Transparenz aus Kundensicht: Bausteine zum Monitoring von Cloud-Umgebungen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von

Matthias Flittner, M.Sc.
aus Fulda

Tag der mündlichen Prüfung: 06. Februar 2019

Erste Referentin: Professorin Dr. Martina Zitterbart
Karlsruher Institut für Technologie

Zweiter Referent: Professor Dr. Ralf Steinmetz
Technische Universität Darmstadt

Danksagung

By the Grace of God I Am What I Am.

VOR Ihnen liegt die Doktorarbeit “Transparenz aus Kundensicht: Bausteine zum Monitoring von Cloud-Umgebungen”. Die Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am *Institut für Telematik* des *Karlsruher Instituts für Technologie (KIT)*. Mein erster Dank gilt meiner Doktormutter und Institutsleiterin Frau Prof. Dr. Martina Zitterbart, die mich während der gesamten Zeit mit hilfreichen Diskussionen und Ratschlägen intensiv begleitete. Ebenso bedanken möchte ich mich bei Herrn Prof. Dr. Ralf Steinmetz, der sich trotz zahlreicher Verpflichtungen in Forschung und Lehre sofort bereit erklärte, das Korreferat für meine Promotion zu übernehmen.

Ebenso gilt allen Kolleginnen und Kollegen des Instituts für Telematik mein Dank. Besonders hervorheben möchte ich hier meine Freunde Martin Florian, Robert Bauer, Hauke Heseding und Roland Bless, die stets ein offenes Ohr für mich hatten, mich mit zahlreichen hilfreichen Ratschlägen hervorragend unterstützten und in wissenschaftlichen Projekten mit mir zusammengearbeitet haben. Weiter gilt mein Dank auch allen Studierenden, die als Bachelor-, Master- oder Diplomarbeiter sowie als wissenschaftliche Hilfskräfte neue Ideen eingebracht und entworfene Konzepte kritisch hinterfragt haben.

Großen Anteil am Gelingen dieser Arbeit haben nicht zuletzt auch meine Familie und Freunde, die mich immer wieder motiviert und auch in schwierigeren Phasen unterstützt haben. Mein spezieller Dank gilt insbesondere meinen Eltern, die mir mein Studium ermöglichten und die mich auf dem Weg zur Promotion zusammen mit meiner Schwester Laura sicher und geduldig begleiteten. Ein sehr großer Dank gilt auch meinem Onkel und Firmpaten Roland, der mir als Vorbild und mit wertvollen Ratschlägen immer zur Seite stand. Darüber hinaus danke ich meinen geschätzten Freunden Paul, Tobi, Julia, Uffel, Michael und Arnold für die jahrelange Unterstützung und Treue. Vielen Dank euch allen namentlich und nicht namentlich Erwähnten!

Mein größter Dank gilt jedoch Gott: Frei nach dem Motto “*Der Mensch denkt und Gott lenkt*” ist sein gnädiges Handeln an mir nicht ohne Wirkung geblieben. Das Gehen mit Christus birgt schließlich viele Überraschungen in sich!

Und nun wünsche ich Ihnen viel Freude beim Lesen dieser Doktorarbeit,
-FliTTi

Karlsruhe, 19. März 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1RRS Singulärer Request-Response-Dienst.

AID Anfrage-ID.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

ARQ Automatic Repeat Request.

BDSG Bundesdatenschutzgesetz.

CMS Cloud-Management-System.

CoCoPAS Corporate Confidentiality Preserving Auditing Scheme.

DOM XML Document Object Model.

EU-DSGVO EU-Datenschutz-Grundverordnung.

EWMA Exponentially Weighted Moving Average.

g1RRS Gruppenbasierter 1RRS.

GID Gruppen-ID.

GMS Group Management Service.

IaaS Infrastructure-as-a-Service.

Log-Server Log-Server.

LVM Logical Volume Manager.

Monitoring-Agent Monitoring-Agent.

Monitoring-Controller Monitoring-Controller.

NIST National Institute of Standards and Technology.

NMR N-Modular Redundancy.

PaaS Platform-as-a-Service.

RPC Remote Procedure Call.

RRS Request-Response Service.

SaaS Software-as-a-Service.

SDN Software-Defined-Networking.

SFC Service-Function-Chaining.

SLA Service-Level-Agreement.

UUID Universally Unique Identifier.

VM Virtuelle Maschine.

VMM Virtual-Machine-Monitor.

Kapitel 1

Einleitung

In den letzten Jahren hat durch die rasch voranschreitende Digitalisierung der Einsatz von Cloud-Computing immer mehr an Bedeutung gewonnen [276, 263, 254]. Vor allem Unternehmen versprechen sich hierdurch große wirtschaftliche Vorteile. So ermöglicht Cloud-Computing unter anderem die flexible Anpassung der bereitgestellten Dienstleistung zur Laufzeit, kürzere Entwicklungszyklen, optimierte Ressourcennutzung und Skalierbarkeit von Anwendungen. Dabei werden virtuelle Ressourcen von unterschiedlichen Kunden häufig auf der gleichen physischen Infrastruktur des Cloud-Anbieters platziert (*resource pooling*) und nur die tatsächlich vom Kunden genutzten Ressourcen in Rechnung gestellt (*pay-per-use*).

Obwohl der Einsatz von Cloud-Computing viele Vorteile verspricht, schrecken Unternehmen vor dessen Nutzung unter anderem aus Sicherheitsbedenken zurück [140, 37]. Insbesondere in sensiblen Anwendungsbereichen fehlt bei einigen Unternehmen das Vertrauen, dass das Cloud-Management-System (CMS) die vom Kunden geforderte Dienstleistung wie gewünscht umsetzt. In der Literatur hat sich in diesem Kontext der Begriff *Lack of Transparency* etabliert. Dabei sind sowohl Kunden als auch Cloud-Anbieter der Funktionalität des CMS praktisch "ausgeliefert". Dabei scheitert die Überprüfung durch Kunden bereits an mangelnden Zugriffsmöglichkeiten auf die Cloud-Infrastruktur. Für Cloud-Anbieter hingegen ist eine vollständige Überprüfung der Konfiguration aller physischen Cloud-Knoten aufgrund der Vielzahl an verwendeten Ressourcen und der Dynamik der Cloud-Umgebung äußerst komplex. In der Konsequenz können Fehler des CMS sowohl vom Cloud-Anbieter als auch vom Kunden unbemerkt bleiben. Darüber hinaus trauen die wenigsten Kunden einem Cloud-Betreiber zu, dass dieser die getroffenen vertraglichen Vereinbarungen ausreichend überwacht [258]. Aus diesem Grund fordern

zahlreiche Veröffentlichungen *zusätzliche Maßnahmen* zur Überprüfung des CMS und des Cloud-Anbieters [149, 77, 114, 76].

Anhand des folgenden Beispiels soll aufgezeigt werden, wie sich mangelnde Transparenz auf Seiten des Kunden auswirkt. Wenn beispielsweise ein Kunde mit einem Cloud-Anbieter vereinbart hat, dass eine angemietete Virtuelle Maschine (VM) nur auf physischen Cloud-Knoten ausgeführt werden darf, auf denen derzeit keine VMs von anderen Kunden ausgeführt werden. Eine Anforderung, die in der Industrie zur Isolation von kritischen Diensten häufig Verwendung findet und von großen Cloud-Anbietern bereits adaptiert wurde [246, 268]. Nachdem der Kunde beim Cloud-Anbieter die Isolation einer bestimmten VM in Auftrag gegeben hat, hat er keine Möglichkeit mehr, zu überprüfen, ob diese Eigenschaft zur Laufzeit vom CMS des Cloud-Anbieters tatsächlich wie angefordert umgesetzt wird. Vielmehr muss er sich auf die Angaben in der Kundenschnittstelle des CMS verlassen. Diese können jedoch inkonsistent zu dem in der Cloud-Infrastruktur etablierten Zustand sein – beispielsweise weil das CMS vom Cloud-Anbieter nicht (wie vom Hersteller empfohlen) aktualisiert wurde. Durch die systemimmanente Opazität von Cloud-Computing erfährt der Kunde weder, dass seine Vereinbarungen mit dem Cloud-Anbieter derzeit gegebenenfalls vom CMS nicht eingehalten werden, noch wird dieser Sachverhalt vom Cloud-Anbieter ausreichend dokumentiert, so dass im Falle einer Fehler-Ursachen-Analyse vor Gericht vorkommen kann, dass ein Versagen des CMS nicht nachgewiesen werden kann.

Darüber hinaus spielt Transparenz im Sinne von Überprüfbarkeit und Nachvollziehbarkeit auch aus rechtlicher Sicht eine große Rolle. Insbesondere dann, wenn personenbezogene Daten zur Auftragsdatenverarbeitung an einen Cloud-Anbieter ausgelagert werden. Hierzu wurde unter anderem mit Inkrafttreten der EU-Datenschutz-Grundverordnung (EU-DSGVO) im Mai 2018 [155] das Bundesdatenschutzgesetz (BDSG) [27] grundlegend neu geregelt. Beispielsweise finden sich hier jetzt Regelungen zur Überprüfung der vertraglichen Vereinbarungen zwischen Kunde und Cloud-Anbieter (vergleiche Art. 28 Abs. 3 Buchstabe h der EU-DSGVO [155], §62 Abs. 5 Ziffer 6 des neuen BDSG [27] oder die aus §11 des alten BDSG [26] abgeleiteten Kontrollpflichten [302]). Außerdem wird für Auftragsdatenverarbeiter nun explizit die Möglichkeit zur datenschutzkonformen Zertifizierung geschaffen. Ähnliche Regelungen zur Überprüfung von vertraglichen Vereinbarungen und für Nachvollziehbarkeit im Fall von Fehler-Ursachen-Analysen finden sich auch abseits der Auftragsdatenverarbeitung, beispielsweise im IT-Sicherheitsgesetz [68] oder im BSI-Gesetz [25]. Der Gesetzgeber fordert hier Transparenz als einen integralen Bestandteil für sichere Systeme und Kritische Infrastrukturen.

Ziel dieser Dissertation ist aus oben genannten Gründen daher die Entwicklung von funktionalen Bausteinen für CMS-unabhängiges Monitoring von Cloud-Umgebungen. Dabei steht explizit Cloud-Monitoring aus Kundensicht im Vordergrund. Durch die im

weiteren Verlauf dieser Dissertation vorgestellten Bausteine sollen die vom CMS in der Cloud-Umgebung vorgenommenen Konfigurationen für Kunden überprüfbar werden. Außerdem soll für Kunden die Aufzeichnung von Monitoringdaten für spätere Fehler-Ursachen-Analysen ermöglicht werden. Auf diese Art und Weise soll dem bei Cloud-Computing vorherrschenden Mangel an Transparenz zumindest in Teilen entgegengewirkt werden.

