44

VM Virtuelle Maschine (engl. Virtual Machine). 45–47

WSN Kabelloses Sensornetzwerk (engl. Wireless Sensor Netzwork). 19

Glossar

Ad-hoc-Netzwerk Ein Ad-hoc-Netzwerk ist ein Zusammenschluss von Netzwerkknoten, die selbstständig ein gemeinsames Mesh-Netzwerk bilden und konfigurieren sowie ohne einen zentralen Zugangspunkt miteinander kommunizieren können [37]. 11, 69

Broadcast Ein Broadcast ist eine Methode zur gleichzeitigen Übermittlung einer Nachricht an alle Teilnehmer eines Netzwerkes. 17

Cloud Die Hardware und Software in einem Rechenzentrum wird als die Cloud bezeichnet [12]. Verwendung findet sie bei dem Cloud Computing, bei dem eine schnelle Bereitstellung von Software, Entwicklungswerkzeuge, Hard- und Software-Infrastruktur, Rechenleistung oder Speicherkapazitäten von Anbietern als Dienstleistungen über das Internet erfolgt und auf Anforderungen von Kunden verwendet und wieder freigegeben werden kann. Sowohl die Anwender dieser Ressourcen als auch die Kunden müssen sich nicht um die unterliegende Cloud-Infrastruktur kümmern [13]. Für mehr Informationen vgl. Kapitel 1. 2, 3, 8, 19, 33, 70, 71

Constrained Application Protocol (CoAP) Das Constrained Application Protocol ist ein Webtransfer-Protokoll für die Verwendung in einer Umgebung aus Objekten und Netzwerken, die in ihren Ressourcen eingeschränkt sind. Während die Objekte oft eine geringe Speicherkapazität aufweisen, sind die Netzwerke mit einer hohen Paketfehlerrate und sehr geringen Durchsatz gekennzeichnet [35]. 9

Diensteanbieter Ein Dienstanbieter (engl. Service Provider) im Sinne des Cloud Computings stellt Dienstleistungen, wie Server-Infrastrukturen und einhergehende Rechenleistungen und Speicherkapazitäten, bereit. Da ein Anbieter es mehrere Kunden gleichzeitig anbieten kann, können er die Kapazitäten besser skalieren. Infolgedessen sinken die Kosten der Dienstleistungen für die Kunden im Vergleich einer eigenen Infrastruktur. 8–10

Durchsatz Der Durchsatz einer Verbindung ist ein Maß für die maximale Menge erfolgreich übertragener Daten pro Zeiteinheit. 10, 16–18, 20, 22–28, 40, 46, 50, 51, 61

Echtzeit Unter Echtzeit (engl. real-time) versteht man die unmittelbare Reaktion ohne spürbare Verzögerung einer Anwendung. Eine genaue zeitliche Festlegung gibt es für den Begriff allerdings nicht [44]. 18

Entwurfsmuster Ein Entwurfsmuster (engl. Design Pattern) ist eine wiederverwendbare

Lösungsschablone für bekannte Probleme bei dem Entwurf einer Softwarearchitektur [55]. 33

Fog Computing Das Fog Computing stellt eine Brücke zwischen Cloud Computing und dem Internet der Dinge dar, bei der die Rechenkapazität für das Internet der Dinge weit verteilt am „Rande des Netzwerks“ bereitgestellt wird. Die Eigenschaften des Fog Computings sind geringe Latenzen, Echtzeit, Standortkenntnis, geografisch weite Verteilung, Mobilität und eine große Anzahl Objekte [15]. 2, 3, 5, 8, 10, 31, 33, 49, 69, 70

Fork Eine Abspaltung (engl. Fork) ist in der Softwareentwicklung ein Entwicklungszweig eines Projektes, bei dem der Quellcode teilweise oder vollständig vom Projekt übernommen und unabhängig weiterentwickelt wird. 46

Gateway Ein Gateway ist ein Gerät einer Netzwerk-Infrastruktur, das zwei Netzwerke unterschiedlicher Technologien miteinander verbindet und die Kommunikation durch eine gegenseitige Übersetzung des Inhaltes ermöglicht. 7–12, 51

Hop Ein Hop ist ein Wegabschnitt zwischen einem Start- und einem Zielknoten für ein im Netzwerk weitergeleitetes Paket, wobei jede Bridge, jeder Router und jedes Gateway als ein Hop zählt. 12, 13, 16, 44, 62–64, 66, 67

IaaS Infrastructure as a Service bezeichnet die Bereitstellung von Rechenleistung, Speicherkapazitäten, Netzwerken und anderen grundlegenden Rechenressourcen, in der die Kunden jegliche Software, Betriebssysteme und Anwendungsapplikationen bereitstellen und verwalten können [13]. 2

ICMP Das Internet Control Message Protocol (ICMP) ist ein unterstützendes Protokoll der Internetprotokollfamilie und wird von Netzwerkgeräten zur Übermittlung von Informations- und Fehlermeldungen verwendet. Das Diagnosewerkzeug ping verwendet ICMP zur Übertragung von „Echo request“ und „Echo reply“ Paketen. 23

Internetschicht Die Internetschicht (engl. Internet Layer) vermittelt die Pakete durch das Netzwerk, indem sie zunächst durch die Wahl von Routen das nächste Zwischenziel für die Pakete ermittelt und die Pakete anschließend dorthin weiterleitet. Im zentralen Fokus liegt hier das Internet Protokoll in seinen verschiedenen Versionen (IPv4, IPv6). 12

Lastverteilung Eine Lastverteilung (engl. load balancing) verbessert die Verteilung von Arbeitsbelastungen über mehrere Ressourcen mit Rechenleistung, wie Computer. Das Ziel ist eine Optimierung der Ressourcen, wie die Maximierung des Durchsatzes, Minimierung von Latenzen und die Vermeidung der Überlastung einzelner Ressourcen. Dafür wird jedoch eine spezielle Hard- oder Software benötigt, die sich mit der Koordinierung beschäftigt. 19

M2M Machine-to-Machine (M2M) steht für den automatisierten Informationsaustausch (meist via Internet) ohne menschlichen Eingriff zwischen Maschinen, Automaten und Fahrzeugen untereinander oder mit einer zentralen Leitstelle [44]. 2, 5, 7

Mehrkosten Als Mehrkosten (engl. Overhead) im Sinne der Datenübertragung werden die Daten bezeichnet, die zusätzlich zu den reinen Nutzdaten gesendet werden müssen. Dazu zählen beispielsweise neben den Headern von Protokollen auch verschiedene Kontroll- und Signalisierungsdaten, die zur Realisierung einer stabilen Verbindung beitragen. 16, 19, 58

Mesh Ein Mesh-Netzwerk ist eine Netzwerktopologie, bei der jeder Netzwerkknoten mit anderen Netzwerkknoten direkt verbunden ist. Alle Knoten kooperieren miteinander und können die Daten innerhalb des Netzwerkes bis zum gewünschten Ziel weiterleiten. 9, 11, 13, 17–20, 22, 24, 25, 45, 46

NAT Die Netzwerkadressübersetzung (engl. Network Address Translation) ist eine Methode, bei der die IP-Adresse eines Datenpaketes während der Übertragung vom Router durch eine andere ersetzt wird. 12

Netzzugangsschicht Die Netzzugangsschicht (engl. Link Layer) ist die unterste Schicht der Internetprotokollfamilie. Sie verpackt die zu sendenden Pakete in Frames und garantiert eine möglichst fehlerfreie Übertragung zwischen benachbarten Netzwerkknoten. 12

PaaS Platform as a Service bezeichnet die Bereitstellung von Kunden entwickelter Anwendungsapplikationen in einer Cloud-Infrastruktur, die mithilfe der vom Anbieter unterstützen Programmiersprachen, Bibliotheken und Entwicklungsumgebungen entwickelt wurden. Die Kunden haben zumeist die Möglichkeit die bereitgestellten Anwendungsapplikationen und Laufzeitumgebungen zu verwalten [13]. 2

PID Eine Prozess ID ist eine eindeutige Nummer, die ein Betriebssystem zur Identifizierung eines einzelnen Prozesses verwendet. 39

SaaS Software as a Service bezeichnet die Bereitstellung von Anwendungssoftware in einer Cloud-Infrastruktur, die über eine Vielzahl von Clients mittels schmalen Client-Interfaces, APIs oder Webbrowsern erreichbar ist [13]. 2

Spanning Tree Protokoll Das Spannbaum-Protokoll (engl. Spanning Tree Protokoll) ist ein Netzwerkprotokoll, das eine Schleifenfreiheit in logischen Netzwerktopologien garantiert und durch Schleifen erzeugte Broadcast-Stürme verhindert. 28

Steuerungsebene Die Steuerungsebene (engl. Control Plane) ist ein Begriff aus dem Software-Defined Networking, bei der aus Routing-Tabellen des Netzwerkes die Forwarding-Tabellen für die Weiterleitungsebene (engl. Forwarding Plane) erstellt werden [59]. Für mehr Informationen vgl. Kapitel 5.1. 44, 47

Taktile Internet Unter dem Taktilen Internet versteht man eine extrem kurze und für den Menschen nicht wahrnehmbare Reaktionszeit (weniger als eine Millisekunde) einer via Internet gesteuerten Anwendung. Damit werden zukünftige Anwendungen, etwa in der Telemedizin oder Car2Car-Kommunikation, möglich. Vor allem aber wird so das Internet der Dinge realisiert [44]. 18

Verhaltensmuster Ein Verhaltensmuster ist eine Kategorie von Entwurfsmustern. Sie charakterisiert die Art und Weise der Interaktion von Klassen und Objekten sowie die

Verteilung der Zuständigkeiten [55]. 37

Virtual Network Emulator Ein Virtual Network Emulator ist eine Testumgebung, die den Aufbau eines virtuelles Netzwerkes und so das Testen der Performance realer Applikationen ermöglicht. 13, 20, 45, 46, 70

Weiterleitungsebene Die Weiterleitungsebene (engl. Forwarding Plane respektive Data Plane) ist ein Begriff aus dem Software-Defined Networking. Kommen Ethernet Frames an einem Switch-Interface eines Netzwerkgerät an, kümmert sich die Weiterleitungsebene um die Weiterleitung dieser zu einem Ausgang. Über Forwarding-Tabellen können Regeln für die Weiterleitung definiert werden [59]. Für mehr Informationen vgl. Kapitel 5.1. 44, 45

Wertobjekt Ein Wertobjekt (engl. Value Object) ist ein einfaches Objekt, dessen Gleichheit nicht auf seine Identität basiert, sondern auf seinen Wert. Es ist vergleichbar mit den primitiven Datentypen einiger Programmiersprachen [57]. 39

Zeitgeber (engl. Timer) Ein Zeitgeber (engl. Timer) realisiert zeitbezogene Funktionen durch das Messen von Zeitintervallen. 41

KAPITEL 1

Motivation

Das Internet der Dinge (IoT) gilt als einer der nächsten großen Schritte der digitalen Vernetzung, das von der ITU sogar als die „Infrastruktur der Informationsgesellschaft“ bezeichnet wird [1]. Physikalischen Objekten, wie RFID-Tags, Sensoren, Aktoren und smarten Geräten soll das Sehen, Hören, Denken und Ausführen von Aufgaben durch eine eindeutige Adressierung und gemeinsame Kommunikation ermöglicht werden, bei der sie Informationen teilen und darauf basierende Entscheidungen für ein gemeinsames Ziel treffen können [2, 3]. Gegenüber dieser Heterogenität, bei der die verschiedensten physikalischen Objekte miteinander kommunizieren, besteht das derzeitige Internet eher aus einheitlichen Geräten, die mit wenigen Ausnahmen gleiche Eigenschaften besitzen und für denselben Anwendungszwecke geschaffen sind [4]. Aus technischer Perspektive gelingt der vollständige Übergang vom Internet zum IoT nur durch die Vereinigung von Technologien, die sich ausgehend von eingebetteten Systemen mit einer allgegenwärtigen Berechnungsfähigkeit, über Sensornetzwerken und Kommunikationstechnologien bis hin zu den Internetprotokollen und -applikationen spannt [3]. Das weltweit vernetzte Internet, wie wir es heute kennen, wird in diesem Szenario jedoch nicht vollständig verschwinden. Da die physikalischen Objekte im IoT anhand ihrer geringen Ressourcen sowohl bei Rechenleistung als auch bei der Energieverfügbarkeit charakterisiert werden können [5], bleibt das Internet mit seiner entscheidenden Rolle als globales Rückgrat für einen weltweiten Informationsaustausch erhalten und verbindet so die physikalischen Objekte mit seiner verfügbaren Rechenleistung und Kommunikationsfähigkeit über die gesamte Bandbreite von Technologien und Anwendungen [6].
Die Anwendungsmöglichkeiten auf Basis des IoT sind außergewöhnlich vielseitig. Das Spektrum erstreckt sich über die verschiedensten Wirtschaftssektoren, von Transport und Logistik, über den Gesundheits- und Sicherheitssektor bis hin zu automatisierten Umgebungen, wie „Smart Homes“, „Smart Grids“ oder „Smart Cities“ [2]. Im Transport sind die Fahrzeuge mit Sensoren, Aktoren und Prozessorleistung ausgestattet, die kollaborativ mit den vernetzten Sensoren der Infrastruktur den Verkehr regeln, optimieren und über ein Verkehrsleitsystem überwachen. Echtzeitinformationen geben Auskunft über die Waren während der gesamten Lieferkette und optimieren so die Logistik. Die Objekte im IoT zeigen im Gesundheitssektor durch bessere Präventions- und Diagnosemöglichkeiten von Krankheiten eine stetige Besserung der Lebensumstände, während im Krankenhaus durch eine kontinuierliche Überwachung der Patienten mithilfe von Sensoren die Behandlungen verbessert

werden [7]. Zuletzt sind in „smarten“ Umgebung weitere Anwendungen zu finden. In „Smart Homes“ können anhand genauer Vorhersagen Aktionen ausgeführt werden, wie das automatische Schließen von Fenstern bei Wetterumbrüchen oder das Melden von Einbrüchen. „Smart Grids“ erlauben beispielsweise durch „Smart Meter“ eine effiziente Überwachung und anschließende Minimierung des Energie- und Ressourcenverbrauch [3, 8]. Vernetzten Städten können schließlich übergreifende Informationen zur Verbesserung der Lebensqualität und öffentlichen Sicherheit bereitzustellen [3, 7].
Diese Anwendungen spiegelt die gewaltige Größe des IoT wider, das sich derzeit sowohl geografisch als auch in die verschiedenen Wirtschaftssektoren segmentieren lässt. Jahr für Jahr steigt die Anzahl der sich mit dem Internet verbindenden physikalischen Objekte unvergleichlich rasant an [3]. Schätzungen des Marktforschungsunternehmen Gartner, Inc. zufolge werden bis um Jahr 2020 bereits insgesamt 26 Milliarden Objekte in das IoT eingebunden sein [9]. Laut den Analysen von Cisco Systems, Inc. bedeutet dies, dass die Verbindungen von Machine-to-Machine (M2M) zwischen diesen Objekten zu dem Zeitpunkt 46% aller weltweiten Verbindungen im Internet ausmachen werden [10]. Dies stellt das gesamte System vor eine große Herausforderung, denn die physikalischen Objekte im IoT erzeugen eine ungeheure Datenmenge, die sowohl gespeichert, verarbeitet als auch ohne Probleme in einer effizienten und einfachen Form präsentiert werden können muss [11].
Die durch all diese physikalischen Objekte im IoT generierten Daten werden in ihrer Gesamtheit als *„Big Data“* bezeichnet. Diese Datenmenge ist für die Verarbeitung in normalen Hardwareumgebungen und Softwareprogrammen der Anwender jedoch zu groß, sodass sich neue Technologien unter dem Begriff *„Cloud Computing“* etabliert haben, bei denen Anwendungssoftware, Entwicklungswerkzeuge, Rechenleistung und auch Speicherkapazitäten als Dienstleistungen in Rechenzentren über das Internet angeboten werden [12]. Zu den klassischen Dienstleistungen innerhalb des Cloud Computings zählen die Software as a Service (SaaS), Platform as a Service (PaaS) und Infrastructure as a Service (IaaS) [13]. Für die Datenverarbeitung der Objekte im IoT können Kunden mittels IaaS sowohl die Speicher- als auch Rechenkapazitäten für ihre Produkte von externen Cloud-Anbietern mieten. Diese garantieren dafür die kontinuierliche Bereitstellung und Wartung der benötigten Infrastruktur, ohne dass sich die Kunden um die Cloud-Infrastruktur kümmern müssen [3, 13]. Doch zeichnet sich mit der Cloud ein für das IoT sehr zentrales und statisches Modell ab, durch das Komplikationen entstehen. Allein die bloße Datenmenge der Objekte im immer weiter wachsenden IoT mit ihren heterogenen Datentypen bereitet große Probleme für die Analysemethoden [14]. Darüber hinaus benötigt das IoT neben der Mobilität und geografischen Verteilung der Objekte zusätzlich Standortkenntnis über die Objekte, sodass schließlich eine Reduktion von Latenzen und Steigerung der Quality of Service (QoS) erreicht werden kann [15, 16].
Diese Anforderungen können durch eine neue, verteilte Plattform namens *„Fog Computing“* erfüllt werden, die als eine Art Brücke zwischen den heterogenen Objekten und dem traditionellen Cloud Computing agiert [3]. Im Fog Computing werden die benötigten Dienste nicht in einer zentralen Cloud-Infrastruktur bereitgestellt, sondern auf verteilten, heterogenen Geräten der neuen Fog-Infrastruktur, die sich von in der physischen Nähe zum IoT befindlichen Gateways, „Edge“-Routern und Switches bis in zu den Objekten des IoT erstreckt. Durch diese verteile Verarbeitung am „Rande des Netzwerks“ (engl. „Edge of the Network“) können Dienste so nah an den datenerzeugenden Sensoren und steuerbaren Aktoren arbeiten wie nur möglich [16]. Als potentielle Kandidaten für die Bereitstellung solcher Dienste kommen beispielsweise die Mobilfunkbetreiber mit ihren bereits vorhandenen Netzwerken

oder sogar innerhalb der einzelnen Funkzellen in Betracht [3]. Eine Alternative zu diesem Szenario wäre die Bereitstellung der Dienste auf den Objekten im IoT selber, bei der die Dienste zwischen den Objekten migriert werden. Ähnlich wie die Cloud, stellt die Fog Daten, Rechenleistung, Speicher und Anwendungsdienste den Endbenutzern bereit [17]. Dabei ersetzt das Fog Computing allerdings nicht die Cloud, vielmehr erweitert es deren Aufgabenbereiche durch die Abbildung neuer Anwendungen und Dienste auf den Edge-Geräten, die schließlich ein Zusammenspiel zwischen Cloud und Fog ermöglichen [18]. Die Cloud bleibt mit ihrer im Vergleich zur Fog großen Rechenleistung, Speicher- und Kommunikationskapazität weiterhin erhalten [3, 15]. So ergeben sich für das IoT neue Möglichkeiten. Neben der Reduktion von Latenzzeiten und dem einhergehendem Potential zu Echtzeitdiensten durch die geografische Verteilung ist es im IoT nun durch die Bildung von kleinen „Mikro-Fogs“ möglich, die Dichte der Objekte weiter zu erhöhen und dabei weiterhin ein skalierbares System beizubehalten. Durch diese Mikro-Fogs werden die Dienste mit limitierter Rechenleistung, Speicher- und Kommunikationskapazitäten so verteilt angeboten, dass die Kosten auf einen Bruchteil von denen der Cloud-Datenzentren minimiert werden können [3]. Wird außerdem die Bereitstellung der Dienste in einer privaten Umgebung angeboten, können zusätzliche sicherheitsrelevante Aspekte Beachtung finden, sodass schließlich ein Maß an Datenschutz durch Zugriffskontrollen ermöglicht wird, bevor die Daten in die externe Cloud-Infrastruktur gelangen [19].
Das Ziel dieser Masterarbeit ist die Entwicklung einer Plattform, die eine automatische Migration der Dienste in die physische Nähe der Objekte umsetzt, um Latenzzeiten zu minimieren. Dazu soll zunächst die möglichen Dienste und ihre Merkmale analysiert werden, die sich für die Bereitstellung und Migration innerhalb des IoT eignen. Für eine optimale Positionierung zur Minimierung von Latenzzeiten sollen bewertbare Systemeigenschaften und passende Aufnahmemöglichkeiten verwendet werden, die Teil einer weiteren Untersuchung sind. Diese stellen schließlich benötigte Daten für die Plattform zur Positionierung der Dienste bereit, mit der in einer abschließenden Evaluation herausgefunden werden soll, in welchen Bereichen des IoT und auf welchen Geräten die besten Resultate im Bezug auf Latenzzeiten und weiteren Leistungsmetriken zwischen den Objekten im IoT und den Diensten für das Fog Computing geliefert werden können.

KAPITEL 2

Die Migration der autonomen Dienste

Die Grundlage all der in Kapitel 1 genannten Anwendungen wird auf der untersten Ebene durch die einzelnen Objekte innerhalb des IoT gebildet. Sie führen Aufgaben aus, die sowohl durch Sensoren auf äußere Einflüsse reagieren als auch proaktiv tätig werden können. Der entscheidende Faktor hier ist die Vernetzung untereinander mittels einer direkten M2M-Kommunikation, die schließlich alle Objekte kollaborativ arbeiten lässt und die Grundlage für die Anbindung der Anwendungsapplikationen an das vernetzte IoT bildet. Bei dieser Vernetzung der Objekte im IoT und der gemeinsamen Verarbeitung von Informationen entstehen Aufgaben, die von den Objekten automatisiert erledigt werden sollen. Einzelne Aufgaben können dabei auch besser auf leistungsstärkere Geräte zentral ausgelagert werden, weil sie zum Beispiel Informationen zentral zusammenfassen. Dies setzt eine Bereitstellung grundlegender Dienste im IoT voraus, mit denen alle Geräte mittels einer M2M-Kommunikation interagieren können. Die Gesamtheit dieser Dienste in der physischen Nähe der Objekte im IoT lassen sich unter dem Begriff Fog Computing zusammenfassen [17].
Zunächst sind diese grundlegenden Dienste im IoT zu definieren. Bei der Definition als auch bei der Konstruktion und Bereitstellung von Diensten muss auch die Heterogenität der Geräte innerhalb des IoT beachtet werden, denn bestimmte Dienste sind aufgrund ihres Funktionsumfangs von dieser Eigenschaft abhängig. Die Heterogenität der Objekte entscheidet im Betrieb darüber, wo die Dienste ausgeführt werden können, ob sich einige Objekte besser zur Ausführung eines Dienstes eignen als andere und zuletzt, ob die Bereitstellung durch eine verteilte Ausführung des Dienstes verbessert werden kann. Die Beachtung dieser Eigenschaft gewährleistet später eine höhere Skalierbarkeit und Verfügbarkeit der Dienste. Ein weiterer entscheidender Aspekt ist die Autonomie, die im Kontext von Diensten im IoT eine sehr wichtige Eigenschaft darstellt. Dienste, die beispielsweise eine kontinuierliche Abhängigkeit zu externen Ressourcen in anderen Bereichen des IoT bilden, benötigen eine gesonderte Betrachtung bei der Bereitstellung. Letzten Endes entscheidet schließlich die Positionierung der Dienste über den Erfolg des Fog Computings mit der einhergehenden Möglichkeit zur Reduktion der Latenzzeit. Es werden verschiedene Szenarien für die Bereitstellung von Diensten im IoT verglichen, bei denen Leistungsmetriken im Netzwerk optimiert werden sollen. Abschließend sind topologischen Verfahren beschrieben, die bei einer Migration von autonomen Diensten im IoT zum Einsatz kommen. Ihre Datenerhebung und den darauf basierenden Auswertungen bilden die Grundlage der Migration.

2.1 Dienste im Internet der Dinge

Der Begriff „Dienst“ ist innerhalb des IoT mehrdeutig definiert und variiert zwischen verschiedenen Projekten [20]. Einige Autoren beschreiben beispielsweise Dienste im IoT als eine direkte Darbietung von Objekten in der physikalischen Welt, sogenannte „real-world services“. Sie stellen die direkten Informationen aus der physikalischen Welt bereit, sodass mit ihnen in der virtuellen Welt interagiert werden kann [21].
Andere Ansätze definieren Dienste im IoT oftmals anhand von Diensten aus dem Paradigma der Serviceorientierte Architektur (SOA) [22, 23]. Die SOA ist ein Architekturmuster, bei dem versucht wird alle Anwendungen in kleine, voneinander getrennte, logische Einheiten zu kapseln, die separat verteilt werden und in Kombination miteinander arbeiten können. Diese Einheiten werden schließlich Dienste genannt [24]. Es besagt unter anderem, dass ein Dienst eine autarke, plattformunabhängige Komponente mit einer über ein Netzwerk verfügbaren Schnittstelle ist [25]. Bezogen auf das IoT werden alle von den Objekten angebotenen Funktionen durch einen solchen Dienst abstrahiert und den Benutzern auf einer höheren Softwareebene angeboten [22].
Die Autoren Thoma et al. versuchen schließlich aus den Bereichen der „real-world services“ und der SOA eine allgemeingültige Definition für Dienste im IoT zu erstellen [26].

> *An IoT-Service is a transaction between two parties, the service provider and the service consumer. It causes a prescribed function enabling the interaction with the physical world by measuring the state of entities or by initiating actions which will cause a change to the entities.*

Für diese Arbeit ist diese Definition jedoch nicht hinreichend genug, da sie eine Beschränkung auf Funktionalitäten bezüglich der Entitäten enthält, also der physikalischen Ressource der Objekte im IoT. Xiaojiang, Jianli und Mingdong gehen indes einen Schritt weiter. Sie sagen, dass eine weitreichende Anzahl von Applikationen in IoT-Dienste eingebunden werden können und diese sich anhand verschiedener Kriterien klassifizieren und kategorisieren lassen [27]. Anhand technischer Eigenschaften lassen sich Dienste im IoT dabei in vier aufeinander aufbauende Klassen einteilen, dies sind die „Identity-Related Services“, die „Information Aggregation Services“, die „Collaborative-Aware Services“ und die „Ubiquitous Services“. Diese Unterteilung ist von besonderem Interesse, da sie eine gesonderte Betrachtung von Diensten mit physikalischen Ressourcen als auch von Diensten mit einer reinen softwareseitigen Logik beinhaltet. Der Interessenfokus dieser Masterarbeit liegt dabei vor allem auf die Klassen der Information Aggregation Services und der Collaborative-Aware Services.
Die „Identity-Related Services“ sind rudimentäre Dienste, die physikalische Objekte mit einem Identifikationsmerkmal, wie zum Beispiel einem RFID-Tag, in die virtuelle Welt übertragen. Ein Lesegerät kann diese Objekte auslesen und über ein Netzwerk mehr Informationen zu den Objekten erhalten. Diese Dienste sind sehr wichtig, da sie die physikalische und die virtuelle Welt verschmelzen. Dadurch sind sie primär ein grundlegender Bestandteil der anderen Dienstklassen [27, 28].
Die „Information Aggregation Services“ sammeln Daten von Sensoren im IoT, verarbeiten diese Daten und geben sie über das IoT an Anwendungsapplikationen weiter. Sie abstrahieren dabei die physikalischen Eigenschaften von Objekten und die Netzwerkzugänge. Über verschiedene Netzwerkprotokolle wie ZigBee oder 6LoWPAN hinweg können sie mithilfe von Gateways die Daten verschiedenster Sensoren abfragen und diese gebündelt über Web

Services den Anwendungsapplikationen bereitstellen [28].
Die „Collaborative-Aware Services“ bauen auf die Information Aggregation Services auf. Sie aggregieren ebenfalls Daten aus dem IoT, starten anschließend eine Auswertung und führen auf Basis dieser Analyse geforderte Funktionen aus [28]. Ein bekanntes Beispiel dafür ist die Hausautomatisierung für deren Funktionalität sowohl eine M2M-Kommunikation als auch eine erhöhte Zuverlässigkeit, Übertragungsgeschwindigkeit und Prozessleistung der Geräte im IoT unabdingbar sind.
Die Klasse der „Ubiquitous Services“ stellt die letzte Stufe der Dienste im IoT dar. Dies sind Collaborative-Aware Services, die *allen Personen alle Objekte* zu *jeder Zeit* zur Verfügung stellen [28]. Die ITU schreibt in ihrem Empfehlungsschreiben zu Netzwerken mit Ubiquitous Services [29]:

> *”5C+5Any” are key features of ubiquitous networks. 5C stands for convergence, content, computation, communication, and connection. 5Any represents anytime, anywhere, any service, any network, and any object.*

Diese Klasse stellt somit das ultimative Ziel aller Dienste im IoT dar, das es zu erreichen gilt [3]. Zum aktuellen Stand ist es jedoch noch nicht absehbar, ob diese Ubiquitous Services überhaupt über das Internet angeboten werden [27]. Zur Erreichung dieses Status müssen sowohl die Heterogenität, verursacht durch Protokollunterschiede, über Technologien hinweg aufgelöst und alle Aspekte des Netzwerks vereinigt werden [28, 30].

2.2 Autonomie von Diensten

Ein weiterer zu betrachtender Aspekt bei der Beschreibung von Diensten im IoT ist die Autonomie von Dienste, die die wichtigste Anforderung an die Dienste im IoT ist. Erl bezeichnet die Autonomie als die Fähigkeit zur Selbstbestimmung ist [31]. Abgeleitet aus dem Paradigma der SOA, sollen Dienste im IoT die volle Kontrolle über die Durchführung ihrer Logik haben, ohne von externen Einflüssen abhängig zu sein [25]. Es ist ein Qualitätsmerkmal, mit dem die Zuverlässigkeit, Performance und Vorhersagbarkeit im Verhalten von Softwareanwendungen gesteigert werden kann. Einhergehend muss die Implementierung solcher Dienste jedoch isoliert und unabhängig von anderen Diensten gehandhabt werden [31, Seite 294-295]. Der Autor differenziert dabei zwischen zwei Arten der Autonomie von Diensten, zum einen die Laufzeitautonomie (engl. Runtime Autonomy) und zum anderen die Entwurfszeitautonomie (engl. Design-Time Autonomy).
Die Laufzeitautonomie ist die Stufe, die ein Dienst Kontrolle über seine auszuführende Logik während der Ausführung hat. Sie soll eine Verbesserung in den bereits genannten Bereichen Zuverlässigkeit, Performance und Vorhersagbarkeit im Verhalten zeigen. Zusätzlich wird durch die Isolation eines einzelnen autonomen Dienstes auch die Sicherheit gestärkt, da weniger Angriffspunkte im Vergleich zu verteilten Diensten existieren, bei denen einzelne Funktionen eines Dienstes über mehrere Ausführungsinstanzen verteilt angeboten werden und gemeinsam den Dienst bilden.
Die Entwurfszeitautonomie wird als die Stufe beschrieben, bei der ein Dienst bereits in der Entwurfsphase eine externe Unabhängigkeit bildet. Die Abhängigkeit geschieht schon zu dem Zeitpunkt, an dem externe Programme oder Dienste auf den eigenen Dienst aufbauen oder in irgendeiner Form verwenden. Doch auch die Skalierbarkeit des Dienstes ist ein Faktor, der in die Entwurfszeitautonomie mit einfließt. Ein Dienst muss beispielsweise für

den Umfang der Nutzungslast ausgelegt und möglicherweise erweiterbar sein. Da das IoT aus einer dynamischen Anzahl Objekten besteht, ist dieses von Bedeutung. Eine weitere externe Abhängigkeit eines Dienstes im IoT ist im Bezug auf die Entwurfszeitautonomie die Möglichkeit zur Erweiterung der Ausführungsumgebung des Dienstes. In Anbetracht der Tatsache, dass das IoT aus einer Vielzahl heterogener Objekten besteht, muss immer eine kompatible Ausführungsumgebung für einen Dienst im IoT gewährleistet sein. Dabei müssen auch die vom Dienst verwendeten Ressourcen und der momentane Prozessstatus von der Ausführungsumgebung so verwaltet werden, dass sie über Migrationen hinweg konsistent verfügbar bleiben (vgl. Kapitel 2.3).