1.1 Problemstellung und Annahmen

Steht das Vertrauen in die Funktionalität des CMS zur Disposition, so kann beim Monitoring einer Cloud-Umgebung nicht auf selbiges zurückgegriffen werden. Andernfalls können Fehlfunktionen des CMS unter Umständen Implikationen auf das Monitoring der Cloud-Umgebung haben.

Im Fokus dieser Dissertation steht daher die *Trennung von Monitoring und Kontrolle*. Während die *Kontrolle* der Cloud-Umgebung weiterhin vom CMS ausgeübt werden kann, soll das *Monitoring* der Cloud-Umgebung unabhängig vom CMS realisiert werden, um gegebenenfalls Fehlfunktionen aufdecken zu können. Im Rahmen dieser Dissertation werden ausgewählte Monitoring-Bausteine zur Realisierung von CMS-unabhängigem Cloud-Monitoring betrachtet. Deren jeweilige spezifische Problemstellung wird nachstehend kurz vorgestellt.

Adaptives Transportsystem für Request-Response in Gruppen. Eine maßgebliche Problemstellung im Kontext der Trennung von Monitoring und Kontrolle ist die effiziente Abfrage der auf den physischen Cloud-Knoten ermittelten Monitoringdaten. Da zur Durchführung von Monitoring im Kontext von Cloud-Computing in der Regel Daten von mehrere physischen Cloud-Knoten benötigt werden, empfiehlt sich die Erweiterung des Request-Response-Kommunikationsmusters auf Szenarien mit Gruppen. Hierdurch ergeben sich allerdings mehrere schwierige Herausforderungen. Zunächst muss dafür die bei Request-Response typische Semantik auf Gruppen erweitert werden. Für Cloud-Monitoring müssen hierfür geeignete gruppenbasierte Request-Response-Klassen identifiziert werden. Außerdem soll die Anzahl der erwarteten Antworten parametrisierbar sein. Dabei stellten sich insbesondere die Ausgestaltung der Verlustbehandlung und effiziente Umsetzung von parametrisierbaren Request-Response für Gruppen als besonders herausfordernd dar. Im Vergleich zu existierenden Lösungsansätzen (siehe unter anderem [173]) wird in dieser Dissertation explizit der Fokus auf parametrisierbare Request-Response-Kommunikation in Gruppen gelegt und die effiziente Übertragung von Nachrichten über das Netz mit berücksichtigt. Bisherige Verfahren bieten entweder Garantien für zuverlässigen unidirektionalen

Multicast oder Request-Response-Kommunikation für Gruppen ohne Berücksichtigung von Effizienz oder Antwort-Vollständigkeit.

Verdeckender Ungleichheitsnachweis für Monitoringdaten. Cloud-Monitoring birgt das grundsätzliche Spannungsverhältnis zwischen Überprüfbarkeit und Schutz von Betriebsgeheimnissen in sich. Zwar sollen die vertraglichen Vereinbarungen zwischen Kunde und Cloud-Anbieter zur Laufzeit überprüfbar und im Nachhinein nachvollziehbar sein, allerdings müssen dabei die Betriebsgeheimnis des Cloud-Anbieters vertraulich bleiben. Dies gilt insbesondere für die beim Cloud-Monitoring ermittelten Monitoringdaten. An vollständiger Offenlegung aller internen Details gegenüber Kunden kann der Cloud-Anbieter aus naheliegenden Gründen kein Interesse haben. Insbesondere für eine Überprüfung auf Ungleichheit von Monitoringdaten (beispielsweise um zu überprüfen, ob zwei VMs auf unterschiedlichen physischen Cloud-Knoten platziert wurden) ergeben sich hier mehrere herausfordernde Problemstellungen. Die Überprüfung auf Ungleichheit muss die konkreten Monitoringdaten vor dem Kunden verstecken und nur Auskunft darüber liefern, ob diese verschieden zueinander sind. Außerdem muss sichergestellt werden, dass der Cloud-Anbieter auf die während der Überprüfung verwendeten Monitoringdaten festgelegt ist und deren Existenz nicht im Nachhinein abstreiten kann. Darüber hinaus muss im Fall von Fehler-Ursachen-Analysen vor Gericht sichergestellt werden, dass die während einer Überprüfung vertraulich behandelten Monitoringdaten eindeutig aufgedeckt werden können. Im Vergleich zu existierenden Lösungsansätzen wird in dieser Dissertation ein neues auf Zero-Knowledge-Beweisen und Hinterlegungsverfahren basierendes Verfahren zur Überprüfung auf Ungleichheit von Monitoringdaten vorgestellt. Insbesondere die Ungleichheit von per Hinterlegungsverfahren hinterlegten Werten ist hier ein besonderer Schwerpunkt. Die Überprüfung von Gleichheit gehört hingegen bereits zum Stand der Forschung.

Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen. Darf zur Erfassung der Cloud-Konfiguration nicht auf Informationen des CMS zugegriffen werden, so müssen Monitoringdaten über die Cloud-Konfiguration direkt auf den physischen Cloud-Knoten ermittelt werden. Dabei ergeben sich mehrere Schwierigkeiten. Zum einen muss sichergestellt werden, dass die Zuordnung zwischen virtuellen Ressourcen einer VM (*Compute*, *Network* und *Storage*) und Kundenkontext wiederhergestellt werden kann. Zum anderen muss die vom CMS verursachte Cloud-Dynamik (Migrationen und Starten beziehungsweise Stoppen von VMs) berücksichtigt werden. So werden beispielsweise VMs vom CMS auf einen anderen physischen Cloud-Knoten migriert, um Energiekosten zu sparen oder dem Cloud-Anbieter Wartungsarbeiten zu ermöglichen. Im Vergleich zu existierenden Lösungsansätzen (siehe u.a. [171, 55, 3]) lässt sich im Rahmen dieser Dissertation die Unabhängigkeit vom CMS als besondere Herausforderung identifizieren. Weiterhin stellt sich im Vergleich mit ver-

wandten Arbeiten die unmittelbare Überprüfung (auf Nachfrage) der aktuellen Cloud-Konfiguration auf den physischen Cloud-Knoten als besonderes Abgrenzungskriterium dar.

Diese Dissertation baut darüber hinaus auf den folgenden Annahmen auf.

Ehrlicher Cloud-Anbieter. Generell gelingt die Absicherung in Szenarien mit unehrlichem Cloud-Anbieter nur sehr schwer. Schließlich hat der Cloud-Anbieter volle Kontrolle über Hard- und Software einer Cloud-Umgebung und kann selbst bei Einsatz der modernsten vertrauenswürdigen Laufzeitumgebungen die für das Cloud-Monitoring relevanten Datenquellen manipulieren. Im Rahmen dieser Dissertation wird allerdings davon ausgegangen, dass solange kein Rechtsstreit zwischen Kunde und Cloud-Anbieter vorliegt, der Kunde seinem Cloud-Anbieter prinzipiell vertrauen kann. Dieses hohe Maß an Vertrauen wird durch das Geschäftsmodell des Cloud-Anbieters begründet. Sollten Kunden in ihrem Vertrauen enttäuscht werden, so würde dies auf kurz oder lang das Geschäftsmodell des Cloud-Anbieters gefährden. Außerdem macht die Auslagerung von IT-Infrastruktur an einen Cloud-Anbieter aus Kundensicht keinen Sinn, wenn dem Cloud-Anbieter nicht grundsätzlich vertraut wird. Allerdings erstreckt sich dieses Vertrauen nicht auf das vom Cloud-Anbieter eingesetzte CMS. Weiter wird davon ausgegangen, dass das Vertrauen zwischen Kunde und Cloud-Anbieter im Fall von Rechtsstreitigkeiten erlischt.

Zusätzliche Monitoring-Sichten. Für diese Arbeit wird grundlegend davon ausgegangen, dass sich zusätzlich zur "Sicht" des CMS weitere davon unabhängige *Monitoring-Sichten* auf die Infrastruktur der Cloud-Umgebung aufbauen lassen. Die verschiedenen voneinander unabhängigen Sichten lassen sich dann beispielsweise in Form eines Mehrheitsentscheids nutzen, um die tatsächliche Cloud-Konfiguration in Gegenwart fehlerhafter Sichten zu ermitteln (verwandte Konzepte finden sich im Bezug zur Gewährleistung von Sicherheit durch N-Modular Redundancy (NMR) [125, 45]). Im Rahmen dieser Arbeit wird allerdings eine stärkere Annahme genutzt. Zum einen wird vorausgesetzt, dass gleichzeitiges Auftreten eines verdeckten Fehlers in der Monitoring-Sicht und der Sicht des CMS sehr unwahrscheinlich ist, so dass davon ausgegangen wird, dass kein Fehler im CMS vorliegt, wenn beide Sichten deckungsgleich zueinander sind. Zum anderen wird davon ausgegangen, dass die Erzeugung einer zusätzlichen Sicht auf die Cloud-Infrastruktur wesentlich weniger Komplexität aufweist als die im CMS vorhandene Komplexität. Für den Fall dass die beiden Sichten sich voneinander unterscheiden wird hier ohne Beschränkung der Allgemeinheit angenommen, dass ein Fehler im CMS vorliegt. Soll auf diese stärkere Annahme verzichtet werden, so ist der Einsatz von mehreren voneinander unabhängigen Sichten und die Durchführung eines Mehrheitsentscheids unabdingbar, was allerdings nicht Gegenstand dieser Dissertation ist.

1.2 Zielsetzung und Beiträge

Ziel dieser Dissertation ist es, dem Mangel an Transparenz bei Cloud-Computing durch geeignete Monitoring-Bausteine entgegenzuwirken. Schwerpunkte dieser Dissertation sind dabei die folgenden drei Monitoring-Bausteine: 1) Adaptives Transportsystem für Request-Response in Gruppen, 2) Verdeckender Ungleichheitsnachweis für Monitoringdaten und 3) Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen. Darüber hinaus wird das Zusammenspiel der drei Bausteine in einem Rahmenwerk für CMS-unabhängiges Cloud-Monitoring in der Praxis demonstriert.

In den folgenden Abschnitten werden die Beiträge der einzelnen Monitoring-Bausteine kurz beleuchtet.

Adaptives Transportsystem für Request-Response in Gruppen. Ein originärer Beitrag dieser Dissertation ist die Entwicklung verschiedener Request-Response-Klassen für bidirektionale-Zuverlässigkeit in Szenarien mit Gruppen. Hier ist insbesondere die effiziente Realisierung von Request-Response-Klassen durch adaptive Auswahl zwischen Unicast und Multicast für den Datentransport hervorzuheben. Im Rahmen dieser Dissertation wurde dazu ein auf Request-Response-Kommunikation aufbauendes Transportsystem für Gruppen entwickelt, das einerseits die Verwaltung der Gruppenmitgliedschaft durch Monitoring-Agenten ermöglicht und anderseits Monitoring-Controllern die Angabe von Parametern für die Durchführung von Request-Response in Gruppen erlaubt (siehe auch [TT-3]). Im direkten Vergleich mit dem Stand der Forschung erreicht das in dieser Dissertation entwickelte Transportsystem eine höhere Antwort-Vollständigkeit und die Steigerung der Effizienz bei Nutzung von Netzressourcen. Die zur Leistungsbeurteilung durchgeführten Parameterstudien zeigen, dass aufgrund der niedrigen Verlustrate in Datenzentren bei Fehlen von Response-Nachrichten in der Regel immer Sendewiederholungen per Unicast sinnvoll sind (anstatt erneut die ganze Gruppe zu adressieren). Weiter zeigen die Parameterstudien auf, dass durch die geschickte Auswahl zwischen Unicast und Multicast für den Versand von Request-Nachrichten ein Einsparungsfakor von bis $7x$ möglich ist.

Verdeckender Ungleichheitsnachweis für Monitoringdaten. Ein weiterer bedeutender Beitrag dieser Dissertation ist der Nachweis von Ungleichheit zweier mittels eines Hinterlegungsverfahrens hinterlegten Werte, ohne dabei die hinterlegten Werte selbst offenlegen zu müssen. Hinzu kommt die formale Analyse des entwickelten Verfahrens hinsichtlich unterschiedlicher Sicherheitseigenschaften. In dieser Dissertation wird dabei aus verschiedenen existierenden kryptographischen Bausteinen ein neuartiges Verfahren entwickelt, das während einer aktiven Überprüfung Monitoringdaten

des Cloud-Anbieters vertraulich behandelt, dem Kunden das Resultat der Überprüfung zugänglich macht und im Nachhinein die Details der Überprüfung gegenüber berechtigten Parteien nicht-abstreitbar offenlegen kann (siehe auch [TT-13]). Die Beurteilung des entwickelten Verfahrens erfolgt aus zwei Gesichtspunkten. Zum einen wird in Form einer analytischen Betrachtung überprüft, unter welchen Bedingungen kryptographische Eigenschaften erreicht werden können. Zum anderen wird anhand eines Prototypen gezeigt, welche Laufzeit und Overhead das entwickelte Verfahren für verschiedene Sicherheitsgrade mit sich bringt. Im Ergebnis wird gezeigt, dass das Spannungsverhältnis zwischen Überprüfbarkeit und Schutz von Betriebsgeheimnissen durch geeignete Verfahren auflösbar ist. Der Verdeckende Ungleichheitsnachweis bietet für die Überprüfung auf Ungleichheit Geheimhaltung von Monitoringdaten bei gleichzeitiger Option zur späteren eindeutigen Wiederaufdeckbarkeit.

Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen. Ein wesentlicher Beitrag dieser Dissertation zum Stand der Forschung ist die systematische Untersuchung der Trennung von Monitoring und Kontrolle für Cloud-Umgebungen. Hierzu werden mögliche Datenquellen für Cloud-Monitoring theoretisch untersucht und anschließend die Implikationen der ermittelten Ergebnisse auf reale Cloud-Umgebungen beleuchtet. Als Resultat wird eine neuartige Taxonomie zur Klassifikation von Monitoring-Unabhängigkeit präsentiert. Außerdem wird untersucht, unter welchen Bedingungen eine konsistente Erfassung in einer verteilten Cloud-Umgebung überhaupt möglich ist. Darüber hinaus wird anhand von Fallstudien die Anwendbarkeit von CMS-unabhängigem Cloud-Monitoring demonstriert und der auf physischen Cloud-Knoten entstehende Overhead untersucht. Im Rahmen dieser Dissertation wird dazu ein Rahmenwerk für unabhängiges Cloud-Monitoring entwickelt (siehe auch [TT-5]) und die Integration der Monitoring-Bausteine besprochen.

1.3 Gliederung

In Kapitel 2 werden für das Verständnis dieser Arbeit notwendige Grundlagen vermittelt. Hierfür wird zunächst eine Einordnung des für diese Arbeit notwendigen Verständnis von Cloud-Computing vorgenommen. Darüber hinaus werden wesentliche Aspekte für Transportsysteme, Kommunikationsdienste, Gruppenkommunikation und Request-Response-Dienste ausgeführt. Abschließend wird kurz auf den Zusammenhang zwischen Transparenz und Vertrauen eingegangen.

In Kapitel 3 wird ein Rahmenwerk für unabhängiges Cloud-Monitoring für den weiteren Verlauf dieser Dissertation zu Grunde gelegt. Zunächst wird sich dabei umfassend mit dem Stand der Forschung zu Cloud-Monitoring beschäftigt. Anschließend wird der

Systementwurf eines Rahmenwerks für CMS-unabhängiges Cloud-Monitoring detailliert wird. Abschließend werden die im Rahmen dieser Dissertation vorgestellten Monitoring-Bausteine eingeordnet.

In Kapitel 4 wird ein Adaptives Transportsystem für Request-Response in Gruppen als Monitoring-Baustein vorgestellt. Dafür wird zunächst das anvisierte Konzept und die adressierten Herausforderungen erläutert. Anschließend wird der *Group Management Service* und *Request-Response Service* ausführlich dargestellt. Danach wird auf Sicherheitsaspekte und verwandte Arbeiten eingegangen bevor abschließend eine umfassende Evaluation des Ansatzes folgt.

Im Mittelpunkt des Kapitels 5 steht ein Verdeckender Ungleichheitsnachweis für Monitoringdaten. Neben den Anforderungen und dem Angreifermodell wird zunächst das Konzept des Verfahrens erörtert. Hiernach werden die drei Phasen: Hinterlegung-, Überprüfung- und Aufdeckungsphase präsentiert. Abschließend erfolgt eine Evaluation und Betrachtung von verwandten Arbeiten.

In Kapitel 6 wird der Begriff *Monitoring-Unabhängigkeit* präzisiert, bevor der verfolgte Forschungsansatz dargestellt wird. Weiter werden mögliche Datenquellen für CMS-unabhängiges Cloud-Monitoring exploriert und anschließend die Implikationen auf reale Cloud-Umgebungen untersucht. Auf Basis der durchgeführten Analyse wird eine Taxonomie für Monitoring-Unabhängigkeit herausgearbeitet. Abschließend erfolgt eine Betrachtung von Seiteneffekten.

In Kapitel 7 wird das Zusammenspiel der im Rahmen dieser Dissertation entwickelten Monitoring-Bausteinen mit unterschiedlichen Monitoring-Mechanismen in dem Rahmenwerk für unabhängiges Cloud-Monitoring beschrieben. Hierfür werden zunächst verschiedene Monitoring-Mechanismen beispielhaft konstruiert und die Anwendbarkeit untersucht.

Die Arbeit schließt in Kapitel 8 mit einer Zusammenfassung der relevanten Ergebnisse dieser Dissertation und einem Ausblick auf mögliche weiterführende Arbeiten.

Kapitel 2

Grundlagen

Im Folgenden werden zum Verständnis dieser Arbeit notwendige Grundlagen erörtert.

2.1 Cloud-Computing

Seit Anfang der 2000er Jahre gewinnt Cloud-Computing im Umfeld von Unternehmen immer mehr an Bedeutung. Aber auch bei Privatpersonen steht die Nutzung von Cloud-basierten Anwendungen voll im Trend. So sieht beispielsweise das Marktforschungsunternehmen *Gartner* im Bereich von Cloud-Computing bis zum Jahr 2019 für Cloud-Anbieter eine Umsatzsteigerung von bis zu 17.3% voraus [264]. Ergänzend hierzu sagt das Wirtschaftsmagazin *Forbes* für das Jahr 2020 voraus, dass rund 83% der von Unternehmen benötigten Computer-Ressourcen (unter anderem Rechenleistung, Arbeitsspeicher und Festspeicher) durch Cloud-Umgebungen bereitgestellt werden [263]. Auch der Netzwerkgerätehersteller und Dienstleister *Cisco Systems* sieht in seinem *Global Cloud Index* bis zum Jahr 2021 voraus, dass der durch Cloud-Computing entstehende Datenverkehr sich jährlich auf bis zu 20.6 Zettabyte addiert und insgesamt 95% des Gesamtverkehrs von Datenzentren ausmachen wird [254].

Als wegweisende Veröffentlichungen auf dem Gebiet von Cloud-Computing sind hier unter anderem die Arbeiten von Armbrust et al. von der *University of California at Berkeley* zu nennen [8, 9] (derzeit über 7.000 beziehungsweise 10.000 Zitierungen).

Im Folgenden wird zunächst eine gängige Definition von Cloud-Computing wiedergegeben. Anschließend wird die Architektur von Cloud-Umgebungen besprochen. Zum Abschluss werden aktuelle Entwicklungen im Bereich Cloud-Computing aufgezeigt.

Definition

In der Literatur wird für Cloud-Computing derzeit mit Abstand am häufigsten auf die Definition des National Institute of Standards and Technology (NIST) Bezug genommen [143] (derzeit ca. 10.000 Zitierungen). Diese Definition unterscheidet in Charakteristika, Dienstleistungs- und Bereitstellungsmodelle. Diese werden im Folgenden kurz wiedergegeben.

Vom NIST identifizierte Charakteristika von Cloud-Computing sind:

On-demand self-service: Die Verwaltung der vom Cloud-Anbieter bereitgestellten Ressourcen kann einseitig durch Kunden ohne zusätzliche menschliche Interaktion mit dem Cloud-Anbieter erfolgen.

Broad network access: Die in der Cloud-Umgebung bereitgestellten Ressourcen sind für den Kunden von außen erreichbar. In der Regel über das öffentliche Internet.

Resource pooling: Der Cloud-Anbieter bündelt seine physischen Ressourcen, um diese für mehrere Kunden nutzbar zu machen. Dabei hat der Kunde prinzipiell keine Einsicht darin, wie seine virtuellen Ressourcen bereitgestellt werden oder welche physischen Ressourcen zum Einsatz kommen. Beispiele für solche physischen Ressourcen sind Festspeicher, Rechenleistung oder Arbeitsspeicher.

Rapid elasticity: Der Kunde kann nach Bedarf die Menge der für ihn bereitgestellten Ressourcen anpassen. Dies soll insbesondere Skalierbarkeit unterstützen. Hierbei sollen theoretisch keine Grenzen gesetzt werden und dem Kunden bei Bedarf zu jedem Zeitpunkt unbegrenzte Ressourcen zur Verfügung stehen.

Measured service: Zur Abrechnung kommt bei Cloud-Computing typischerweise das *pay-per-use*-Konzept zum Einsatz. Hierzu überwacht die Cloud-Umgebung automatisch die vom Kunden angeforderten virtuellen Ressourcen und stellt diese in Rechnung. Außerdem kann sich der Cloud-Anbieter so einen Überblick über die vom Kunden genutzten Ressourcen verschaffen.