2.3 Positionen der autonomen Dienste

Velasquez et al. zeigen bereits erste Untersuchungen zur Reduktion von Latenzzeiten durch die Platzierung von Diensten im IoT [32]. Ihre Betrachtung fokussiert sich auf „Smart Cities“, bei der ein zusammenhängendes IoT die Grundlage zur Bereitstellung von Diensten bildet. Doch das globale IoT, in dem die verschiedene Wirtschaftssektoren separat Einfluss nehmen, kann anders als das Internet zurzeit eher als ein geografisch und technisch segmentiertes Netzwerk betrachtet werden. Es ist nicht wie das Internet ein global verbundenes Netzwerk, bei dem alle Teilnetzwerke miteinander interagieren können. Einzelne Segmente sind sowohl geografisch als auch funktional durch die Wirtschaftssektoren abgegrenzt. Möchte man darauf Anwendungsapplikationen aufbauen und die Wirtschaftssektoren vollständig miteinander verschmelzen, müssen zunächst weitere Analysen durchgeführt und die Segmentierung aufgehoben werden. Dies spiegelt auch den Zustand des Fog Computings wider, bei der eine Interoperabilität der Unterteilungen angestrebt wird [3]. Aus technischer Sicht grenzen die Segmente des IoT am Internet an und können jeweils als eigenständige Subnetzwerk betrachtet werden. Zur Überwindung der Inkompatibilität sorgen derzeit spezielle Edge-Gateways für die Kommunikation zwischen dem Internet und dem IoT [16]. Die Abbildung 2.1 skizziert ein solches segmentiertes IoT zusammen mit Internet. Die für das IoT angebotenen Dienste können sich an jeder beliebigen Position in diesem segmentiertem IoT oder im Internet befinden. Bekannte Cloud-Diensteanbieter (engl. Service Provider) wie Cisco Systems, Inc. und Amazon.com, Inc. versuchen die Transformation ihrer Cloud-Infrastruktur zu einer Bereitstellung und Anbindung von Objekte im IoT an ihre Applikationen mittels Fog Computing zu meistern [33, 34]. Doch nicht für alle IoT-Dienste ist es von Vorteil, wenn diese durch Diensteanbieter abgebildet werden. Je nach Arbeitsaufgabe der autonomen Dienste kann es zusätzlich essenziell sein, dass diese sich in bestimmten Bereichen eines IoT befinden. Anhand eines Beispiels für das Fog Computing lässt es sich einfach nachvollziehen.
Ein Collaborative-Aware Service könnte beispielsweise die einfache Aufgabe haben, die Sensordaten von den Objekten im IoT zu aggregieren (vgl. Kapitel 2.1). Anschließend soll er nach einer Auswertung an bestimmten Positionen im IoT wieder neue Aktionen von Aktoren auszulösen. Eine Möglichkeit wäre die Übertragung der Sensordaten durch das IoT über ein Gateway zu einem zuvor definierten zentralen Cloudserver eines Diensteanbieters, bei dem der Dienst platziert ist. Der Server verarbeitet die aggregierten Sensordaten. Die gewünschten Aktionen der Aktoren sendet er zurück an einen Vermittler, der als externe Schnittstelle für das IoT gilt. Schließlich verteilt der Vermittler die Aktionen an die rele-

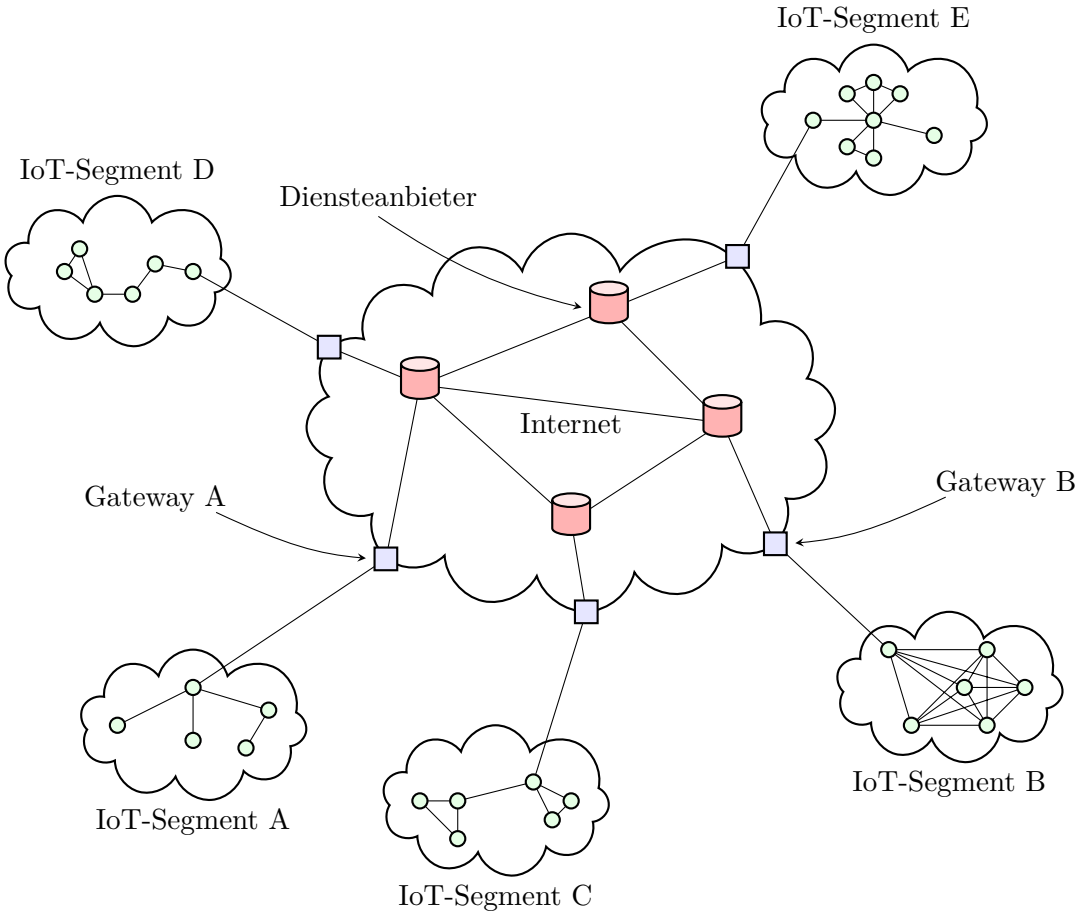


Abbildung 2.1: Das Internet und die derzeitigen IoT-Segmente als Skizze. Geografisch und technisch betrachtet grenzen die Segmente des IoT am Internet an. Sie sind jeweils eigenständige Netzwerke, die über spezielle Edge-Gateways mit dem globalen Internet verbunden sind.

vanten Objekte innerhalb des IoT, die abschließend die Aktionen ausführen.
Dabei sind allerdings zwei Punkte zu beachten. Der Server befindet sich in diesem Systemaufbau bei einem externen IoT-Diensteanbieter in einem weit entfernten Netzwerk, wodurch lange Übertragungslaufzeiten entstehen. Bilden die Objekte im IoT außerdem noch ein kabelloses Mesh-Netzwerk und sind nicht direkt an das Gateway angeschlossen, müssen die Sensordaten über mehrere Objekte weitergeleitet werden, zusätzliche Paketverluste und ein hoher Kommunikationsaufwand können entstehen. Um diese Paketverluste anwendungsseitig zu vermeiden, werden beispielsweise spezielle Anwendungsprotokolle wie das Constrained Application Protocol (CoAP) auf den Objekten im IoT benötigt, die bei einem Verlust der Sensordaten eine erneute Übertragung der Daten initiieren [3, 35].
Ein solches System mit Diensteanbieter erfordert zudem viele Anforderungen an die gesamte Infrastruktur. Es müssen leistungsstarke Gateways eingesetzt werden, die die massiven Daten aller im IoT befindlichen Objekte an die Diensteanbieter weiterleiten können. Sind viele Objekte im IoT vorhanden, die über ein Gateway mit den Diensten verbunden sind, wird das Gateway zum Flaschenhals des Systems. Eine Ausfallsicherheit zwischen Gateways

und Internet muss daher gewährleistet werden. Die Diensteanbieter müssen ferner eine kontinuierliche Bereitstellung ihrer Dienste sicherstellen. Ohne diese Verfügbarkeit der Dienste können Fehlfunktionen an den Objekten entstehen. Da die Datenverbindungen durch das Gateway geleitet werden, muss zusätzlich der maximale Durchsatz (engl. Throughput) im IoT in die Richtung der Gateways hoch sein.
Eine Lösung zur Minimierung dieser Anforderungen ist die Bereitstellung eines autonomen Dienstes innerhalb des IoT, wie es das Fog Computing vorsieht [15, 16, 18]. Dies kann entweder auf den bereits vorhandenen oder auf dedizierten Objekten im IoT geschehen, die durch eine erhöhte Leistung speziell für die Ausführung von Diensten ausgelegt sind. Diese Bereitstellung reduziert bereits die Auslastung an den Gateways zwischen dem IoT und dem Internet. Wird der autonome Dienst zusätzlich immer in der physisch unmittelbaren Umgebung der interagierenden Objekte im IoT bereitgestellt, werden die Übertragungswege weitestgehend minimiert und damit die Verlustrate, die durchschnittliche Round-Trip-Time (RTT) und der im gesamten IoT vorhandene Datenverkehr reduziert. Doch ist es für die Bereitstellung des autonomen Dienstes notwendig stetig auf Veränderungen am Systemzustand reagieren zu können. Verändern sich mit der Zeit die mit dem Dienst interagierenden Objekte, kann es dazu führen, dass der Dienst nicht mehr an der bisherigen Position benötigt wird und eine *Migration des autonomen Dienstes* schließlich unabdingbar wird. Die Migration transferiert den autonomen Dienst auf ein neues Objekt im IoT, das sich wieder in der physisch unmittelbaren Umgebung der interagierenden Objekte befindet [36]. Dadurch kann eine fortwährende Minimierung der Übertragungswege gewährleistet und die Interaktion zwischen autonomen Dienst und den interagierenden Objekten im IoT beschleunigt werden. Wird der autonome Dienst zwischen vielen Objekten oft migriert, spricht man von einer *Wanderung des Dienstes* durch das IoT.
Der Fokus dieser Masterarbeit liegt auf die Unterstützung zur Migration und Ausführung autonomer Dienste, wie sie in den vorangegangenen Kapiteln 2.1 und 2.2 beschrieben sind. Im Interesse einer langfristigen Kompatibilität sollen die autonomen Dienste im IoT sowohl eine vollständige Isolation der Programmlogik als auch der Datenressourcen vorweisen (vgl. [31, Tabelle 10.2]). Dies sichert eine vereinfachte Migration ohne externe Abhängigkeiten sowie angelehnt an der SOA die Modularität und eine einhergehende Wartbarkeit der Dienste [25]. Zusätzlich vereinfacht es die Erstellung einer einheitlichen Testumgebung für den in Kapitel 4 definierten Service Manager, der eine Plattform für die Ausführung und Migration solcher autonomen Dienste bietet.
In Kapitel 5 soll schließlich evaluiert werden, ob eine Migration durch den Service Manager an strategisch sinnvollen Positionen zur gewünschten Reduktion der Verlustrate und der durchschnittlichen RTT führt. Dazu werden im Folgenden sechs unterschiedliche Szenarien verglichen, die einen autonomen Dienst in allen Bereiche des IoT und des Internets platzieren. Einige Szenarien lassen den autonomen Dienst durch IoT-Segmente wandern, bei den restlichen befindet sich der autonome Dienst an fixierten Positionen. Diese Szenarien werden schließlich in zwei unterschiedlichen Systemaufbauten untersucht. Im ersten Systemaufbau befinden sich die anfragenden Objekte zufällig verteilt im IoT-Segment A. In einem zweiten Systemaufbau wird eine Verdoppelung der anfragenden Objekte durchgeführt, die in das IoT-Segment B zufällig verteilt hinzugefügt werden (vgl. Abbildung 2.1). Diese Szenarien stellen stufenweise unterschiedliche Anforderungen an die verschiedenen Komponenten. Die wandernden Dienste erfordern eine Rechenleistung auf den einzelnen Objekten im IoT, während die Szenarien mit interagierenden Gateways eine hohe Auslastung dieser darstellt, da alle Daten über diese Flaschenhälse übertragen werden müssen. Ein in der Praxis nicht

zu vernachlässigbarer Aspekt ist die Beachtung von Sicherheitsaspekten. Befindet sich ein angebotener Dienst am Gateway eines anderen IoT-Segments oder innerhalb des anderen IoT-Segments, so entstehen Problematiken mit Datenschutz und Zugriffsberechtigungen [3].

Szenario 1: Der Dienst wandert im IoT-Segment A.

Szenario 1a: Der Dienst wandert im IoT-Segment A mit dedizierten Dienstobjekten.

Szenario 2: Der Dienst befindet sich am Gateway A.

Szenario 3: Der Dienst befindet sich bei einem IoT-Dienstanbieter im Internet.

Szenario 4: Der Dienst befindet sich am Gateway des IoT-Segments B.

Szenario 5: Der Dienst wandert im IoT-Segment B.

2.4 Topologische Verfahren der Migration

Die in Kapitel 2.3 bereits definierten Szenarien zeigen, dass autonome Dienst durch das IoT wandern sollen. Wie bei diesen Wanderungen jedoch speziell die einzelnen Migrationen durchgeführt werden, ist noch nicht genau vorgegeben. Grundvoraussetzung für die Migrationen durch ein IoT-Segment und den in diesem Kapitel vorgestellten topologischen Verfahren ist ein durch die Objekte bereitgestelltes Mesh-Netzwerk. In einem Mesh-Netzwerk sind alle Netzwerkknoten mit einem oder mehreren anderen verbunden. Ist das Netzwerk beispielsweise kabellos, so können all diejenigen Objekte miteinander verbunden sein, die sich in der physischen Nähe zueinander befinden. Mithilfe von Routing-Protokollen werden auf dem Mesh-Netzwerk basierend Übertragungswege zwischen nicht direkt verbundenen Objekten ermöglicht. Informationen werden über die Netzwerkknoten bis zum Ziel weitergereicht [37].
Bei der Analyse der Kommunikation zur Durchführung einer Migration können verschiedene Verfahren eingesetzt werden. Diese wurden für die Bereitstellung von Diensten in Ad-hoc-Netzwerken entwickelt, können aber auch für das IoT adaptiert werden. Dabei haben alle im Folgenden vorgestellten Verfahren mehrere Dinge gemein. Zunächst werden alle Verfahren auf dem Objekt ausgeführt, auf dem auch der zu migrierende autonome Dienst ausführt wird und damit eine Unabhängigkeit zwischen allen Objekten gewährleistet. Während der Ausführung wird die Laufzeit in beliebige, aber konstante Intervalle quantisiert; zugleich werden die am Objekt ein- und ausgehenden Datenpakete der anderen mit ihm interagierenden Objekte beobachtet. Zum Ende eines jeden Intervalls wird auf Grundlage der beobachteten Menge der Datenpakete eine Entscheidung zur Migration getroffen und zwar entweder ob der Dienst weiter auf dem Objekt ausgeführt oder auf ein anderes Objekt im IoT migriert werden soll. Mit der Annahme, dass alle Objekte im IoT den TCP/IP-Stack implementieren, können die folgenden Verfahren zur Migration von autonomen Diensten verwendet werden. Auch eine Migration zwischen verschiedenen IoT-Segmenten ist mit diesen Methoden möglich. Wenn Adressen durch Verfahren wie NAT ersetzt werden, erfolgt zunächst eine Migration zum Gateway des Netzwerks, der die Ausführung des autonomen Dienstes zwischenzeitlich übernimmt.

2.4.1 LinkPull

Das *LinkPull*-Verfahren betrachtet alle ein- und ausgehenden Datenpakete an der Netzzugangsschicht (engl. Link Layer) [36]. Zur Laufzeit des autonomen Dienstes werden kontinuierlich die Paketheader der Netzzugangsschicht entpackt. Diese beinhalten bei eingehenden Datenpaketen die Informationen des letzten und bei ausgehenden Paketen die Informationen des nächsten Hops. So wird innerhalb eines Intervalls die Größe der gesamten gesendeten und empfangenen Kommunikationsdaten erhoben, und zwar aufgeteilt auf die direkten Nachbarn des dienstausführenden Objektes. Eine Migration am Ende des Intervalls wird schließlich zu dem direkten Nachbarn durchgeführt, über den die meisten Datenpakete übertragen wurden. In jedem Intervall findet somit jeweils eine Migration über einen einzelnen Hop im IoT statt.

2.4.2 PeerPull

Bei dem *PeerPull*-Verfahren werden die Datenpakete auf der Internetschicht (engl. Internet Layer) betrachtet [36]. Ähnlich dem LinkPull-Verfahren wird innerhalb eines Intervalls die Größe der gesamten gesendeten und empfangenen Kommunikationsdaten erhoben. Allerdings teilt sich die Gesamtgröße nicht auf die direkten Nachbarn auf, vielmehr findet eine Aufgliederung der Gesamtgröße auf die Netzwerkadressen statt, mit denen der autonome Dienst kommuniziert. Bei eingehenden Datenpaketen sind es die IP-Adressen der Quellen und bei ausgehenden Datenpaketen entsprechend die IP-Adressen der Ziele. Am Ende des Intervalls wird der Dienst schließlich über mehrere Hops hinweg zu dem Objekt migriert, mit dem der bisherige Dienst am meisten kommuniziert hat.

2.4.3 NetCluster

Das *NetCluster* ist ein einfaches Verfahren, bei dem die autonomen Dienste immer in das aktivste Cluster migriert werden [36]. Ein Cluster bildet sich dabei immer aus den Objekten, die die Datenpakete über den selben 1-Hop-Nachbarn des derzeit dienstausführenden Objektes übertragen und sich somit ihren Übertragungsweg über ihn teilen. Anhand der während des Intervalls erhobenen Größe der gesamten gesendeten und empfangenen Kommunikationsdaten wird am Ende jeden Intervalls der aktivste Cluster berechnet. Aus diesem Cluster wird zufällig ein Objekt als neues dienstausführendes Objekt ausgewählt und abschließend eine Migration zu diesem durchgeführt. Wie bei dem LinkPull-Verfahren ist dazu eine Extraktion des Paketheaders der Netzzugangsschicht aller Datenpakete notwendig.

2.4.4 TopoCenter(n)

Einen anderen Ansatz verfolgt das Verfahren *TopoCenter(n)* [36]. Einem durch das Mesh-Netzwerk übertragenes Datenpaket werden zu Beginn die Verbindungsinformationen des sendenden Objektes angehängt. Diese beinhalten die n-Hop-Nachbarschaft des Objektes. Im einfachsten Fall ist $n = 1$. Bei jeder Weiterleitung auf dem Weg zum autonomen Dienst durch das Mesh-Netzwerk werden die Informationen zur Topologie weiter akkumuliert. Jedes Objekt im IoT entpackt dazu das ausgehende Datenpaket, speichert die zuvor angehängten Nachbarschaften ab und hängt ebenfalls seine n-Hop-Nachbarschaft an das Datenpaket an.

Schließlich wird das Datenpaket zum nächsten Objekt in die Richtung des Dienstes weitergeleitet. Alle Objekte können so einen Teil des Mesh-Netzwerkes aufbauen, in dem sie sich befinden. Das dienstausführende Objekt speichert zusätzlich zu allen Nachbarschaften intern die Größe der gesendeten und empfangenen Kommunikationsdaten.
Am Ende eines Intervalls berechnen alle Objekte die kürzesten Pfade zu allen anderen bekannten Objekten im Mesh-Netzwerk. Das dienstausführende Objekt berechnet zusätzlich auf Grundlage dieser kürzesten Pfade die anfallenden Migrationskosten zu allen anderen Objekten und auf Basis der gesammelten Größe der Kommunikationsdaten die im nächsten Intervall voraussichtlichen Kommunikationskosten zu allen Objekten. Diese beiden Variablen sollen möglichst minimiert werden, weshalb die Migration zu dem Objekt durchgeführt wird, bei dem beide Variablen in der Berechnung minimal sind.

2.4.5 Globale Informationen

Die hier abschließend betrachtete Alternative unterscheidet sich von den anderen Verfahren zur Migration von autonomen Diensten darin, dass sie *globale Informationen* über das zu betrachtende System kennt. Dieser Informationsgehalt kann in realen Umgebungen wie dem IoT oder auch dem Internet bei der Berechnung eines geeigneten Objekts zur Ausführung eines autonomen Dienstes nicht vorausgesetzt werden, da das IoT beispielsweise aus unzähligen heterogenen Objekten besteht und allein aufgrund der Größe die Informationen nicht bereitgestellt werden können. In geschlossenen Emulationsumgebungen, wie das in Kapitel 3.2 vorgestellte Münster Internet of Things (MIOT)-Testbed oder das in Kapitel 5.1 beschriebene Software-Defined Networking (SDN), ist es jedoch durchaus möglich, da sowohl die Anzahl der Objekte als auch die Anzahl der Verbindungen untereinander in einem überschaubaren Rahmen sind. Bereits für die Erstellung des Virtual Network Emulators ist die Kenntnis über Informationen aller Objekte und den Verbindungen zwischen in den Objekten im Mesh-Netzwerk notwendig, doch dazu später im Kapitel 5.1 mehr.
Das Verfahren mit globalen Informationen kann als eine Abwandlung des TopoCenter(n)- und des PeerPull-Verfahren angesehen werden. Innerhalb eines Intervalls werden wie bei diesen Verfahren die Größen der gesendeten und empfangenen Datenpakete zusammen mit der IP-Adresse des mit dem autonome Dienst kommunizierenden Objektes gespeichert. Am Ende eines Intervalls wird schließlich eine Berechnung auf Basis des kürzesten Pfades und der aufgenommenen Datenpaketgrößen durchgeführt. So kann ein zentrales Objekt zur Ausführung des autonomen Dienstes gefunden werden, bei dem die vorausberechneten Kommunikationskosten minimal sind.

KAPITEL 3

Topologische Charakteristika und das Internet der Dinge

Das IoT besteht aus einer unzähligen Anzahl heterogener Objekten [2]. Da die Objekte mittels verschiedenster kabelgebundener und kabelloser Übertragungsmedien miteinander verbunden sein können, kann eine vielseitige Zusammensetzung verschiedener Topologien innerhalb des IoT angenommen werden. Diese zusammengesetzte Topologie des IoT soll in diesem Kapitel als ein Graph $G = (V, E)$ betrachtet werden. Die Menge der Objekte im IoT wird in dem Graphen G durch die Menge der Knoten V und die vorhandenen Verbindungen zwischen den Objekten als die Menge der Kanten E repräsentiert.
Die in Kapitel 2.4 beschriebenen topologischen Verfahren zeigen bereits die Migration einer einzelnen Instanz eines autonomen Dienstes im IoT, die zwischen den Objekten innerhalb des IoT wandert. Die Betrachtung der optimalen Positionierung mehrerer verteilt ausgeführter Instanzen eines einzelnen autonomen Dienstes im IoT lässt sich auf das grundlegende p-Median-Problem in einem Graphen zurückführen. Angewandt auf das vorliegende Szenario wird bei dem p-Median-Problem versucht, p verschiedene Instanzen eines autonome Dienstes so im IoT zu positionieren, dass der durchschnittliche Abstand zwischen den anfragenden Objekten und der zu ihnen nächstgelegenen Instanz minimiert wird. Die Minimierung gewichtet sich dabei auf Grundlage der Nachfragegröße der anfragenden Objekte, die bei den topologischen Verfahren in Kapitel 2.4 beschrieben ist. Auf allgemeine Graphen ohne bekannten Aufbau der Topologie, wie es in dem Szenario des IoT der Fall ist, kann das p-Median-Problem allerdings nicht angewendet werden, es ist $\mathcal{NP}$-schwer [38, Kapitel 2]. Der Fokus dieser Arbeit richtet sich daher auf die Bereitstellung einer einzelnen Instanz eines autonomen Dienstes im IoT. Für die Bereitstellung mehrerer Instanzen eines autonomen Dienstes in einem segmentierten, verteilten IoT wäre hier die Bereitstellung in den einzelnen IoT-Segmenten durch die Bildung von Mikro-Fogs möglich, wie es bereits in Kapitel 1 beschrieben ist.
Zur Ausführung der topologischen Verfahren werden zunächst die bewertbaren Systemeigenschaften aus dem zugrundeliegenden IoT benötigt. Dies sind statistischen Daten, die sogenannten *Leistungsmetriken*, die entweder zuvor oder von dem in Kapitel 4 definierten Service Manager während der Laufzeit des autonomen Dienstes im IoT erhoben werden. Darauf folgt die Auswahl und die Aufnahme der geeigneten Leistungsmetriken aus dem MI-

OT-Testbed zur Emulation der Szenarien aus Kapitel 2.3. Wie diese dem später definiertem Service Manager für die Emulation bereit gestellt werden, wird durch die Definition einer Datenstruktur gezeigt. Abschließend werden die aus dem MIOT-Testbed aufgenommenen Leistungsmetriken ausgewertet und miteinander verglichen.

3.1 Leistungsmetriken im Internet der Dinge

Die nachstehenden Leistungsmetriken bilden die Grundlage für mögliche Migrationen eines autonomen Dienstes im IoT. Sie lassen sich anhand ihres Aufnahmezeitpunktes in zwei Gruppen einteilen. Die Leistungsmetriken der erste Gruppe werden vor der eigentlichen Ausführung eines autonomen Dienstes bei der Vermessung des Systems als eine Momentaufnahme ermittelt (vgl. Kapitel 3.2). Die Ermittlung der Leistungsmetriken der zweite Gruppe geschieht kontinuierlich während der eigentlichen Prozessausführung des autonomen Dienstes und des Service Managers aus Kapitel 4.

3.1.1 Minimaler Hop Count

Die einfachste Leistungsmetrik in einem Netzwerk ist der *Hop Count*. Er gibt an über wie viele routingfähige Netzwerkgeräte eine Nachricht durch das Netzwerk geleitet wird bis sie zum Ziel gelangt. Der Hop Count wird im IP-Header eines Netzwerkpakets zu Beginn auf einen vordefinierten Wert festgelegt und von jedem routingfähigen Netzwerkgerät auf seinem Weg herunter gezählt. Im IPv4-Header ist dafür das Feld *Time to live (TTL)* und im IPv6-Header das Feld *hop limit* vorgesehen. Damit Netzwerkpakete nicht unendlich lange durch ein Netzwerk weitergeleitet werden, wird die weitere Weiterleitung bei einem Hop Count Wert von 0 unterbunden.
Sind die Routingpfade durch das Netzwerk durch ein statisches Routing bereits im Voraus bekannt, können die Hop Count Werte für alle Verbindungen zwischen allen Geräten bereits vorberechnet werden. Diese Leistungsmetrik ist jedoch sehr ungenau, um die optimale Länge einer Route zwischen dem Start- und Zielobjekt zu bestimmen, da alle Verbindungen zwischen zwei Geräten gleich behandelt und die Geschwindigkeit, Auslastung, Zuverlässigkeit und die Latenzen auf den Leitungen zwischen den einzelnen Hops hier nicht berücksichtigt werden.

3.1.2 Durchsatz

Der *Durchsatz (engl. Throughput)* einer Verbindung zwischen zwei Netzwerkgeräten ist im Sinne dieser Masterarbeit die maximal erreichbare Datenrate während der erfolgreichen Übertragung einer Nachricht über einen Kommunikationskanal. Im IoT ist der Kommunikationskanal die Verbindung zwischen zwei Objekten, welche sowohl kabelgebunden als auch kabellos erfolgen kann. Die Einheit des gemessenen Durchsatzes ist Bits pro Sekunde ($bits/s$). Fällt die Betrachtung auf die Menge der übertragbaren Daten, die als reine Nutzdaten auf Applikationsebene verfügbar sind, wird der Durchsatz als Datendurchsatz (engl. Goodput) spezifiziert. Dabei ist zu beachten, dass sowohl die Mehrkosten (engl. Overhead) der Protokolle als auch die erneute Datenpaketübertragung im Fehlerfall in der Angabe zum Datendurchsatz ausgeschlossen werden.

Da der Durchsatz eine entscheidende Messgröße für eine Verbindung ist und die obere Schranke für die Datenübertragung über einen Kommunikationskanal darstellt, ist es ein wichtiges topologisches Charakteristika im IoT. Aus diesem Grund erfolgt im Laufe dieser Arbeit eine detaillierte Messaufnahme, die im nachstehenden Kapitel 3.2 beschrieben wird. Bei der Messaufnahme ist der Durchsatz abhängig von der Größe der gesendeten Nachrichten, sodass diese sich über alle Messungen hinweg für eine Vergleichbarkeit nicht verändern sollte [39].

3.1.3 Expected Transmission Count (ETX)

Eine weitere Leistungsmetrik ist die erwartete Übertragungsrate (engl. Expected Transmission Count (ETX)) nach [40]. ETX ist eine unidirektionale Metrik, die separat für beide Richtungen einer kabellosen Verbindung zwischen zwei Netzwerkgeräten den Effekt der Verlustrate beider Verbindungen beschreibt. Sie wurde entwickelt, um eine hohe Güte in der Übertragung auf verlustbehafteten Verbindungen zu erreichen und so den Durchsatz im Netzwerk zu erhöhen. Zwar unterstützen die Mechanismen von IEEE 802.11 bereits den erneuten Versand einer Nachrichten bei Verlust, doch das auf Kosten des Durchsatzes. Mit dieser Metrik sollen bereits während der Suche einer geeigneten Route die besten Zwischenverbindungen durch ein Netzwerk gewählt werden, sodass die Häufigkeit der erneuten Versendungen minimiert wird.
Die ETX einer Verbindung ist definiert als die prognostizierte Anzahl von Datenübertragungen samt Wiederholungen, um ein Datenpaket über eine Verbindung zu senden. Dabei liegt der Wert von ETX zwischen 1 und unendlich. Je näher der Wert an 1 ist, desto besser ist die Verbindung. Auf einer Route ist ETX die Summe der einzelnen ETX-Werte der beteiligten Verbindungen. Die Berechnung von ETX einer Verbindung erfolgt auf Basis der zwei gemessenen Wahrscheinlichkeiten nämlich, dass ein Paket vom Sender erfolgreich beim Empfänger ankommt d_f und dass die Bestätigung des Pakets vom Empfänger am Sender ankommt d_r. Die erwartete Gesamtwahrscheinlichkeit für den erfolgreichen Abschluss der Sendung und Bestätigung eines Pakets ist somit $d_f \times d_r$. Da jeder Versandversuch eines Pakets als Bernoulli-verteilt angesehen werden kann, liegt der Erwartungswert für ETX bei der folgenden Gleichung 3.1:

$$\mathrm{ETX} = \frac{1}{d_f \times d_r} \tag{3.1}$$

Durch genaue Messungen der Verbindungsverluste soll ETX präzise zwischen unterschiedlichen Routen entscheiden können. De Couto et al. präsentieren dazu ein Messverfahren, bei dem innerhalb eines Zeitfensters spezielle Pakete mit einer festgelegten Größe zur Verbindungsuntersuchung zwischen den Netzwerkgeräten als Broadcast-Nachrichten gesendet werden. Durch die Übertragung als Broadcast-Nachrichten werden diese durch IEEE 802.11 nicht bestätigt oder erneut versandt [40].
Da ETX eine genauen Repräsentation der einzelnen Netzwerkverbindungen im Mesh-Netzwerk widerspiegelt, wird es als eine grundlegende Leistungsmetrik für das zu untersuchende IoT aufgenommen. Die genaue Messung wird dazu in Kapitel 3.2 beschrieben.

3.1.4 Expected Transmission Time (ETT)

Mit der erwartete Übertragungszeit (engl. Expected Transmission Time (ETT)) zeigen die Autoren in [41] eine Metrik für kabellose Mesh-Netzwerke, in denen mehreren, parallele Funkkanäle zum Einsatz kommen. Die Metrik ist eine Kombination aus dem Durchsatz und der ETX für ein Paket, das über eine Verbindung übertragen wird, wobei die ETT als eine an den Durchsatz angepasste ETX bezeichnet wird. Bei der folgenden Gleichung 3.2 ist S als die Größe der Pakete und B als der Durchsatz einer Verbindung definiert:

$$\mathrm{ETT} = \mathrm{ETX} \times \frac{S}{B} \tag{3.2}$$

So zeigt sich hier die Anpassung, dass sich bei einer Größenveränderung individueller Pakete ebenfalls die ETT ändert. Wie bei der ETX wird die ETT einer Route durch die Summe der einzelnen ETT-Werte der beteiligten Verbindungen berechnet. Die Besonderheit liegt hier jedoch in einer Erweiterung, bei der zusätzlich zur Bildung von Summen auch die Interferenzen zwischen zwei Verbindungen zum Ausdruck kommen. Bei einer Route, bei der zwei Verbindungen in der Nähe den gleichen Funkkanal verwenden, kann schließlich nur eine Verbindung zur gleichen Zeit senden. Daher werden bei der Auswahl von Routen durch das Netzwerk die Verbindungen bevorzugt, bei denen eine hohe Vielfalt an Funkkanälen vorkommt.
Die ETT bietet sich als Erweiterung zu ETX bei der Auswahl von Routen in Mesh-Netzwerken mit mehreren, parallelen Funkkanälen an. Vorausschauend auf die Emulation des MIOT-Testbeds durch ein SDN in Kapitel 5.1 ist die Metrik jedoch nicht weiter von Vorteil, da die parallelen Funkkanäle durch das SDN nicht emuliert werden können und die ETT daher bei dem Aufbau des Netzwerkes sowie der Bildung von Routen nicht weiter verwendet werden kann.

3.1.5 Latenzzeit

Eine essenzielle Kennzahl während der Übertragung von Daten durch ein Netzwerk ist die Latenzzeit und im Speziellen die RTT. Um das IoT in eine greifbare Nähe und der Interaktion mit Menschen bringen zu können, müssen die Latenzzeiten zwischen dem Auslösen eines Ereignisses bis zur Ausführung eines Prozesses weitestgehend minimiert werden [42]. Fettweis beschreibt dazu das Taktile Internet (engl. Tactile Internet) [43, 44]. Dienste, wie die im IoT (vgl. Kapitel 2.1), werden als Echtzeit (engl. real-time) definiert, wenn die Antwortzeit der Kommunikation geringer ist als die Laufzeitkonstanten der Anwendung selber und somit die durch Kommunikation und internen Berechnungen entstandene Verzögerungen vernachlässigbar sind. Die Laufzeitkonstanten verschiedener Anwendungen teilt Fettweis dabei in vier verschiedene Typen mit absteigender Größe ein: muskulär mit 1 s, audio mit 100 ms, visuell mit 10 ms und taktil mit 1 ms [42]. Erst wenn der taktile Typ bei Anwendungen respektive den Diensten mit einer RTT von 1 ms oder weniger erreicht ist, spricht man vom taktilen Internet. Dieses gilt es durch neue Technologien im Bereich der Datenkommunikation sukzessiv zu erreichen. So gibt es bereits bei den kabellosen Kommunikationstechnologien von 5G, dem Nachfolger von LTE, die Bestrebungen, eine Ende-zu-Ende-Latenzzeit von 1 ms zu erreichen [42].
Dienste im IoT, die diesen Typen gerecht werden wollen, sollten also diese Laufzeitkon-

stanten in ihrer Kommunikation einhalten. Strategien, wie die Migration der Dienste nach Kapitel 2, können bei der Verwirklichung solcher Anforderungen helfen.

3.1.6 Eigenschaften der Objekte

Neben den topologischen Charakteristika, bei denen die Kommunikation zwischen den Objekten im IoT zum Ausdruck kommt, haben die Objekte selber weitere Eigenschaften, die für die Migration eines Dienstes ebenso relevant sind. Durch die eingeschränkt vorhandenen Ressourcen der Objekte werden weitere Kriterien geschaffen, auf deren Basis Entscheidungen für die auf dem Objekt laufenden Dienste getroffen werden müssen [2, 3]. Die Ressourcen können beispielsweise die verfügbare Energie, Prozessorleistung und die Kapazitäten von Arbeits- und Datenspeicher sein. Beansprucht ein Dienst auf einem Objekt im IoT einen großen Anteil einer bestimmten Ressource, könnte der Dienst zu einer anderen Position migriert werden, bei der die Gesamtkapazität dieser Ressource höher ist. Naheliegend ist als Alternative die Duplikation des Dienstes auf mehrere parallel laufender Objekte im IoT. Mehrere Instanzen des gleichen Dienstes würden sich dabei die Aufgaben der anfragenden Clients teilen. Dadurch entstehen allerdings gleich mehrere nichttriviale Probleme, die es zu lösen gilt. Zunächst ist eine Lastverteilung (engl. load balancing) notwendig, sodass eine gleichmäßige Aufteilung der Clients auf die verschiedenen Instanzen des Dienstes im IoT erfolgt. Dies entspricht der Unterteilung in einzelne kleine Mikro-Fogs [3]. Weiterhin wird durch die Duplikation eine Abhängigkeit der Instanzen des Dienstes untereinander gebildet. Da diese Instanzen der Dienste im IoT die Daten der verschiedenen Objekte bearbeiten, kann eine inkonsistente Datenhaltung zwischen den dienstausführenden Objekten im IoT entstehen, die entweder eine Synchronisation zwischen den Instanzen des Dienstes oder eine Interaktion mit einer entfernten Cloud-Infrastruktur erfordern [18]. Die dabei entstehende Kommunikation führt schließlich zu einem nicht zu vernachlässigbaren Anteil an Mehrkosten. Die Kommunikation bei der Synchronisierung einer solchen verteilten Anwendung mag auf der lokalen Ebene noch zuverlässig funktionieren, ist jedoch auf einer globalen Ebene, wie die des IoT, nur sehr schwer zu realisieren [45, Kapitel 1.2]. Letzten Endes muss über die Entfernung der anfragenden Objekte zu dem dienstausführenden Objekt und der Ressourcenknappheit entschieden werden, ob eher eine Migration oder eine Duplikation für einen Dienst während der Ausführung sinnvoller ist.

3.2 Analyse der Leistungsmetriken im MIOT-Testbed

Die Untersuchungen der in Kapitel 2 beschriebenen Migration autonomer Dienste im IoT soll mithilfe einer Emulation des MIOT-Testbeds an der Westfälischen Wilhelms-Universität in Münster geschehen. Das MIOT-Testbed besteht aus derzeit 60 sogenannten MIOT-Nodes zur Forschung an kabellosen Mesh-Netzwerken und dem kabelloses Sensornetzwerk (engl. Wireless Sensor Netzwork (WSN)). Die Nodes und das MIOT-Testbed sind dabei Äquivalente zu den Objekten im IoT. Für die nachstehenden Untersuchungen enthält jede dieser Nodes ein eingebettetes System und bis zu drei IEEE 802.11 Netzwerkkarten. Das Betriebssystem des eingebetteten Systems basiert auf der Linux-Distribution Ubuntu. Durch die drei Netzwerkkarten in den MIOT-Nodes bilden sie zusammen ein gemeinsames Mesh-Netzwerk, das in drei IP-Subnetze aufgeteilt ist. Über unterschiedliche Frequenzen spannen sie dabei

drei physikalisch voneinander getrennte IEEE 802.11 Netzwerke im Frequenzbereich um 2,4 Ghz auf [46]. Dadurch resultiert eine variable Erreichbarkeit zwischen den im gesamten Campus verteilten MIOT-Nodes unter anderem durch die räumliche Entfernung zwischen den einzelnen Nodes als auch durch die Ausrichtung der Antennen.
Aufgrund von kontinuierlichen Erweiterungen und laufenden Umbaumaßnahmen sind jedoch nicht immer alle Nodes des MIOT-Testbeds konstant verfügbar. Dadurch können die Messaufbauten zwischen den zu untersuchenden Szenarien aus Kapitel 2.3 bei der Emulation im MIOT-Testbed variieren. Zur Lösung dieses Problems soll das MIOT-Testbed zusätzlich mithilfe von SDN durch ein Virtual Network Emulator emuliert werden [47]. Die Emulation schafft außerdem die Möglichkeit, die Anforderungen der verschiedenen Szenarien einfacher abzubilden (vgl. Abbildung 2.1), sodass die Migration autonomer Dienste auch innerhalb einer globalen Größe eines IoT emuliert werden kann. Die Untersuchung dazu ist in Kapitel 5.1 zu finden.
Für eine realitätsnahe Emulation durch ein SDN ist allerdings die Analyse und Aufnahme der gesamten Netzwerktopologie des MIOT-Testbeds anhand ausgewählter Leistungsmetriken aus Kapitel 3.1 notwendig. Durch die Aufnahme sollen schließlich die Objekte und all ihre Verbindungen untereinander genau definiert werden. Diese Art der Analyse der Netzwerktopologie kann zwar in einer realen Umgebung mit vielen Objekten im IoT aufgrund der Größe nicht durchgeführt werden, jedoch vereinfacht sie die Untersuchungen der Szenarien erheblich. Neben der Möglichkeit zur Emulation des MIOT-Testbeds ist mit diesem Vorgehen auch die Realisierung des in Kapitel 2.4 vorgestellte topologischen Verfahren zur Migration autonomer Dienste im IoT auf Grundlage der globalen Informationen möglich. Große Anpassungen bei der Kommunikation zwischen den Clients und dem autonomen Dienstes, wie das Übertragen der Nachbarschaft der Objekte im IoT, wie sie bei TopoCenter(n) gemacht werden müssen, sind hier nicht notwendig (vgl. Kapitel 2.4.4).
Zunächst ist zu analysieren, welche Leistungsmetriken aus Kapitel 3.1 sich für die Darstellung des MIOT-Testbeds durch ein SDN eignen. Bei der Betrachtung einer einzelnen Verbindung zwischen zwei Objekten im IoT sind vor allem zwei Leistungsmetriken maßgebend, die ETX und der Durchsatz. Durch ETX lassen sich zum einen die Paketverluste bei Verbindungen emulieren. Zum anderen bestimmt der Durchsatz die obere Schranke der Datenmenge, die über eine Verbindung pro Zeiteinheit übertragen werden kann. Diese beiden Leistungsmetriken werden auch von dem Virtual Network Emulator in Kapitel 5.1 zur Erstellung einer in ihren Ressourcen eingeschränkten Verbindung benötigt. Die Latenzzeit respektive die RTT ist hingegen abhängig von einigen Störfaktoren. Dies können beispielsweise die momentane Auslastung des autonomen Dienstes auf den Objekten sein oder auch die momentane Auslastung einer Verbindung zwischen zwei Objekten. Sie eignet sich daher nicht direkt als grundlegende Leistungsmetrik zur Bereitstellung eines SDNs. Doch die RTT kann bei der Evaluation in Kapitel 5 als eine variable Leistungsmetrik verwendet werden, um die Szenarien untereinander zu vergleichen (vgl. Kapitel 5.3). Die weiteren Leistungsmetriken, wie die Eigenschaften der Objekte, können darüber hinaus als Entscheidungskriterien für weitere Aktionen bei der Ausführung des autonomen Dienstes in Verwendung treten (vgl. Kapitel 4.2.4).
Die Autoren Das et al. untersuchten bereits einige Leistungsmetriken in Mesh-Netzwerken. Dabei präsentieren sie vier für diese Analyse relevante Ergebnisse, die zur Bildung von Rahmenbedingungen für die nachstehende Aufnahme beitragen [39]:

- Die Qualität aller Verbindungen innerhalb eines Netzwerkes variiert über die Zeit unabhängig von ihrer Güte.
- Typische Verbindungen verändern die Qualität allerdings nicht innerhalb weniger Sekunden.
- Die aufgenommenen Metriken hängen von dem Verfahren der Messaufnahme ab.
- Die Leistungsmetriken werden durch Hintergrundübertragungen externer Einflüsse negativ beeinflusst.

Im Detail zeigen die Autoren, dass die Stabilitäten der Metriken über den Tag verteilt signifikante Unterschiede aufweisen, die in externen Einflüssen begründet sind. Während der Arbeitszeiten werden die Verbindungen durch Bewegungen von Personen gestört, in der Nacht hingegen sind sie weitestgehend stabil. Dabei zeigt die Leistungsmetrik Durchsatz über mehreren Messungen hinweg zusätzlich eine viel höhere Stabilität gegenüber der ETX auf. Einen hohen Einfluss auf die Leistungsmetrik ETX soll zudem das ausgewählte Messverfahren haben. Zur Erzielung genauerer Messergebnisse empfehlen die Autoren bei der Messaufnahme von ETX beispielsweise die Verwendung von Unicast- anstelle von Broadcast-Messungen.
Auf dieser Grundlage aufbauend sollen nun die beiden Leistungsmetriken Durchsatz und ETX im MIOT-Testbed zur Bereitstellung eines SDNs aufgenommen werden. Dies geschieht innerhalb des MIOT-Testbeds mittels zwei verschiedener Programme, die über das Webinterface, das der Verwaltung von Experimenten auf dem MIOT-Testbed dient, auf den einzelnen MIOT-Nodes initialisiert und ausgeführt werden können. Bei der Messung des Durchsatzes kommt das Netzwerkdurchsatz-Messtool *iperf* zum Einsatz [48]. Über eine Client-Server-Architektur sendet es als Anwendung über TCP oder UDP Datenpakete und misst so den Durchsatz zwischen zwei Netzwerkgeräten. Alle verfügbaren MIOT-Nodes stellen zunächst parallel einen iperf-Server bereit. Darauf folgt eine sequenzielle Messung zwischen diesen Nodes über die iperf-Clients. Da die drei IEEE 802.11 Netzwerkkarten jeweils ein eigenes IP-Subnetz aufbauen, soll auch in allen drei Teilnetzen des Mesh-Netzwerk jeweils eine Messung durchgeführt werden. Abschließend werden alle Messergebnisse über das MIOT-Testbed aggregiert. Da die MIOT-Nodes standardmäßig keine Routing-Informationen besitzen, können die iperf-Messungen nur zwischen den Clients und Servern durchgeführt werden, die sich auch in einer unmittelbaren Nähe befinden. Der folgende Quellcode 3.1 beschreibt eine Messdurchführung mittels iperf als ein Experiment über das Webinterface des MIOT-Testbeds. Der in Zeile 2 gestartete iperf-Server wird zur finalen Terminierung des Experiments nach 85.500 s (23,75 h) automatisch gestoppt, darauf folgt die Zusammenfassung der Kommandozeilenausgaben mithilfe des dem MIOT-Testbed bereitgestellten Python-Skripts `save-stdout.py`. Der iperf-Client in Zeile 4 wird sequenziell zwischen allen Nodes und allen Netzwerkkarten durchgeführt. Eine sequenzielle Aufrufreihenfolge aller Nodes verhindert, dass sich diese bei ihrer Untersuchung im MIOT-Testbed gegenseitig beeinflussen. Würden MIOT-Nodes in der gleichen Nachbarschaft zur gleichen Zeit eine Messung durchführen, könnten die Netzwerke belastet sein und die Messungen signifikant beeinflusst werden. Aufgrund der Stabilität der Leistungsmetrik Durchsatz ist hier eine Messdauer von 60 s ausreichend [39]. Nach Durchlauf aller Messungen wird auch die Kommandozeilenausgabe mithilfe des Python-Skripts `iperf-results.py` aggregiert.

```
1  true;
2  iperf -s
3  sleep 60
4  iperf -c {all.*wireless} -t 60
```

Quellcode 3.1: MIOT-Experiment zur Messung des Durchsatzes im MIOT-Testbed mittels iperf

Die Messung der ETX zwischen allen MIOT-Nodes erfolgt mithilfe des Diagnosewerkzeugs *ping* [49]. Ping ermöglicht eine einfache Überprüfung der Erreichbarkeit der MIOT-Nodes untereinander durch den Versand sogenannter ICMP-Pakete, die Bestandteil von IPv4 sind. Ist IPv6 im Einsatz, werden hier ICMPv6-Pakete verwendet. Ähnlich dem Vorgehen zur Ermittlung des Durchsatzes werden Unicast-Nachrichten zwischen allen Nodes gesendet. Da ping bereits Bestandteil von IPv4 ist, muss keine spezielle Anwendung gestartet werden. Die Nodes, die sich in der Nähe befinden, antworten von allein auf die Nachrichten. Der Quellcode 3.2 zeigt die Messdurchführung mittels ping als ein Experiment über das Webinterface des MIOT-Testbeds.

```
1  true;
2  sleep 60
3  sudo ping -q -B -c 10000 -s 484 -l 3 -p 0f1e2d3c4b5a6978 -f {all.*wireless}
```

Quellcode 3.2: MIOT-Experiment zur Messung der ETX im MIOT-Testbed mittels ping

Der ping-Befehl wird dabei mit verschiedenen Parametern gestartet, die im folgenden beschrieben werden:

- *-q:* Es wird nur die finale Zusammenfassung auf der Kommandozeile ausgegeben.
- *-B:* Die zu Beginn ausgewählte Quelladresse wird über den gesamten Verlauf von ping beibehalten.
- *-c 10000:* Pro Verbindung werden jeweils 10.000 ICMP-Request-Pakete gesendet. Durch die hohe Wiederholungszahl wird eine erhöhte Genauigkeit über einen längeren Messzeitraum erlangt.
- *-s 484:* Die Datenmenge des ICMP-Pakets beträgt 484 Bytes, in der Standardkonfiguration sind es 56 Bytes. Zusammen mit dem 8 Bytes großen ICMP-Header und den 20 Bytes des IPv4-Headers wird die MTU auf 512 Bytes abgestimmt. Sehr schwache Verbindungen sollen so zuverlässig erkannt werden.
- *-l 3:* Es werden immer drei Nachrichten parallel ausgesendet, ohne dass bereits eine Antwort zur vorherigen Nachricht erhalten wurde (ergänzt sich mit dem Flag *-f*).
- *-p 0f1e2d3c4b5a6978:* Das Pattern füllt das Paket mit bis zu 16 Bytes im Datenbereich auf, sodass das Datenfeld nicht nur aus Nullen besteht.
- *-f:* Die ICMP-Requests werden sequenziell sofort nach Erhalt der Antwort zur letzten Nachricht ausgesendet. Falls die Antworten zu langsam sind, wird die Rate auf mindestens 100 pro Sekunde festgelegt.

Schließlich wird die Kommandozeilenausgabe ebenfalls mittels des Python-Skripts `savestdout.py` zusammengetragen. Während der langanhaltenden Messung des Experiments von neun Tage wurden die einzelnen Messungen zu verschiedenen Tageszeiten erhoben. Da externe Einflüsse einen Einfluss auf die ETX nehmen, können Unterschiede in den Aufnahmen entstanden sein [39]. Allerdings ist eine Aufnahme ohne diese Schwankungen

nicht möglich, weshalb sie für die weitere Verarbeitung hingenommen werden.

3.3 Datenstruktur der topologischen Charakteristika

Die in Kapitel 3.2 im MIOT-Testbed aufgenommenen Leistungsmetriken Durchsatz und ETX werden den beiden Programmen iperf und ping als Rohdaten entnommen. Da der Service Manager in Kapitel 4 die Kalkulationen zur Findung eines besseren Knotens auf Grundlage der Leistungsmetriken durchführt und das SDN in Kapitel 5.1 aus diesen Rohdaten erzeugt wird, muss eine effiziente Datenstruktur erstellt werden, mit der diese Leistungsmetriken zwischen den Knoten transportiert, strukturiert verwaltet und bereitstellt werden können.
Da das IoT durch einen Graphen $G = (V, E)$ mit V als Menge der Knoten respektive Objekten im IoT und E als Menge der Kanten respektive Verbindungen zwischen den Objekten repräsentiert werden kann, sollen die erhobenen Rohdaten zunächst in einen Graphen eingebettet werden. Ohne Beachtung der Messungen der Leistungsmetriken können die folgenden Eigenschaften bereits aus der Definition eines Mesh-Netzwerkes in die Graphentheorie übersetzt werden:

- Aus der Eigenschaft des Mesh-Netzwerkes, dass eine Node sich stets selbst erreichen kann, folgt ein vollständig mit Schleifen besetzter Graph.
- Die multiplen Netzwerkkarten pro Node erlauben Mehrfachkanten zwischen zwei Knoten des Graphens.
- Teilgraphen über drei oder mehr Nodes können vollständige Graphen bilden. Der Graph mit allen Nodes enthält somit Zyklen.

Zu diesen Eigenschaften lassen sich zwei weitere Eigenschaften des Graphens aus den Ergebnissen der Messungen entnehmen. Zum einen existieren zwischen den Nodes des MIOT-Testbeds nicht nur bidirektionale, sondern auch vereinzelt unidirektionale Verbindungen. Diese werden durch gerichtete Kanten im Graphen repräsentiert. Zum anderen sind auch bei den bidirektionalen Verbindungen Messschwankungen vorhanden. Somit sind alle Kanten im Graphen in beiden Richtungen gerichtet und mit unterschiedlicher Gewichtung dargestellt. Diese Eigenschaften des Graphens G spielen neben der Konstruktionszeit der Datenstruktur und der Anfragezeit auf die Datenstruktur eine große Rolle bei der Auswahl einer geeigneten Datenstruktur.
Auf Grundlagen dieses Graphens G kann schließlich eine Datenstruktur gefunden werden. Dabei ist zu beachten, dass die Eigenschaft bezüglich der Zyklen bereits die Auswahl einschränkt. Baumartige Datenstrukturen können nicht verwendet werden, da sie per Definition stets azyklisch sind.
Zwei mögliche Methoden zur Speicherung eines Graphens mit diesen Eigenschaften sind die *Adjazenzliste* und die *Adjazenzmatrix*. Bei der Adjazenzliste wird ein ungerichteter Graphen $G = (V, E)$ durch ein Array mit $|V|$ Listen dargestellt. In den Listen werden für jeden Knoten $v \in V$ all seine Nachbarn v' mit $\{v' \in V : (v, v') \in E\}$ gespeichert. In einem gerichteten Graphen wird die Adjazenzliste so angepasst, dass anstelle der Nachbarn die Nachfolger in diesen Listen enthalten sind [50]. Sind die Kanten $(v, v') \in E$ zusätzlich gewichtet, so können diese Gewichtungen zu den entsprechenden Nachbarn respektive Nachfolgern v' in der Listen geschrieben werden.

Eine alternative Darstellung eines Graphens $G = (V, E)$ zur Adjazenzliste ist die Adjazenzmatrix. Die Knoten $v \in V$ des Graphens G bilden eine zweidimensionale Matrix $A = (a_{i,j})$ mit $|V| \times |V| = \left|V^2\right|$ Elementen. Jedes Element $a_{i,j} \in A$ in der i-ten Zeile und j-ten Spalte repräsentiert die Kante von Knoten i nach Knoten j mit

$$a_{i,j} = \begin{cases} 1, & \text{wenn } (i,j) \in E, \\ 0 & \text{sonst.} \end{cases}$$

Sind die Kanten der Graphens gewichtet, können die Gewichtungen auch anstelle der Wertigkeit 1 in die Elemente $a_{i,j}$ der Adjazenzmatrix eingetragen werden [50]. Diese beiden Datenstrukturen sind für die Testszenarien hinreichend.
Die allgemeine Auswahl von Datenstrukturen beruht auf der Grundlage des benötigten Speicherplatzes und der Zugriffszeiten auf diese Datenstrukturen. Diese Entscheidung zwischen der Adjazenzliste und Adjazenzmatrix als geeignete Datenstruktur zur Darstellung der Netzwerktopologie wird schließlich durch Anzahl der vorhandenen Kanten bestimmt. Ist der Graph $G = (V, E)$ dünn mit Kanten $e \in E$ besetzt, ist die Adjazenzliste vorzuziehen. Bei Annäherung von $|E|$ an $\left|V^2\right|$ sollte die Adjazenzmatrix gewählt werden [50]. Denn für das zu betrachtende Mesh-Netzwerk gilt $|E| \ll \left|V^2\right|$, sodass die Adjazenzliste effizienter ist (vgl. Kapitel 3.4). Durch diese geringe Dichte $d = |E| \,/\, \left|V^2\right|$ hat die Adjazenzliste einen Vorteil. Sie benötigt genau so viel Speicherplatz, wie Knoten und Kanten vorhanden sind. Mathematisch liegt der Speicherplatzbedarf der Adjazenzliste in der Klasse $\Theta(|V| + |E|)$. Demgegenüber wird bei der Adjazenzmatrix unabhängig von der Anzahl der Kanten stets ein Speicherplatzbedarf in der Klasse $\Theta(\left|V^2\right|)$ benötigt.
Weiterhin ist zu überprüfen, wann die Zugriffe auf die Datenstruktur erfolgen. Bei der Ausführung des Service Managers, wie in Kapitel 4 beschrieben, wird zu Beginn einmalig auf den einzelnen MIOT-Nodes eine statische Kostenmatrix mit der vollständigen Routing-Gewichtung von der jeweilgen Node zu allen anderen Nodes aus der Datenstruktur erstellt (vgl. Kapitel 4.2.3). Da für das Auffinden von geeigneten Routen in der Datenstruktur alle Nachfolger des gerichteten Graphens betrachtet werden müssen, liegt hier ebenfalls der entscheidende Vorteil bei der Adjazenzliste gegenüber der Adjazenzmatrix. Die Adjazenzliste liefert für einen Knoten v direkt alle Nachfolger v' mit $M_{v'} = \{v' \in V : (v, v') \in E\}$ in der Laufzeitklasse $\Theta(|M_{v'}|)$. Bei einer geringen Dichte d gilt für den Graph G hier $|M_{v'}| < |V|$. Bei der Adjazenzmatrix muss hingegen immer eine vollständige Reihe mit $|V|$ Elementen in der Laufzeitklasse $\Theta(|V|)$ durchsucht werden, um die Liste der Nachfolger zu finden. Der Vorteil der Adjazenzmatrix, ein schneller Zugriff auf einzelne Elemente in der Laufzeitklasse $\mathcal{O}(1)$, ist in den zu betrachtenden Untersuchungen nicht vorhanden. Da die Einfügung von Knoten und Kanten nur während des initialen Aufbaus geschieht sowie das Löschen von Knoten und Kanten nicht benötigt wird, werden Überlegungen zu diesen Auswahlkriterien außer Acht gelassen. Nach erfolgreicher Erstellung der Datenstruktur wird diese als statisch betrachtet. Aus den dargelegten Gründen fällt die Wahl eindeutig auf die Adjazenzliste als Datenstruktur zur Speicherung der Netzwerktopologie. Innerhalb der Elemente in den Listen werden die Leistungsmetriken Durchsatz und ETX sowie die verwendete Netzwerkkarte eingetragen.

3.4 Auswertung und Messergebnisse

Bei den Messaufnahmen der Leistungsmetriken ETX und Durchsatz sind von den insgesamt $|V_{\max}| = 60$ vorhandenen MIOT-Nodes nur eine Teilmenge von $|V| = 41$ Nodes verfügbar. Dies hat zum einen eine direkte Auswirkung auf die Gesamtanzahl der Verbindungen zwischen den Nodes im MIOT-Testbed. Zum anderen können die Knoten beim Routing in den weiterführenden Untersuchungen in Kapitel 5 nicht beachtet werden. In einem voll vermaschten Netzwerk, in dem sich alle Nodes über die drei Teilnetzwerke untereinander erreichen können, wären mit $|V|$ als Menge der verfügbaren Nodes genau $|E_{\max}| = 3 \times |V|^2 = 3 \times 41^2 = 5.043$ Verbindungen insgesamt möglich. Im Folgenden werden zunächst die Auswertungen der einzelnen Messungen dargestellt und daraufhin eine Kombination der beiden Messungen durchgeführt. Die gesamte Evaluation erfolgt mithilfe des Python-Skripts `topology_evaluation.py`. In einem weiteren Schritt werden nur die besten Mehrfachkanten in die in Kapitel 3.3 beschrieben Adjazenzliste eingefügt, da das in Kapitel 5.1 beschriebene SDN nur diese zur Simulation des MIOT-Testbeds benötigt.
Die Aufnahme der ETX erzeugt über die drei voneinander getrennten Netzwerke hinweg die in der Tabelle 3.1 angegebenen Messwerte. Hier ist die erwartete Übertragungswahrscheinlichkeit $P(\text{TX}) = ETX^{-1}$ interessant, die sowohl über das kombinierte Netzwerk als auch über die einzelnen Netzwerke betrachtet werden kann. In dieser Messung sind auch die durch die drei Netzwerkkarten entstehenden Mehrfachkanten zwischen den Nodes im MIOT-Testbed enthalten.

Netzwerk	Verbindungsanzahl	$M_{P(\text{TX})}$	$SD_{P(\text{TX})}$
Gesamt	2.071	50,726%	34,375%
Netzwerk 1	496	57,907%	28,540%
Netzwerk 2	739	50,601%	35,804%
Netzwerk 3	836	46,573%	35,563%

Tabelle 3.1: Vergleich der Knotenanzahl und erwarteten Übertragungswahrscheinlichkeit des aufgenommenen MIOT-Testbeds anhand der ETX-Messung. Das Netzwerk ist dabei in drei Subnetzwerk aufgeteilt, die durch die drei Netzwerkkarten bereitgestellt werden.

Würde eine Minimierung des Netzwerkes auf Basis von ETX durchgeführt werden, sodass immer die Beste der maximal drei vorhandenen Mehrfachkanten einer Verbindung zwischen zwei Nodes verwendet wird und dennoch alle Nodes untereinander erreichbar sind, reduziert sich die Anzahl der Verbindungen in dem kombinierten Netzwerk auf $|E| = 1.043$ (aufgeteilt: Netzwerk 1 = 167, Netzwerk 2 = 390, Netzwerk 3 = 486). Für das gesamte Netzwerk würde dies eine Veränderung der kombinierten $P(\text{TX})$ auf $M_{P(\text{TX})} = 51{,}746\%$, $SD_{P(\text{TX})} = 35{,}344\%$ bedeuten.
Das MIOT-Testbed soll allerdings so aufgenommen werden, dass bei allen betrachteten Kanten sowohl die Werte für ETX als auch des Durchsatzes vorhanden sind, denn nur so kann ein Netzwerk vollständig beschrieben werden. Bei der im Vergleich zur ETX-Messung kurzzeitigen Messung des Durchsatzes konnten nicht alle Verbindungen wiederhergestellt werden. Die Messung des Durchsatzes enthalten bei einigen Knotenpaare nicht alle Mehrfachkanten, obwohl die Messung der ETX dort möglich war. Daher werden im nächsten Schritt zunächst die Ergebnisse der Durchsatz-Messung dargestellt. Die Tabelle 3.2 zeigt

die in der Messung aufgefundene Kantenanzahl zwischen den selben 41 MIOT-Nodes wie aus der ETX-Messung. Weiterhin wird der durchschnittliche Durchsatz der Verbindungen in der Einheit Mbit/s dargestellt.

Netzwerk	Verbindungsanzahl	$M_{\text{Durchsatz}}$ (in Mbit/s)	$SD_{\text{Durchsatz}}$ (in Mbit/s)
Gesamt	1.296	2,769	1,211
Netzwerk 1	290	2,303	1,437
Netzwerk 2	476	2,991	1,121
Netzwerk 3	530	2,826	1,088

Tabelle 3.2: Vergleich der Knotenanzahl und des Durchsatzes des aufgenommenen MIOT-Testbeds anhand der Durchsatz-Messung. Das Netzwerk ist dabei in drei Subnetzwerk aufgeteilt, die durch die drei Netzwerkkarten bereitgestellt werden.

Darauf folgt eine Zusammenführung der Messergebnisse aus den beiden ETX- und Durchsatz-Messungen durch Bildung der Schnittmenge beider gemessener Graphen. Dabei werden zunächst alle Werte der Durchsatz-Messung in den Graphen der ETX-Messung eingearbeitet und schließlich alle Kanten aus den Graphen gestrichen, die nicht beide Messwerte enthalten. Mehrfachkanten sind in diesem Graphen jedoch noch existent. Die Messergebnisse der neuen Schnittmenge sind in der folgenden Tabelle 3.3 dargestellt.

Netzwerk	Verbindungsanzahl	$M_{P(\text{TX})}$	$SD_{P(\text{TX})}$	$M_{\text{Durchsatz}}$ (in Mbit/s)	$SD_{\text{Durchsatz}}$ (in Mbit/s)
Gesamt	1.167	62,352%	31,562%	2,873	1,178
Netzwerk 1	271	67,105%	21,240%	2,387	1,413
Netzwerk 2	400	65,337%	32,621%	3,148	1,060
Netzwerk 3	496	57,379%	34,597%	2,917	1,041

Tabelle 3.3: Vergleich der Knotenanzahl, erwarteten Übertragungswahrscheinlichkeit und des Durchsatzes des aufgenommenen MIOT-Testbeds anhand der Schnittmenge der ETX- und Durchsatz-Messung. Das Netzwerk ist dabei in drei Subnetzwerk aufgeteilt, die durch die drei Netzwerkkarten bereitgestellt werden. Verbindungen, die nicht beide Leistungsmetriken aufwiesen, wurden aus dem unterliegenden Graphen entfernt.

Interessant ist mit diesen Ergebnissen ein Vergleich zwischen ETX und den Durchsatz innerhalb der Zusammenführung. Abbildung 3.1 zeigt diesen Vergleich aufgeteilt auf die drei noch nicht weiter minimierten Netzwerke. Während in den Netzwerken 2 und 3 die Werte deutliche über die gesamte Breite der erwarteten Übertragungswahrscheinlichkeit streuen, konzentrieren sich die Verbindungen in Netzwerk 1 größtenteils bei einer erwarteten Übertragungswahrscheinlichkeit zwischen 60% und 80%. Vergleichsweise viele Verbindungen der Netzwerke 2 und 3 sind zusätzlich im Bereich von über 90% zu finden. Darüber hinaus zeigt sich ein erhöhter Gesamtdurchsatz bei den Netzwerken 2 und 3 im Vergleich zu Netzwerk 1. Während sich die Verbindungen von Netzwerk 1 hier gleichmäßig verteilen, ist der größte Anteil der Verbindungen in den Netzwerken 2 und 3 in der oberen Hälfte wiederzufinden. Das Minimum aller Durchsatz-Messungen liegt bei $0,0179$ Mbit/s und das interessante-

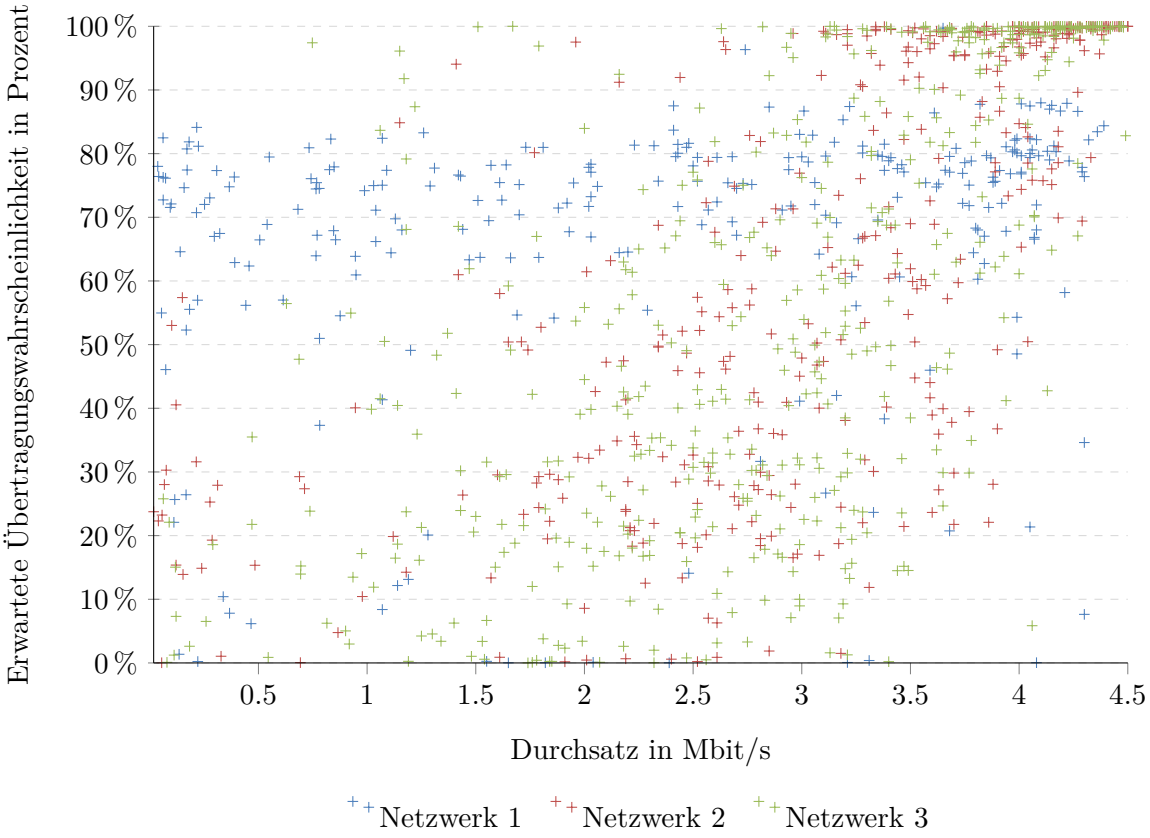


Abbildung 3.1: Gegenüberstellung der erwarteten Übertragungswahrscheinlichkeit und des Durchsatzes, aufgeteilt auf die drei separaten, kabellosen Netzwerke des MIOT-Testbeds. Die erwartete Übertragungswahrscheinlichkeit wurde dabei mit der Formel $P = \text{ETX}^{-1} * 100$ berechnet.

re Maximum bei $4,6$ Mbit/s. Abschließend erfolgt durch Reduktion der Mehrfachkanten die Erstellung eines finalen Graphens, der für die Verwendung des SDNs in einer Adjazenzliste abgespeichert wird. Die Mehrfachkanten geben einer Verbindung zwischen zwei MIOT-Nodes zwar eine höhere Stabilität, da diese bei Verlust einer einzelnen Kante als Ausweichroute verwendet werden können, doch müssten diese auch beim Routing Beachtung finden. Vorausschauend ist dies bei dem SDN-Controller in Kapitel 5.1 nicht der Fall, da der SDN-Controller mittels Spanning Tree Protokoll das Auffinden der kürzesten Pfade durchführt. Sind diese einmal gesetzt, werden sie immer weiter verwendet und der Ausfall einer Verbindung führt zum Verlust der gesamten Route.
Die Reduktion der Mehrfachkanten wird anhand der ETX entschieden. Je näher der Wert von ETX dem Minimum 1 ist, desto besser ist die Verbindung. Zwar könnte die Reduktion auch anhand des Durchsatzes geschehen, doch die Entscheidung fiel auf ETX in Hinblick auf die Tatsache, dass eine gesicherte Übertragung ohne erneuten Versand wichtiger ist als die Maximierung des Durchsatzes. Der finale Graph besteht schließlich aus $|E| = 634$ Verbindungen (aufgeteilt: Netzwerk 1 = 85, Netzwerk 2 = 215, Netzwerk 3 = 334) zwischen den 41 Nodes. Für das gesamte Netzwerk bedeutet dies eine Veränderung der kombinierten $P(\text{TX})$ auf $M_{P(\text{TX})} = 60{,}584\%$, $SD_{P(\text{TX})} = 33{,}586\%$ und einen Durchsatz mit

$M_{\text{Durchsatz}} = 2{,}827$ Mbit/s, $SD_{\text{Durchsatz}} = 1{,}167$ Mbit/s.
Ein weiterer Aspekt ist die Konnektivität der MIOT-Nodes, die in der Graphentheorie durch den Knotengrad d_G beschrieben wird. Der finale Graph mit 41 Knoten hat einen durchschnittlichen Knotengrad $d_G(v \in V)$ von $M_{d_G(v)} = 15$, $SD_{d_G(v)} = 6{,}083$. Das Maximum liegt bei einem Knotengrad von $d_G(v) = 24$ bei drei unterschiedlichen MIOT-Nodes. Das Minimum mit $d_G(v) = 1$ existiert bei einer MIOT-Node. Die MIOT-Nodes mit einem hohen Knotengrad könnten sich als sehr geeignet zur Ausführen eines Dienstes im IoT herausstellen, da sich viele andere Nodes in der unmittelbaren Umgebung befinden und die Routen durch das Netzwerk gering sind.
Schließlich zeigt sich auch hier entscheidende Vorteil der Adjazenzliste gegenüber der Adjazenzmatrix. Die Kantendichte d mit $d = |E| \big/ \left|V^2\right|$ liegt bei diesem finalen Graphen, der keine Mehrfachkanten enthält, bei gerade einmal $d = 634 \big/ \left|41^2\right| = 37,72\%$ (vgl. Kapitel 3.3).

KAPITEL 4

Der Service Manager als dienstorientierte Plattform

Die in Kapitel 2 vorgestellten autonomen Dienste im IoT benötigen eine einfache Möglichkeit, ihre Anwendungen den heterogenen Objekten im IoT bereitstellen zu können. Eine Option zur Bereitstellung von Diensten wäre die Ausführung auf statischen, zentralen Servern. In kabelgebundenen oder kleinen kabellosen Netzwerken mag dies funktionieren, bei einer immer weiter wachsender Anzahl vernetzter Objekte im IoT entstehen jedoch große Probleme. Paketverluste und hohen Latenzzeiten treten ein [3], die es im IoT zu minimieren gilt [43]. Eine Idee zur Lösung der Probleme ist die Verkürzung der Kommunikationswege zwischen den anfragenden Objekten und den Diensten im IoT durch die Bereitstellung der Dienste in physischer Nähe zu den anfragenden Objekten. Das Modell hinter dieser Idee wird als Fog Computing bezeichnet [17].
Die Bereitstellung der Dienste erfolgt dabei über eine Middleware, die auf den Objekten im IoT ausgeführt wird, so die Heterogenität der Objekte in ihrer Leistungsfähigkeit sowie Kommunikationsmöglichkeiten ausgleicht und schließlich eine Interoperabilität zwischen Anwendungen und Diensten schafft [51, 52]. Die Middlewares analysieren das Nutzungsverhalten des Dienstes sowie der Clients und stellen darauf aufbauend erforderliche Kapazitäten für den Dienst in den Bereichen des IoT bereit, wo sie benötigt werden. Der Nachteil dieser Middlewares ist die Anpassungsnotwendigkeit der Dienste, denn diesen wird von den Middlewares eine API angeboten, die auf der Dienstebene implementiert werden muss. Losgelöst von einer eigenen von den Diensten zu implementierenden API wird in diesem Kapitel die Service Manager Plattform vorgestellt.
Die Service Manager Plattform hat das Ziel beliebige autonome Dienste im IoT ohne proprietäre dienstseitige Anpassungen bereitzustellen. Dienste, die bereits zuvor auf den einzelnen Objekten im IoT ausgeführt werden konnten, sollen durch die Plattform ebenfalls ohne notwendige Anpassungen möglichst strategisch optimal im IoT positioniert werden. Dies gelingt durch die Ausführung der Plattform auf allen für den Dienst vorgesehen Objekten im IoT und einer zyklischen Migrationen der Dienste zwischen den Objekten (vgl. Kapitel 2.3).
Die Abbildung 4.1 zeigt die allgemeine Übersicht über die Plattform mit der Client-Server-Architektur eines Service Managers, die die eigenständigen autonomen Dienste vollständig

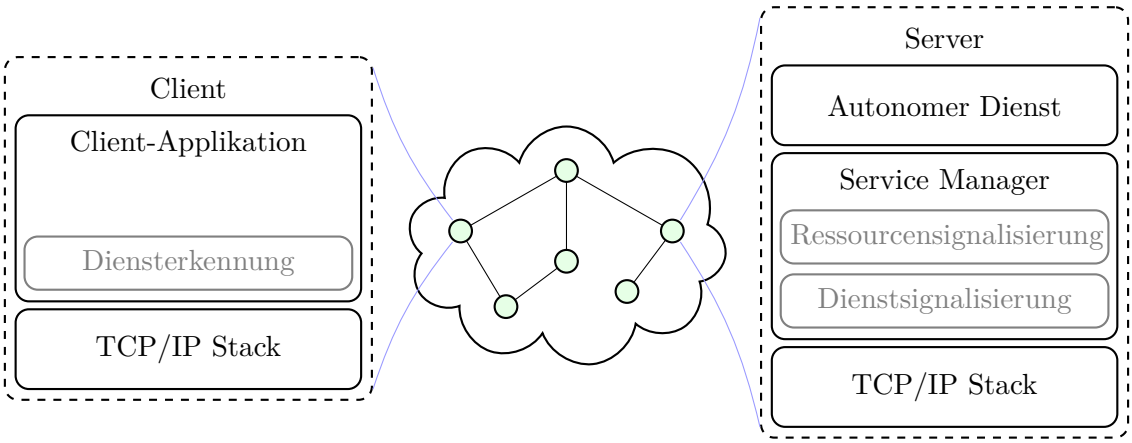


Abbildung 4.1: Allgemeine Übersicht über die Plattform mit der Client-Server-Architektur eines Service Managers, die auf allen Objekten im IoT ausgeführt werden kann, die Ausführung des eigenständigen autonomen Dienst vollständig verwaltet und die Signalisierung von Änderungen am Dienstzustand übernimmt. Die Clients können sich für diese Änderungen mit einer Komponente zur Diensterkennung registrieren.

verwaltet. Der Service Manager unterstützt mit der Dienst- und Ressourcensignalisierung dabei zwei wesentlicher Funktionen für die Bereitstellung eines verteilten Systems im IoT. Diese werden in den folgenden Unterkapiteln näher beschrieben.
Voraussetzung zur Ausführung der Plattform ist lediglich eine Unterstützung von Python 2.7 seitens der Objekte. Die komponentenbasierte Architektur des Service Managers ist so modular aufgebaut, dass eine Erweiterung ohne große Anpassungen möglich ist. Ist ein Dienst an einer zuvor festgelegten Position in die Plattform eingespeist worden, organisiert diese anhand des Nutzungsverhalten des Dienstes und der Clients alle notwendigen Schritte zur optimalen Positionierung im IoT. Dabei deaktivieren die Service Manager auf den Objekten, auf denen sie gerade keinen Dienst ausführen, die nicht notwendigen Teilkomponenten ihres Systems, um wichtige Ressourcen der Objekte im IoT einzusparen. Während der Bereitstellung des Dienstes auf dem Objekt im IoT werden die Teilkomponenten aktiviert, sodass bei der Suche der optimalen Positionierung im IoT verschiedene Schritte unternommen werden können bis der Dienst schließlich migriert werden kann. Das dazugehörige zyklische Verfahren ist in Abbildung 4.2 illustriert.