Weiter führt das NIST die folgenden Dienstleistungsmodelle auf:

Software-as-a-Service (SaaS): Die Cloud-Umgebung stellt dem Kunden den Zugriff auf eine bestimmte Anwendung zur Verfügung. Im Hintergrund stellt die Cloud-Umgebung die für die Anwendung benötigten Ressourcen automatisch bereit und skaliert diese bei Bedarf. Der Kunde kontrolliert dabei nicht direkt die zur Verfügung gestellten Ressourcen. Die Anwendung ist in der Regel für den Kunden von außen erreichbar (beispielsweise über das Internet). Beispiele für Software-as-a-Service (SaaS) sind *Office 365* [288] oder *Dropbox* [261].

Platform-as-a-Service (PaaS): Dem Kunden wird durch die Cloud-Umgebung Zugriff auf Programmierschnittstellen und Bibliotheken zur Anwendungsentwicklung gewährt. Die auf Basis dessen vom Kunden entwickelten Anwendungen werden anschließend von der Cloud-Umgebung ausgeführt. Während bei SaaS der Kunde wirklich nur die von der Anwendung bereitgestellte Dienstleistung nutzen kann, hat der Kunde hier Einfluss auf die Logik der Anwendung und kann diese nach Belieben anpassen. Allerdings erhält der Kunde auch in diesem Dienstleistungsmodell keine Kontrolle über die von der Cloud-Umgebung zur Verfügung gestellten Ressourcen. Beispiele für Platform-as-a-Service (PaaS) sind die *Google App Engine* [266] oder *AWS Elastic Beanstalk* [250].

Infrastructure-as-a-Service (IaaS): In diesem Fall werden dem Kunden virtuelle Ressourcen wie Rechenleistung, Arbeitsspeicher und Festspeicher von der Cloud-Umgebung als VM zur Verfügung gestellt. Über diese Ressourcen kann der Kunde dann frei verfügen und innerhalb einer VM beliebige Anwendungen ausführen. Außerdem hat der Kunde hier direkten Einfluss auf die Menge der bereitgestellten Ressourcen und kann diese nach Belieben horizontal oder vertikal skalieren. Unter horizontaler Skalierung versteht man dabei den Einsatz weiterer VMs. Vertikale Skalierung hingegen umfasst die Erweiterung der für eine VM bereitgestellten Ressourcen. Beispiele für Infrastructure-as-a-Service (IaaS) sind *Amazon EC2* [245] oder *Microsoft Azure* [280].

Als Bereitstellungsmodell von Cloud-Computing unterscheidet das NIST in:

Private cloud: Die Cloud-Umgebung steht exklusiv nur einem einzigem Kunden zur Verfügung. Dabei kann der Kunde selbst die Rolle des Cloud-Anbieters einnehmen oder auch eine dritte Partei mit der Verwaltung der Cloud-Umgebung betrauen. In einigen Fällen wird die Cloud-Umgebung dabei sogar lokal bei dem Kunden selbst aufgebaut.

Community cloud: Bei diesem Bereitstellungsmodell wird speziell für eine bestimmte Gruppe von Unternehmen eine eigne Cloud-Umgebung zur Verfügung gestellt. Dabei können sich Unternehmen zu Gruppen zusammenschließen und selbst eine *Community cloud* aufbauen, oder ein Cloud-Anbieter stellt seine Dienstleistung exklusiv einer bestimmten Branche zur Verfügung, beispielsweise für alle Finanzdienstleister.

Public cloud: Die Nutzung der Cloud-Umgebung steht der allgemeinen Öffentlichkeit zur Verfügung. Ein Cloud-Anbieter bietet hierfür in der Regel über das Internet seine Dienstleistung an.

Hybrid cloud: Kommt eine Kombination der oben beschriebenen Bereitstellungsmodelle zum Einsatz, so wird dies als *Hybrid cloud* bezeichnet. Dabei bleiben die einzelnen Bestandteile eigenständig und der Zusammenschluss entsteht beispielsweise

durch Nutzung gemeinsamer Standards um Portabilität zwischen den verschiedenen Cloud-Umgebungen und Bereitstellungsmodellen zu erlauben.

Im Rahmen dieser Dissertation ist insbesondere das Dienstleistungsmodell IaaS von besonderem Interesse. Die beiden anderen Dienstleistungsmodelle werden zunächst ausgeklammert und können Gegenstand für weiterführende Arbeiten sein. Als Bereitstellungsmodell wird im Rahmen dieser Dissertation zumindest die Trennung in Cloud-Anbieter und Kunde vorausgesetzt (siehe auch nächsten Abschnitt). Eine *Private cloud*, die von einem Kunden selber betrieben wird, ist daher nicht Gegenstand der in dieser Dissertation durchgeführten Untersuchungen. Außerdem wird davon ausgegangen, dass der Cloud-Anbieter in der Regel mehrere Kunden bedient und das daher *Resource pooling* zum Einsatz kommt (vergleiche auch Charakteristika weiter oben).

Architektur

In der Literatur existieren zahlreiche Ansätze zur Beschreibung einer Cloud-Umgebung [69, 157, 1, 142, 22, 195]. Für diese Dissertation wird das architekturelle Rahmenwerk von Schöller et al. [161] zu Grunde gelegt und im Folgenden verfeinert (vergleiche auch [TT-9, TT-14]). Die von Schöller et al. [161] beschriebene Architektur eignet sich besonders, weil diese auch im Kontext von rechtlichen Fragestellungen zu Transparenz bei Cloud-Computing entstanden ist.

Anhand von Abbildung 2.1 wird erläutert, welche Akteure und Domänen von Cloud-Computing für diese Dissertation relevant sind. Dabei findet folgende Unterscheidung statt:

Cloud-Anbieter und Cloud-Anbieter-Domäne: Der Cloud-Anbieter hat vollen Zugriff auf die physische Cloud-Infrastruktur in seiner Domäne. Weiter bietet der Cloud-Anbieter für Kunden IaaS an. Das CMS des Cloud-Anbieters erlaubt Kunden das Einrichten von virtuellen Datenzentren.

Kunde und Kunden-Domäne: Kunden nutzen die per IaaS angebotene Infrastruktur, um ihrerseits eine Dienstleistung für Benutzer zu realisieren. Beispielsweise die Bereitstellung einer bestimmten Applikation. Zwischen Kunde und Benutzer kommt im Rahmen dieser Dissertation kein Cloud-Computing zum Einsatz.

Benutzer und Benutzer-Domäne: In dieser Domäne finden sich die Benutzer der eigentlichen durch Cloud-Computing erbrachten Dienstleistung. Beispielsweise eine Sekretärin, die einem Buchhaltungsprogramm Buchungen vornimmt.

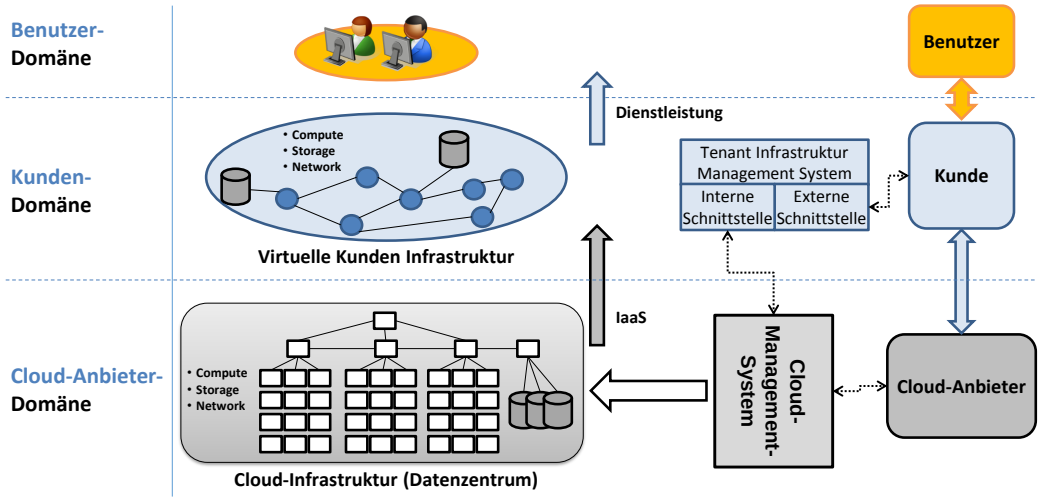

Abbildung 2.1: *Architekturelles Rahmenwerk für Cloud Computing.*

Abweichend zu [TT-9, TT-14, 161] werden hier die Domänen *Tenant Infrastructure Level* und *Critical Infrastructure Service Level* als *Kunden-Domäne* zusammengefasst. Die feine Unterscheidung der Kunden-Domäne ist hier nicht sinnvoll, da es im weiteren Verlauf dieser Dissertation nur auf die Beziehung zwischen Kunde und Cloud-Anbieter ankommt. In der Architektur von [TT-9, TT-14, 161] kann ein Kunde die Dienstleistungen der Cloud auch an seine eigenen Benutzer weiterreichen, womit der Kunde selbst die Rolle eines Cloud-Anbieters einnimmt. Die dadurch entstehende Indirektion ist für diese Dissertation allerdings uninteressant. Stattdessen ist das Rollenverhältnis der Akteure klar vorgegeben: der Kunde nutzt die vom Cloud-Anbieter bereitgestellte Cloud-Umgebung – klassisches Cloud-Computing. Die Benutzer hingegen nutzen nur die vom Kunden bereitgestellte Dienstleistung – beispielsweise eine Office-Anwendung (hier kommt kein Cloud-Computing zum Einsatz).

Daher wird im weiteren Verlauf dieser Dissertation vor allem das Verhältnis zwischen Kunde und Cloud-Anbieter betrachtet. Die entwickelten Monitoring-Bausteine sollen dabei für mehr Transparenz der vom Cloud-Anbieter bereitgestellten IaaS-Lösung sorgen. Dabei sollen sowohl vertragliche Vereinbarungen überprüfbar als auch Monitoringdaten für spätere Fehler-Ursachen-Analysen aufgezeichnet werden.

Aktuelle Entwicklungen

Rund um Cloud-Computing lassen sich derzeit auch einige aktuelle Entwicklungen und Trends ausmachen [181, 197], die die Wichtigkeit von Transparenz aus Kundensicht noch einmal unterstreichen.