Abbildung 4.2: Die zyklische Verarbeitung eines Service Managers in der Plattform, bei der die notwendigen Schritte zwischen dem Erhalt und weiteren Migrationen eines Dienstes sequentiell aufeinanderfolgenden. Die verschiedenen Farben der einzelnen Schritte deuten auf die Komponenten des Service Managers hin, die diesen Schritt verarbeiten. Diese sind in Abbildung 4.3 dargestellt.

Zunächst werden Anforderungen an die Service Manager Plattform beschrieben, die beim Entwurf der Architektur und der darauffolgenden Entwicklung zu beachten sind und in funktionale und nicht-funktionale Anforderungen kategorisiert werden.
Abschließend wird das der Service Manager Plattform zugrundeliegende Entwurfsmuster (engl. Design Pattern) und die damit einhergehend entworfene Softwarearchitektur beschrieben. Die einzelnen Komponenten des Service Managers werden hier genau erläutert und die in Abbildung 4.2 dargestellten Funktionen detailliert beschrieben. Dabei ist schließlich zu sehen, dass die Berechnung der Positionierung auf Basis der topologischen Verfahren aus Kapitel 2.4 mit den definierten Leistungsmetriken aus Kapitel 3.1 erfolgt.

4.1 Anforderungen an die Plattform

Die Service Manager Plattform stellt im Allgemeinen einen Dienst bereit und differenziert sich von einer Middleware dadurch, dass sie die Komplexität eines einzelnen Objektes, in diesem Fall die des IoT, nicht für den Dienst vollständig abstrahiert [53]. Durch dieses Vorgehen können die autonomen Dienste wie die in Kapitel 2 eigenständig entwickelt werden und bleiben autonom. Dienste müssen sich bei der Plattform zwar mit der eigenständigen Bereitstellung ihrer Anwendung beispielsweise durch Webservices befassen, wie es auch bei Applikationen im Internet notwendig ist, doch die Problematiken des unterliegenden IoT-Gesamtsystems werden von der Service Manager Plattform gelöst. Bei der Bereitstellung einer solchen Plattform im IoT muss zunächst ein Anforderungsrahmen geschaffen werden, der bei dem Entwurf und der Implementierung des Service Managers schließlich zur Geltung kommt. Die Anforderungen des Rahmens lassen sich in zwei Untergruppen eingeteilt, den nicht-funktionalen und den funktionalen Anforderungen.
Aus zwei Forschungsbereichen zu IoT lassen sich acht nicht-funktionalen Anforderungen an Middlewares entnehmen, die ebenfalls für die Service Manager Plattform maßgeblich sind. Bei der Bereitstellung allgemeiner Middlewares für das IoT sind diese bereits spezifiziert worden [52], doch darüber hinaus existieren weitere Anforderungen an den Architekturvorschlag der OpenFog Consortium Architecture Working Group zu Fog Computing [51]:

Autonomie: So wie die Dienste eine Autonomie aufweisen (vgl. Kapitel 2.2), so besteht auch die Anforderung eines gewissen Grades Autonomie an den Service Manager. Er soll autonom den Betrieb des Dienstes garantieren und diesen während des gesamten Betriebslebenszyklus verwalten. Auf lokalen Entscheidungen basierend soll der Betrieb des Dienstes an strategisch sinnvollen Positionen im IoT selbstoptimierend und selbstorganisierend gewährleistet sein [30]. Dieser dezentrale Ansatz zeigt die Unterschiede zu Diensten, die in der Cloud bereitstellt werden. Ein weiterer Punkt der Autonomie ist die Signalisierung für mögliche Konsumenten des Dienstes, die die Plattform bei der Bereitstellung des Dienstes vorzunehmen hat (vgl. Abbildung 4.1).

Offenheit: Zur Erreichung des ultimativen Ziels eines Netzwerks mit „Ubiquitous Services“ (vgl. Kapitel 2.1) muss eine Offenheit gegenüber anderen Systeme und Diensten gewährleistet sein. Der Service Manager soll eine transparente Entwicklung und zusätzlich einen hohen Freiheitsgrad bei der Auswahl des bereitzustellenden Dienstes aufweisen. Dies ist durch eine starke Entkopplung von Dienst und Service Manager möglich und sorgt für eine hohe Portabilität des Dienstes. Eine Entkopplung aller Komponenten des Service Managers ermöglicht zusätzlich für die Möglichkeit einer

schnellen Erweiterung. Abschließend sorgt eine offene Kommunikation im Netzwerk für eine effiziente Bereitstellung des Dienstes, indem der Service Manager alle notwendigen Informationen, wie Prozessleistung, Speicherkapazitäten, Sensorfähigkeit und die Leistung kabelloser Verbindungen, der Objekte im Netzwerk erhält und auch selber bereitstellt.

Skalierbarkeit: Die Skalierbarkeit ermöglicht es dem Service Manager die Dynamik des IoT zu adressieren. Wird aus den topologischen Verfahren in Kapitel 2.4 ein Ansatz auf der Basis lokaler Information gewählt, so kann der Service Manager in einem stetig wachsenden IoT angewandt werden, vorausgesetzt, die neuen Objekte unterstützen die Plattform respektive den Dienst in ihren Interaktionen. Die Leistungsfähigkeit des Service Managers soll so skalierbar sein, dass er den Anwendungsanforderungen entgegen kommen kann, wie beispielsweise geringe Latenzen zwischen Sensoren und Aktoren auf Dienstebene. Die Adressierung der vielen Anwendungen, Benutzer und Objekte im IoT kann schließlich durch die Unterstützung von IPv6 erreicht werden [52, 2]. Abschließend kann eine Verbesserung in der Skalierbarkeit durch eine lose Kopplung der Plattformkomponenten und eine vollständige Entkopplung der Dienstlogik von der Plattform erreicht werden.

Sicherheit: Bei der Sicherheit müssen verschiedene Aspekte Beachtung finden. Zum einen haben der Dienst und der Service Manager in allen Komponenten Datenschutz zu gewährleisten. Doch auch die Privatsphäre der Anwender muss geschützt werden. So könnte die Kontextabhängigkeit des Service Managers persönliche Informationen, wie den Standort von Objekten oder Personen, offenbaren. Daraus ist zu schließen, dass eine Migration zwischen verschiedenen Segmenten des IoT nur durch eine explizite Zustimmung möglich sein darf (vgl. Abbildung 2.1). Damit einhergehend sollten neu hinzugefügte Objekte nicht direkt als mögliche Kandidaten für eine Ausführung des Dienstes in Betracht gezogen werden, sie müssen zunächst in einer bestimmten Form Vertrauen aufbauen.

Echtzeit, Aktualität: Der Service Manager muss Dienste mit Echtzeitanforderungen unterstützen und dabei die Aktualität der Daten garantieren können. Verzögerte Informationsbereitstellung machen das System in Bereichen wie Transport und Gesundheitswesen unbrauchbar und möglicherweise sogar gefährlich. Dies kann durch eine Bereitstellung des autonomen Dienstes nahe der konsumierenden Objekte erreicht werden.

Zuverlässigkeit: Auch bei Fehlern soll der Service Manager weiterhin zuverlässig funktionsfähig bleiben. Die einzelnen Komponenten und der Dienst müssen daher ebenfalls zuverlässig sein, so dass der Service Manager die vollständige Funktionsfähigkeit gewährleisten kann. Dies schließt sowohl die Kommunikation, Datenhaltung als auch die verwendete Technologien mit ein. Diese Zuverlässigkeit bildet die Grundlage für die Verfügbarkeit des Systems.

Verfügbarkeit: Zu jeder Zeit muss der Service Manager die Verfügbarkeit des Dienstes und seiner Funktionen garantieren. Dies gilt auch bei aufgetretenen Fehlern im System, bei denen die Erholungszeit und die Fehlerfrequenz zu minimieren gilt. Durch die Zusammenarbeit der Anforderungen von Zuverlässigkeit und Verfügbarkeit wird schließlich eine hohe Fehlertoleranz gewährleistet, die für den Dienst unabdingbar ist.

Einfache Verteilung: Die Service Manager Plattform, der Dienst und die Updates dieser beiden Komponenten sollen einfach ohne großen Aufwand auf die Objekte im IoT verteilt werden können. Dies kann durch eine einfache und schmale Konfiguration erreicht werden.

Neben diesen nicht-funktionalen Anforderungen existieren die folgenden fünf funktionale Anforderungen, die sich größtenteils an die Service Manager Plattform richten, aber auch teilweise die zu unterstützende Dienst betreffen [52]:

Ressourcenentdeckung: Im IoT werden durch die heterogenen Objekte viele verschiedene Ressourcen bereitgestellt, die angefangen von der grundlegenden Peripherie wie Sensoren, über die Prozessorleistung, Speichergröße und Kommunikationsmodule bis hin zur Netzwerktopologie in verschiedene Kategorien klassifiziert werden können (vgl. Kapitel 3.1.6). Hier können zusätzlich auch von den Objekten angebotene Dienste, die der Service Manager nicht verwaltet, als eine Ressource betrachtet werden.
Wie bereits in Kapitel 2.4.5 beschrieben wurde, ist eine globale Bereitstellung dieses Wissens in einem realen System nicht umsetzbar, da die Infrastruktur und Umgebung des IoT schier zu groß und dynamisch ist. Ebenso wäre eine manuelle Bereitstellung einer Ressourcenübersicht undenkbar, was schließlich zu dem Schluss führt, dass sie automatisiert werden muss. Dies sollte nicht wie bei zentralen System über einen dafür bestimmten Server laufen, sondern über automatisierte Mechanismen zur Ressourcensignalisierung in jedem einzelne Objekt, die die Objektpräsenz und die angebotenen Ressourcen bekannt geben. Diese Aufgabe soll der Service Manager durch die Ressourcen- und Dienstsignalisierung übernehmen (vgl. Abbildung 4.1), wobei die Skalierbarkeit aufgrund der im IoT vorhandenen Ressourcenknappheit Beachtung finden muss.

Ressourcenverwaltung: Werden die Ressourcen eines dienstausführenden Objektes knapp, können Probleme bei der Bereitstellung einer akzeptablen QoS entstehen. Aus diesem Grund muss für die globale Anwendung, respektive dem unterliegenden Dienst die Ressourcennutzung des Objektes überwacht und auf auftretende Konflikte reagiert werden. Als Konfliktlösung eignen sich beispielsweise die Beschaffung neuer Ressourcen durch eine Migration auf Objekte, die ein größeres Ressourcenkontingent aufweisen, bis die Bedürfnisse des Dienstes erfüllt sind.

Datenverwaltung: Daten sind die Grundlage für Anwendungen im IoT. Dabei kann man zwischen den aufgenommenen Sensordaten und den Daten aus der Netzwerkinfrastruktur differenzieren, die über verschiedene Verfahren erfasst werden (vgl. Kapitel 2.4). Während für den Service Manager die Daten aus der Netzwerkinfrastruktur für die Auswertungen der Netzwerkauslastungen relevant sind, übernimmt der Dienst die Datenverwaltung aufgenommener Sensordaten der Objekte. Neben der Verwaltung ist er auch für die Anschaffung, Verarbeitung, Aggregation und Speicherung dienstrelevanter Daten zuständig.

Ereignisverwaltung: Durch die hohe Anzahl vieler heterogener Objekte entstehen im IoT erkleckliche Mengen an Ereignissen, die sowohl für den Dienst als auch für den Service Manager relevant sind. Beide Softwarekomponenten sollen die für sie relevanten Teile der Ereignisse separat entnehmen, auswerten und daraus Rückschlüsse für ihre Aufgaben ziehen. Würde der Service Manager diese, wie bei Middlewares, bereitstellen verlören die Dienste einen großen Teil ihrer geforderten Autonomie.

Quellcodeverwaltung: In einer IoT-Umgebung kann die Verteilung von Quellcode herausfordernd sein. Durch Migrationen und eines global einheitlichen Versionsstandes soll die Service Manager Plattform diese Aufgabe für den Dienst zwischen den verschiedenen Objekten im IoT übernehmen.

4.2 Entwurfsmuster und Softwarearchitektur

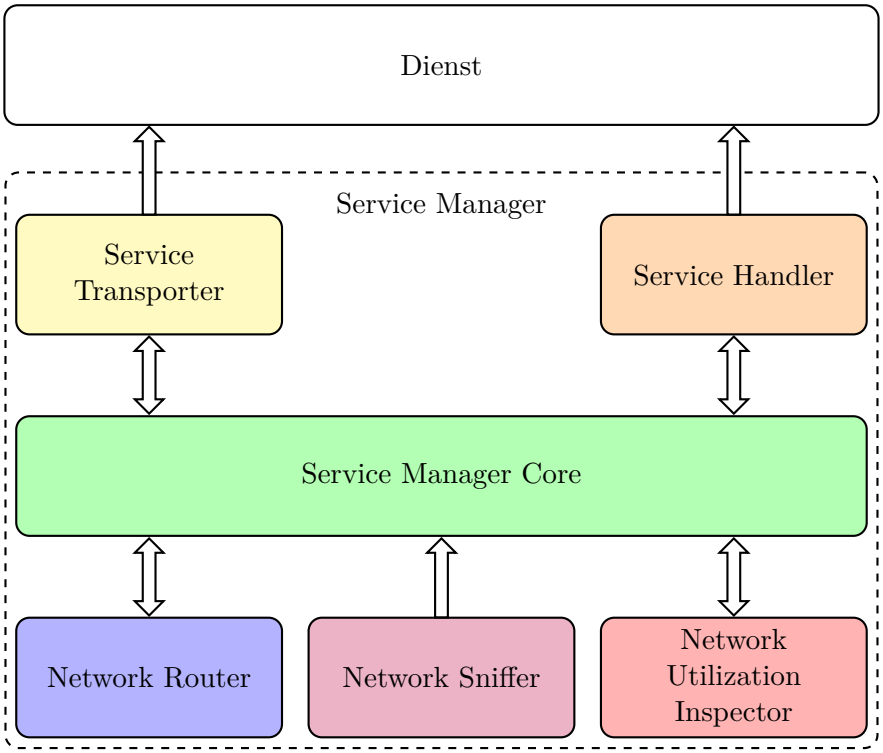


Abbildung 4.3: Komponentenarchitektur des Service Managers, bei der der Sevice Manager Core die Vermittlerrolle zwischen den Komponenten übernimmt und diese miteinander verknüpft. Während die unterste Schicht die Funktionen der Analyse, zyklischen Verarbeitung und Entscheidung zur Migration bereitstellt, bildet die oberste Schicht Funktionalitäten für die Migration und Prozessbildung des eigentlichen Dienst und seinen Dateien ab. Die Pfeile stellen den Kommunikationsverlauf zwischen den Komponenten dar, wobei die Kommunikation in Richtung des Dienstes nur auf Systemebene durch die Verwaltung von Dienstdateien gegeben ist.

Die Anforderungen an die Service Manager Plattform aus Kapitel 4.1 bilden einen Rahmen, aus der sich eine Softwarearchitektur ableiten lässt. Neben diesen Anforderungen existiert noch ein weiterer Punkt, der speziell die Architektur der Service Manager Plattform betrifft, nämlich eine hohe Flexibilität, so dass das Hinzufügen und die Weiterentwicklung von Funktionalitäten ohne große Umstrukturierung und Neuimplementierung möglich ist [52]. Durch eine Gliederung aller Aufgabenbereiche des Service Managers in einzelne, dedizierte Komponenten ist es möglich diese Flexibilität zu erreichen. Einhergehend steigert diese Modularität auch die Wartbarkeit der gesamten Service Manager Plattform dadurch, dass die einzelnen

Komponenten unabhängig voneinander getestet werden können. Bei der Erstellung einer geeigneten Softwarearchitektur, die diese Flexibilität unterstützt, müssen zunächst die Eigenschaften von Komponenten und ihre Kommunikationswege analysiert werden. In einer komponentenbasierten Architektur haben die Komponenten einige Eigenschaften [54]:

- Generell wiederverwendbar in verschiedenen Szenarien und Anwendungen. Teilweise können Komponenten jedoch auch spezifisch sein.
- Ersetzbar durch ähnliche Komponenten.
- Nicht kontextspezifisch und somit über verschiedene Umgebungen hinweg einsetzbar. Spezifische Daten wie Zustandsdaten sollen der Komponente übergeben und nicht durch diese angefragt werden.
- Erweiterbar um neue Funktionen und neues Verhalten.
- Eingekapselt mit einem Interface nach außen, über das Funktionen ausgeführt werden können aber keine Details über innere Prozesse oder Variablen offenbart.
- Unabhängig zu anderen Komponenten, sodass sie zwischen verschiedenen Umgebungen verteilt werden können.

Diese Eigenschaften werden durch verschiedene Architekturmuster adressiert [55]. Am besten eignet sich dabei das Verhaltensmuster (engl. behavioral pattern) *Mediator* nach [55, Kapitel 5.5] als Grundlage für die Softwarearchitektur. Bei dem Verhaltensmuster Mediator verknüpft eine Hauptkomponente, die als Vermittler (engl. Mediator) bezeichnet wird, alle vorhandenen Komponenten miteinander. Durch die Verarbeitung aller Anfragen und Interaktionen der Komponenten durch den Vermittler wird eine ereignisorientierte Kommunikation zwischen den einzelnen Komponenten erzeugt. Bei Bedarf reagiert der Vermittler mit weiteren auszuführenden Aufgaben und steuert so das gesamte Verhalten des Programms. Eine lose Kopplung wird dadurch begünstigt, dass die Komponenten sich untereinander nicht gegenseitig referenzieren müssen.
Ein Kriterium für die Auswahl dieses Verhaltensmusters ist, dass die ereignisorientierte Kommunikation zwischen den Komponenten des Service Managers zwar wohldefiniert erfolgt, diese jedoch sehr komplex sein kann. Einzelne Komponenten warten gegenseitig auf Ereignisse bevor sie ihren geforderten Arbeitsschritt weiter ausführen. Ohne das Verhaltensmuster würden diese Komponenten eine große Abhängigkeit bilden. Ein weiteres Kriterium stellt die einfache Anpassbarkeit des verteilten Verhaltens im Vermittler dar, denn bestehende und neue Komponenten können hier einfach miteinander verknüpft werden. Eine vollständige Erweiterung neuer Funktionalitäten des Service Managers ist durch Änderung bestehender oder direktes Hinzufügen neuer Komponenten auch ohne weitreichende Unterklassenbildung stets gewährleistet.
Im Service Manager lassen sich diese eigenständigen Komponenten in eine dreischichtige, komponentenbasierte Architektur aufteilen (vgl. Abbildung 4.3). Die Basis des Server Managers wird durch den *Service Manager Core* gebildet, der als Vermittler nach dem Verhaltensmuster Mediator arbeitet. Eine weitere Schicht wird durch die drei Komponenten *Network Utilization Inspector*, *Network Routing* und *Network Sniffer* gebildet. Wird ein Dienst durch den Service Manager auf dem Objekt ausgeführt, werden diese Komponenten aktiv geschaltet und überprüfen die momentane Ressourcenauslastung des Objektes. Eine Ressourcensignalisierung könnte in dieser Architekturschicht stattfinden, ist aber aufgrund des in Kapitel 5.1 durchgeführten SDN nicht aktiv. Mit einem Verfahren aus Kapitel 2.4

wird anhand der der topologischen Charakteristika (vgl. Kapitel 3) zyklisch überprüft, ob ein anderes bekanntes Objekt im IoT besser für die Ausführung des Dienstes geeignet ist. Ist das der Fall, übermitteln diese Komponenten ein Signal für eine mögliche Migration des Dienstes an den Service Manager Core weiter. Die dritte und oberste Softwareschicht besteht aus zwei Komponenten, die der Verwaltung einer laufenden Dienstinstanz und zum Transport des Dienstes zu anderen Objekten im IoT dient. Zusätzlich wird hier die Dienstsignalisierung erledigt. Aufgeteilt ist diese Softwareschicht in den *Service Transporter* und den *Service Handler*. Die technische Umsetzung und erwähnenswerte Besonderheiten dieser Komponenten werden in den folgenden Unterkapiteln beschrieben.

4.2.1 Service Manager Core

Der Service Manager Core ist für die Hauptfunktionalitäten des Systems zuständig und initialisiert als Vermittler des Systems alle vom Service Manager benötigten Komponenten. Konfiguriert werden kann der gesamte Service Manager über die in Tabelle 4.1 angegebenen Übergabeparameter, die der Service Manager Core verarbeitet und den anderen Komponenten bereitstellt. Darüber hinaus stellt der Service Manager Core vordefinierte

Parameter	Typ	Standardwert	Beschreibung
-a	String	./adjacency_list.json	Speicherort der Adjazenzliste.
-p	int	6001	Port des Service Transporters zur Übertragung des Services.
-s	String	./service.py	Speicherort des Services.
-r	Boolean	False	Angabe, ob der Service direkt gestartet werden soll.
-t	Boolean	False	Angabe, ob der Durchlauf ein Test ist.
-u	String	a	Liste nicht erreichbarer Hosts im Netzwerk.
-v	String	a	Liste der möglichen Serverhosts.
-m	Boolean	True	Angabe, ob der Service überhaupt migriert werden soll.
-c	int	30	Zeit zwischen zwei Migrationszyklen.
-g	float	2.0	Migrationsschwelle in Prozent [%].

Tabelle 4.1: Konfigurationsargumente zur Einstellung einer Instanz des Service Managers. Diese Argumente können dem Service Manager zu Beginn übermittelt und so die Hauptfunktionen gesteuert werden. Der Service Manager Core stellt die Argumente den anderen Komponenten bereit.

Callback-Funktionen für die Komponenten bereit. Da die Komponenten eine Referenz auf den Service Manager Core haben, können sie mithilfe der Funktionen Ereignisse auslösen und komponentenübergreifende Statusinformationen, wie die Konfigurationsparameter, abfragen. Durch diese Entkopplung wird im Service Manager eine Ausfallsicherheit gewährleistet, sodass bei auftretenden Fehlern nicht der gesamte Programmablauf zum Erliegen kommt, sondern nur einzelne Komponenten. Löst eine Komponente ein Ereignis im Service Manager Core aus, reagiert dieser unmittelbar mit den angefragten Ergebnissen. Lediglich

für die Migrationsfunktionalitäten relevante Ereignisse werden nicht sofort vom Service Manager Core abgearbeitet, sondern zunächst nur bei ihm registriert. Diese werden schließlich in einer separaten Hauptschleife in einer definierten Reihenfolge sequenzielle abgearbeitet, sodass ein dauerhaft logischer Status des Systems gewährleistet werden kann. Komponenten können somit keine inkonsistenten Zustände der Kernkomponente auslösen, wie beispielsweise parallel das Starten eines Dienstes und eine Löschung der für das Starten des Dienstes benötigten Dateien.

4.2.2 Network Sniffer

Jeder beliebige Netzwerkdienst soll als wandernder Dienst im IoT ausgeführt werden können, ohne dass explizite Anpassungen, wie die Implementierung einer API, für die Service Manager Plattform vorgenommen werden muss. Um dennoch eine Analyse über das Nutzungsverhalten des Dienstes im Netzwerk durchführen zu können, wird eine Komponente benötigt, die den Datenverkehr des Services von außen beobachten kann. Diese Aufgabe übernimmt die Network Sniffer Komponente.
Nach einem erfolgreichen Start des Dienstes durch den Service Handler (vgl. Kapitel 4.2.6) erhält der Network Sniffer die Prozess ID (PID) des gestarteten Dienstes auf dem Objekt. Es wird davon ausgegangen, dass die für den Service Manager relevanten Netzwerkdienste mit den Transportprotokollen TCP und UDP arbeiten. Aus diesem Grund bringt der Network Sniffer zunächst die vom ausführenden Dienst geöffneten Ports in Erfahrung. Dies wird durch das Diagnose-Werkzeug `netstat` mit den Argumenten `-tlnup` durchgeführt. Steht `netstat` nicht zur Verfügung, können die offenen Ports auch direkt aus den Dateien `/proc/net/tcp` und `/proc/net/udp` des dem Service Manager unterliegendem Linuxsystems ausgelesen werden. Innerhalb der Dateien enthält jede Zeile das Feld `local_address` mit der lokalen IP-Adresse und dem offenen Port in hexadezimalen Schreibweise sowie das dazugehörige Feld `inode`. Über die `inode` kann schließlich mit dem Befehl `find /proc -lname "socket:\[$INODE\]" 2> /dev/null` geprüft werden, zu welchem Prozess mit zugehöriger PID der jeweilige Eintrag gehört.
Darauf folgt eine fortwährende Untersuchung des Netzwerks auf neue ein- und ausgehende Pakete am Objekt im IoT. Dabei wird der Paket-Sniffer `tcpdump` mit den Argumenten `-ntlqxx -Q inout (-i <interface>)*` verwendet [56]. Von Interesse ist hier das Argument zu den untersuchenden Netzwerkinterfaces. Sind mehrere Netzwerkinterfaces vorhanden und soll der Dienst zum Beispiel nur in einem kabellosen Netzwerk nach IEEE 802.11 wie im MIOT-Testbed angeboten werden, so sind für die gesamte Analyse auch nur entsprechende kabellose Netzwerkinterfaces relevant. Diese können mit dem Argument `-i` bereits zum Aufnahmezeitpunkt mit `tcpdump` herausgefiltert werden, wodurch die Rechenlast des Service Managers reduziert wird.
Die entnommenen Netzwerkpakete von `tcpdump` werden jeweils in ein Wertobjekt (engl. Value Object) [57, Seite 486] der Klasse `servicemanager.utils.NetworkPacket` entpackt. Stimmt der Port im TCP- oder UDP-Header mit den zuvor ermittelten Ports und die IP-Adresse des Empfängers respektive Senders im IP-Header mit der eigenen IP-Adresse überein, wird das Netzwerkpaket über eine Callback-Funktion dem Service Manager Core als Vermittler des Service Managers weitergegeben. Auch für den Dienst irrelevante Netzwerkpakete werden über eine separate Callback-Funktion dem Service Manager Core übermittelt. So könnte die Auslastung durch den Dienst im Verhältnis zur allgemeinen Netzwerkauslas-

tung am Objekt im IoT ermittelt werden.

4.2.3 Network Router

Hinter dem Network Router verbirgt sich die Logik zum Auffinden der Objekte im IoT, die nach der momentanen Situation am besten zum Ausführen des Dienstes geeignet sind. Diese Komponente bestimmt zusammen mit dem Network Sniffer das Verfahren zur Aufnahme und Verarbeitung der Leistungsmetriken (vgl. Kapitel 3.1). Bei der Wahl einer anderen Leistungsmetrik müssen diese Komponenten angepasst werden.
Mithilfe des *Dijkstra*-Algorithmus zum Lösen von *Shortest Path*-Problemen wird zu Beginn der Ausführungszeit des Service Managers eine $m \times m$ Kostenmatrix K erstellt, bei der m die Anzahl der Knoten im aktuellen IoT definiert. Die Daten für die Berechnung werden aus der zuvor generierten Adjazenzliste entnommen (vgl. Kapitel 3.3). Sie enthält alle Knoten und deren Nachbarn des Netzwerkes. Mit den Eigenschaftsfeldern einer Kante zwischen zwei Knoten, wie zum Beispiel ETX oder Durchsatz, kann die Adjazenzliste als ein gerichteter und gewichteter Graphen angesehen werden. Die Stabilität der Leistungsmetriken bei schwankenden Netzwerkeigenschaften erlaubt eine Verwendung aus den zuvor durchgeführten Messungen [39]. Der Dijkstra-Algorithmus berechnet daraus die kürzesten Pfade zwischen allen Knoten des Netzwerkes. Die Gesamtkosten eines jeden Pfades werden in die Kostenmatrix eingetragen und für eine spätere Verwendung bereitgestellt. Durch die vorberechnete Kostenmatrix ist es möglich, das Abrufen während der Laufzeit auf $\mathcal{O}(1)$ zu beschleunigen, da für jedes Knotenpaar keine Berechnungen mehr nötig sind. Die Berechnung des am besten zum Ausführen des Dienstes geeigneten Objektes im IoT erfolgt auf Basis der letzten Netzwerkaktivitäten am Dienst. Die dafür benötigten Informationen werden durch den Network Sniffer ermittelt und vom Network Utilization Inspector bereitgestellt.
Die Informationen beinhalten die vom Dienst empfangenen Datenmengen D_i und gesendeten Datenmengen D_o jeweils in Byte und sind aufgeteilt nach den zuletzt verbundenen Hosts $h \in H$. Bei der Berechnung werden die n Knoten aus der Menge der möglichen dienstausführenden Objekte N_{Service} betrachtet und eine absteigende Rangliste R nach der Formel 4.1 erstellt. Die Funktion $g(n)$ nutzt dazu die Kostenmatrix K und die Datenmengen D_i und D_o der zuletzt verbundenen Hosts $h \in H$ und gibt die Güte eines Dienstknotens an.

$$n \in N_{\text{Service}}, n \leq m \tag{4.1a}$$

$$g(n) = \sum_{h \in H} D_{i_h} \times K_{hn} + D_{o_h} \times K_{nh} \tag{4.1b}$$

$$R = \left\{ \left(n_j, g(n_j)\right)_{j \in \{1,\ldots,|N_{\text{Service}}|\}} \middle| g(n_j) > g(n_{j+1}) \right\} \tag{4.1c}$$

Die Kostenmatrix könnte in einem großen IoT nicht erstellt werden, da sie bei jedem Hinzufügen eines neuen Objektes um $2 \times n$ weitere Elemente steigen würde. Die Verwendung der Kostenmatrix ist auch nur bedingt notwendig, denn es beschleunigt lediglich die Analyse, die je nach topologischen Verfahren anders vonstatten geht (vgl. Kapitel 2.4).

4.2.4 Network Utilization Inspector

Der Network Utilization Inspector bildet das Grundgerüst des Funktionsbereiches der Migration (vgl. Abbildung 4.2). Wird ein Dienst vom Service Manager ausgeführt, startet der Network Utilization Inspector einen wiederholenden Zeitgeber (engl. Timer) mit der in der Konfiguration angegebenen Migrationszykluszeit. Die Überlegungen zur Verwendung eines zyklischen Migrationsprozesses entstammt aus den in Kapitel 2.4 vorgestellten topologischen Verfahren der Migration. Alle während der Netzwerkanalyse vom Netzwork Sniffer gewonnenen Daten werden innerhalb des Migrationszyklus im Network Utilization Inspector protokolliert und zu allen verbundenen Clients die ein- und ausgehende Datenmengen zugeordnet und gespeichert, da diese für die nachstehenden Berechnungen relevant sind. Nach Ablauf der Migrationszykluszeit wird ein Ereignis des wiederholenden Timers ausgelöst. Der Network Utilization Inspector lässt sich von einer Funktion des Network Routers aus den gespeicherten Netzwerkdaten eine Rangliste R der bekannten Objekte im IoT berechnen, die am besten zum Ausführen des Dienstes geeignet sind. Konnten in dem Zyklus keine Netzwerkdaten gefunden und keine Rangliste aufgebaut werden – es gab also keine Anfragen von Clients an den Dienst – wird dies dem Service Manager Core mitgeteilt. Wurde jedoch eine Rangliste erstellt und das beste Objekt in der Rangliste ist ein anderes Objekt im IoT als der bisherige, wird zusätzlich noch die in der Konfiguration angegebene Migrationsschwelle überprüft. Bei der Überprüfung werden die Gütewerte der möglichen dienstausführenden Objekte aus der Rangliste R mit dem Gütewert des bisherigen dienstausführenden Objektes verglichen, die ebenfalls in der Rangliste R stehen. Alle dienstausführenden Objekte, deren Gütewerte außerhalb der prozentualen Migrationsschwelle liegen und somit besser als neue dienstausführende Objekte geeignet sind, werden als mögliche neue Kandidaten zur Ausführung des Dienstes in Betracht gezogen. Die Migrationsschwelle gilt in erster Linie der Kompensation einer ständigen Fluktuation des ausführenden Dienstknotens nach jedem Migrationszyklus. Haben zwei Knoten eine fast identische Güte, wird ein Wechsel somit unterbunden. Anhand anderer Leistungsmetriken, die die Auslastung einer Ressource beschreiben, könnte der Network Utilization Inspector zu diesem Zeitpunkt auch eine Duplikation anstelle einer Migration des Dienstes im Service Manager Core auslösen, wenn beispielsweise eine hohe Prozessorlast oder eine hohe Speicherlast im Objekt vorliegen. Wie in Kapitel 5.1 jedoch beschrieben ist, kann dies in einem virtuellen Netzwerk nicht simuliert werden, weshalb diese Funktion nur vorbereitend im Service Manager enthalten ist. Die Informationen zur Migration oder Duplikation sendet der Network Utilization Inspector schließlich durch Auslösung eines Ereignisses im Service Manager Core weiter.

4.2.5 Service Transporter

Für eine erfolgreiche Migration eines Dienstes ist der Service Transporter zuständig. Über ein internes auf TCP aufsetzendes Protokoll sichert er die Migration eines Dienstes auf ein anderes Objekt im IoT und somit den Erhalt aller für den Dienst notwendigen Daten ab. Das Protokoll einer erfolgreichen Migration eines Dienstes im IoT mit der Behandlung aller möglichen Ausnahmen ist in Abbildung A.1 illustriert. Zum einen wird durch das Protokoll unterbunden, dass ein Service Manager sich unnötigerweise zu einem Knoten migriert, wenn dieser bereits eine Dienstinstanz ausführt. Dieser Fall kann eintreten, falls in einem vorherigen Migrationszyklus bereits eine Duplikation stattfand.

Zum anderen werden durch das Protokoll die möglichen Fehlerfälle jedes einzelnen Schritts gesondert abgefangen. Da die Verbindungen bei der Übertragung in einem kabellosen Netzwerk sehr instabil sind, können in jedem Schritt Timeouts und einhergehende Verbindungsverluste auftreten. Auch könnten während der Übertragung mehrerer Dateien Timeouts entstehen, sodass eine Löschung der vorherigen Daten oder eine Neuübertragung der restlichen Daten veranlasst werden muss. Timeouts behandelt der Service Transporter an diesen Stellen immer mit einem Wert über 60 s, sodass das TCP-Protokoll die Möglichkeit einer erneuten Übertragung hat, bevor der Service Transporter abbricht [58].

4.2.6 Service Handler

Der Service Handler ist die Schnittstelle zum eigentlichen Dienst. Er verarbeitet alle direkten Interaktionen mit dem auszuführenden Dienst im IoT und den zugrundeliegenden Dateien des Dienstes auf dem Objekt. Vom Service Manager Core beauftragt, startet und stoppt der Service Handler den Dienst innerhalb der gleichen Prozessgruppe in einem separaten Subprozess wie der Service Manager. In Fehlerfällen kann der Service Handler die lokalen Dateien des Dienstes löschen.
Bei einer Veränderung der Dienstinstanz macht der Service Handler dies als Dienstsignalisierung mittels Broadcast über UDP den Clients im IoT bekannt. Übermittelt wird in diesen Fällen der Name des Netzwerkdienstes, eine im IoT eindeutige Service-Instanz ID und das aufgetretene Ereignis, ob der Netzwerkdienst gestartet oder gestoppt wurde. Die im IoT eindeutige Service-Instanz ID wird vor einer Migration durch den Service Transporter jeweils inkrementiert und hilft den Clients beim Erhalt der asynchronen Broadcasts für eine eindeutige Zuordnung der laufenden Service-Instanz. Sie wird zusammen mit dem aktuellen Instanzstatus und den Netzwerkdienstport als Konfiguration im Service Handler gespeichert.
Aufgrund hoher Verluste im verlustbehafteten Netzwerk scheitern viele verbindungslose Nachrichtenübermittlungen, wie die der Broadcast-Nachrichten der Dienstsignalisierung, sodass diese nicht alle Objekte erreichen. Fehlt den Objekten die Information einer stattgefundenen Migration, versuchen sie weiterhin Verbindungen zum alten dienstausführenden Objekt aufzubauen, die schließlich fehlschlagen. Aus diesem Grund reagiert der Service Handler auch auf sogenannte `who_is`-Anfragen, die die Objekte im IoT über Broadcast-Nachrichten senden können, falls ihnen keine aktive Dienstinstanz bekannt ist. Die Clients sollen diese Nachricht senden, wenn sie bei einem versuchten Verbindungsaufbau keine Antwort vom Dienst erhalten und schließlich Timeouts auftreten. Dieser Timeout soll im Client als ein *Fehlversuch* gewertet werden. Bevor weitere Verbindungsversuche unternommen oder verbindungslose Nachrichten an den Dienst übermittelt werden, sollen die Clients auf eine eingehende `who_is_answer`-Nachricht warten. Erhalten sie diese nicht innerhalb von 30 Sekunden, soll dieser Prozess ebenfalls als ein Fehlversuch gewertet und eine neue `who_is`-Broadcast gesendet werden. Nach maximal zehn Fehlversuchen sollen die Clients mit dem Nachrichtenversand abbrechen.

KAPITEL 5

Die Evaluation der Dienstemigration im Internet der Dinge

Im letzten Schritt werden die in den vorherigen Kapiteln beschrieben Komponenten des IoT miteinander kombiniert, indem die in Kapitel 2 vorgestellten autonomen Dienste mit dem in Kapitel 4 definierten Service Manager miteinander vereint werden, sodass sie schließlich als eine gemeinsam Plattform beurteilt und abschließend Schlüsse gezogen werden können. Zum einen soll so die Realisierbarkeit des Service Managers und seinen Funktionsweisen überprüft werden. Zum anderen kann anhand dessen die Migration autonomer Dienste im IoT durchgeführt werden, die in verschiedenen Testfällen auf ihre Machbarkeit und Effizienz evaluiert wird.
Die gesamte Evaluation findet in einer emulierten IoT-Umgebung mittels SDN statt. Die zuvor in Kapitel 3 ermittelten Leistungsmetriken sowie die Datenstruktur mit den Informationen über den Netzwerkgraph finden hier ihre Anwendung. Zunächst wird ein Vergleich von Emulationsplattformen und die Emulation des MIOT-Testbeds beschrieben.
Darauf folgt eine Definition des Aufbaus des autonomen Dienstes für die Evaluation. Hier ist auch die Beschreibung des sogenannten Performance-Client zu finden, der auf mehreren Hosts im virtuellen Netzwerk ausgeführt wird und durch seine Eigenart bei der Nachrichtenübertragung eine große Bedeutung für die Messaufnahme vergleichbarer Testfälle spielt.
Die eigentliche Evaluation findet schließlich durch die Überprüfung verschiedener Testfälle statt. Dabei werden mithilfe des autonomen Dienstes, des Service Managers und des Performance-Clients zunächst verschiedene Variablengrenzen des virtuellen Umgebung ausgemessen. Auf diese kann schließlich in den finalen Testfällen zurückgegriffen werden, sodass die definierten Szenarien in einem segmentierten IoT durch einen festgelegten Testrahmen betrachtet werden können.

5.1 Emulation durch Software-Defined Networking (SDN)

In Kapitel 3.2 ist bereits das MIOT-Testbed als auch die Aufnahme der Leistungsmetriken in diesem Testbed beschrieben. Diese aufgenommenen Leistungsmetriken sollen nun in einem weiteren Schritt als Grundlage zur Erstellung eines virtuellen Netzwerkes mittels

SDN dienen. Das SDN ist ein Ansatz zur Virtualisierung von Netzwerken, bei der die unterliegende Hardwareinfrastruktur durch Software abstrahiert wird. Die SDN-Architektur erlaubt dabei die Entkopplung der im Netzwerk vorhandenen Steuerungs- von den Weiterleitungsfunktionalitäten. Diese Funktionalitäten werden im Netzwerk in der *Steuerungsebene (engl. Control Plane)* und der *Weiterleitungsebene (engl. Forwarding Plane)* verarbeitet.
Die Steuerungsebene bestimmt durch Regeln die Signalpfade für die zu übertragenden Daten durch das unterliegende Netzwerk in der Weiterleitungsebene und optimiert diese. In einem einzelnen Router werden diese Informationen beispielsweise lokal in Routing-Tabellen festgehalten. Beim SDN wird die Steuerungsebene üblicherweise durch einen zentralen *Controller* für das gesamte Netzwerk verwaltet und gesteuert. Dies ist eine Softwarekomponente auf einem zentralen Server, der die gesamte Netzwerkstruktur bekannt ist. Durch das Hinzufügen, Ändern und Entfernen von Regeln kann der Controller sogenannte *Flows* für ganze Paketmengen zwischen einer Quelle und einem Ziel bestimmen, die er als Forwarding-Tabellen an die einzelnen Netzwerkgeräte der Weiterleitungsebene verteilt [59].
Die Weiterleitungsebene ermöglicht die Datenübertragung zwischen den Hosts, indem sie die eingehenden Pakete bearbeitet und zum nächsten Hop weiterleitet. In dieser Ebene befindet sich üblicherweise die Hardware in Form von Routern und Switches. Doch die Weiterleitungsebene kann nicht nur durch Hardware dargestellt werden. Es existieren auch rein virtuelle Lösungen, die ein Netzwerk vollständig in Software durch virtuelle Router und Switches abbilden.
Im SDN erhalten die Geräte der Weiterleitungsebene vom zentralen Controller aus der Steuerungsebene die Weiterleitungsregeln für Pakete als Forwarding-Tabellen und fragen diesen für den Umgang mit neuen Paketen an. Im Gegenzug senden sie dem Controller Informationen über ihre Auslastung und dem momentanen Datenverkehr [60]. Die Regeln einzelner Flows können beispielsweise die MAC- und IP-Adressen sowie Ports von der Quelle und dem Ziel oder auch Parameter für VLAN und QoS enthalten [59]. Die Kommunikation zwischen der Steuerungsebene und der Weiterleitungsebene kann über das standardisierte und herstellerunabhängige Protokoll OpenFlow geschehen [61]. Hersteller können dies in ihre Geräte zur Anbindung an die Controller im SDN implementieren. Dabei ist auch die Kombination von virtuellen und aus Hardware bestehenden Netzwerken möglich.

5.1.1 Auswahl der Komponenten für das SDN

Die Grundlage für ein SDN wird durch die Steuerungsebene und die Weiterleitungsebene gebildet. Die beiden Ebenen sollen zu Emulationszwecken das in Kapitel 3.2 vorgestellte MIOT-Testbed durch Softwarekomponenten darstellen und die notwendigen Funktionsanforderungen bereitstellen. Zu diesen Funktionsanforderungen zählen zum einen, dass die in der Adjazenzliste gespeicherten Leistungsmetriken programmierbar in das Simulationsnetzwerk übertragen werden können (vgl. Kapitel 3.3). Dabei soll eine hohe Flexibilität gewährleistet sein, wenn das Netzwerk durch Entfernen von Kanten aus der Adjazenzliste modifiziert werden muss. Zum anderen sollen sowohl der Controller als auch der Emulator für das virtuelle Netzwerk ohne großen Konfigurationsaufwand über das OpenFlow-Protokoll zueinander kompatibel sein. In [62] stellen die Autoren einige der Controller und Emulatoren für virtuelle Netzwerke vor.
Eine davon ist die *OpenDaylight SDN Platform*, die sowohl einen SDN-Controller als auch eine vollständige Open Source Plattform anbietet, mit der unter anderem eine Network

Resource Optimization (NRO) durchgeführt werden kann [63]. Die Plattform wird in einer eigenen Virtuelle Maschine (VM) ausgeführt. Für eine einfache Bereitstellung des Netzwerks ist diese Plattform sehr umfangreich und benötigt einen hohen Konfigurationsaufwand. Für jeden Anwendungsfall müssen sogenannte Features passend nachinstalliert werden.
Ein weiteres Projekt ist der *discrete-event network simulator ns-3* [64]. Es ist in C++ geschrieben und ermöglicht die Anbindung an Python. Der Hauptunterschied zwischen ns-3 und anderen Projekten liegt jedoch in der Ausführung der Software. Diese wird simuliert und nicht emuliert. Für eine wirkliche Adaption des MIOT-Testbeds ist eine Emulation vorzuziehen. Zwar unterstützt ns-3 die Simulation kabelloser Verbindungen und die Bewegung von Objekten innerhalb des Netzwerks, doch diese Anforderungen sind bei den ausgewählten Szenarien nicht weiter relevant. Ein weiterer Punkt ist die eingeschränkte Unterstützung für OpenFlow und SDN-Controllern. Die Simulation des gesamten Netzwerks müsste dazu vollständig innerhalb von ns-3 entwickelt werden [65].
Als eigenständiger Controller kann der Python-basierend *POX-Controller* verwendet werden [66]. Es ist einer der ersten verfügbaren SDN-Controller auf Grundlage des NOX-Controllers, die Weiterentwicklung stagniert allerdings in den letzten Jahren. Darüber hinaus muss zur Verbesserung der Performance und auch zur Erkennung von Zyklen in Graphen, wie sie im Fall des vorhandenen Mesh-Netzwerks vorkommen, dieser Controller durch die Entwicklung eigener Komponenten konfiguriert werden [67].
Ein für die Emulation des MIOT-Testbeds optimaler SDN-Controller ist der in Java geschriebene *Floodlight OpenFlow SDN-Controller* [68]. Ohne weitere Konfiguration unterstützt er das Weiterleiten von Paketen in Mesh-Netzwerken, die Zyklen enthalten, sofern diese vollständig durch OpenFlow angebunden sind. Dies ist im Fall der vollständigen Emulation des MIOT-Testbeds gegeben. Wie der Service Manager Objekte zur Ausführung des Dienstes im IoT auffindet (vgl. Kapitel 4.2.3), berechnet der Floodlight-Controller den kürzesten Pfad zwischen zwei Objekte zur Weiterleitung der Pakete. Zyklen im Netzwerk werden so erkannt.
Als Virtual Network Emulator mit Anbindung an SDNs durch OpenFlow eignet sich *Mininet* [69]. Auf einer einzelnen Maschine mit Linux-Kernel erstellt Mininet ein vollständiges, virtuelles Netzwerk, bestehend aus Hosts, Layer-2 Switches, Layer-3 Routern und den Verbindungen zwischen diesen. Es emuliert dadurch die zuvor beschriebene Weiterleitungsebene des SDN. Sie verhalten sich wie echte Geräte, die über virtuelle Ethernet-Schnittstellen miteinander verbunden sind und auf denen Programme ausgeführt werden können. Für die Emulation des MIOT-Testbeds zeichnet sich Mininet sehr flexibel bei der Anpassung der Topologie aus und verspricht ein realistisches Verhalten im Umgang mit der Ausführung von Programmen im Vergleich zu Simulationsumgebungen wie ns-3 [69]. Eine Einschränkung besteht jedoch bei Mininet dadurch, dass die Heterogenität der Objekte nicht genau abgebildet werden kann. Bis auf die Leistungsfähigkeit der CPU sind alle im virtuellen Netzwerk vorhandenen Objekte homogen. Durch die SDN-Anbindung mittels OpenFlow zeigt Mininet außerdem eine hohe Kompatibilität zu verschiedenen SDN-Controllern, wie auch zu dem bereits definierten Floodlight-Controller. Mittels einer Python-API können in Mininet alle notwendigen Geräte im Netzwerk erzeugt werden (vgl. Kapitel 5.1.2). Bereitgestellt wird Mininet als eigene Installation oder über eine eigene auf Ubuntu 14.04 LTS basierenden VM [70].

5.1.2 Emulation des MIOT-Testbeds

In diesem Kapitel wird schließlich die Emulation des MIOT-Testbeds durch den Virtual Network Emulator Mininet und dem SDN-Controller Floodlight beschrieben (vgl. Kapitel 5.1.1). Die Emulation wird über die vom Mininet-Projekt bereitgestellte VM durchgeführt. Mininet in der Version 2.2.1 ist hier bereits vorinstalliert. Die VM basiert auf Ubuntu 14.04 LTS (64bit) und wird mithilfe von VMWare Workstation Pro 12 virtualisiert. Die Basis bildet ein Computer mit einer Intel Core i5 3570k CPU, 16 GB RAM und als Betriebssystem Windows 10 Education N. Davon stehen der VM 12 GB RAM und zwei CPUs mit jeweils zwei Kernen pro CPU, insgesamt somit 4 Prozessorkerne, zur Verfügung. Zusätzlich zu Mininet wird innerhalb der VM der SDN-Controller Floodlight und einhergehend das Java Development Kit (JDK) in der Version 8 benötigt. Die Installation wird anhand der Instruktionen in [68, Getting Started] durchgeführt.
Die Ausführung des virtuellen Netzwerks erfolgt mithilfe des Skripts `mesh_topo.py` für Python in der Version 2.7.6 (vgl. Anhang A.3). Die Python-API von Mininet verfolgt dabei von der Erstellung des virtuellen Netzwerks bis hin zur Ausführung von Software auf den virtuellen Hosts einen objektorientierten Ansatz. Über ein Objekt der Klasse `mininet.net.Mininet` kann das gesamte Netzwerk erzeugt, gestartet und schließlich gestoppt werden. Die `pingAll`-Methode der Klasse erlaubt die Ausführung einer ping-Messung zur Überprüfung der Erreichbarkeit zwischen allen Hosts des Netzwerks. Bei der Erstellung des Objektes können optional eine Topologie sowie Optionsparameter für Hosts, Switches und Verbindungen (engl. Links) im Netzwerk übergeben werden. Diese einzelnen Parameter werden im Folgenden detaillierter beschrieben.
Die Topologie beinhaltet bei Mininet die Kombination aller im virtuellen Netzwerk vorhandenen Hosts, Switches und Links. Durch Ableitung der Basisklasse `mininet.topo.Topo` und durch Überschreibung der `build`-Methode kann die Klasse für die Topologie erzeugt werden. Hier wird das Netzwerk mithilfe der in Kapitel 3.3 definierten Adjazenzliste erstellt. Allen Hosts der Klasse `mininet.node.Host` wird ein Name und eine aufsteigende IPv4 im Netzwerk 10.0.0.0/24 zugewiesen. Da Hosts in Mininet nicht direkt in einem Mesh-Netzwerk miteinander verbunden werden können, erhält jeder Host einen eigenen Switch der Klasse `mininet.node.Switch`. Abhilfe könnte dazu der Fork des Mininet-Projektes Mininet-Wifi schaffen [71], da es die Simulation kabelloser Mesh-Netzwerke unterstützt. Bei der Erstellung eines Mesh-Netzwerkes können allerdings keine Verbindungen, sondern nur die absoluten Positionen von jedem Host in einem zweidimensionalen Koordinatensystem angegeben werden, die zunächst aus den relativen Kanten des Graphens in der Adjazenzliste berechnet werden müssten. Aus diesem Grund wird die Simulation mit dem originalen Mininet-Projekt durchgeführt.
Innerhalb eines Tupels sind alle Hosts mit ihrem jeweiligen Switch über einen verlustfreien Link der Klasse `mininet.link.Link` miteinander verbunden. Zwischen den Switches erfolgen schließlich die Verbindungen anhand der in der Adjazenzliste zusammengetragenen Informationen über den Durchsatz und die ETX. Ein Link der Klasse `mininet.link.TCLink` (Traffic Control Link) erlaubt eine in ihren Ressourcen eingeschränkte Verbindung durch die Parameterangabe bezüglich Durchsatzes und der aus der ETX berechneten Verlustrate als Gütemaße.
Das Netzwerk selber ist durch die Topologie bereits definiert. Doch zusätzliche Informationen über die verschiedenen Hosts, Switches und über die Links können über die weiteren Optionsparameter von Mininet definiert werden. Mit der Parameterangabe der Klasse

`mininet.node.CPULimitedHost` können die Hosts in ihrer CPU eingeschränkt werden. Abhängig von der gesamten CPU, die der Mininet-VM zur Verfügung steht, bekommt jeder Host so einen gleichgroßen Anteil. Versuche mit dieser Angabe zeigten jedoch, dass die Hosts durch die Einschränkung der CPU der Ausführung der Testfälle nicht mehr gerecht wurden, weshalb diese Option im weiteren Verlauf außer Acht gelassen wird. Darüber hinaus können weitere Leistungsmetriken einzelner Hosts wie die Größe von RAM und Speicher nicht weiter eingeschränkt werden. Für die Testfälle ist der Parameter `mininet.node.OVSSwitch` für die Switches im Netzwerk jedoch notwendig. Die Switches werden so in *Open vSwitches* umgewandelt, die zum einen das OpenFlow-Protokoll unterstützen [72]. Über eine direkte Verbindung an die Steuerungsebene sind sie ohne die Einschränkungen der TCLinks mit dem noch zu integrierenden SDN-Controller verbunden. Zum anderen ist durch die Verwendung der Open vSwitches das Spannbaum-Protokoll (engl. Spanning Tree Protokoll (STP) einsetzbar.
Ferner fehlt die Anbindung zwischen Mininet und dem SDN-Controller Floodlight. Dieser muss vor dem Start von Mininet bereits eigenständig in der Java-Laufzeitumgebung (engl. Java Runtime Environment (JRE)) ausgeführt werden. Ohne weiteren Konfigurationsaufwand stellt er standardmäßig den für das OpenFlow-Protokoll verwendeten Port 6653 bereit. Durch die Angabe eines Controllers als `mininet.node.RemoteController` kann sich Mininet schließlich in der lokalen VM mit ihm verbinden. Durch die Anbindung kennt der Floodlight-Controller zwar bereits die gesamte Topologie, Routen hat er durch das Netzwerk jedoch noch nicht gebildet. Mithilfe der Ausführung von `pingAll` durch Mininet kann der Controller die Weiterleitungsregeln für alle Switches eigenständig aufbauen, sodass die Wege zwischen allen Hosts zur Ausführungszeit der Messungen in den Testfällen garantiert bekannt sind. Sofern sie nicht durch die Verlustrate wegfallen, muss der der Controller diese nicht neu berechnen. Abschließend kann die Durchführung der Testfälle mithilfe der in Kapitel 5.2 definierten Testapplikationen beginnen.

5.2 Dienst und Performance-Client

Zur Untersuchung des Service Managers aus Kapitel 4 und der Testfälle, die in Kapitel 5.3 spezifiziert sind, werden im Wesentlichen zwei Testapplikationen benötigen. Neben einen eigenständigen Dienst, der über die Service Manager Plattform zwischen den Hosts im IoT migriert werden kann, wird auch ein Client benötigt, der die Performance des Systems in verschiedenen Testfällen testet. Die grundlegende Idee ist die Abbildung des Anwendungsfalls, bei dem mehrere Client-Objekte mit einem zentralen Dienst im IoT kommunizieren. Die Client-Objekte sollen dem zentralen Dienst in einer konstanten Rate Nachrichten senden, auf die der Dienst unverzüglich antwortet. Die Client-Objekte fassen bei jeder Kommunikation mit dem Dienst die generierten Messdaten zusammen. Der Dienst ist dabei fortwährend zustandslos. Die anfragenden Client-Objekte sind in den jeweiligen Testfällen immer die selben, während sich das dienstausführende Objekt zur Verbesserung der Leistungsmetriken (vgl. Kapitel 3.1) durch Migration des Dienstes stetig ändert. Über eine Diensterkennung in der Client-Applikation und Dienstsignalisierung in der Service Manager Plattform sollen die Clients die Migrationen der Dienstobjekte erkennen.
Der Dienst und die Clients werden über das bereits in Kapitel 5.1.2 beschriebene Skript `mesh_topo.py` nach dem Erzeugen des Netzwerks auf den virtuellen Hosts im Netzwerk von Mininet gestartet. Während alle potentiell dienstausführenden Objekte nur die Ser-

vice Manager Plattform in Betrieb nehmen, übernimmt ein Objekt im IoT zusätzlich die erste Ausführung des Dienstes. Der Dienst zur Durchführung der Testfälle ist ein einfaches Programm, das einen Server startet und einen TCP-Socket bereitstellt. Auf einem den Clients zuvor bekannten Port horcht er auf die Anfragen der Clients und führt selber proaktiv keine eigenen Aufgaben aus. Verbindet sich ein Client, startet dieser direkt eine Datenübertragung zum Dienst. Nach Erhalt der Daten antwortet der Dienst ohne weitere Verarbeitung und Verzögerung dem jeweiligen Client. Durch dieses einfache Vorgehen können die geforderten Leistungsmetriken bis zur Applikationsebene gemessen werden, ohne dass der Dienst durch einen Bearbeitungsaufwand die Ergebnisse beeinflusst.
Die Clients hingegen besitzen mehr Logik in ihrer Ausführung, starten bei der Datenübertragung zum Dienst fortwährend Messungen von Leistungsmetriken und speichern die Ergebnisse für eine nachstehende Auswertung ab. Dabei werden alle im System vorhandenen Clients mit den gleiche Ausführungsinformationen gestartet. Zur Durchführung des Nachrichtenversands in einer konstanten Rate erhalten die Clients die folgenden Informationen als Parameter:

- Die Größe der zu sendenden Nachrichten in Bytes.
- Die Anzahl aller Nachrichten, die insgesamt gesendet werden sollen.
- Die Anzahl der Nachrichten, die pro Minute gesendet werden sollen.
- Die Wartezeit bevor der Client mit seinen Messungen anfangen soll.

Dabei spielt es für das gesamte System und den jeweiligen Testfällen keine Rolle, wie die zu übertragenen Nachrichten auf der Applikationsebene aussehen. Sie könnten theoretisch durch die Aufnahme der Daten von Sensoren entstehen. Da die Objekte beim SDN jedoch nur virtuelle Hosts sind, ist es für die Untersuchungen in Kapitel 5.3 hinreichend, randomisierte Daten fester Größe von den Clients generieren zu lassen und diese durch das IoT zum Dienst zu übermitteln. Bei der Nachrichtenübermittlung in einer konstanten Rate entsteht allerdings schnell das Problem, dass sich das System bereits nach wenigen Sekunden einpendelt. Aus diesem Grund ist die Software des Performance-Client mit einer Funktion versehen, die zwar eine Unregelmäßigkeit bei der Kommunikation zwischen den Nachrichten generiert, der Nachrichtenstrom über die Zeit jedoch weiterhin konstant bleibt.
In der Funktion sei die vom Client ausgehende konstante Nachrichtenrate als R in Nachrichten pro Minute definiert. Anstelle einer iterativen Übertragung, bei der vor jeder Nachrichtenübermittlung auf den vollständigen Abschluss der vorherigen Übermittlung gewartet wird, sollen sie zyklisch mit einer durchschnittlichen Wartezeit w mit $w = 1/R$ im Minuten respektive $w = 60/R$ in Sekunden vor dem Versand jeder Nachrichten übermittelt werden. Doch so entsteht die nicht gewünschte Regelmäßigkeit in der Übertragung; die Zeit zwischen dem Absenden von zwei Nachrichten ist konstant.
Die Idee ist die Erzeugung einer Unregelmäßigkeiten dadurch, dass die Nachrichten nicht in einem Intervall mit 60 s versenden werden, sondern bereits in einem verkürzten Zeitintervall I in Sekunden mit $I < 60\ s$. Pro Minute wird durch dieses neuen Zeitintervall ein Intervall $I_{\text{frei}} = 60 - I$ Sekunden frei. Die gesamte Menge der zuvor definierten Nachrichten, die pro Minute hätten versandt werden sollen, ist so bereits nach I Sekunden vom Client versendet worden. Die dabei frei gewordene Zeit wurde der Wartezeit w zwischen den Nachrichten entnommen. Aus diesem Grund verkürzt sich die durchschnittliche Wartezeit w_{neu} zwischen den einzelnen Nachrichten um den Faktor $I/60$ auf $w_{\text{neu}} = I/R$.
Diese Sekunden könnten am Ende jeden Intervalls in einer Sequenz als Wartezeit verwendet

werden. Doch stattdessen wird bei jeder Nachricht der Anteil der frei gewordenen durchschnittliche Wartezeit w_{frei} mit $w_{\text{frei}} = I_{\text{frei}}/R = (60 - I)/R$ auf ein Pausenkonto gutgeschrieben. Das Pausenkonto beinhaltet die eingesparte Wartezeit aller bereits versendeten Nachrichten und muss innerhalb der gesamten Ausführung, jedoch zu einem beliebigen Zeitpunkt aufgebraucht werden, da es sonst eine Erhöhung der Nachrichtenrate R zur Folge hat. Daher wird das Pausenkonto mit einer frei festgelegten Wahrscheinlichkeit P_{Leerung}, hier mit $P_{\text{Leerung}} = 1/(2 \times R)$, vor dem Versand einer Nachricht vollständig geleert. So wird bei den Clients zu unterschiedlichen Zeitpunkten die Leerung ihres Pausenkontos angestoßen und insgesamt ein unregelmäßiger Nachrichtenstrom am Dienst erzeugt, der abhängig von der vorherigen konstanten Nachrichtenrate R ist.
Eine weitere Fluktuation ist außerdem bei der neuen durchschnittlichen Wartezeit w_{neu} vorhanden, die eine weitere Variabilität in die Nachrichtenrate bringt. Anstelle der Berechnung eines festen Wertes für alle Wartezeiten, wird ein zufälliger Wert aus dem Intervall $[0.5 \times w_{\text{neu}}, 1.5 \times w_{\text{neu}}]$ gewählt. Dies betrifft allerdings nur die tatsächlich zu wartende Zeit, für die Gutschrift auf das Pausenkonto wird weiterhin die durchschnittliche Wartezeit w_{neu} verwendet.
Neben dieser unregelmäßigen Nachrichtenübermittlung besitzt die Client-Software eine weitere Funktion, die für die Diensterkennung zuständig ist. Dabei horchen die Clients über einen UDP-Port auf neue Broadcast-Mitteilungen der dienstausführenden Objekte, die von der Service Manager Plattform versendet werden (vgl. Kapitel 4.2.6). Da die Verbindungen im Netzwerk verlustbehaftet sind, können die Broadcast-Mitteilungen verloren gehen und die Clients die Mitteilungen nicht erhalten. Zur Lösung dieses Problems besitzen sie einen selbstorganisierenden Mechanismus, der bei mehrfach erfolglosen Verbindungsaufbau zum Dienst sogenannte `who_is`-Nachrichten per Broadcast sendet. Alle Clients und der dienstausführende Service Manager antworten über eine `who_is_answer`-Nachricht mit der Adresse, unter dem der Dienst erreichbar ist. Dieses System aus Dienst und den Performance-Clients sorgt damit für einen reibungslosen Ablauf der Messungen in Kapitel 5.3.

5.3 Messungen und Auswertung verschiedener Testfälle

In verschiedenen Testfällen sollen abschließende Messungen überprüfen, ob die Bereitstellung eines Dienstes über die in Kapitel 4 vorgestellten Service Manager Plattform aus Sicht der Clients eine Verbesserung der Latenzzeit im virtuellen Netzwerk zur Folge hat. Die Latenzzeit ist einer der wichtigsten Leistungsmetrik (vgl. Kapitel 3.1.5), die bei dem Fog Computing und der einhergehenden Analyse beobachtet werden kann und im Folgenden bei den verschiedenen Testfällen verglichen wird [3, 15, 16, 18]. Die Messungen erfolgen auf den in Kapitel 5.2 vorgestellten Dienst und Performance-Client, bei denen die Clients mindestens 2.000 Messungen durchführen. In einigen Testreihen ergeben sich Abweichungen in der Anzahl der Durchführungen aufgrund einer zu geringen Messdauer; sie sind in den nachstehenden Unterkapiteln jeweils gekennzeichnet. Eine weitere Auswertung aller im Netzwerk vorhandenen Service Manager erfolgt im Bezug auf die Migrationen des Dienstes. Der Vergleich erfolgt hierbei auf Basis aller möglichen Migrationen und wie sich die Service Manager diesbezüglich verhalten haben. Unterschieden wird hier zwischen vollständig abgeschlossenen Migrationen, abgelehnten Migrationen und Migrationsfehlern. Die Ergebnisse aller Testfälle wurden dabei mit dem Python-Skript `final_evaluation.py` erstellt.
Da bisher mit dem in Kapitel 5.1 von Mininet erstellten virtuellen Netzwerk noch keine

Messungen durchgeführt wurden, werden zunächst dessen Grenzen anhand der folgenden Variablen in vier Testfällen überprüft. In ihrem jeweiligen Testfall werden sie näher analysiert:

1. Gesendete Nachrichten pro Minute und Host
2. Migrationszykluszeit
3. Gesendete Nachrichtengröße auf Applikationsebene
4. Anzahl Migrationen bei konstanter Datenrate

Neben der Überprüfung der Grenzen erfolgt auch eine sukzessive Festlegung der Variablen auf einen jeweiligen Standardwert, wodurch die Messungen der verschiedenen Testfälle auch untereinander vergleichbar bleiben. Zu beachten ist hier, dass die konstante Datenrate in Punkt 4 keine eigene Variable ist, sondern eine Kombination aus den gesendeten Nachrichten pro Minute und Host sowie der Größe dieser Nachrichten. Das den Messungen zugrundeliegende Netzwerk wird aus der in Kapitel 3.4 final erstellten Adjazenzliste erzeugt (vgl. Testfälle 1-4 und Abbildungen 5.1-5.8). Der dazugehörige Graph enthält insgesamt 41 Objekte und 634 Verbindungen, von denen alle einen Service Manager bereitstellen und somit potentielle Kandidaten für die Ausführung einer Dienstinstanz sind (vgl. Abbildung A.2). 25% von diesen 41 Objekten, aufgerundet auf insgesamt 11 Objekte, werden als Clients verwendet. Schrittweise werden die einzelnen Variablen in den jeweiligen Testfällen solange weiter erhöht, bis sie an eine maximale Grenze stoßen. Eine Voruntersuchung zeigte, dass sich hier eine weitere Erhöhung der Client-Anzahl sehr negativ auf die Variablen auswirkt. Die obere Grenze wurde dabei so schnell erreicht, dass in keinem Testfall vergleichbare Messungen vorgenommen werden konnten. Die Grenzen zeichnen sich bei den Messungen in Mininet primär durch eine der nachstehenden drei verschiedene Merkmalen aus:

- Trotz Erhöhung der jeweiligen Variable können bei den Messungen keine Veränderungen in der gemessenen Leistungsmetrik erkannt werden.
- Die Erhöhung der jeweiligen Variable erzeugt eine so große Anfragemenge, dass die virtuellen Hosts im Netzwerk schließlich überlastet sind.
- Der Floodlight SDN-Controller kann die große Menge anfragender Hosts bei dem Auffinden von Routen nicht mehr standhalten und stürzt dauerhaft ab.

Bei der Ausmessung der maximalen Grenzen in dem großen Netzwerk stört die Verlustrate der Verbindungen, die durch die Angabe von ETX bei der Verwendung von `mininet.link.TCLink` übergeben wird (vgl. Kapitel 5.1.2). Da die Grenzen zunächst nicht von der Verlustrate abhängig sind, sondern durch die maximale Leistungsfähigkeit von Mininet und dem SDN-Controller bestimmt werden, wurde die Verlustrate für die ersten Messungen zunächst unterbunden. Das Auffinden eines geeigneten, dienstausführenden Objektes durch den Service Manager (vgl. Kapitel 4.2.3) erfolgt daher in den ersten Testfällen nicht auf der Basis der Leistungsmetrik ETX, sondern auf den Durchsatz, der ebenfalls in der Adjazenzliste zu finden ist. Dadurch sind die Grenzen weiterhin aussagekräftig, nur die Migrationen werden anhand eines anderen Kriteriums durchgeführt.
Ohne die Messungen der vier Variablengrenzen könnte kein Testrahmen gebildet werden, der für die abschließenden Messreihen relevant ist (vgl. Testfälle 5-6 und Abbildungen 5.9-5.12). Bei diesen Messreihen werden abschließend die bereits aufgestellten Szenarien aus Kapitel 2.3 miteinander verglichen. Diese sind nachstehend nochmals erläutert:

Szenario 1: Der Dienst wandert im IoT-Segment A.

Szenario 1a: Der Dienst wandert im IoT-Segment A mit dedizierten Dienstobjekten.

Szenario 2: Der Dienst befindet sich am Gateway A.

Szenario 3: Der Dienst befindet sich bei einem IoT-Dienstanbieter im Internet.

Szenario 4: Der Dienst befindet sich am Gateway des IoT-Segments B.

Szenario 5: Der Dienst wandert im IoT-Segment B.

Für die Testfälle 5-6 wird jedoch eine Segmentierung des IoT benötigt, bei der zwei getrennte IoT-Segmente durch ein Internet miteinander verbunden sind. Das daraus entstehende Netzwerk soll schließlich bei der Emulation durch Mininet Verwendung finden. Hier wird die Bedeutung der Adjazenzliste evident. Das Netzwerk kann durch die Entfernung von Kanten aus der Adjazenzliste respektive von Nachbarn aus den Listen innerhalb der Adjazenzliste sehr einfach beschnitten werden (vgl. Kapitel 3.3). Aus dem Graphen in Abbildung A.2 mit 41 Knoten wird zunächst ein Knoten extrahiert, sodass eine gerade Anzahl übriger Knoten entsteht. Alle Kanten von und zu diesem Knoten werden für das Vorhaben entfernt. Die übrigen 40 Knoten werden anschließend zufällig auf zwei Teilgraphen aufgeteilt und alle Kanten zwischen den beiden Teilgraphen entfernt. Diese bilden schließlich die zwei voneinander getrennten IoT-Segmente. Zur Emulation der zwei Gateway-Knoten werden aus den beiden Teilgraphen jeweils ein Knoten ausgewählt. Durch erneutes Einfügung des zuvor extrahierten Knotens werden die beiden Gateway-Knoten über jeweils eine Kante miteinander verbunden. Aufgrund der Annahme, dass die Gateways mit dem Internet verbunden sind, repräsentiert dieser neue Knoten das Internet als ein fester Knotenpunkt zwischen den IoT-Segmenten. Selbstverständlich besteht das reale Internet nicht nur aus einem Knoten. Doch wird zur Emulationsmöglichkeit angenommen, dass die Verbindungen im globalen Internet bereits so weit ausgereift sind, dass mögliche Effekte durch die IoT-Segmentierung entstehen und nicht durch Verluste im Internet. Die Kanten zwischen den Gateway-Knoten und dem Internetknoten stellen die Verbindungen zum Internet dar. Zur Abbildung eines realistischen Netzwerks werden sie mit einem Durchsatz über 50 Mbit/s in beide Richtungen ohne Verlustrate angenommen. Die Abbildung A.3 zeigt den beschnittenen Graphen, der dem gewünschten Netzwerk zugrunde liegt.