Im Zuge des Vormarsches von *Internet of Things*, *5G* und *Smart Cities* wird die Forderung nach dezentralisierten Cloud-Umgebungen am *Rand des Internets*, sogenanntes *Edge* und *Fog Computing*, immer lauter [163, 167, 23]. Die Cloud-Umgebung soll dabei mit geringer Latenz große Rechenleistung mit hoher Bandbreite für eher beschränkte Geräte zur Verfügung stellen. Beispielsweise zur Analyse von Video-Streams von Verkehrsüberwachungskameras. Große Herausforderungen in diesem Bereich sind unter anderem die optimale Platzierung der Cloud-Umgebung, Mobilität der Endsysteme und die Realisierung von Privatsphäre. Gerade im Kontext von Privatsphäre spielt die Überprüfung der vom Cloud-Anbieter versprochenen Dienstleistung eine entscheidende Rolle. Beispielsweise, wenn vereinbart wurde, dass eine VM nicht zusammen mit VMs anderer Kunden auf einem physischen Cloud-Knoten ausgefürt wird.

Eine weitere aktuelle Entwicklung im Bereich von Cloud-Computing ist der Einsatz von Cloud-Umgebungen zur Analyse von großen Datenmengen wie es beispielsweise bei *Machine Learning* erforderlich ist [194, 127]. Der Einsatz von Cloud-Computing bietet hier große Vorteile, da mit wachsenden Datenmengen die Cloud-Umgebung passend skaliert werden kann. Auch ist gerade für *Machine Learning* teure Spezialhardware erforderlich, die im Fall von Cloud-Computing geteilt und dem *pay-per-use*-Konzept folgend abgerechnet werden kann. Eine große Herausforderung in diesem Bereich ist das Management von großen verteilten Datenspeichern hinsichtlich Ausfallsicherheit und Fehlertoleranz. Auch in diesem Fall wäre eine zusätzliche Überprüfungsmöglichkeit der vom Betreiber vorgehaltenen Reservekapazität zur Realisierung von Ausfallsicherheit wünschenswert.

Ein weiterer Trend im Bereich von Cloud-Computing ist die Fokussierung auf Nachhaltigkeit. Forschung hierzu wird häufig unter dem Stichwort *Green Cloud* zusammengefasst (siehe zum Beispiel [12]). Dabei ist der durch Cloud-Computing benötigte Energiebedarf zum Betrieb von physischen Cloud-Knoten immens. Hierbei ist insbesondere die Entwicklung geeigneter Heuristiken für Energie-Management von Datenzentren herausfordernd (vergleiche hierzu [14, 28]). Jüngere Arbeiten beschäftigen sich beispielsweise auch mit der Entwicklung geeigneter Metriken zur Beurteilung der Nachhaltigkeit eines Datenzentrums oder aber auch mit der Entwicklung von energieeffizienten Algorithmen für Platzierung von VMs [58, 121]. Gerade für Cloud-Anbieter die auf Nachhaltigkeit setzen kann es zusätzlich interessant sein, den Kunden Überprüfungsmöglichkeit für das Nachhaltigkeitsversprechen mit an die Hand zu geben. So ist beispielsweise denkbar, dass Kunden sich mit geeignet Monitoring-Mechanismen eigenständig einen Eindruck von der Nachhaltigkeit einer Cloud-Umgebung verschaffen.

2.2 Transportsystem und Kommunikationsdienste

Ein klassisches Referenzmodell im Kontext von Rechnernetzen ist das *OSI Modell* [101, 199]. Dieses unterscheidet in sieben Abstraktionsschichten: Anwendungs-, Darstellungs-, Sitzungs-, Transport-, Vermittlungs-, Sicherungs- und Physikalische Schicht. Eng damit verwandt sind die vier Abstraktionsschichten des *Internet Modells* [203]:

Anwendungsschicht: Oberste Schicht des Referenzmodells in der Protokolle für Anwendungen realisiert sind (Nutzer-zu-Nutzer-Kommunikation). Zwischen zwei Protokollinstanzen werden dazu *Nachrichten* ausgetauscht. Typische Protokolle dieser Schicht sind *HTTP* [233], *IMAP* [213] und *BGP* [219]. Die Anwendungsschicht des *Internet Modells* umfasst die Anwendungs-, Darstellungs-, Sitzungsschicht des *OSI Modells*.

Transportschicht: Bietet Ende-zu-Ende-Kommunikationsdienste für Anwendungen an. Zwischen zwei Protokollinstanzen werden dazu *Segmente* ausgetauscht. Typische Protokolle dieser Schicht sind *TCP* [240] und *UDP* [237]. Die Transportschicht des *Internet Modells* umfasst die Transportschicht des *OSI Modells*.

Internetschicht: Verbindungsloser Dienst zum Austausch zwischen nicht direkt benachbarter Systeme. Zwischen zwei Protokollinstanzen werden dazu *Datagramme* ausgetauscht. Ein typisches Protokoll dieser Schicht ist *IP* [239, 208]. Die Internetschicht des *Internet Modells* umfasst die Vermittlungsschicht des *OSI Modells*.

Netzzugangsschicht: Dient zur Kommunikation mit dem direkt angeschlossenen Netz. Zwischen zwei Protokollinstanzen werden dazu *Rahmen* ausgetauscht. Ein typisches Protokoll dieser Schicht ist *Ethernet* [135] oder auch *ARP* [243]. Die Netzzugangsschicht des *Internet Modells* umfasst die Sicherungsschicht und Physikalische Schicht des *OSI Modells*.

In dieser Dissertation wird für den weiteren Verlauf das *Internet Modell* als Referenz verwendet.

In Abbildung 2.2 werden die beiden Referenzmodelle noch einmal nebeneinander dargestellt. Zusätzlich wird abgebildet, welche Schichten dabei typischerweise als sogenanntes *Transportsystem* aufgefasst werden. Im *OSI Modell* sind das die transportorientierten Schichten Physikalische Schicht, Sicherungsschicht, Vermittlungsschicht und Transportschicht. Für das *Internet Modell* sind dies die transportorientierten Schichten Netzzugangsschicht, Internetschicht und Transportschicht.

Neben verschiedenen Abstraktionsschichten wird zusätzlich zwischen vertikaler und horizontaler Kommunikation unterschieden. Bei vertikaler Kommunikation kooperieren

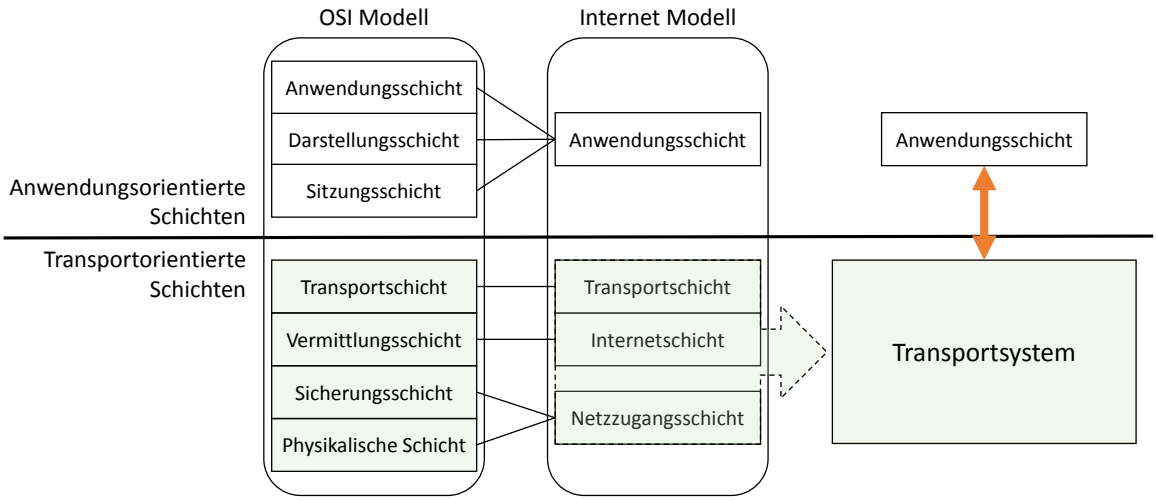

Abbildung 2.2: *Zusammenhang zwischen dem typischen Transportsystem und den Schichten des Internet beziehungsweise OSI Modells.*

Protokollinstanzen benachbarter Schichten miteinander. Horizontale Kommunikation hingegen umfasst den Austausch von Daten zwischen Protokollinstanz einer Schicht.

Des Weiteren zeichnen sich die von einer Abstraktionsschicht angebotenen Dienste durch Dienstdefinitionen, Protokollspezifikationen und Dienstprimitive aus.

2.3 Gruppenkommunikation und Multicast

Bei Kommunikation in Netzen wird typischerweise zwischen *Sender* und *Empfänger* unterschieden. In Abbildung 2.3 ist weiterhin eine Übersicht über gängige Kommunikationsmuster dargestellt.

Bei *Unicast* handelt es sich dabei um Punkt-zu-Punkt-Kommunikation zwischen einem Sender und einem Empfänger. Bei *Multicast* hingegen handelt es sich um Punkt-zu-Mehrpunkt-Kommunikation (oder Gruppenkommunikation) zwischen einem Sender und mehreren Empfängern. Dabei werden die Empfänger häufig mit Hilfe einer Gruppenadresse adressiert. Bei der Zusammensetzung einer solchen Gruppen wird unterschieden, ob der Sender oder die Empfänger die Mitgliedschaft in der Gruppe verwalten. Bei *Broadcast* kommuniziert ein Sender mit allen am Netz angeschlossenen Systemen. Hierzu müssen nicht zuerst Gruppen gebildet werden. Als *Multipeer*-Kommunikation wird eine Mehrpunkt-Kommunikation bezeichnet, bei der alle teilnehmenden Systeme sowohl die Rolle von Sender als auch Empfänger einnehmen können. Bei *Anycast* hingegen wird vom Sender eine Gruppe von Empfängern adressiert, wobei eine Nachricht allerdings

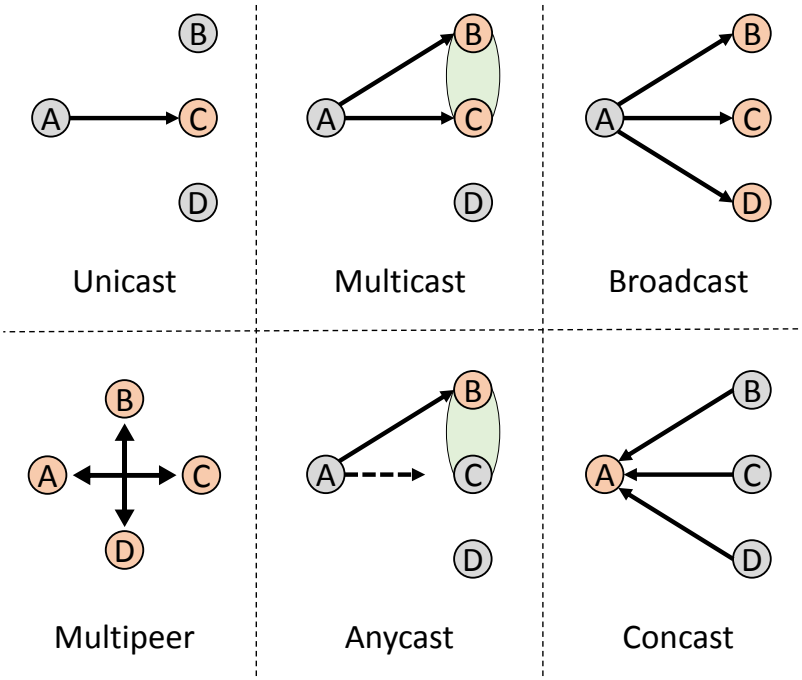

Abbildung 2.3: *Darstellung verschiedener klassischer Kommunikationsmuster. In Orange dargestellt: Empfänger. In Grün dargestellt: Gruppe zur Adressierung von Empfängern.*

nur an ein einzelnes Gruppenmitglied zugestellt werden muss. Werden von vielen Gruppenmitgliedern gleichzeitig Nachrichten an einen Empfänger geschickt, so wird dies als *Concast* bezeichnet.