Im ersten Testfall werden die gesendeten Nachrichten pro Minute und Host solange weiter gesteigert, bis das System eine der drei Grenzen erreicht hat (vgl. Abbildungen 5.1-5.2). Aufgrund der verwendeten TCP-Verbindungen und einer MTU von 1.500 Bytes im Netzwerk ist die Nachrichtengröße des Performance-Clients auf 1.460 Bytes festgelegt. Dies entspricht der MSS, sodass eine Nachricht durch genau ein Netzwerkpaket abgebildet werden kann. Zur Erlangung eines aussagekräftigen Ergebnisses senden die Hosts respektive Performance-Clients bei den ersten drei Messreihen insgesamt 2.000 Nachrichten. Ab 120 Nachrichten pro Minute und Host erhöht sich die Gesamtanzahl der Nachrichten auf 10.000. Dies führt zur Verlängerung ihrer Ausführungszeit und zu einer Angleichung der Ausführungszeit zwischen allen Messreihen. Ohne die Angleichung läge die Ausführungszeit bei den letzten Messreihen jeweils unter 10 Minuten. Die Abbildung 5.1 stellt die Ergebnisse des Testfalls mit Boxplots dar. Zur Darstellung werden die Ergebnisse pro Messreihe aufsteigend sortiert. Innerhalb der Kästen befinden sich all diejenigen Ergebnisse einer Messreihe, die größer als die nied-

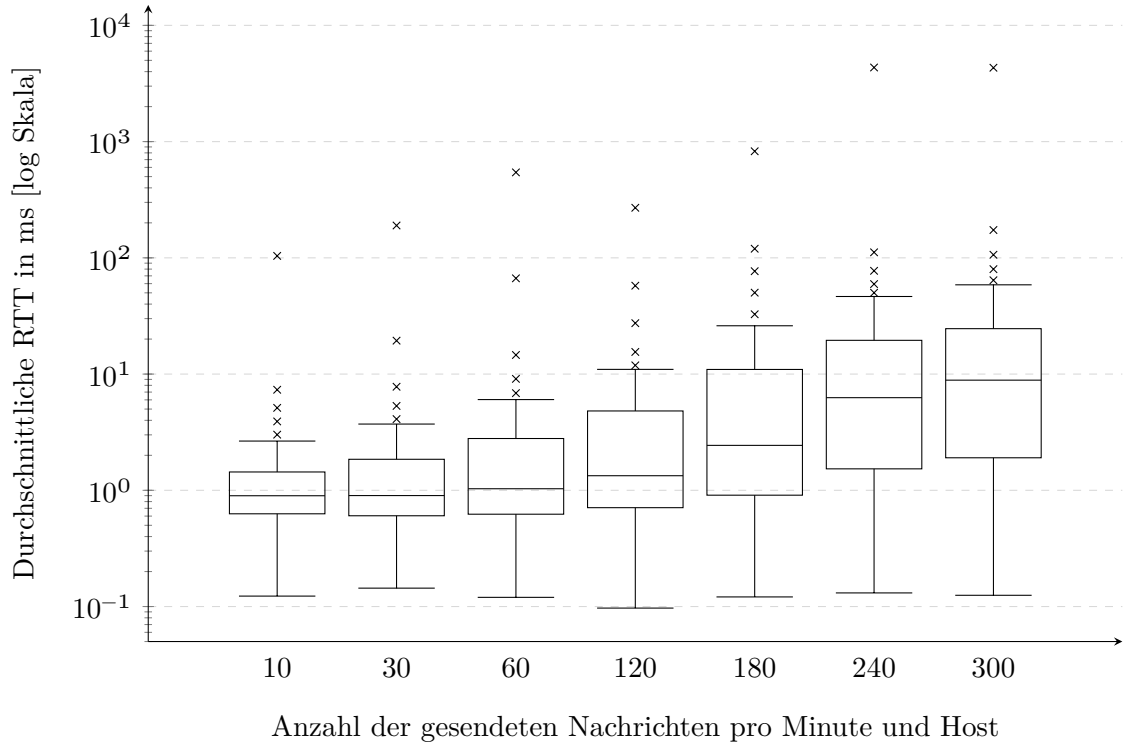


Abbildung 5.1: Vergleich der RTT gegenüber der gesendeten Nachrichten pro Minute und Host. In dem Testfall werden die gesendeten Nachrichten pro Minute und Host immer weiter erhöht, bis das SDN-System schließlich an seine Grenzen kommt.

rigsten 25% (unteres Quartil) und kleiner als die höchsten 25% (oberes Quartil) sind. Die Mittelbalken in den Kästen kennzeichnen als Median der Messreihe die 50%-Marke. Die beiden Antennen stellen jeweils das 1,5-Fache des Interquartilsabstands dar und werden als „Whisker“ bezeichnet. Falls in den Messdaten kein Wert an den Whisker-Positionen aufzufinden ist, werden die Whisker auf den zum Kasten nächsten Messpunkt festgelegt. Die Punkte außerhalb der 1,5-Fachen des Interquartilsabstände stellen die Ausreißer der jeweiligen Messreihe dar. In der Abbildung ist die logarithmische Skala zu beachten.
Beginnend bei 10 Nachrichten pro Minute und Host wird bei den Messungen das Maximum bei 300 Nachrichten pro Minute und Host erreicht, die die 11 Clients zum Dienst senden. Eine weitere Erhöhung verursacht einen Absturz des Floodlight SDN-Controllers mit einer `java.lang.NullPointerException` in der `link discovery update loop`, von der er sich nicht mehr erholen kann. Diese Experimente müssen daher abgebrochen werden. Während die Erhöhung von 10 auf bis 60 Nachrichten pro Minute und Host kaum Veränderungen in dem Median der RTT zwischen Client und Dienst zeigen, steigt dieser ab 120 Nachrichten pro Minute und Hosts bereits auf das Doppelte an und erhöht sich in den nächsten Messreihen weiter. Die unteren Whisker repräsentieren in der Abbildung 5.1 die minimalen Werte der Messreihen. Diese sind bei allen Messreihen sehr gering und fast identisch, da der Service Manager den Dienst auch auf einen Host migrieren kann, auf dem der Performance-Client ausgeführt wird. So müssen bei diesen Messungen keine externen Verbindungen in der Kommunikation zwischen Performance-Client und Dienst aufgebaut werden.
Die Abbildung 5.2 zeigt die zusammengefassten Resultate der Migrationen aller dienstaus-

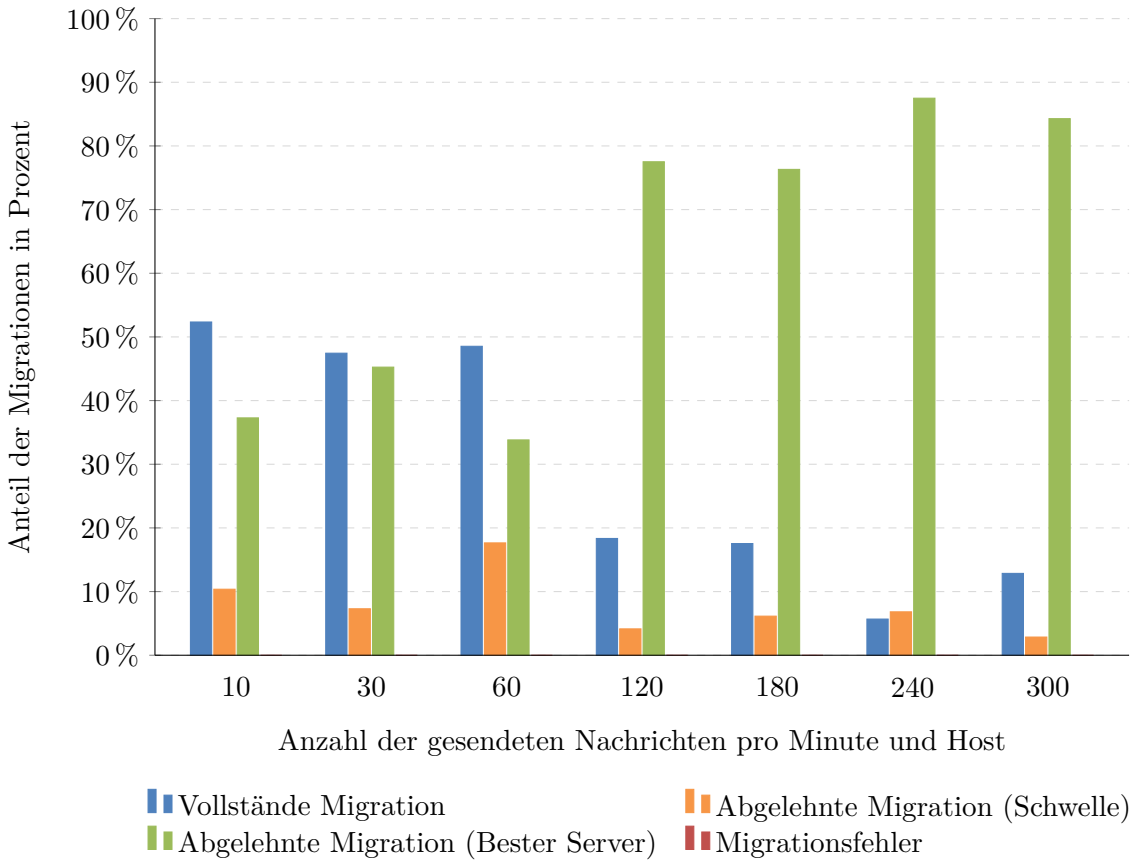


Abbildung 5.2: Vergleich der Migrationen gegenüber der gesendeten Nachrichten pro Minute und Host. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

führenden Objekte respektive Service Manager, bei denen der Migrationszyklus auf 30 Sekunden festgelegt ist. Die ersten beiden Messreihen zeigen eine geringe Erfolgsrate bei der Bereitstellung des Dienstes auf der richtigen Position im Netzwerk. Durch die wenigen Nachrichten, die innerhalb eines Migrationszyklus gesendet werden, wird der Dienst oft von den Service Manager zwischen den Objekten migriert. Es folgt eine stetige Steigerung der Erfolgsrate je mehr Nachrichten pro Minute und Host gesendet werden. Mögliche Migrationen werden vom Service Manager immer öfters abgelehnt, was auf der Tatsache beruht, dass der dienstausführende Host laut der Überprüfung bereits der besten Host für die Bereitstellung ist oder der Schwellwert von 2% noch nicht überschritten wurde, um eine Migration durchzuführen (vgl. Kapitel 4.2.3). Die Messreihe mit 60 Nachrichten pro Minute und Host stellt einen kleinen Ausreißer in den Messungen dar, denn hier ist im Vergleich zu den vorherigen Messreihen vor allem eine Steigerung in der Ablehnung aufgrund der Schwelle zu vermerken. Die Steigerung der Ablehnung von Migrationen bei den weiteren Messreihen zeigen, dass die Bereitstellung eines Dienstes sehr von der Anzahl der gesendeten Nachrichten pro Minute und Host abhängig ist, zumindest solange innerhalb eines Zyklus alle Clients gleichmäßige Datenmengen und Anfragen senden. Aufgrund dieser Anwendungsabhängigkeit wird jedoch der Wert 10 als Standard für die Anzahl der gesendeten Nachrichten pro Minute und Host in

den weiteren Messungen festgehalten. Durch die geringe Nachrichtenrate und der Gesamtanzahl von 2.000 Nachricht führen die Service Manager mehr wünschenswerte Migrationen durch und erlauben einen Test aller Komponenten des Service Managers. Außerdem werden die einzelnen Messreihen mit diesen Werten jeweils eine Zeit von über drei Stunden in Anspruch nehmen. So kann sichergestellt werden, dass sich das virtuelle Netzwerk durch die anfängliche Erstellung von Routen eingependelt hat.

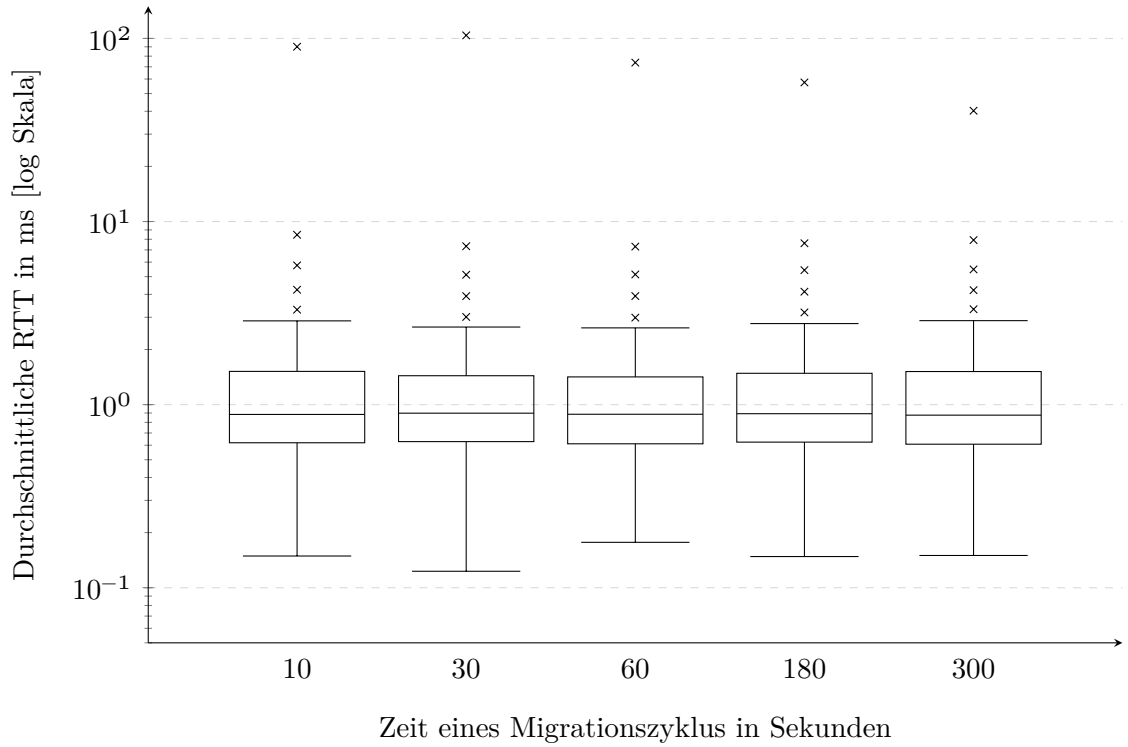


Abbildung 5.3: Vergleich der RTT gegenüber der Zeit eines Migrationszyklus. In dem Testfall wird die Zeit eines Migrationszyklus immer weiter erhöht, bis keine Veränderungen mehr festzustellen sind.

Im zweiten Testfall soll die Zeit eines Migrationszyklus des Service Managers immer weiter erhöht werden, während die anderen Variablen festgelegt sind (vgl. Abbildungen 5.3-5.4). Wie zuvor im ersten Testfall festgehalten wurde, sollen die 11 Performance-Clients mit einer konstanten Nachrichtenrate 10 Nachrichten pro Minute an den Dienst senden. Die übertragene Datenmenge auf Applikationsebene liegt für jede Nachricht bei 1.460 Byte. Insgesamt werden wieder 2.000 Nachrichten von einem Performance-Client pro Messreihe übermittelt. Für die Vergleichbarkeit entspricht hier die zweite Messreihe mit einem Migrationszyklus von 30 Sekunden der ersten Messreihe aus dem ersten Testfall (vgl. Abbildung 5.1).
Die Abbildung 5.3 zeigt die Ergebnisse der Clients während der Nachrichtenübermittlung. Der Migrationszyklus wurde zwischen den Messreihen von 10 Sekunden bis auf 300 Sekunden immer weiter erhöht. Durch die konstante Nachrichtenrate zeigt sich aus der Sicht der Clients auf die RTT insgesamt nur eine moderate Veränderungen zwischen den Messreihen, lediglich eine leichte Minimierung bei den Ausreißern ist feststellen. Diese Minimierungen

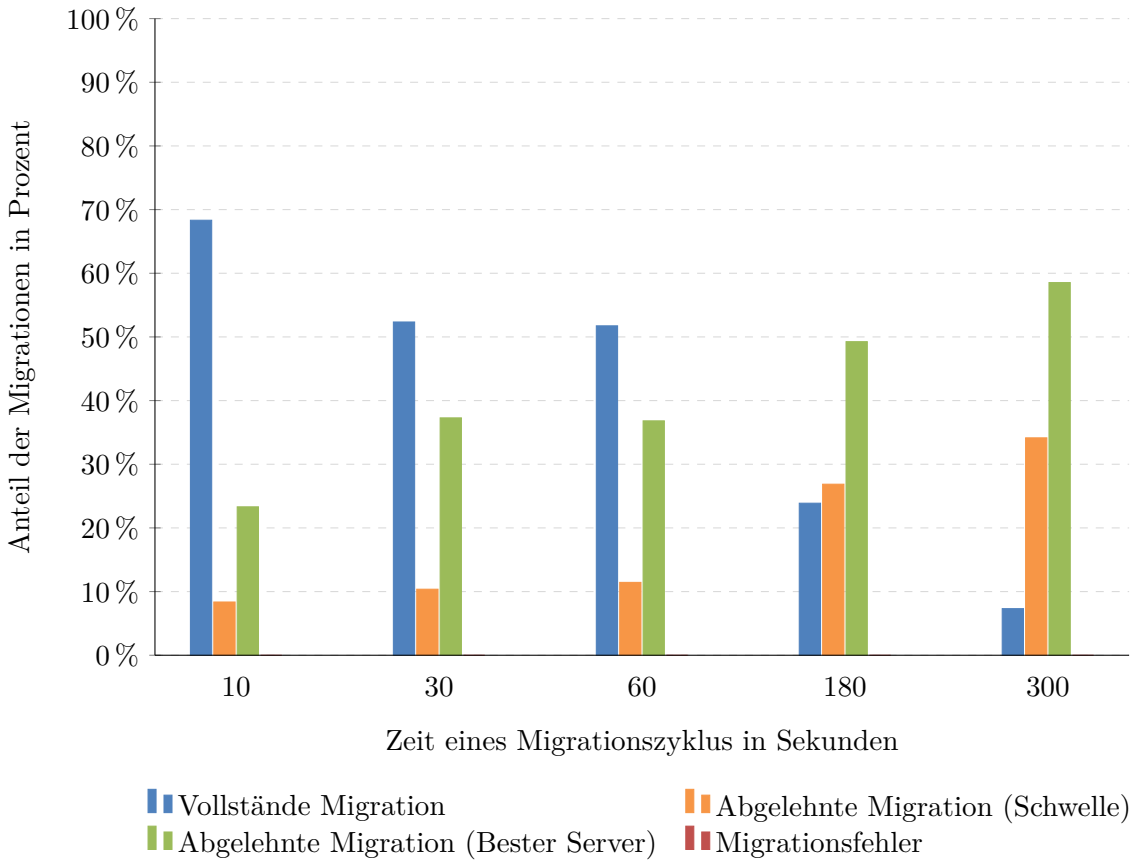


Abbildung 5.4: Vergleich der Migrationen gegenüber der Zeit eines Migrationszyklus. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

lassen sich jedoch auch durch Schwankungen zwischen den einzelnen Messreihen erklären. Insgesamt haben die Änderungen des Migrationszyklus in diesem Testfall kaum einen Einfluss auf die RTT der Nachrichten. Zum einen ist durch die geringe Nachrichtenrate und der zur MSS passenden Nachrichtengröße das Netzwerk nicht ausgelastet. Zum anderen muss beachtet werden, dass das hier untersuchte Netzwerk nicht verlustbehaftet ist und alle Nachrichten garantiert den Dienst erreichen. Bei einem verlustfreien Netzwerk dieser Größe und geringer Nachrichtengröße sind die möglichen Routen zwischen den Clients und den Diensten sehr ausgeglichen. In einem Netzwerk mit verlustbehafteten Kanten könnte das Ergebnis hingegen anders ausfallen (vgl. Testfälle 5-6 und Abbildungen 5.9-5.12).
Im Vergleich zur Betrachtung aus der Sicht der Clients ist die Veränderung des Migrationszyklus aus Sicht der Service Manager deutlich wahrnehmbar, wie in Abbildung 5.4 zu sehen ist. Durch die schrittweise Erhöhung des Migrationszyklus wird zunächst die Gesamtanzahl möglicher Migrationen deutlich von 1.184 bis auf ein Minimum von 41 verringert. Im Vergleich zum ersten Testfall zeigt sich weiterhin auch ein signifikanter Trend in den gemessenen Ergebnissen. Mit steigender Zeit eines Migrationszyklus erhöht sich prozentual die Ablehnung einer Migration aufgrund der Schwelle, die bei allen Testfällen und Messreihen 2% beträgt. Dies deutet darauf hin, dass in dem Netzwerk zwei Objekte existieren, die sich beide mittig (im Sinne der Leistungsmetrik) zwischen den Clients befinden. Die

Schwelle verhindert eine fluktuierende Migration des Dienstes zwischen den beiden Objekten (vgl. 4.2.3). Darüber hinaus hat die Erhöhung des Migrationszyklus zur Folge, dass das Auffinden des besten Servers immer präziser wird, denn Migrationen werden prozentual nicht mehr so häufig durchgeführt. Diese Erkenntnis ist einfach zu begründen. Der erste dienstausführende Service Manager sammelt mit einem größeren Migrationszyklus bereits so viele auswertbare Informationen über das Verhalten der anfragenden Clients, dass er nach einem Zyklus bereits ein optimales Gesamtbild über diese besitzt. Würden die Clients mit einer geringeren Nachrichtenrate als 10 Nachrichten pro Minute senden, würde sich das Phänomen wieder auflösen und der Service Manager würde prozentual öfters den Dienst erneut migrieren. Dies zeigt, dass die Zeit eines Migrationszyklus der Anwendungsart des Dienstes entsprechend optimiert werden könnte. Anwendungen mit einer geringen Übertragungsrate von Client-Nachrichten benötigen bei dem Service Manager einen größeren Migrationszyklus ihrer Dienste. Bei einer größeren Nachrichtenrate kann die Zeit eines Migrationszyklus entsprechend geringer ausfallen, da binnen eines Zyklus genug auswertbare Informationen gesammelt werden können. Wie bereits im ersten Testfall wird die Zeit für einen Migrationszyklus für die nächsten Testfälle weiterhin bei 30 Sekunden festgehalten.

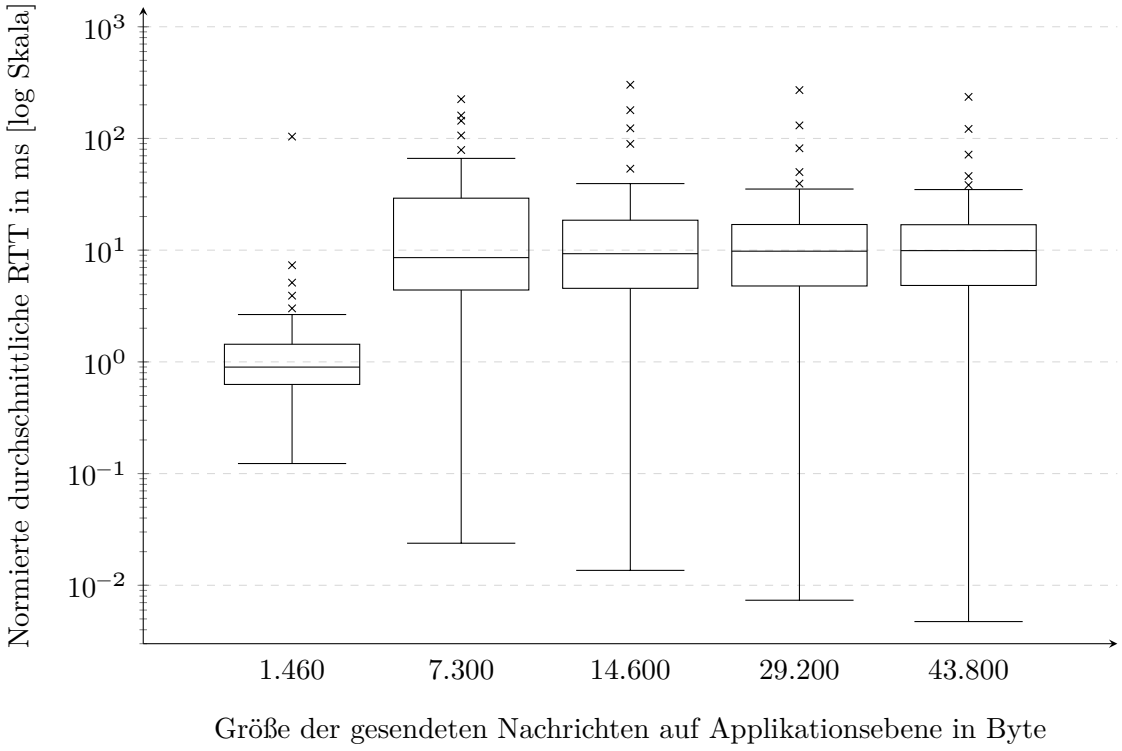


Abbildung 5.5: Vergleich der RTT gegenüber der Nachrichtengröße auf Applikationsebene. In dem Testfall wird die Größe der von den Clients gesendeten Nachrichten immer weiter erhöht, bis keine Veränderungen mehr festzustellen sind. Die Ergebnisse der Messreihen, bei denen pro Nachrichten mehr als 1.460 Byte gesendet werden, sind für eine Vergleichbarkeit mit dem jeweiligen Faktor 1.460/Nachrichtengröße normiert dargestellt.

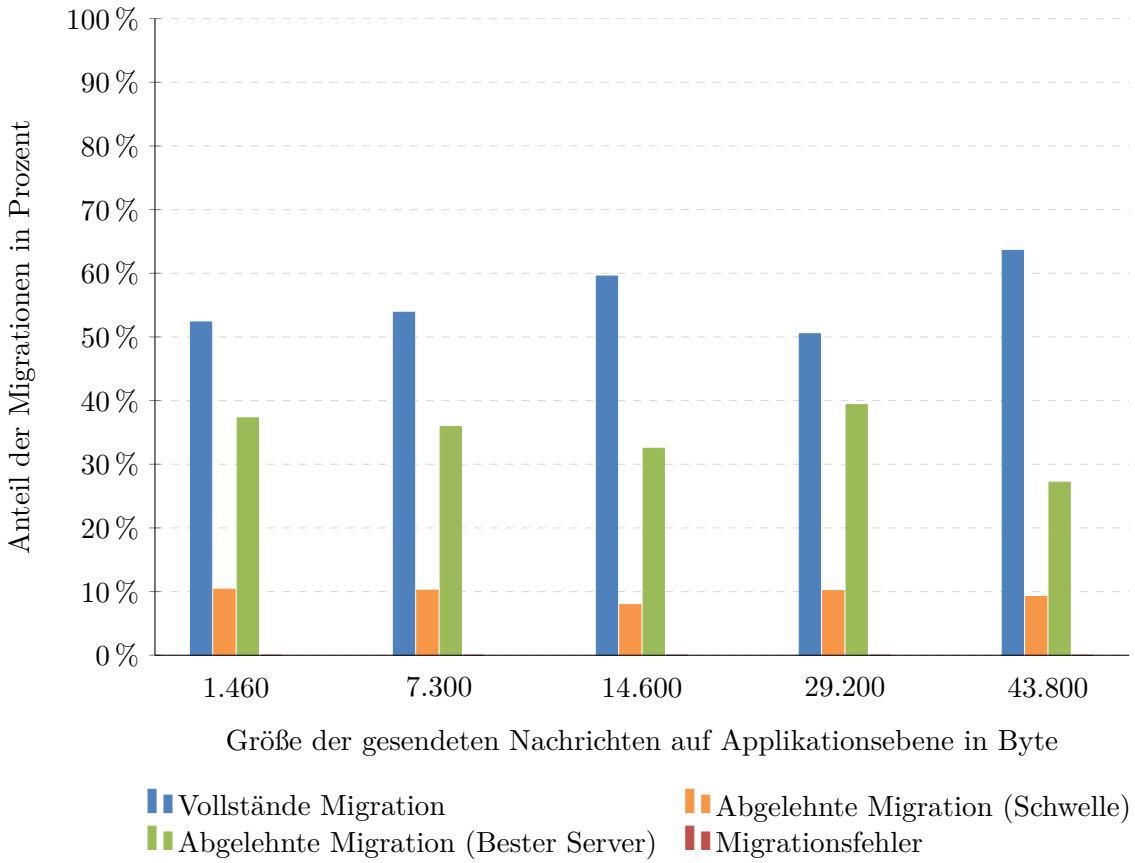


Abbildung 5.6: Vergleich der Migrationen gegenüber der Größe der gesendeten Nachrichten auf Applikationsebene. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

Der dritte Testfall vergleicht die gesendete Nachrichtengröße auf Applikationsebene, indem die sie zwischen den Messreihen ausgehend von der MSS mit 1.460 Bytes um die Faktoren 5, 10, 20 und 30 schrittweise erhöht werden (vgl. Abbildungen 5.5-5.6). Die 11 Clients senden die insgesamt 2.000 Nachrichten bei allen Messreihen mit einer konstanten Nachrichtenrate von 10 Nachrichten pro Minute. Die Service Manager auf den dienstausführenden Objekten führen die Migrationen in einem Zyklus über 30 Sekunden durch. Aufgrund der Vergleichbarkeit entspricht die erste Messreihe des Testfalls wieder den Messreihen aus den vorherigen Testfällen mit 1.460 Bytes Nachrichten bei 10 Nachrichten pro Minute und Host sowie 30 Sekunden Zeit eines Migrationszyklus.
Die Abbildung 5.5 zeigt die Ergebnisse der Clients des Testfalls. Damit die Messreihen auch untereinander vergleichbar sind, ist bei den Messreihen jeweils eine Normierung um ihren gegebenen Faktor vorgenommen worden. Die Ergebnisse der Messreihen stellen somit die normierte RTT dar, wie sie für ein Datenpaket zu erwarten wäre. Ohne Normierung wäre erkennbar, dass die Erhöhung der gesendeten Nachrichtengröße eine höhere durchschnittliche RTT zur Folge hat. Diese wäre in der Abbildung linear zum Faktor der jeweiligen Messreihe zu verzeichnen. Durch die Normierung kann hingegen eine gleichermaßen konstante Differenz in der RTT zwischen der Messreihe, bei der die Nachrichtengröße 1.460 Bytes

beträgt, und den anderen Messreihen erfasst werden. Diese konstante Differenz ist durch entstehende Mehrkosten begründbar. Eine Nachrichten, die mehr Daten als die MSS beinhaltet, muss in mehrere kleinere Pakete aufgeteilt werden. Diese kleineren Pakete werden mit eigenen Protokollheadern einzeln durch das Netzwerk verschickt und schließlich wieder auf Applikationsebene zusammengefügt mit der Folge, dass um den jeweiligen Faktor der Messreihen zusätzliche Daten über das Netzwerk gesendet werden müssen. Zu beachten ist jedoch der am unteren Whisker erkennbare Fehler durch die Normierung. Wie bereits bei den vorherigen Testfällen, wird das Minimum in der RTT durch die Bereitstellung des Dienstes auf einem Host erreicht, auf dem auch ein Client ausgeführt wird. Dieses Minimum wäre eigentlich die untere Schranke in der RTT, die durch die Normierung jedoch ebenfalls abgesenkt wird.
Die Abbildung 5.6 vergleicht die Migrationen der Service Manager im dritten Testfall. Da die Clients aufgrund der konstanten Nachrichtenrate durchschnittlich gleichmäßig senden, ist hier kaum eine Veränderung zwischen den Messreihen zu vernehmen. Die Wahrscheinlichkeit, dass ein Service Manager den Dienst auf ein anderes dienstausführendes Objekt migriert, liegt bei allen Messreihen ungefähr zwischen 50% und 65%. Daraus ist zu schließen, dass die Migrationen der Service Manager nicht von der Last im gesamten Netzwerk abhängen. Dies ist sehr positiv, da es bedeutet, dass bei konstanter Erhöhung der Nachrichtengröße die Service Manager gleich reagieren. Nur bei einer Veränderung des Anfrageschemas durch die Clients könnte sich das Migrationsverhalten ändern. Die Veränderung des Anfrageschemas könnte beispielsweise durch die Einführung einer größeren Unregelmäßigkeit in dem Nachrichtenversand zwischen den Clients und dem Dienst geschaffen werden. Dies könnte durch Veränderung der Wahrscheinlichkeit zur Leerung des Pausenkontos P_{Leerung} (vgl. 5.2) geschehen, durch eine Anpassung des Clientverhaltens, wie es im vierten Testfall zu sehen ist, oder aber durch eine Erhöhung der gesendeten Nachrichtengröße bei einzelnen Clients. Für die abschließenden Testfälle 5 und 6, bei denen die Verbindungen im Netzwerk verlustbehaftet sind, soll die Nachrichtengröße auf 14.600 Bytes angehoben werden. Zum einen wird so der Fall getestet, dass pro Nachricht mehr als ein Paket übermittelt wird. Zum anderen ist das Netzwerk durch die erhöhte Nachrichtengröße im Vergleich zu kleineren Nachrichten ausgelasteter.