Für den weiteren Verlauf dieser Dissertation sind vor allem die Kommunikationsmuster *Multicast* und *Unicast* für den Versand von Request-Nachrichten relevant. Werden anschließen viele Response-Nachrichten von einer Gruppe zurückgesandt, so handelt es sich dabei um *Concast*.

Für *Multicast* wird weiter zwischen *Routing* und *Transport* unterschieden. Während sich *Routing* mit der Berechnung optimaler Pfade vom Sender zu allen Gruppenmitgliedern beschäftigt, wird für den *Transport* möglichst späte Replikation von Nachrichten angestrebt (nah am Empfänger). Außerdem lassen sich für den Transport typischerweise verschiedene Zuverlässigkeitgrade unterscheiden. Weiterhin lässt sich beim Transport von Multicast-Nachrichten die Reihenfolge aufrechterhalten oder der Dienst atomar auslegen. Das im Rahmen dieser Dissertation benötigte und entwickelte Adaptive Transportsystem für Request-Response in Gruppen benötigt dabei keine Reihenfolgetreue oder Atomarität. Die für diese Dissertation relevante Details werden jeweils an der entsprechenden Stelle in Kapitel 4 besprochen.

Das *Multicast*-Kommunikationsmuster kann dabei auf verschiedenen Abstraktionsschichten des Internet Modells umgesetzt werden. Nachstehend werden diese Möglichkeiten kurz vorgestellt.

Application-Layer-Multicast: Hierbei wird die benötigte Funktionalität innerhalb der Anwendungsschicht umgesetzt. Dies hat den Vorteil, dass Endsysteme volle Kontrolle über die Multicast-Funktionalität haben. Das hat allerdings den Nachteil, dass das Netz nicht ohne weiteres Unterstützung für Multicast anbieten kann – in der Regel werden Protokolle der Anwendungsschicht nicht von Zwischensystemen ausgewertet.

IP-Multicast: Bei *IP-Multicast* hingegen kann das Netz Unterstützung für *Multicast* innerhalb der Internetschicht des *Internet Modells* anbieten. Beispielsweise durch den Einsatz von effizienten Routing-Algorithmen zur Weiterleitung der Nachrichten vom Sender zu den Empfängern. Weiterhin werden zur Adressierung einer Gruppe spezielle IP-Adressen verwendet und es existieren zusätzliche Protokolle zur Gruppenverwaltung.

Ethernet-Multicast: Wird Multicast innerhalb der Netzzugangsschicht des *Internet Modells* umgesetzt, so kann das Netz dafür sorgen, dass Nachrichten wirklich nur an den benötigten Stellen dupliziert werden.

Für den weiteren Verlauf dieser Dissertation wird vor allem *IP-Multicast* und *Ethernet-Multicast* benötigt. Weiterhin liegt der Fokus dieser Arbeit auf dem *Transport* von Multicast-Nachrichten. Aus diesem Grund werden Aspekte wie Routing und der dadurch entstehende Overhead zunächst nicht mit betrachtet.

2.4 Request-Response-Dienste

Bei *Request-Response* [86, 282, 314] handelt es sich typischerweise um ein verbindungsloses 1:1-Kommunikationsmuster (Punkt-zu-Punkt), bei dem nicht nur Daten von einem Sender zu einem einzigen Empfänger (unidirektional) übertragen werden, sondern bei dem Daten zwischen beiden Teilnehmern ausgetauscht werden (bidirektional). Zum Einsatz kommt dieses Kommunikationsmuster beispielsweise bei Remote Procedure Call (RPC) [232, 226, 21, 173, 82, 162] oder bei dem *Request/Response Exchange* von *HTTP/2* [235]. Der Initiator der Kommunikation wird als Client-Instanz bezeichnet und die Gegenpartei als Server-Instanz. Zusätzlich wird zwischen synchronem und asynchronem Datenaustausch unterschieden. Für Monitoring ist vor allem der synchrone Datenaustausch zwischen Client- und Server-Instanz interessant. Hierbei sendet eine Client-Instanz eine Request-Nachricht an eine Server-Instanz. Die Server-Instanz wertet dann den in der Request-Nachricht enthaltenen Monitoringauftrag aus und sendet

ihrerseits Monitoringdaten an die anfragende Client-Instanz per Response-Nachricht zurück.

Durch die Bidirektionalität der Kommunikation von *Request-Response* werden auf diesem Kommunikationsmuster aufbauende Dienste in der Regel oberhalb der Transportschicht auf Ebene der Anwendungsschicht realisiert. Die klassischen Dienstdefinitionen der Transportschicht lassen sich dabei nicht einfach übertragen. Beispielsweise definiert sich ein zuverlässiger Dienst der Transportschicht als ein Dienst, der Daten vollständig und korrekt, in der richtigen Reihenfolge und ohne Duplikate oder Phantom-Daten vom Sender zum Empfänger überträgt (vergleiche beispielsweise [116], Abschnitt 3.4). Allerdings muss für die Dienstdefinition von *Request-Response* auch die Übertragung in Gegenrichtung mit berücksichtigt werden. Außerdem müssen bei *Request-Response* auch Fehler auf Anwendungsebene der Server-Instanz bei Abarbeitung von Request-Nachrichten abgedeckt werden [21]. Beispielsweise können auf Anwendungsebene der Server-Instanz kurzfristige Lastspitzen nach dem Empfang einer Request-Nachricht zum Ausbleiben der Response-Nachricht führen, ohne dass die Server-Instanz insgesamt nicht mehr betriebsbereit wäre [153]. Im Rahmen dieser Dissertation wird ein Dienst, der das Kommunikationsmuster *Request-Response* zur Verfügung stellt, als *Request-Response-Dienst* bezeichnet und wie in Definition 2.1 definiert.

Definition 2.1 *[Request-Response-Dienst] Ein Dienst, der Daten vollständig und korrekt, in der richtigen Reihenfolge und ohne Duplikate oder Phantom-Daten von einer Server-Instanz abruft. Die Fehlerbehandlung des Dienstes deckt dabei sowohl den Verlust von Nachrichten in beiden Richtungen zwischen Client- und Server-Instanz ab (bidirektional), als auch den Verlust von Nachrichten in der Anwendungsebene der Server-Instanz.*

An der Definition von Request-Response-Diensten ist besonders, dass die Dienstdefinition sich auf den bidirektionalen Datenaustausch zwischen Client- und Server-Instanz erstreckt. Ein *Request-Response-Dienst* ist also insbesondere erst dann erfüllt, wenn die per Request-Nachricht angeforderte Response-Nachricht von der Client-Instanz empfangen wurde. Da Nachrichtenverluste in Server-Instanzen nicht ausgeschlossen werden können, ist bei Request-Response-Diensten die Client-Instanz allein für die Behandlung von Nachrichtenverlust verantwortlich – beispielsweise für den Verlust von Request-Nachrichten zwischen Client- und Server-Instanz, für Response-Nachrichten zwischen Server- und Client-Instanz oder Nachrichtenverlust in der Server-Instanz. Hierzu überwacht die Client-Instanz auf Basis der sogenannten *Synchronous Communication Assumption* [105, 20, 153] (das unterliegende Netz stellt Nachrichten innerhalb einer begrenzten Zeit zu) die angeforderten Response-Nachrichten mit einem Zeitgeber und führt gegebenenfalls Sendewiederholungen von Request-Nachrichten mittels eines Automatic Repeat Request (ARQ)-Verfahrens durch. Für das ARQ-Verfahren dienen dabei Response-Nachrichten

als implizite Bestätigungen für die erfolgreiche Übertragung von Request-Nachrichten zu Server-Instanzen. Der restliche Teil der Fehlerkontrolle (Erkennung von Duplikaten, Erkennung von Phantom-Daten, Korrektheit und Reihenfolgetreue) wird wie bei anderen Diensten auch sowohl durch die Client- als auch durch die Server-Instanz umgesetzt.

Ein Spezialfall eines Request-Response-Dienstes stellt der Datenaustausch zwischen Client- und Server-Instanz dar, bei dem die Kommunikation nach genau einer Nachricht je Kommunikationsrichtung abgeschlossen ist [304, 86]. Beispielsweise wenn ein Client zu Monitoringzwecken (*probing* oder *Messaging Query* [86]) eine Anfrage als Request-Nachricht an einen Server sendet und das Ergebnis der Anfrage als einzelne Antwort per Response-Nachricht vom Server zurückbekommt. Für das bessere Verständnis wird im Folgenden für diesen Spezialfall eines Request-Response-Dienstes, der nach genau einer Nachricht je Kommunikationsrichtung abgeschlossen ist, die Bezeichnung *Singulärer Request-Response-Dienst (1RRS)* verwendet. Für 1RRS spielt Reihenfolgetreue keine Rolle, da bei nur einer Nachricht je Kommunikationsrichtung keine Reihenfolge definiert werden kann. Davon unberührt bleibt jedoch die Forderung nach korrekter und vollständiger bidirektionaler Datenübertragung ohne Duplikate oder Phantom-Daten zwischen Client- und Server-Instanz.

Das als Teil der Fehlerkontrolle bei 1RRS zum Einsatz kommende ARQ-Verfahren ist mit *Stop-and-Wait* [169, 172, 122] vergleichbar. Nach dem Senden einer Request-Nachricht wartet die Client-Instanz mittels eines Zeitgebers auf das Eintreffen der Response-Nachricht des Server-Instanz. Dabei dient die Response-Nachricht als implizite Bestätigung für den Empfang (und Verarbeitung) der Request-Nachricht durch die Server-Instanz. Trifft die Response-Nachricht nicht vor Ablauf des Zeitgebers ein, so wird die Request-Nachricht von der Client-Instanz wiederholt. Nach Übertragung der Request-Nachricht vom Client-Instanz zu Server-Instanz und der korrekt empfangenen Response-Nachricht von Server-Instanz zu Client-Instanz ist der Dienst erfüllt und die Kommunikation abgeschlossen.