In dem vierten und damit letztem Testfall, das auf das Netzwerk ohne Verlustrate aufbaut, wird eine Kombination aus dem ersten und dem dritten Testfall dargestellt (vgl. Abbildungen 5.7-5.8). Dabei wird der Ansatz verfolgt, dass über alle Messreihen hinweg ein gleichbleibender Durchsatz bestehen bleibt. Das bedeutet, dass bei Verringerung der Nachrichtengröße die Anzahl Nachrichten pro Minute und Host entsprechend erhöht werden muss. Umgekehrt gilt dies bei Erhöhung der Nachrichtengröße. Der durchschnittliche Durchsatz liegt in dem Testfall bei 14.600 Bytes pro Minute und Host. Die Zeit eines Migrationszyklus liegt wie in den anderen Testfällen ebenfalls bei 30 Sekunden. Die Referenzmessreihe mit 1.460 Bytes als Nachrichtengröße und 10 Nachrichten pro Minute und Host ist in diesem Testfall in der zweite Messreihe dargestellt. Bei den letzten drei Messreihen mit 300 Nachrichten pro Minute und Host ist mit 10.000 wie zuvor eine erhöhte Anzahl der gesamten Nachrichten zur Verlängerung der Ausführungszeit vorzufinden.
In Abbildung 5.7 zeigt sich bei der Untersuchung der normierten Ergebnisse von den Clients der gleiche Effekt, der auch zuvor bereits festgestellt wurde. Die schrittweise Erhöhung der Nachrichten pro Minute und Host (Messreihe 3 und 4) belastet zum einen das Netz-

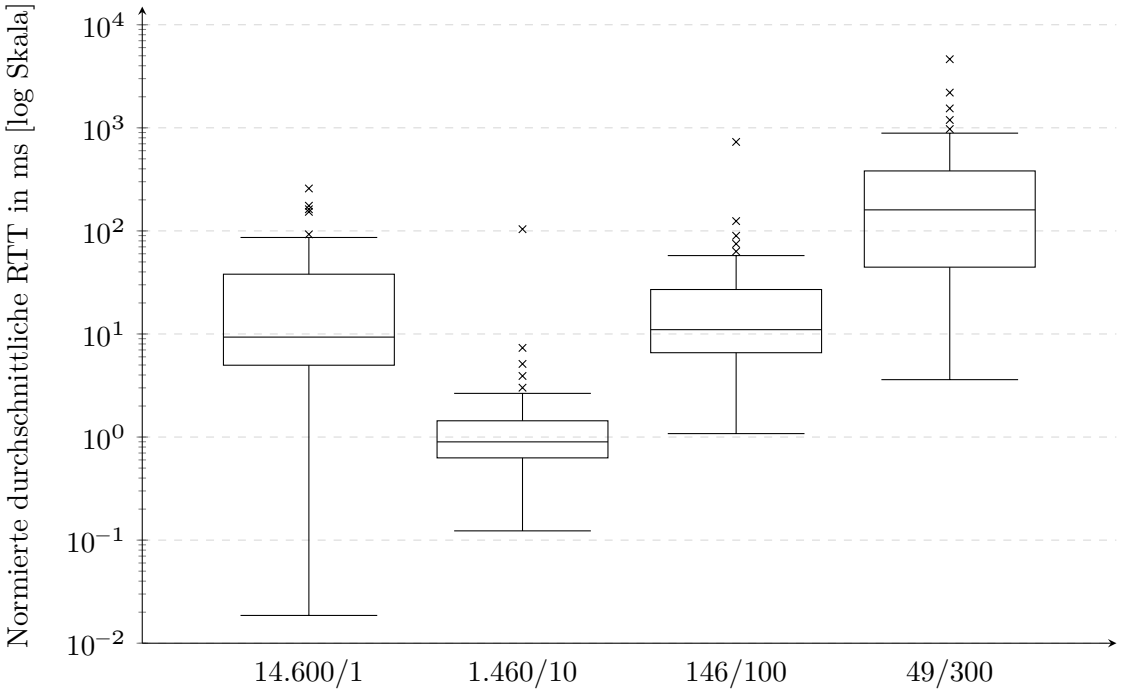


Abbildung 5.7: Vergleich der RTT gegenüber der Nachrichtengröße und Nachrichtenrate auf Applikationsebene. In dem Testfall wird die Größe der von den Clients gesendeten Nachrichten über die Messreihen und die durchschnittliche Nachrichtenrate so angepasst, dass der Durchsatz pro Client immer konstant bleibt. Die Ergebnisse der Messreihen, bei denen pro Nachrichten mehr als 1.460 Byte gesendet werden, sind für eine Vergleichbarkeit mit dem jeweiligen Faktor 1.460/Nachrichtengröße normiert dargestellt.

werk analog zum ersten Testfall (vgl Abbildung 5.1). Zum anderen führt die Erhöhung der Nachrichtengröße (Messreihe 1) zu einer höheren Anzahl gesendeter Pakete und so zu einer erhöhten RTT (vgl. Abbildung 5.5). Auch hier ist der durch die Normierung niedrige untere Whisker zu beachten.
Interessanter ist in diesem Testfall jedoch die Auswertung der Service Manager in Abbildung 5.8. In der ersten Messreihe ist deutlich die kontinuierliche Wanderung des Dienstes zwischen verschiedenen Service Managern im virtuellen Netzwerk festzustellen. Innerhalb eines Migrationszyklus gehen genau so viele Nachrichten von unterschiedlichen Clients beim Dienst ein, dass er auf der Grundlage dieser letzten Nachrichten fast immer an einer anderen Position besser geeignet wäre. Die geringe Nachrichtenrate von einer Nachricht pro Minute und Host und den einhergehenden sehr langen Wartezeiten zwischen den einzelnen Nachrichten führt dazu, dass der Dienst in über 90% der 3.995 Fälle migriert wird. Dies bekräftigt nochmals die Aussage, die im zweiten Testfall bereits bezüglich der Migrationszeit getroffen wurde (vgl. Abbildung 5.4). Bei einer seltenen Verwendung des Dienstes durch die Clients muss die Migrationszeit an das Anfrageverhalten der Clients angepasst werden. Dies führt schließlich zu einer Reduktion der Wanderungen im Netzwerk, wie an den an-

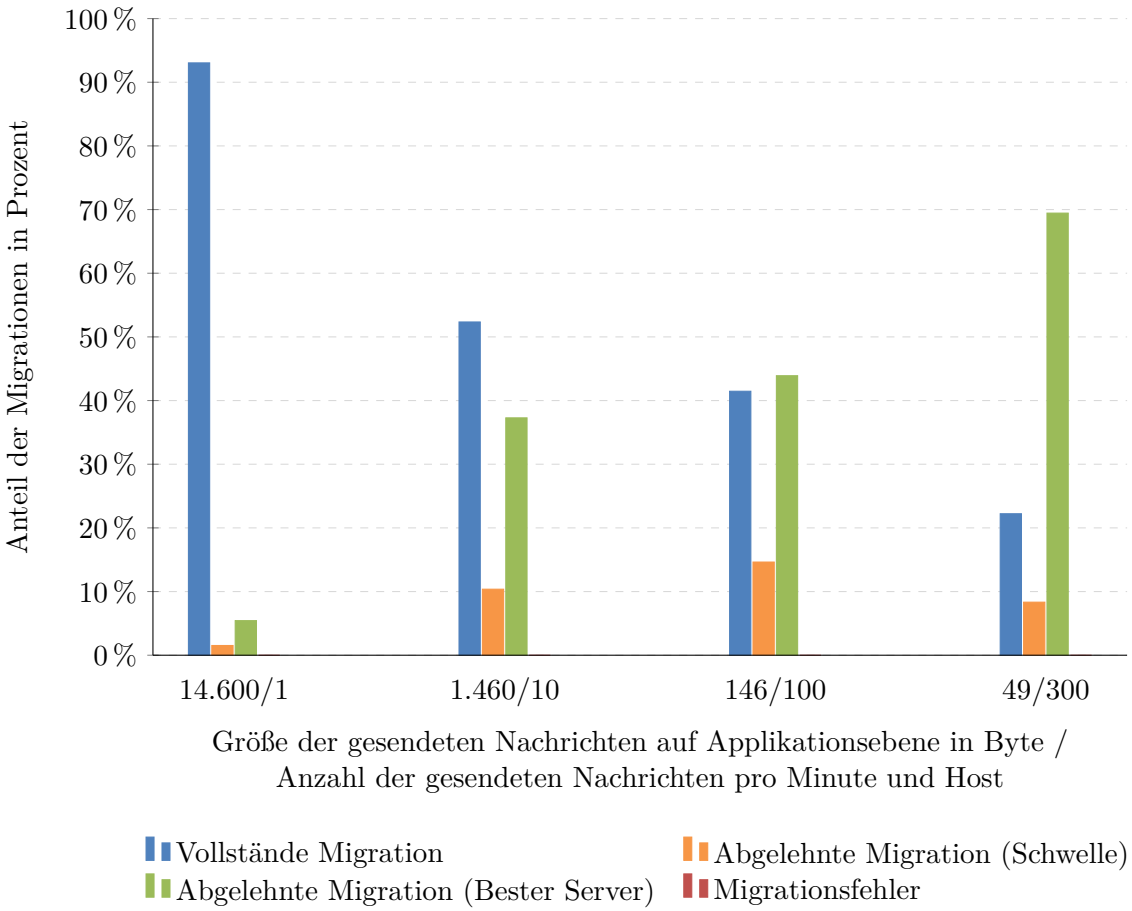


Abbildung 5.8: Vergleich der Migrationen bei einer durchweg konstanten Nachrichtenrate. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

deren Messreihen dieses Testfalls zu erkennen ist. Schrittweise werden mehr Migrationen abgelehnt, weil der beste Host zur Ausführung des Dienstes bereits gefunden wurde.

Abschließend erfolgt die Untersuchung der Dienstemigration durch den Service Manager in den in Kapitel 2.3 aufgestellten Szenarien. In den folgenden beiden Testfällen wird dies jedoch nicht in einem verlustfreien Netzwerk geschehen, sondern in einem verlustbehafteten. Die Besonderheit des verlustbehafteten Netzwerks liegt in der genaueren Emulation kabelloser Verbindungen, denn bereits aufgebaute Verbindungen zwischen zwei Objekten im Netzwerk können während der Übertragung abbrechen und auch bereits gesendete Nachrichten verloren gehen. Das in Kapitel 4.2.6 beschrieben Verfahren der Dienstsignalisierung und der Diensterkennung hat hierbei eine besondere Bedeutung für die Datenerfassung der Performance-Clients. Die dort beschriebenen Verluste der Broadcast-Nachrichten können in diesen Testfällen fortwährend bei den Messungen der RTT geschehen, was den Informationsverlust über eine stattgefundene Migration und das damit verbundene momentane Wissen über den aktuellen Dienst zur Folge hat. Ist die Verbindung einer Messnachricht zum Dienst erst einmal hergestellt, werden die Nachrichten größtenteils ohne weitere Probleme übertragen. Doch die Probleme entstehen bereits beim Aufbau der Verbindung.

Wird das in Kapitel 4.2.6 beschriebene Maximum von zehn Fehlversuchen beim Aufbau einer Verbindung und dem anschließenden Ausbleiben von `who_is_answer`-Nachrichten erreicht, bricht der Performance-Client als Lösung des Problems mit der RTT-Messung bei der Nachricht ab, sodass diese nicht in die Auswertung der durchschnittlichen RTT aller erfolgreichen Nachrichten einfließen kann (vgl. Abbildungen 5.9 und 5.11). Da die fehlerhaften Nachrichtenübermittlungen jedoch nicht vollständig ausgeschlossen werden sollen, werden diese anschließend in beiden Testfällen über eine weitere Auswertung anhand der Tabellen 5.1 und 5.2 beschrieben.
Das verlustbehaftete Netzwerk im fünften und sechsten Testfall emuliert die zwei geforderten IoT-Segmente aus je 20 Objekten respektive Hosts, die über zwei Gateways und einem gemeinsamen Internetknoten miteinander verbunden sind (vgl. Abbildung A.3). Die mit dem Internet verbundenen Knoten sind alle über eine Verbindung mit einem Durchsatz von 50 Mbit/s miteinander verbunden. Die Clients und Service Manager in diesem Netzwerk verwenden die in den ersten drei Testfällen festgelegten Werte für die verschiedenen Variablen bei allen Messreihen:

- Die Performance-Clients senden 10 Nachrichten pro Minute und Host.
- Die Migrationszykluszeit des Service Managers ist auf 30 Sekunden festgelegt.
- Die Nachrichtengröße auf Applikationsebene der Clients umfasst 14.600 Bytes.

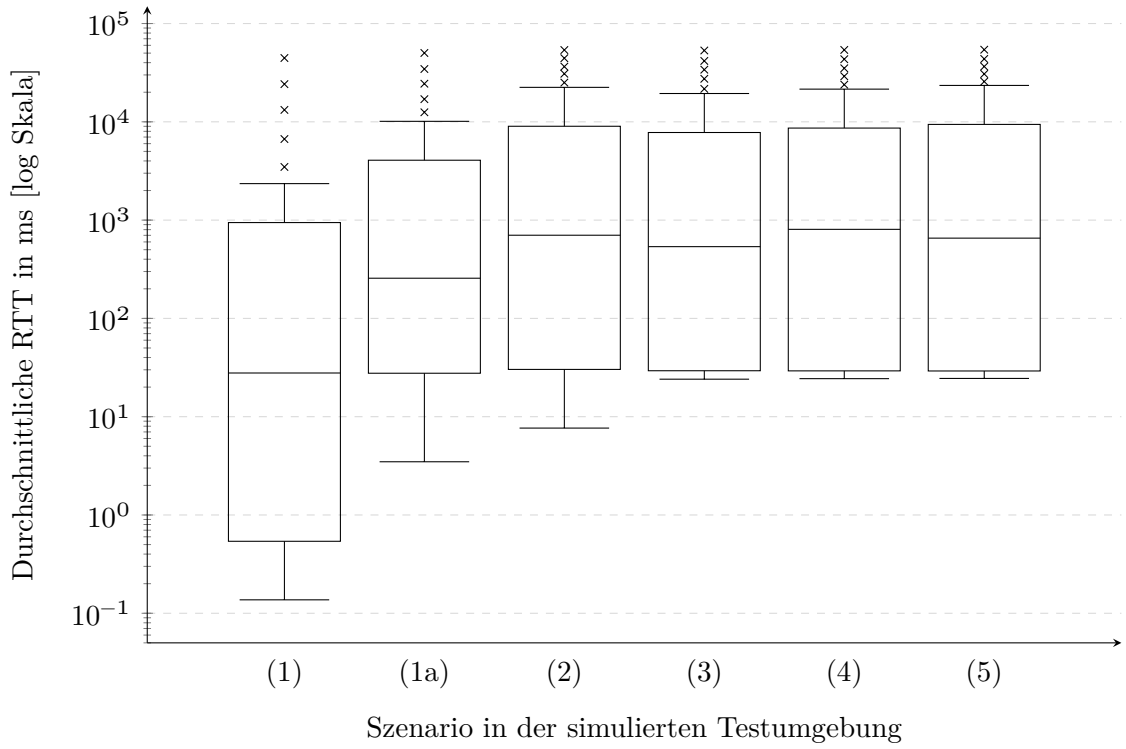


Abbildung 5.9: Vergleich der RTT zwischen unterschiedlichen Szenarien im verlustbehafteten, segmentierten IoT mit fünf Clients aus einem IoT-Segment. Die Szenarien stellen eine schrittweise Veränderung der Positionierung eines Dienstes im IoT dar.

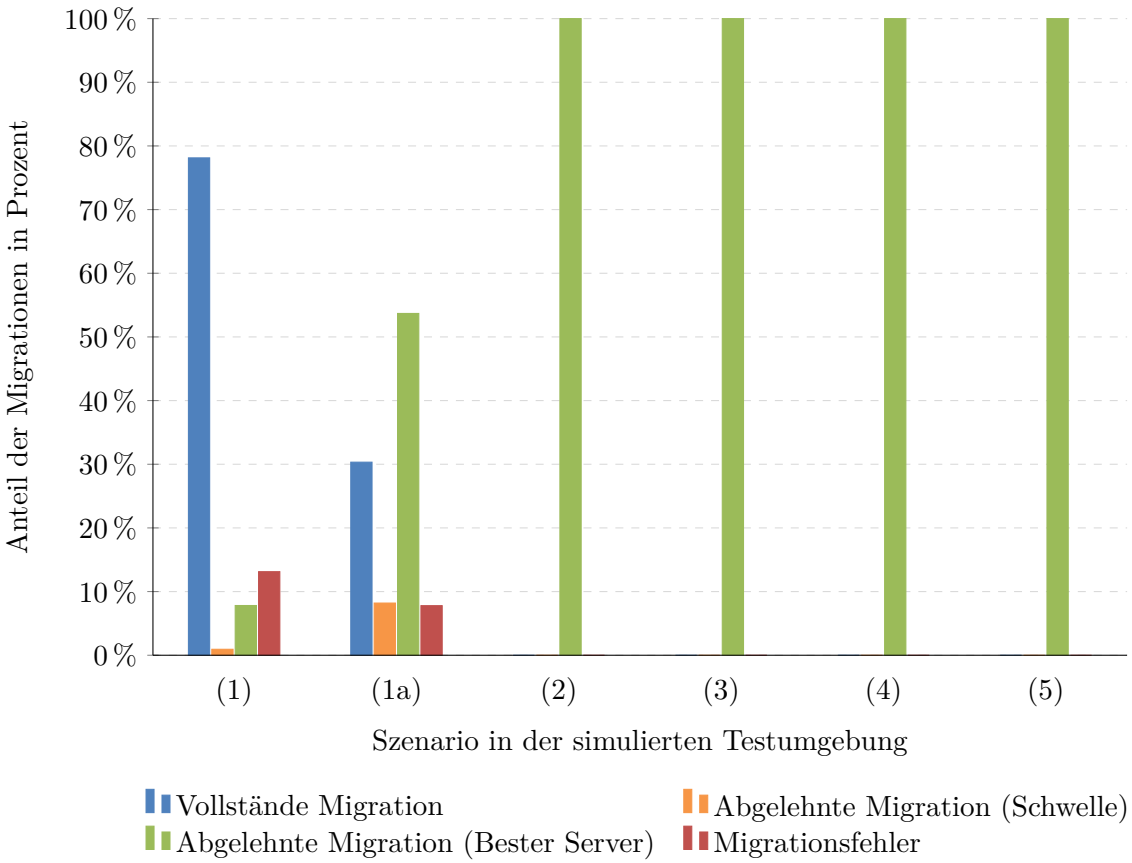


Abbildung 5.10: Vergleich der Migrationen zwischen unterschiedlichen Szenarien im verlustbehafteten, segmentierten IoT mit fünf Clients aus einem IoT-Segment. Die Szenarien stellen eine schrittweise Veränderung der Positionierung eines Dienstes im IoT dar. Die Szenarien 3-5 sind fixiert, weshalb keine Migrationen des Dienstes stattfinden und sich dieser immer an der „besten“ Position im IoT befindet. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

Im fünften Testfall senden zunächst fünf zufällig im IoT-Segment A verteilte Performance-Clients Nachrichten zum Dienst. Der Testfall soll überprüfen, ob die Bereitstellung eines wandernden Dienstes innerhalb eines IoT-Segments zu einer Reduktion der RTT, der Verlustrate und der Hops im Vergleich zu fixierten Diensten an definierten Positionen führt (vgl. Kapitel 2.3). Während die Performance-Clients aufgrund der Vergleichbarkeit in allen Messreihen ihre Position beibehalten, wird bei der Bereitstellung der Service Manager unterschieden auf welchen Hosts diese ausgeführt werden. Die erste Messreihe erlaubt Migrationen des Dienstes auf allen Hosts im IoT-Segment A. Diese beinhalten auch die Hosts, die als Performance-Clients deklariert sind, und den Host, der als Gateway bestimmt ist. Analog gilt dies für die sechste Messreihe (Szenario 5), nur dass sich dort die möglichen dienstausführenden Hosts im IoT-Segment B befinden. Bei der zweiten Messreihe (Szenario 1a) werden die Hosts mit dem Performance-Client aus der Menge der potentiell dienstausführenden Hosts entfernt, sodass der Dienst nicht an dieselben Positionen migrieren werden kann. Die erste Ausführung des Dienstes übernimmt in diesen Messreihen der Gateway-

Knoten des jeweils betrachteten IoT-Segments. Die anderen Messreihen (Szenarien 2-4) dienen dem Vergleich der Migration zu klassischen Systemen. So wird der Dienst hier an fixierten Positionen gesetzt und verrichtet von dort seine Aufgabe, zum einen sind es die beiden Hosts am Gateway der IoT-Segmente sowie der Internetknoten. In diesem Testfall wird angenommen, dass der Dienst in den ersten beiden Messreihen immer in die Richtung der Performance-Clients migriert wird, wie es bereits in den anderen Testfällen festzustellen ist. Durch die Reduktion der Menge dienstausführender Hosts im zweite Szenario ist hier eine Erhöhung der RTT zu erwarten, da Performance-Client und der Dienst sich nicht mehr zur gleichen Zeit auf demselben Host befinden können und sehr kurze RTTs ohne externen Verbindungsaufbau wegfallen.
Die Abbildung 5.9 zeigt die Messungen der RTT der Clients. Dabei ist ein deutlicher Unterschied zwischen der ersten Messreihe (Szenario 1), bei der eine Wanderung des Dienstes im IoT-Segment A durchgeführt wird, und den anderen Messreihen zu verzeichnen. Der Median der RTT ist in der ersten Messreihe, bei der eine Migration stattfindet, bereits um Faktor 20 im Vergleich zu den Messreihen 3-6 (Szenario 2-5) geringer, bei denen sich die dienstausführenden Hosts an fixierten Positionen befinden. Doch auch die zweite Messreihe zeigt eine leichte Verbesserung um Faktor 3 im Vergleich zur Bereitstellung des Dienstes an fixierten Positionen, wie Gateways oder im Internet. Auch die maximalen Werte innerhalb des Interquartilsabstandes sind bei der ersten Messreihe um Faktor 4 geringer als die der zweite Messreihe sowie um Faktor 10 geringer als die Messreihen mit fixierten Positionen. Die Ausreißer im System lassen sich hier durch die in der Software definierten Timeouts von 60 Sekunden sowie durch die Anfragen an den Floodlight SDN-Controller genau erklären. Die Unterschiede der Minima zwischen der ersten und zweiten Messreihe um Faktor 40 bestätigen hier die Annahme der höheren RTT in der zweiten Messreihe aufgrund des Wegfalls derselben Hosts für die Ausführung von Clients und dem Dienst. Ein negativ zu bewertender Punkt für die Migration zeigt sich in der Größe des Interquartilsabstandes der ersten Messreihe. Die Differenz ist hier deutlich höher als bei allen anderen Messreihen, die Messungen streuen mehr. Unterschiede bei der Bereitstellung von Diensten lassen sich bei den Messreihen der Szenarien 2-5 kaum feststellen, es sind lediglich leichte Schwankungen der minimalen Messwerte in der dritten Messreihe zu vernehmen. Dies zeigt deutlich, dass die RTT von den Verbindungseigenschaften des IoT abhängig ist und nicht von denen des beständigen Internets. Hier zeigt sich auch, dass die sechste Messreihe (Szenario 5) kaum eine Verbesserung aufweist. Die Wanderungen in einem anderen IoT-Segment haben keine Auswirkungen auf die RTT zwischen Clients und dem Dienst.

Szenarien	M_{RTT} (in ms)	SD_{RTT} (in ms)	$M_{\text{Fehlvers.}}$	$SD_{\text{Fehlvers.}}$	M_{Hop}	SD_{Hop}
1	3.895,43	10.188,28	0,27	1,19	1,25	0,98
1a	5.699,09	11.808,28	0,37	1,37	1,60	0,76
2	7.781,60	13.446,87	0,09	0,40	1,60	0,49
3	7.122,52	12.841,05	0,11	0,65	2,60	0,50
4	7.560,46	13.081,65	0,06	0,34	3,60	0,49
5	7.682,78	13.232,65	0,09	0,54	3,60	0,50

Tabelle 5.1: Vergleich der Szenarien im fünften Testfall anhand von Leistungsmetriken über Mittelwert und Standardabweichung. Der Vergleich findet über die der Leistungsmetriken RTT, Hop und der Fehlversuche bei den Messungen der Performance-Clients statt.

Auch anhand des Mittelwerts M_{RTT} ist die Verbesserung in der RTT durch die Bereitstellung eines wandernden Dienstes in einem verlustbehafteten IoT zu erkennen, allerdings ist der Unterschied zwischen den verschiedenen Szenarien deutlich geringer als bei dem Median. Der über alle Messreihen hinweg hohe Standardabweichung deutet zusätzlich auf eine hohe Anzahl von Ausreißern hin (vgl. Tabelle 5.1). Eine durch die Migration bedingte Verschlechterung zeigt der Mittelwert $M_{\text{Fehlversuch}}$ in den ersten beiden Messreihen. In beiden Messreihen verursachen die Migrationen den Informationsverlust über den aktuell dienstausführenden Host und einhergehend fehlschlagende Verbindungsversuche. Interessant ist auch ein Vergleich in der Anzahl der Hops zwischen den Szenarien. Zunächst ist hier eine leichte Minimierung der Hops durch die Wanderung in der ersten Messreihe zu verzeichnen. Doch zeigt sich auch, dass in diesem kleinen IoT-Segment aus 20 Knoten die durchschnittliche Anzahl der Hops zwischen der zweiten und dritten Messreihe nicht verbessert, obwohl der Dienst in der dritten Messreihe nicht migriert wurde. Lediglich eine Differenz in der Standardabweichung ist vernehmbar. Die sukzessive Steigerung der Hops zwischen den Messreihen 3-6 (Szenarien 2-5) zeigen die Verbindungen über die Gateways und dem Internetknoten, wobei der im anderen IoT-Segment B wandernde Dienst (Szenario 5) auf den Gateway-Knoten platziert wird.
Die Abbildung 5.10 vergleicht die Migrationen der Service Manager im fünften Testfall. Da bei den Messreihen 2-4 keine Migrationen stattgefunden haben, ist der Dienst dort immer an der „bestmöglichen“ Position aufzufinden. In der sechsten Messreihe hingegen bestätigt sich hier nochmals, was bereits durch den Mittelwert der Leistungsmetrik Hop zu verzeichnen war. Es wurden keine Migrationen durchgeführt. Da die erste Ausführung des Dienstes auf dem Gateway-Knoten stattfindet und die Clients den Dienst nur über den Internetknoten erreichen können, wird er ohne Migration dort kontinuierlich ausgeführt. Die erste Messreihe verzeichnet jedoch insgesamt eine sehr hohe Anzahl Migrationen im Vergleich zum ersten Testfall im verlustfreien Netzwerk (vgl. Abbildung 5.2, Messreihe 1). In nur 10% aller Migrationszyklen blieb eine Migration aus. Im Vergleich ist das Ergebnis der zweiten Messreihe deutlich besser. Die Anzahl der Migrationen wurde deutlich verringert, abgelehnt durch die Findung des bereits besten Knotens zur Ausführung des Dienstes und durch die Betrachtung des Schwellwertes. Das lässt vermuten, dass durch die geringere Anzahl Hosts weniger Möglichkeiten übrig bleiben, den Dienst näher an die Clients zu platzieren und so eine ausgewogenere Bereitstellung zwischen den Clients möglich ist. Die in den Messreihen zu verzeichnende hohe Anzahl Migrationsfehler ist durch ungewollte Timeouts im Netzwerk zu erklären. Ein Problem, dass durch die Timeouts bei der Migration hervorgerufen werden kann und hier zu lösen gilt, ist eine ungewollte Duplikation des Dienstes auf zwei Hosts. Schlägt der letzte Handshake während des Migrationsprozesses fehl, kann nicht sichergestellt werden, dass nur eine Instanz des Dienstes im System existiert, was schließlich ohne Synchronisierung der Instanzen zu einer Dateninkonsistenz führen kann.

Der sechste und damit letzte Testfall emuliert ebenfalls die Bereitstellung eines Dienstes in einem verlustbehafteten IoT und vergleicht dieselben Szenarien untereinander. Einzig das Szenario 1a mit dedizierten Dienstknoten im IoT-Segment A wird außer Acht gelassen, da eine Untersuchung der Szenarien mit mehreren IoT-Segmenten im Vordergrund steht. Im Unterschied zum fünften Testfall wird hier überprüft, ob auch eine Veränderung in den Leistungsmetriken RTT, der Verlustrate und der Hops festzustellen ist, wenn gleichzeitig in beiden IoT-Segmenten jeweils fünf und somit insgesamt zehn Performance-Clients Anfragen

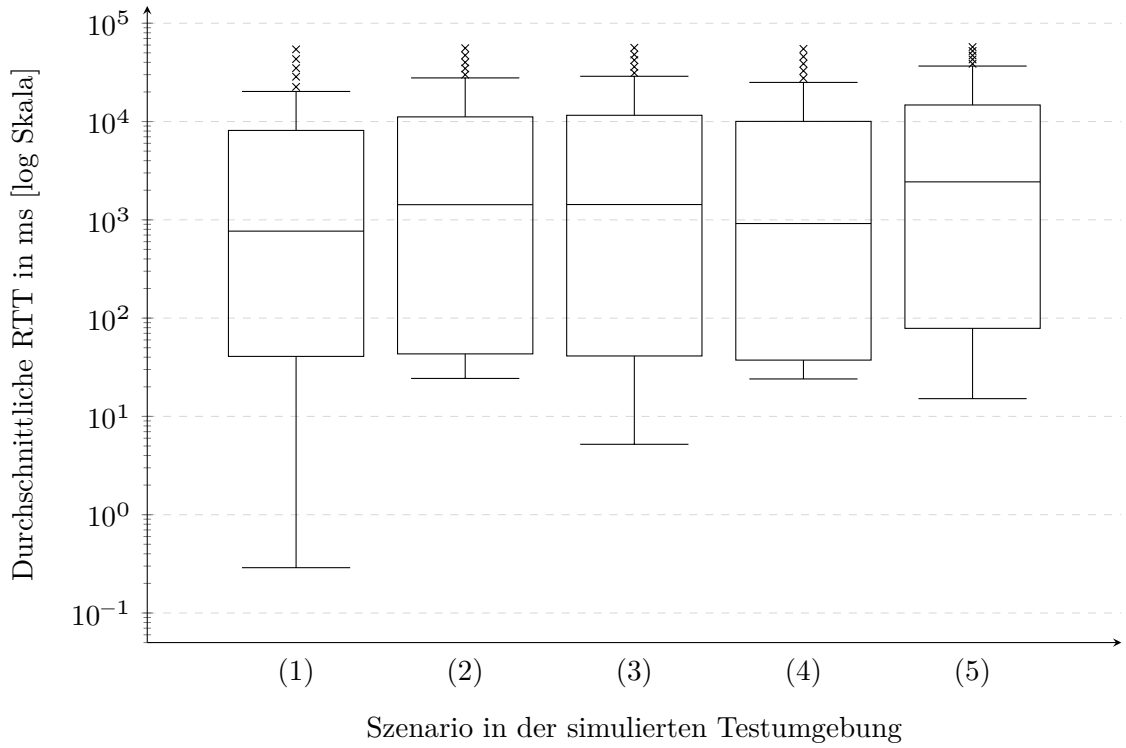


Abbildung 5.11: Vergleich der RTT zwischen unterschiedlichen Szenarien im verlustbehafteten, segmentierten IoT mit zehn Clients aus zwei IoT-Segmente. Die Szenarien stellen eine schrittweise Veränderung der Positionierung eines Dienstes im IoT dar.

an den Dienst stellen.
Die Abbildung 5.11 zeigt die Ergebnisse der RTT-Messung der 10 Performance-Clients. Hier zeigt sich ein großer Unterschied der ersten Messreihe zu der vom fünften Testfall in Abbildung 5.9. Durch die fünf zusätzlichen Clients in dem anderen IoT-Segment nähert sich die RTT deutlich den Ergebnissen der anderen Messreihen 2-5 an. Einzig das Minimum, angegeben durch den unteren Whisker, scheint in der Messreihe deutlich niedriger zu sein. Der Grund dafür könnte eine stattgefundene Migration innerhalb des IoT-Segments A sein. Hat ein Client erst sehr spät eine Leerung seines Pausenkontos K vorgenommen, verzögerte sich das Ende seiner Ausführung. Dadurch könnte der Dienst am Ende der gesamten Messung zu diesem Client migriert worden und schließlich eine kurze RTT entstanden sein. Die restlichen Messreihen weisen nur sehr mäßige Unterschiede auf, die sich durch Schwankungen bei dem Anfrageschema der Performance-Clients oder auch durch leichte Unterschiede in der Struktur zwischen den beiden IoT-Segmenten A und B begründen lassen. Sind die Wege zwischen den Performance-Clients und dem dienstausführenden Objekten in einem der IoT-Segmente kürzer, so hat dies Einfluss auf die erhobene RTT. Zu beachten ist hier, dass für die Verbindungen zum Internet ein Durchsatz von 50 Mbit/s und keine Verlustrate angenommen wurde. Es ist zu erwarten, dass eine Veränderung der Verbindungen Auswirkungen auf die Messungen in diesem Testfall haben.
Analog zur Abbildung 5.11 zeigt sich auch bei den Mittelwerten eine Angleichung zwischen den Szenarien (vgl. Tabelle 5.2). Wie bereits eine Veränderung im ersten Szenario zu er-

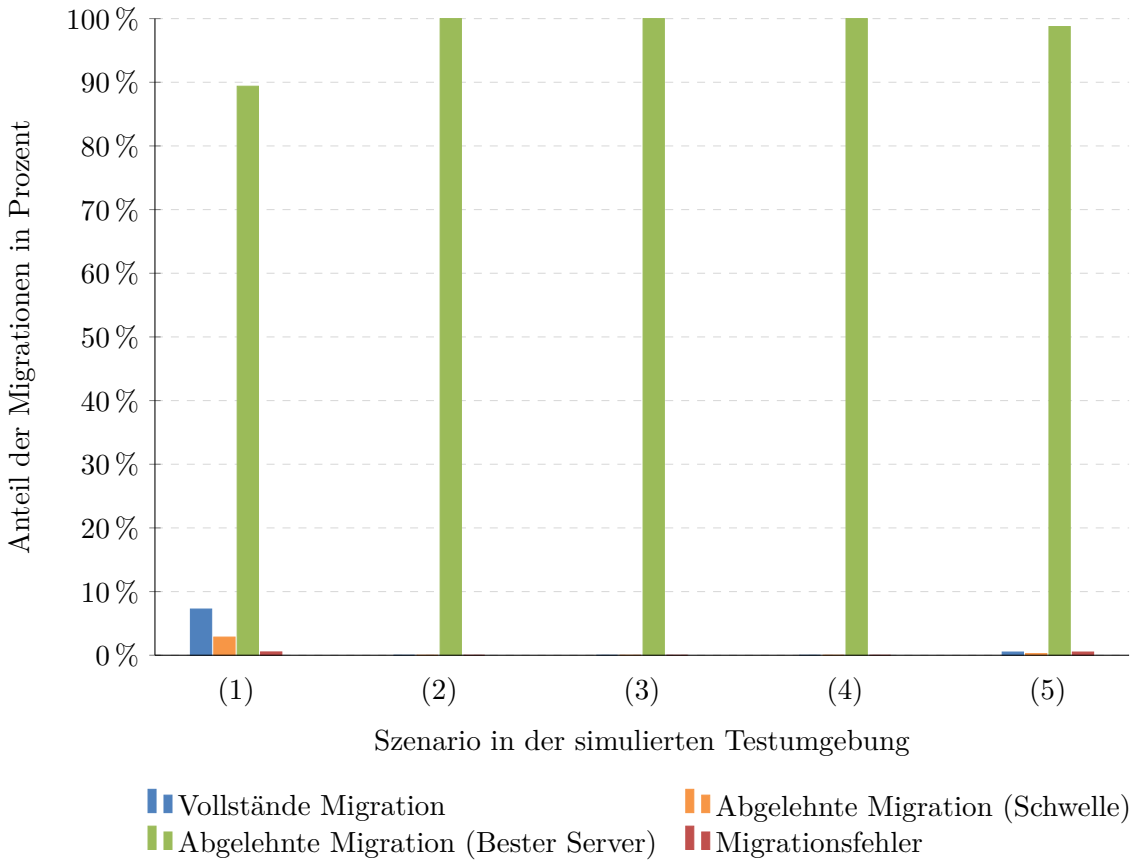


Abbildung 5.12: Vergleich der Migrationen zwischen unterschiedlichen Szenarien im verlustbehafteten, segmentierten IoT mit zehn Clients aus zwei IoT-Segmenten. Die Szenarien stellen eine schrittweise Veränderung der Positionierung eines Dienstes im IoT dar. Die Szenarien 3-5 sind fixiert, weshalb keine Migrationen des Dienstes stattfinden und sich dieser immer an der „besten“ Position im IoT befindet. Innerhalb einer Messreihe werden die möglichen Migrationen aller dienstausführenden Objekte im SDN-Testsystem zusammengefasst und teilen sich in vier Möglichkeiten auf.

kennen war, wird diese auch hier im Mittelwert der RTT widergespiegelt. Untereinander ist ein einziger Unterschied zwischen dem ersten und fünften Szenario festzustellen, der durch die geringen RTT-Werte in der ersten Messreihe zu begründen ist. Insgesamt haben sich die Standardabweichungen auf einem hohen Niveau eingependelt. Im Vergleich zum fünften Testfall sind die Fehlversuche durch die fünf zusätzlichen Clients in allen Szenarien insgesamt angestiegen. Sie befinden sich auf dem gleichen Level, wie die ersten Messreihe aus dem fünften Testfall, bei der eine Migration im IoT-Segment A stattfand. Die Hops haben sich hingegen auf ein einheitliches Niveau eingependelt. Der kleine Anstieg im vierten Szenario ist durch die Gleichheit der beiden IoT-Segmente und dem genau mittig zwischen ihnen platzierten Internetknoten entstanden.
Ein Vergleich des Migrationsverhaltens der Service Manager zwischen den Szenarien in Abbildung 5.12 zeigt deutlich die bereits beschriebenen Migrationen im ersten Szenario. Während im fünfte Szenario die Service Manager in weniger als 1% der Zyklen Migrationen durchführen, weist das erste Szenario in 7,2% aller Zyklen Migrationen und in 2,8% der Fälle

Szenarien	M_{RTT} (in ms)	SD_{RTT} (in ms)	$M_{\text{Fehlvers.}}$	$SD_{\text{Fehlvers.}}$	M_{Hop}	SD_{Hop}
1	7.453,13	13.153,01	0,26	1,04	2,40	0,94
2	8.658,44	13.933,47	0,26	0,97	2,40	0,92
3	8.748,81	13.966,59	0,34	1,16	2,50	0,68
4	8.039,76	13.565,04	0,16	0,67	2,40	1,28
5	10.003,12	14.582,86	0,21	0,89	2,40	1,28

Tabelle 5.2: Vergleich der Szenarien im sechsten Testfall anhand von Leistungsmetriken über Mittelwert und Standardabweichung. Der Vergleich findet über die der Leistungsmetriken RTT, Hop und der Fehlversuche bei den Messungen der Performance-Clients statt.

eine Migrationsablehnung aufgrund des Schwellwertes auf. Die anderen Messreihen enthalten durch die Festlegung des Diensten auf eine fixierte Position im gesamten Netzwerk wie bereits im fünften Testfall keine Migrationen auf.