Der Entwurf von Verfahren zur Fluss- und Staukontrolle bei 1RRS ist nach der Empfehlung von [241] nicht relevant, da solche Verfahren bei lediglich einer Nachricht je Kommunikationsrichtung nicht ihre volle Wirkung entfalten können. Hierzu sind längerfristige Übertragungen notwendig. Nach [241] (Abschnitt *3.1.3 Low Data-Volume Applications*) wird lediglich empfohlen, auf Seiten des Senders die Senderate so zu limitieren, dass im Durchschnitt nicht mehr als eine Nachricht pro ermittelter Umlaufzeit in jede Richtung gesendet wird. Dies soll einer Überlast des Netzes vorbeugen.

Im weiteren Verlauf dieser Dissertation wird der 1RRS auf Gruppen von Server-Instanzen ausgedehnt, so dass ein gruppenbasierter Request-Response-Dienst entsteht, der für Monitoringzwecke geeignet ist (siehe Kapitel 4).

2.5 Transparenz und Vertrauen

Typischerweise wird Transparenz im Sinne von mehr Offenheit bezüglich interner Vorgänge als geeignetes Mittel zur Steigerung von Vertrauen angesehen. Durch Transparenz sollen Handlungen nachvollziehbar und zu Grunde liegende Verantwortlichkeiten aufgedeckbar gemacht werden. Dies wird insbesondere für Regierungshandlungen gefordert (vergleiche beispielsweise [41, 73]). Aber auch bei Unternehmen oder für ganze Industriezweige soll Transparenz für mehr Vertrauen sorgen (vergleiche beispielsweise [85]).

In der Literatur werden allerdings auch potentiell negative Aspekte von Transparenz diskutiert (vergleiche beispielsweise [63]). So führt die Bereitstellung von vielen internen Informationen nicht automatisch zu einem besseren Verständnis, sondern im Gegenteil, das Verständnis kann durch die Informationsflut sogar erschwert werden. Außerdem führt Cucciniello et al. [41] an, dass noch genauer untersucht werden muss, unter welchen Voraussetzungen Transparenz bei Regierungen für mehr Vertrauen sorgt.

Im Bereich von Cloud-Computing erlangt die Forderung nach mehr Transparenz gerade im Zusammenhang mit Zertifizierung immer mehr an Bedeutung (vergleiche unter anderem [124, 123, 113]). Aber auch die Haftung des Cloud-Anbieters für die Einhaltung der zuvor mit dem Kunden getroffenen vertraglichen Vereinbarungen ist hier Thema (vergleiche beispielsweise [77, 114, 88]). Übergeordnet ist *Zurechenbarkeit*, im Englischen als *Accountability* bezeichnet, ein wichtiges Schutzziel im Bereich der Informationssicherheit (vergleiche beispielsweise Abschnitt 2.1 von Cherdantseva [35]). Eng verwandt ist hiermit auch das Schutzziel der *Nichtabstreitbarkeit*.

Diese Dissertation greift dabei die genannten Forderung auf und möchte durch die Entwicklung geeigneter Monitoring-Bausteine Transparenz aus Kundensicht ermöglichen.

Kapitel 3

Rahmenwerk für unabhängiges Cloud-Monitoring

In diesem Kapitel wird das dieser Dissertation zu Grunde liegende Rahmenwerk für CMS-unabhängiges Monitoring vorgestellt. Außerdem wird ein Überblick zum aktuellen Stand der Forschung im Bereich von Cloud-Monitoring präsentiert. Darüber hinaus werden die in dieser Dissertation besprochenen Monitoring-Bausteine im dargestellten Rahmenwerk verortet und es wird aufgezeigt, welche Herausforderungen hier primär behandelt werden und welcher Beitrag zur Wissenschaft jeweils geliefert wird.

Ergebnisse dieses Kapitels basieren unter anderem auf der folgenden Publikation:

- M. Flittner, S. Balaban und R. Bless. „CloudInspector: A Transparency-as-a-Service Solution for Legal Issues in Cloud Computing“. In: *IEEE International Conference on Cloud Engineering Workshop* (Apr. 2016) [TT-5]

3.1 Überblick zum Stand der Forschung

Eine umfassende Bestandsaufnahme zu Cloud-Monitoring findet sich unter anderem in [171, 44, 55, 186, 3]. Neben der Kategorisierung aktueller Arbeiten sind vor allem die herausgearbeiteten Unterscheidungsmerkmale interessant. In Tabelle 3.1 wird analysiert, welche Aspekte die oben genannten Publikation jeweils als *Monitoringzweck*, *Anforderungen* und *Herausforderungen* identifiziert haben. In der jeweiligen Kategorie sind die Zeilen mit großer Übereinstimmung grau hinterlegt. Wird der jeweilige Aspekt in der Publikation berücksichtigt, so wird das Symbol • verwendet. Kommt der Aspekt

	Aspekt	In Veröffentlichung					∑
		Aceto+ [3]	Fatema+ [55]	Ward+ [186]	Rodrig.+ [44]	Syed+ [171]	
Metainformationen	Anzahl Autoren	4	5	2	8	5	–
	Anzahl Seiten	23	16	30	6	16	–
	Erscheinungsjahr	2013	2014	2014	2016	2017	–
	Herausgeber	Elsevier	Elsevier	Springer	ACM	Elsevier	–
	Monitoringsysteme	28	30	30	14	25	–
	Veröffentlichungsort	*ComNet*	*JPDC*	*JoCCASA*	*SAC*	*JNCA*	–
	Verwendete Referenzen	150	96	98	41	73	–
	Zitierungen (Stand 11/18)	484	119	54	22	6	–
Monitoringzweck	Abrechnung	•	•	–	–	•	3
	Fehler-Ursachen-Analysen	•	•	•	–	–	3
	Lastverteilung	–	–	•	–	–	1
	Leistungsüberwachung	•	–	•	(•)	•	**4**
	Lokalisierung	–	–	•	–	–	1
	Management	•	•	–	(•)	•	**4**
	Netzüberwachung	•	–	–	–	–	1
	Planung	•	•	–	–	•	3
	Service-Level-Agreements	•	•	•	–	–	3
	Sicherheit	•	•	–	(•)	–	3

Übersicht zum derzeitigen Stand der Forschung im Bereich von Cloud-Monitoring (Fortsetzung nächste Seite).

	Aspekt	In Veröffentlichung					∑
		Aceto+ [3]	Fatema+ [55]	Ward+ [186]	Rodrig.+ [44]	Syed+ [171]	
Anforderungen	Anpassungsfähig	•	(•)	•	(•)	–	**4**
	Akkurat	•	–	–	•	–	2
	Aktuell	•	–	•	–	–	2
	Anpassbar	–	•	–	–	–	1
	Archivierend	–	•	–	–	–	1
	Autonom	•	•	•	•	–	3
	Effizient	–	•	–	–	–	1
	Elastisch	•	(•)	•	•	–	**4**
	Erschwinglich	–	•	–	–	–	1
	Erweiterbar	•	•	–	–	–	2
	Interoperabel / Portabel	–	•	–	–	–	1
	Mehrmandantenfähig	(•)	•	–	–	–	2
	Multi-granular	–	–	•	–	–	1
	Nicht-invasiv	(•)	•	–	–	–	2
	Nutzbar	–	•	–	–	–	1
	Ressourcenschonend	–	(•)	–	–	•	2
	Skalierbar	•	•	•	•	•	**5**
	Umfassend	•	–	•	•	–	3
	Zuverlässig	•	–	(•)	–	–	2

Übersicht zum derzeitigen Stand der Forschung im Bereich von Cloud-Monitoring (Fortsetzung nächste Seite).

	Aspekt	In Veröffentlichung					∑
		Aceto+ [3]	Fatema+ [55]	Ward+ [186]	Rodrig.+ [44]	Syed+ [171]	
Herausforderungen	Abrechnung	–	–	–	–	•	1
	Anpassungsfähigkeit	–	•	–	–	•	2
	Cloud-Netz	•	–	–	–	–	1
	Datenspeicherung	–	–	•	–	•	2
	Domainübergreifend	•	–	–	–	–	1
	Effektivität	•	(•)	–	(•)	–	3
	Effizienz	•	•	–	–	–	2
	Einsicht in VMs	–	–	–	–	•	1
	Interoperabilität	–	•	–	–	–	1
	Konfiguration	–	–	•	(•)	–	2
	Kundenaktivität	–	•	–	–	–	1
	Lastgeneratoren	•	–	–	–	–	1
	Mehrmandantenfähig	–	–	–	•	–	1
	Monitoringstrategien	–	–	•	–	–	1
	Ohne Agenten	–	–	–	–	•	1
	Portierbarkeit	–	•	–	–	–	1
	Schichtenübergreifend	•	–	–	•	–	2

Übersicht zum derzeitigen Stand der Forschung im Bereich von Cloud-Monitoring (Fortsetzung nächste Seite).

	Aspekt	In Veröffentlichung					∑
		Aceto+ [3]	Fatema+ [55]	Ward+ [186]	Rodrig.+ [44]	Syed+ [171]	
Herausforderungen	Service-Level-Agreements	–	–	–	•	•	2
	Sicherheit	–	•	–	–	–	1
	Skalierbarkeit	–	•	–	–	•	1
	Smart Grid	–	–	–	–	•	1
	Standards	•	–	–	–	–	1
	Wissenstransfer	–	–	•	–	–	1

Tabelle 3.1: *Übersicht zum derzeitigen Stand der Forschung im Bereich von Cloud-Monitoring (Fortsetzung vorherige Seite).*

nur Teilweise zur Sprache oder in einem leicht abgewandelten Kontext das Symbol (•). Wird der jeweilige Aspekt nicht berücksichtigt, das Symbol –.

3.1.1 Monitoringzweck

Als Haupteinsatzzweck für Monitoring von Cloud-Umgebungen wird derzeit *Leistungsüberwachung* und *Management* angesehen. Allerdings spielen auch Aspekte wie *Abrechnung*, *Fehler-Ursachen-Analysen*, *Planung*, *Service-Level-Agreements* und *Sicherheit* eine Rolle. Neben dem Einsatzzweck für Monitoring wird außerdem unterschieden, ob es sich um *aktives* oder *passives* Monitoring handelt und ob die Monitoringinformationen für den Kunden oder den Cloud-Anbieter bereitgestellt werden. Weiter wird unterschieden, ob das Monitoring innerhalb oder außerhalb der VMs von Kunden stattfindet und wie die Kommunikation des Monitoringsystems aufgebaut ist (zum Beispiel *Pull* oder *Push*). Eine detaillierte Betrachtung möglicher Architekturen für Monitoringsysteme findet sich in [29]. Der in dieser Dissertation verfolgte Ansatz wird in Abschnitt 3.3 näher ausgeführt.