In den verschiedenen Testfällen der Evaluationen konnte insgesamt ein positives Ergebnis bei der Bereitstellung migrierender Dienste im IoT festgestellt werden. So zeigten die ersten vier Testfälle die Beeinflussung der verschiedenen Variablen auf die Entscheidung zur Migration. Die Steigerung der Anzahl Nachrichten pro Minute verursachte eine Erhöhung der RTT aufgrund der Auslastung des Netzes, einhergehend reduzierte sich prozentual die Anzahl durchgeführter Migrationen. Dies zeigte sich auch bei der Steigerung des Migrationszyklus, denn hier konnte die Steigerung zur Minimierung der Anzahl Migrationen beitragen. Der dritte Testfall machte deutlich, dass die Erhöhung aller Nachrichten der Clients im gleichbleibenden Verhalten untereinander keinen Einfluss auf die Migrationen hat. Erst als sich das Anfragemuster der Performance-Clients änderte, verhielt sich die Service Manager Plattform unterschiedlich bei dem Migrationsverhalten zwischen den Messreihen. Je mehr Nachrichten pro Minute gesendet wurden, desto besser hat dieser ein Objekt im IoT gefunden, das für die Bereitstellung optimal war. Aufgrund der vielen Nachrichten pro Minute stieg jedoch die RTT dieser Messreihe.
Abschließend konnte mithilfe der letzten beiden Testfällen die Frage geklärt werden, auf welchen Objekten die Bereitstellung der Dienste die besten Resultate auf die Latenzzeit respektive RTT hat. Durch die Migration der Dienste im verlustbehafteten IoT zeigte sich eine positiv zu verzeichnende Minimierung in der RTT um den Faktor 20 im Median und Faktor 2 im Durchschnitt gegenüber festgesetzten Standorten, wie die Edge-Gateways des IoT oder ein Standort im Internet. Dieser Faktor wurde jedoch reduziert, wenn für den Dienst speziell dedizierte Objekte im IoT ausgewählt und die Objekte der Clients für die Ausführung des Dienstes ausgeschlossen wurden. Einhergehend konnten durch die Migration jedoch auch ein leichter Anstieg von Fehlversuchen beim Verbindungsaufbau der Clients gemessen werden, da zum Messzeitpunkt der Standort des Dienstes einigen Clients nicht bekannt war. Der sechste Testfall zeigte die Problematiken der Migration eines segmentierten IoT. Ist eine Segmentierung des IoT vorhanden und sind die Clients in allen Bereichen des globalen IoT vertreten, so hat die Migration innerhalb eines einzelnen IoT-Segments keine Auswirkung mehr.
Bei der Emulation des verlustbehafteten IoT entstanden jedoch auch Probleme, denn der SDN-Controller des virtuellen Netzwerkes hat in dem verlustbehafteten Netzwerk nicht immer alle Routen zwischen den vorhandenen Switches finden können. Darauf folgten schließ-

lich Verbindungsverluste und einhergehend ein Abbruch der jeweiligen Messung in dem virtuellen Netzwerk. Auch entstanden Probleme im SDN-Controller bei der Vergrößerung des Netzwerkes. Die maximalen Grenzen waren sehr schnell erreicht und hatten einen Absturz des SDN-Controllers zur Folge, wenn weitere Objekte oder Verbindungen hinzugefügt wurden.

KAPITEL 6

Fazit und Ausblick

Die vorangegangenen Kapitel demonstrieren die Möglichkeiten der Bereitstellung von Diensten im IoT für das Fog Computing und einhergehend die Beantwortung der zu Beginn aufgestellten Fragen. Zunächst konnten anhand bereits existierender Dienste für das IoT festgestellt werden, dass viele abweichende Interpretationen für den Begriff „Dienst“ im IoT von verschiedenen Autoren aufgrund unterschiedlicher Betrachtungsweisen existieren. Diese ließen sich jedoch schließlich anhand verschiedener Merkmale in vier aufeinander aufbauende Klassen unterteilen, die sowohl eine gesonderte Betrachtung von Diensten mit physikalischen Ressourcen als auch mit einer reinen softwareseitigen Logik erlauben. Die darauf folgende Definition zur Autonomie der Dienste im IoT ist für die Positionierung der Dienste von Interesse. Bei deren Untersuchungen konnte festgehalten werden, dass das globale IoT derzeit eher in einzelne IoT-Segmente aufgeteilt ist und innerhalb dessen unterschiedliche Positionen im IoT zur Bereitstellung der Dienste vorhanden sind. Hierzu wurde überprüft, welche bekannten topologischen Verfahren aus der Erforschung von Adhoc-Netzwerken auch bei der Migration von Diensten im IoT anwendbar sind. Darauf folgte eine Betrachtung topologischer Charakteristika und passende Aufnahmemöglichkeiten, die als bewertbare Systemeigenschaften dazu verwendet werden konnten eine optimale Positionierung der Dienste im IoT bei der Migration zu erlangen. Die Grundlage, die eine Bereitstellung von Diensten im IoT ermöglichte, bildete schließlich die vollständig konzipierte und implementierte dienstorientierte Service Manager Plattform, dessen komponentenbasierte Architektur eine Erweiterung um beliebige Komponenten erlaubt. Diese bilden gemeinsam einen selbstorganisierenden Mechanismus, der die topologischen Charakteristika lokal auf den Objekten aufnimmt, gewichtet und anhand von definierten Entscheidungskriterien den von ihr verwalteten Dienst im IoT an strategisch optimale Positionen migriert. Die erfolgreiche Migration teilt der jeweilige Service Manager den Objekten mithilfe einer Dienstsignalisierung mit. Die Verwaltung des Dienstes führt die Plattform so durch, dass dieser als ein vollständig autonomes Programm auf den Objekten arbeitet und keine spezifische API der Plattform implementieren muss. Da die abschließende Evaluation aufgrund von Umbaumaßnahmen in einem durch SDN emulierten Netzwerk und nicht in einer realen IoT-Umgebung stattfand, mussten zunächst topologische Charakteristika in dem MIOT-Netzwerk der Westfälische Wilhelms-Universität Münster aufgenommen und in einer effizienten Datenstruktur bereitgestellt werden. Diese konnte schließlich bei der Emulation

des virtuellen Netzwerkes durch den Virtual Network Emulator Mininet und der Migration durch den Service Manager verwendet werden. Die Evaluation wurde schließlich mithilfe des gesamten Systems durchgeführt, das sich von einer Client-Anwendung über einen durch die Service Manager Plattform bereitgestellten Dienst im IoT bis hin zu den bewertbaren Systemeigenschaften als Vergleichsmerkmale erstreckt. In diesem Messaufbau sendete die Client-Anwendung gezielt Nachrichten an den im IoT migrierenden Dienst und überprüfte die RTT. Die Service Manager Plattform protokollierte die durchgeführten oder abgelehnten Migrationen und alle aufgetretenen Fehler.
Insgesamt zeigen die Ergebnisse eine positiv zu verzeichnende Minimierung der Latenzzeiten respektive RTT um den Faktor 20 im Median und um den Faktor 2 im Durchschnitt durch die Bereitstellung eines zwischen den Objekten im IoT migrierenden Dienstes im Vergleich zu festgelegten Dienstpositionen, wie die Edge-Gateways des IoT oder ein Standort im Internet, beispielsweise einer Cloud-Infrastruktur. Negativ ist hingegen die verzeichnete Steigerung der Fehlversuche bei dem Aufbau von Verbindungen der Clients zum migrierenden Dienst zu bewerten, die auf fehlende Informationen der Dienstpositionen zurückzuführen sind. Zusätzlich zu diesen Messergebnissen ist auch eine Abhängigkeit zwischen der festgelegten Migrationszeit durch die Plattform und der Anfrageverhalten der Clients festzustellen. Bei einer seltenen Verwendung des Dienstes durch die Objekte im IoT muss eine Anpassung der Migrationszeit vorgenommen werden, die schließlich zu einer Reduktion der Anzahl Migrationen im IoT führt. Darüber hinaus ist den Ergebnissen zu entnehmen, dass die Migrationen nur für die lokale Verarbeitung innerhalb eines IoT-Segments eine zeitliche Verbesserungen ergeben. Ist ein segmentiertes IoT mit anfragenden Clients aus mehreren Segmenten vorhanden und wandert der Dienst in einem der beiden Segmente, so nähern sich die gemessenen Ergebnissen denen der Bereitstellung an den Edge-Gateways und im Internet an. Die Ergebnisse lassen sich plausibel nachvollziehen, da der Dienst für eine passende Bereitstellung beider IoT-Segmente mittig zu den Edge-Gateways migriert wird.
Auf dem Weg zu diesen Ergebnissen sind mehrere Probleme entstanden, die ihren Ursprung primär in den verlustbehafteten Verbindungen des Netzwerkes hatten. Aufgrund dieser verlustbehafteten Verbindungen des IoT erreichen die Signalisierungen einer stattgefundenen Migration eines Dienstes im IoT nicht alle Clients. Die Lösung ist ein zustandsloses Protokoll zwischen den Clients und der Service Manager Plattform, das eine Anfrage zum aktuellen Dienststandort erlaubt. Dieses Protokoll ist in Hinblick auf sehr große, dynamische IoT-Netze ohnehin unabdingbar, da zu jedem Zeitpunkt neue Objekte dem IoT hinzugefügt werden können. Ein weiteres Problem entsteht bei die Migration im verlustbehafteten Netzwerk. Die Nachrichtenverluste sorgen sehr oft für Verbindungsabbrüche bei der Migration zwischen zwei Service Managern, sodass schließlich der momentane Status während der Übertragung nicht erkannt wird. Abhilfe schafft hier ebenfalls ein Migrationsprotokoll, das diese Verbindungsabbrüche in jedem möglichen Zustand behandelt.

Die Ergebnisse dieser Masterarbeit sind für das Fog Computing mit dem globale Ziel der Laufzeitverbesserung von besonderem Interesse. So kann festgehalten werden, dass die Migration der Dienste in die unmittelbare Umgebung der Produzenten und Verbrauchern der Daten innerhalb der IoT-Segmente bei der Erfüllung dieses Ziels deutliche Vorteile gegenüber der Bereitstellung von Diensten nur in der physischen Nähe der Objekte zeigen. Eines von vielen Anwendungsbereichen, die von dieser lokalen Bereitstellung profitieren, sind Verkehrsleitsystemen, bei denen Dienste beispielsweise innerhalb einzelner Ampelanlagen des

globalen IoT zunächst lokale Aufgaben für die vernetzten Autos mit kurzen Latenzzeiten erledigen und die entstehenden Daten schließlich zur Auswertung an eine zentrale Cloud-Infrastruktur der Verkehrsleitstelle leiten können. Bei der lokalen Bereitstellung der Dienste zeigt sich auch der weitere Vorteil, dass nicht immer eine dauerhafte Verbindung zum Internet gewährleistet sein muss. Aufgrund des positiven Einflusses sollten weitere Untersuchungen in diesem Bereich stattfinden. Ein erster Ansatz könnte dabei die Bereitstellung mehrerer gleicher, autonomer Dienste in den einzelnen IoT-Segmenten sein, die kollaborativ über einer Cloud-Infrastruktur zusammen arbeiten und so globale Funktionen den Clients im IoT bereitstellen. Zu diesem Thema wäre auch eine standortverteilte Untersuchung auf Basis echter IoT-Netzwerke und echter Cloud-Infrastruktur anstelle eines virtuellen Netzwerkes von Interesse. Da bisher die Entscheidungen der Service Manager Plattform für die Migration eines Dienstes auf zuvor bekannten Informationen über die Routen des Netzwerkes basieren, sollte hier eine Implementierung in Richtung flüchtiger, lokaler Informationen eines einzelnen Objektes im IoT erfolgen. Dadurch könnte nachgewiesen werden, dass die Bereitstellung von Diensten durch die Plattform auch in einer globalen Größe möglich wäre. Die verschiedenen topologischen Verfahren können dazu Anreize geben, wie eine Migration im IoT durchgeführt werden kann. Ein weitere Analyse sollte in die Richtung der Softwareverteilung und Autonomie von Dienste gehen. Eine automatisierte Softwareverteilung zur Einspeisung von Updates während des Betriebes könnte für eine dauerhafte Bereitstellung des Dienste ohne Ausfallzeiten und für eine aktuelle Software der Plattform und des Dienste zu jedem Zeitpunkt sorgen. Auch die Unterstützung Container-basierter Systeme wie LXC oder Docker [73, 74] wäre ein großer Gewinn für die Plattform, da sie eine Isolation und Autonomie von Anwendungen während der Ausführung erlauben. So wäre es bereits in naher Zukunft möglich allen Personen alle Objekte des Internets der Dinge zu jeder Zeit zur Verfügung zu stellen.

Literatur

[1] ITU-T. *Overview of the Internet of things*. Recommendation Y.4000/Y.2060. International Telecommunication Union, Juni 2012.

[2] Luigi Atzori, Antonio Iera und Giacomo Morabito. „The internet of things: A survey“. In: *Computer networks* 54.15 (2010), S. 2787–2805.

[3] Ala Al-Fuqaha et al. „Internet of things: A survey on enabling technologies, protocols, and applications“. In: *Communications Surveys & Tutorials, IEEE* 17.4 (2015), S. 2347–2376.

[4] Alessandro Bassi und Geir Horn. „Internet of Things in 2020: A Roadmap for the Future“. In: *European Commission: Information Society and Media* (2008).

[5] Daniel Giusto et al. *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010.

[6] Daniele Miorandi et al. „Internet of things: Vision, applications and research challenges“. In: *Ad Hoc Networks* 10.7 (2012), S. 1497–1516.

[7] Geng Wu et al. „M2M: From mobile to embedded internet“. In: *IEEE Communications Magazine* 49.4 (2011), S. 36–43.

[8] Bundesamt für Sicherheit in der Informationstechnik. *Smart Metering Systems: Intelligente Messsysteme*. Okt. 2015. URL: `https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/SmartMeter/smartmeter_node.html` (besucht am 08. 02. 2017).

[9] Gartner, Inc. *Gartner Says the Internet of Things Will Transform the Data Center*. März 2014. URL: `https://www.gartner.com/newsroom/id/2684616` (besucht am 08. 02. 2017).

[10] Cisco Systems, Inc. *The Zettabyte Era: Trends and Analysis*. White Paper. Cisco Systems, Inc., Juni 2016.

[11] Jayavardhana Gubbi et al. „Internet of Things (IoT): A vision, architectural elements, and future directions“. In: *Future Generation Computer Systems* 29.7 (2013), S. 1645–1660.

[12] Michael Armbrust et al. „A view of cloud computing“. In: *Communications of the ACM* 53.4 (2010), S. 50–58.

[13] Peter Mell, Tim Grance et al. „The NIST definition of cloud computing“. In: (2011).

[14] HV Jagadish et al. „Big data and its technical challenges“. In: *Communications of the ACM* 57.7 (2014), S. 86–94.

[15] Flavio Bonomi et al. „Fog computing and its role in the internet of things“. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, S. 13–16.

[16] Ivan Stojmenovic und Sheng Wen. „The fog computing paradigm: Scenarios and security issues“. In: *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE. 2014, S. 1–8.

[17] Cisco Systems, Inc. *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. White Paper. Cisco Systems, Inc., 2010.

[18] Flavio Bonomi et al. „Fog computing: A platform for internet of things and analytics“. In: *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, S. 169–186.

[19] Dennis Lehmkuhl. „Persönliches sicheres Cloud-System auf Open Source Basis“. Masterarbeit. Westfälische Wilhelms-Universität, 2017.

[20] Farah Hussein Mohammed und Roslan Esmail. „Survey on IoT Services: Classifications and Applications“. In: *Int J Sci Res* 4 (2015), S. 2124–7.

[21] Suparna De et al. „Service modelling for the Internet of Things“. In: *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*. IEEE. 2011, S. 949–955.

[22] Patrik Spiess et al. „SOA-based integration of the internet of things in enterprise services“. In: *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE. 2009, S. 968–975.

[23] Feng Chen et al. „A comprehensive device collaboration model for integrating devices with web services under internet of things“. In: *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE. 2011, S. 742–743.

[24] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[25] Philip Bianco, Rick Kotermanski und Paulo F Merson. „Evaluating a service-oriented architecture“. In: (2007).

[26] Matthias Thoma et al. „On iot-services: Survey, classification and enterprise integration“. In: *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE. 2012, S. 257–260.

[27] Xing Xiaojiang, Wang Jianli und Li Mingdong. „Services and key technologies of the internet of things“. In: *ZTE Communications* 2 (2010), S. 011.

[28] Matthew Gigli und Simon Koo. „Internet of things: services and applications categorization“. In: *Advances in Internet of Things* 1.02 (2011), S. 27.

[29] ITU-T. Recommendation Y. 2002. „Overview of ubiquitous networking and of its support in NGN.“ In: (2010).

[30] Michele Zorzi et al. „From today's intranet of things to a future internet of things: a wireless-and mobility-related view“. In: *IEEE Wireless Communications* 17.6 (2010).

[31] Thomas Erl. *SOA: principles of service design*. Bd. 1. Prentice Hall Upper Saddle River, 2008. Kap. 10.

[32] Karima Velasquez et al. „Service placement for latency reduction in the internet of things“. In: *Annals of Telecommunications* (2016), S. 1–11.

[33] Cisco Systems, Inc. *Transforming to a Next Generation IoT Service Provider*. Techn. Ber. Cisco Systems, Inc., 2015.

[34] Amazon.com, Inc. *How the AWS IoT Platform Works*. 2016. URL: `https://aws.amazon.com/iot-platform/how-it-works/` (besucht am 08. 02. 2017).

[35] Z. Shelby, K. Hartke und C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. RFC Editor, Juni 2014. URL: `http://www.rfc-editor.org/rfc/rfc7252.txt`.

[36] Hongzhou Liu et al. „Design and implementation of a single system image operating system for ad hoc networks“. In: *Proceedings of the 3rd international conference on Mobile systems, applications, and services*. ACM. 2005, S. 149–162.

[37] Georg Wittenburg und Jochen Schiller. „Service placement in ad hoc networks“. In: *PIK-Praxis der Informationsverarbeitung und Kommunikation* 33.1 (2010), S. 21–25.

[38] Gilbert Laporte, Stefan Nickel und Francisco Saldanha da Gama. *Location science*. Springer, 2015.

[39] Saumitra M Das et al. „Studying wireless routing link metric dynamics“. In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM. 2007, S. 327–332.

[40] Douglas SJ De Couto et al. „A high-throughput path metric for multi-hop wireless routing“. In: *Wireless Networks* 11.4 (2005), S. 419–434.

[41] Richard Draves, Jitendra Padhye und Brian Zill. „Routing in multi-radio, multi-hop wireless mesh networks“. In: *Proceedings of the 10th annual international conference on Mobile computing and networking*. ACM. 2004, S. 114–128.

[42] Gerhard P Fettweis. „A 5G wireless communications vision“. In: *Microwave Journal* 55.12 (2012), S. 24–36.

[43] Gerhard P Fettweis. „The tactile internet: applications and challenges“. In: *IEEE Vehicular Technology Magazine* 9.1 (2014), S. 64–70.

[44] *Taktiles Internet*. 2016. URL: `https://www.fraunhofer.de/de/forschung/aktuelles-aus-der-forschung/taktiles-internet.html` (besucht am 08. 02. 2017).

[45] Andrew S Tanenbaum und Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[46] Communications Westfälische Wilhelms-Universität Münster und Networked Systems. *Website of the MIOT-Testbed*. 2016. URL: `https://www.uni-muenster.de/MIOT-Lab/` (besucht am 08. 02. 2017).

[47] Faris Keti und Shavan Askar. „Emulation of Software Defined Networks Using Mininet in Different Simulation Environments“. In: *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*. IEEE. 2015, S. 205–210.

[48] Canonical Ltd. *Manual page of the network throughput measurement tool iperf*. 2016. URL: `http://manpages.ubuntu.com/manpages/zesty/man1/iperf.1.html` (besucht am 08. 02. 2017).

[49] linux.die.net. *Manual page of the ICMP-based reachability testing tool ping*. 2016. URL: https://linux.die.net/man/8/ping (besucht am 08. 02. 2017).

[50] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[51] OpenFog Consortium Architecture Working Group. „OpenFog Architecture Overview“. In: *White Paper* (Feb. 2016).

[52] Mohammad Abdur Razzaque et al. „Middleware for internet of things: a survey“. In: *IEEE Internet of Things Journal* 3.1 (2016), S. 70–95.

[53] Steve Neely, Simon Dobson und Paddy Nixon. „Adaptive middleware for autonomic systems“. In: *Annales des télécommunications*. Bd. 61. 9-10. Springer. 2006, S. 1099–1118.

[54] Microsoft Corporation. *Microsoft Application Architecture Guide, 2nd Edition*. 2009. URL: https://msdn.microsoft.com/en-us/library/ee658117.aspx#ComponentBasedStyle (besucht am 08. 02. 2017).

[55] Erich Gamma et al. *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp Verlags GmbH & Co. KG, 2015.

[56] tcpdump. *Offical website of tcpdump and libpcap*. 2016. URL: http://www.tcpdump.org/ (besucht am 08. 02. 2017).

[57] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[58] V. Paxson et al. *Computing TCP's Retransmission Timer*. RFC 6298. Internet Engineering Task Force, Juni 2011, S. 1–11. URL: http://www.rfc-editor.org/rfc/rfc6298.txt.

[59] EtherealMind.com. *OpenFlow and Software Defined Networking: Is it Routing or Switching?* 2011. URL: http://etherealmind.com/openflow-software-defined-networking-routing-or-switching/ (besucht am 08. 02. 2017).

[60] techtarget.com. *software-defined networking (SDN)*. 2015. URL: http://searchsdn.techtarget.com/definition/software-defined-networking-SDN (besucht am 08. 02. 2017).

[61] Open Networking Foundation. *Software-Defined Networking (SDN) Definition*. 2016. URL: https://www.opennetworking.org/sdn-resources/sdn-definition (besucht am 08. 02. 2017).

[62] Fei Hu, Qi Hao und Ke Bao. „A survey on software-defined network and openflow: from concept to implementation“. In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), S. 2181–2206.

[63] OpenDaylight. *Open Source SDN Platform*. 2016. URL: https://www.opendaylight.org/ (besucht am 08. 02. 2017).

[64] nsname. *ns-3: a discrete-event network simulator for Internet systems*. 2016. URL: https://www.nsnam.org/ (besucht am 08. 02. 2017).

[65] Min-Cheng Chan et al. „Opennet: A simulator for software-defined wireless local area network“. In: *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2014, S. 3332–3336.

[66] Stanford University. *Open Network Lab: POX SDN Controller*. 2015. URL: https://openflow.stanford.edu/display/ONL/POX+Wiki (besucht am 08. 02. 2017).

[67] Silvan Streit und Christos Kalialakis. „A POX OpenFlow Loop Solution for Mininet Network Emulations“. In: 4th International Conference on Modern Circuits und Systems Technologies (MOCAST), 2015.

[68] Project Floodlight. *OpenFlow SDN Controller Floodlight*. 2016. URL: http://www.projectfloodlight.org/floodlight/ (besucht am 08. 02. 2017).

[69] Bob Lantz, Brandon Heller und Nick McKeown. „A network in a laptop: rapid prototyping for software-defined networks“. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM. 2010, S. 19.

[70] Mininet. *Mininet: An Instant Virtual Network on your Laptop (or other PC)*. 2016. URL: http://mininet.org/ (besucht am 08. 02. 2017).

[71] Mininet Wifi. *Emulator for Software-Defined Wireless Networks (Fork of Mininet)*. 2016. URL: https://github.com/intrig-unicamp/mininet-wifi (besucht am 08. 02. 2017).

[72] Open vSwitch. *Official website of the Open vSwitch: Production Quality, Multilayer Open Virtual Switch*. 2016. URL: http://openvswitch.org/ (besucht am 08. 02. 2017).

[73] LinuxContainers.org. *Infrastructure for container projects*. 2016. URL: https://linuxcontainers.org/ (besucht am 08. 02. 2017).

[74] Docker, Inc. *Docker - Build, Ship, Run Any App, Anywhere*. 2016. URL: https://www.docker.com/ (besucht am 08. 02. 2017).

Anhang

A.1 Der Service Manager als dienstorientierte Plattform

A.1.1 Sequenzdiagramm zur Migration im Service Transporter

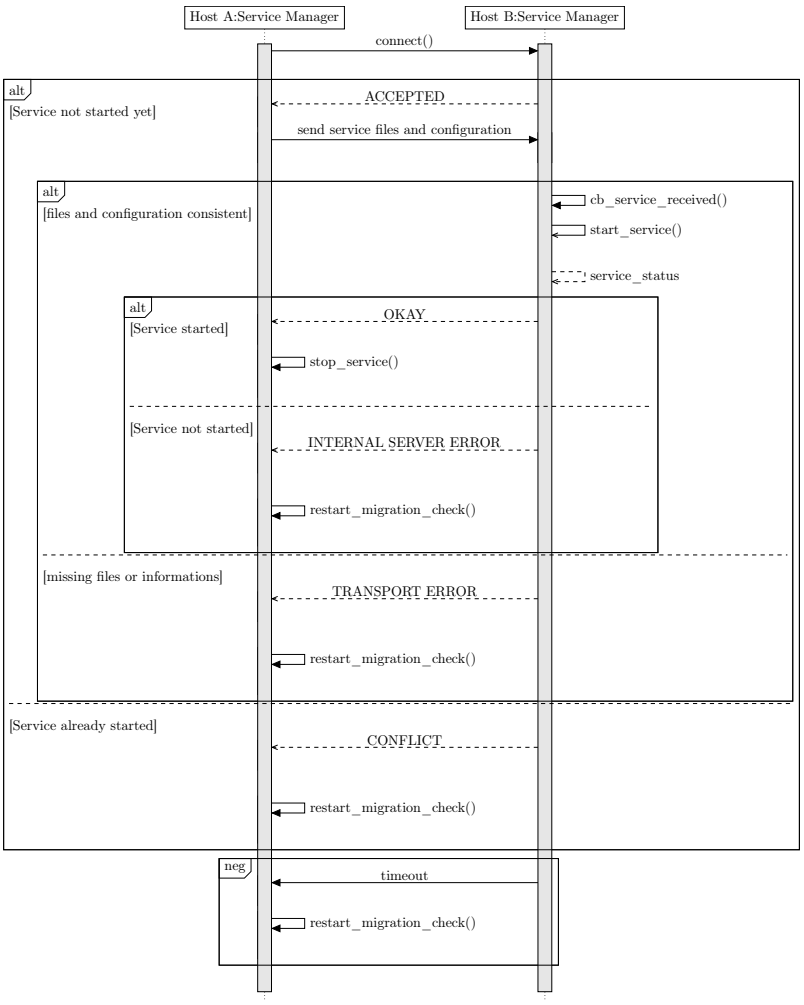


Abbildung A.1: Sequenzdiagramm zur Migration im Service Transporter

A.2 Die Evaluation der Dienstemigration im Internet der Dinge

A.2.1 Vollständiger Graph des MIOT-Testbeds zur Emulation eines IoTs

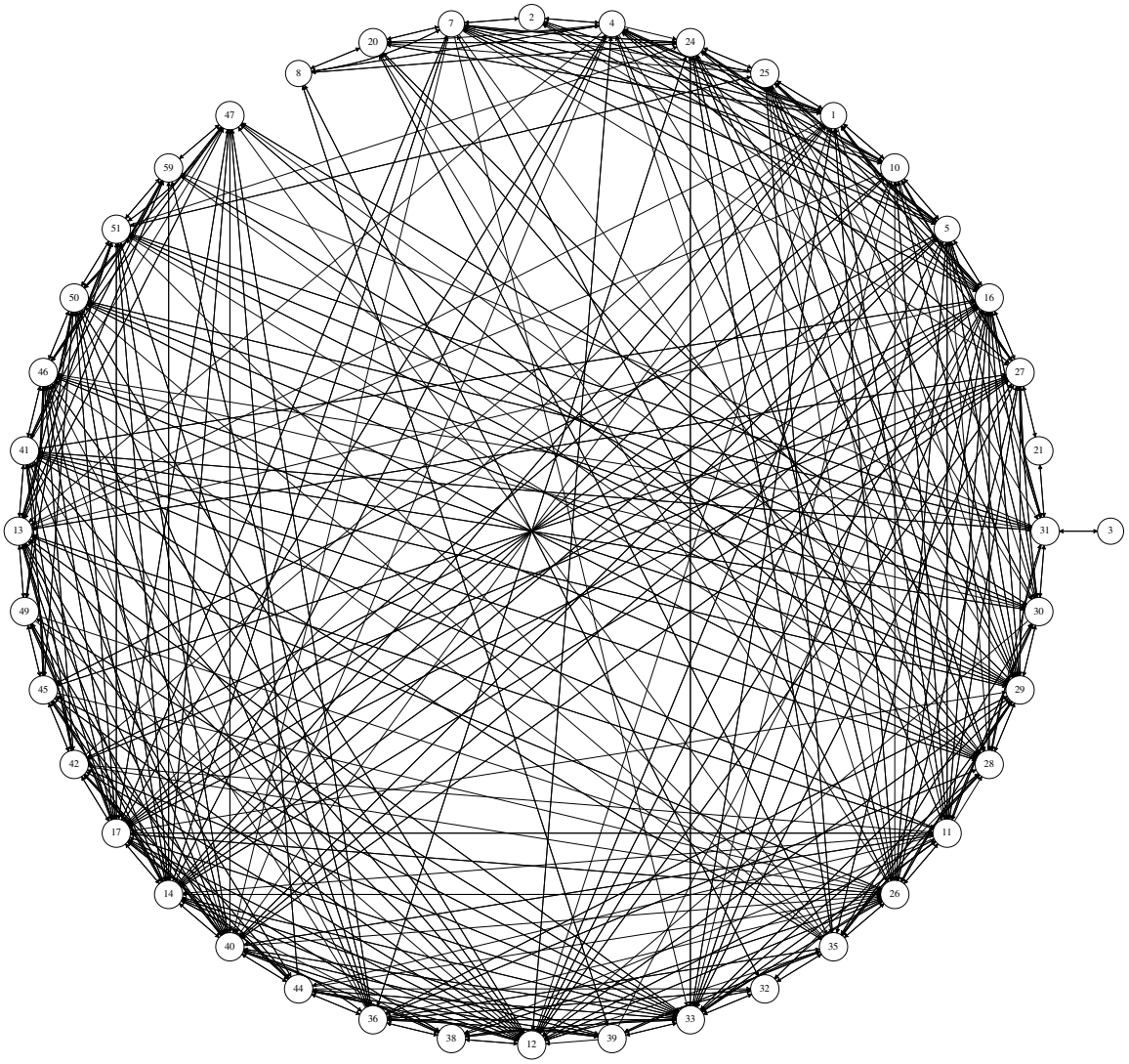


Abbildung A.2: Vollständiger Graph des MIOT-Testbeds zur Emulation eines IoTs

A.2.2 Beschnittener Graph des MIOT-Testbeds zur Emulation eines segmentierten IoTs

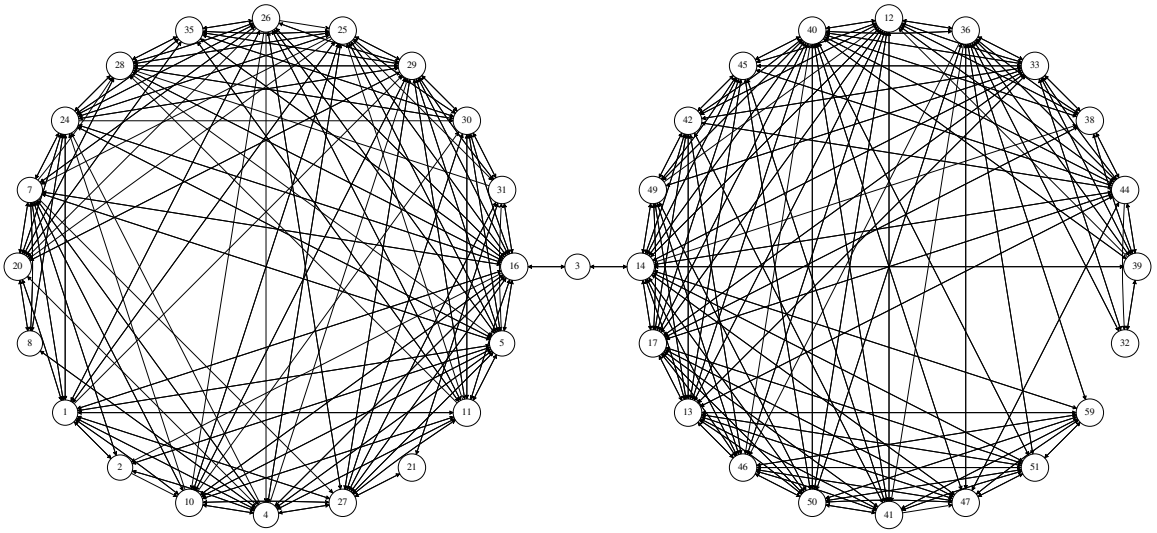
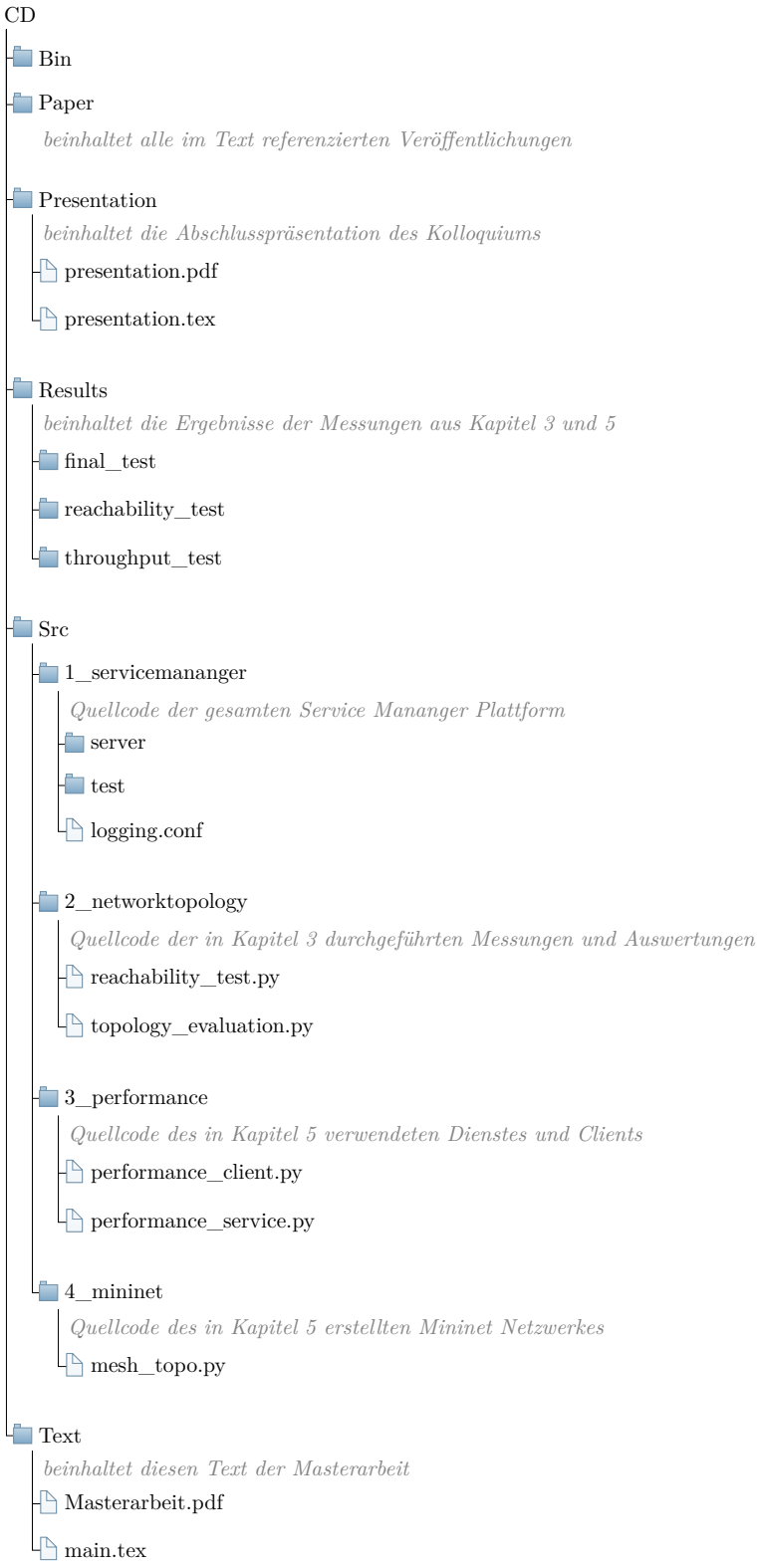


Abbildung A.3: Beschnittener Graph des MIOT-Testbeds zur Emulation eines segmentierten IoTs

A.3 Inhalt der CD



Hiermit versichere ich, dass die vorliegende Arbeit mit dem Titel *Migration autonomer Dienste im Internet der Dinge anhand topologischer Charakteristika* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Münster, 8. Februar 2017

(Simon Lansing)