3.1.2 Anforderungen

In Tabelle 3.1 werden insgesamt *19* verschiedene Anforderungen an Monitoringsysteme untersucht. Im Ergebnis zeigt sich, dass ein Monitoringsystem für Cloud-Umgebungen im wesentlichen *anpassungsfähig*, *elastisch* und *skalierbar* sein sollte. Unter *anpassungsfähig* wird hier die Fähigkeit zur selbstständigen Anpassung des Monitorsystems an die aktuelle Lastsituation innerhalb der Cloud-Umgebung verstanden. Beispielsweise das automatisierte Verringern von Abfrageintervallen bei hoher Last. Ein Monitoringsystem ist dann *elastisch*, wenn es das Cloud-typische Verhalten wie Migration von VMs oder Hinzufügen von Kunden mit berücksichtigt. In der Literatur auch häufig als *cloud-aware* bezeichnet (siehe zum Beispiel [186]). Die *Skalierbarkeit* eines Monitoringsystems ist nach dem derzeitigen Stand der Forschung von besonderem Interesse – als einziges haben alle in der Analyse enthaltenen Veröffentlichungen diesen Aspekt als eine wesentliche Anforderung identifiziert. Neben den drei genannten Kernanforderungen spielen auch Eigenschaften wie *akkurat*, *aktuell*, *autonom*, *erweiterbar*, *mehrmandantenfähig*, *nicht-invasiv*, *ressourcenschonend*, *umfassend* und *zuverlässig* eine wichtige Rolle für Monitoringsysteme. Ein Monitoringsystem wird dabei als *umfassend* bezeichnet, sofern es die Überwachung verschiedenster Ressourcen innerhalb einer Cloud-Umgebung unterstützt (siehe zum Beispiel [44]). Zwischen den Anforderungen *Anpassbar* und *Anpassungsfähigkeit* wird wie folgt unterschieden: Die Anforderung *Anpassungsfähigkeit* ist für Monitoringsysteme dann erfüllt, wenn sich diese adaptiv an die jeweilige Lastsituation und Monitoringaufgabe anpassen kann. Es ist *Anpassbar*, sofern der Cloud-Anbieter das Monitoringsystem

für seine Zwecke konfigurieren kann. Die weiteren aufgeführten Anforderungen sind selbsterklärend.

3.1.3 Herausforderungen

Während sich in den betrachteten Publikationen viele Gemeinsamkeiten im Bezug auf den Monitoringzweck und Anforderungen finden lassen, sind die angeführten Herausforderungen breiter gefächert. Lediglich die *Effektivität* eines Monitoringsystems scheint unter den *17* analysierten Aspekten bei immerhin drei Übereinstimmungen eine besondere Rolle zu spielen. Unter *Effektivität* eines Monitoringsystems wird dabei gemeinhin die Fähigkeit zur Filterung der Monitoringinformationen verstanden, so dass nur die wirklich relevanten Informationen angezeigt werden. Nach Aceto et al. [3] sind hierbei die wesentlichen Herausforderungen, die Entwicklung von *cleveren* Algorithmen zur Korrelation von Monitoringdaten, von Techniken zur Fehler-Ursachen-Analyse und von Messmethoden für virtuelle Umgebungen. Weitere Herausforderungen für Monitoringsysteme sind *Anpassungsfähigkeit*, *Datenspeicherung*, *Effizienz*, *Konfiguration*, *Schichtenübergreifend* und *Service-Level-Agreements (SLAs)*.

3.1.4 Weiteres Forschungsumfeld

In den folgenden Abschnitten werden kurz ausgewählte Themen rund um Cloud-Monitoring beleuchtet, die in Tabelle 3.1 nicht explizit erfasst werden, die aber für diese Dissertation von Bedeutung sind.

3.1.4.1 Dienstleistungsverwaltung

Im übergeordneten Kontext soll Cloud-Monitoring auch zur Dienstleistungsverwaltung, im Englischen *Service Level Management*, beitragen (siehe unter anderem [88, 54]). Die Einhaltung der in der Regel zwischen Kunde und Cloud-Anbieter vereinbarten SLAs (vergleiche [146]) sollen dabei durch Monitoringsysteme überwacht werden. Herausforderungen in diesem Kontext sind unter anderem die Platzierung von Monitoringeinheiten [164, 165], Überprüfung der Verfügbarkeit einer Cloud-Umgebung [166], oder die Auswahl von Cloud-Anbietern nach Quality-of-Service-Metriken [79]. Für letzteres schaffen insbesondere die Arbeiten von Knode und Egan [112] und Ardagna et al. [7] neue Schnittstellen für Kunden zur Überprüfung der von Cloud-Anbietern bereitgestellten Dienstleistung. Dies soll die Transparenz der verschiedenen Cloud-Angebote verbessern und das Vertrauen in einzelne Cloud-Anbieter steigern. In der Literatur werden solche

Ansätze auch im Kontext der Beurteilung der Sicherheit einer Cloud-Umgebung durch Kunden besprochen [71].

3.1.4.2 Vertrauenswürdige Laufzeitumgebungen

Wie Ko et al. [113] und Cheng et al. [34] zeigen, können Cloud-Umgebungen so entwickelt werden, dass Zuverlässigkeit und Sicherheit integraler Bestandteil des Entwurfs sind. Hierbei kommen beispielsweise Ansätzen aus dem Bereich *Trusted Computing* zum Einsatz. Durch Verlagerung von kritischer Funktionalität auf physische Faktoren (wie *Trusted Platform Module* [174, 95] und *Software Guard Extensions* [272]) sollen Auswirkungen von Angriffen minimiert werden. Zusätzlich sollen Kunden dann die Cloud-Konfiguration der physischen Cloud-Knoten überprüfen und sicherstellen können, dass die physische Hardware den vertraglichen Vereinbarungen entspricht. Allerdings bringen solche Mechanismen enorme Komplexität und Inflexibilität in eine bestehende Cloud-Infrastruktur. Außerdem können Fehlkonfigurationen zur Laufzeit nicht sicher erkannt werden, da sich die Überprüfung nur auf die Kompatibilität der physischen Cloud-Knoten an sich bezieht. Wenn zum Beispiel durch die Überprüfung sichergestellt wird, dass die physischen Cloud-Knoten kompatibel sind, so können zur Laufzeit trotzdem Fehlkonfigurationen des CMS trotz Kompatibilität vorliegen.

3.1.4.3 Richtliniendurchsetzung

Vertragliche Vereinbarungen zwischen Kunde und Cloud-Anbieter lassen sich auch automatisiert überprüfen und durch zusätzliche Mechanismen, so sogenannte Richtliniendurchsetzung (im Englischen *Policy Enforcement*), in der Cloud-Infrastruktur erzwingen [176, 48, 106]. Hierzu übermitteln Kunden dem Cloud-Anbieter vorab ein Menge an Regeln, deren Umsetzung dann durch zusätzliche Mechanismen in der Cloud-Infrastruktur erzwungen wird. Auf diese Art und Weise soll das Vertrauen in Cloud-Umgebungen erhöht werden. Allerdings bedeutet dies einen erheblichen Einschnitt in die Cloud-Infrastruktur. Beispielsweise kann dann der Cloud-Anbieter nicht mehr abwägen, vertragliche Vereinbarungen von Kunden kurzfristig zu verletzen, um ein höheres Gut zu schützen – zum Beispiel um bleibenden Schaden in der Cloud-Infrastruktur oder einen Komplettausfall des Dienstes zu verhindern. Auch kann es bei der Durchsetzung von Richtlinien schnell zu Konflikten kommen und es müssen geeignet Maßnahmen zur Priorisierung geschaffen werden.

3.1.4.4 Zertifizierungen

Neben den verschiedenen Monitoringlösungen ist die Forderung nach Zertifizierung von Cloud-Umgebungen und Cloud-Anbietern etabliert (siehe unter anderem [170]). Dabei sollen sowohl das vom Cloud-Anbieter aufgebaute Datenzentrum, die internen Betriebsabläufe, als auch die vertraglichen Regelungen zwischen Cloud-Anbieter und Kunde nach einem einheitlichen Maßstab evaluiert und zertifiziert werden. Die von einer vertrauenswürdigen dritten Partei ausgestellten Zertifikate sollen dann Kunden helfen, zu entscheiden welcher Cloud-Anbieter am vertrauenswürdigsten ist. Mit Hilfe der Zertifikate soll transparent werden, welcher Cloud-Anbieter in der Lage ist, welchen Standard einzuhalten. So bringen Zertifikate gleich zwei Vorteile mit sich: zum einen helfen sie Cloud-Anbietern sich auf dem Markt von anderen abzusetzen, zum anderen dienen sie Kunden als Orientierung bei der Auswahl eines Cloud-Anbieters. In der Praxis haben sich hierfür zahlreiche Ansätze etabliert [257, 303, 98, 99, 100, 148, 249, 256]. Auch ist Zertifizierung von Cloud-Umgebungen Gegenstand mehrerer laufender Projekte (siehe unter anderem [248]). Für zukünftige Systeme lässt sich außerdem ein Trend in Richtung automatischer und kontinuierlicher Überprüfung der zertifizierten Eigenschaften ausmachen [124]. Allerdings bringen Zertifikate eine gewisse Inflexibilität was den Überprüfungsgegenstand und den Überprüfungszeitpunkt anbelangt mit sich. So können Kunden derzeit nicht nach Belieben Rezertifizierunge nanstoßen, wenn die Funktionalität des CMS in Frage steht.

3.1.5 Zwischenfazit

Die Forderung nach Monitoring von Cloud-Umgebungen zur Erhöhung der Transparenz spiegelt sich in der gegenwärtigen Literatur wider. Eine besondere Triebfeder spielt dabei das Verhältnis zwischen Kunde und Cloud-Anbieter. Dieses wird häufig in Form von Verträgen oder SLAs festgehalten und ist dann Gegenstand von Cloud-Monitoring. Von besonderem Interesse ist dieses Thema auch aus rechtlicher Sicht, da neben Zertifizierungen, wenn möglich, zusätzliche Überprüfungsmöglichkeiten vorgehalten werden sollen. Aber nicht nur der Überprüfung zur Laufzeit wird in der Literatur Rechnung getragen, sondern auch der Vorhaltung von Monitoringdaten zur späteren Fehler-Ursachen-Analyse ist Thema.

Im Rahmen dieser Dissertation werden verschiedene Bausteine entwickelt, die sich zu einem Rahmenwerk für unabhängiges Cloud-Monitoring zusammensetzen lassen. Dabei werden von den oben identifizierten wichtigen Eigenschaften für Monitoringsysteme *Elastizität*, *Skalierbarkeit* und *Effizienz* mit berücksichtigt. Weniger zentrale Anforderungen wie *Anpassungsfähigkeit* spielen im Kontext dieser Arbeit hingegen nur eine

untergeordnete Rolle. Allerdings wird in dieser Dissertation eine ganz neue Anforderung an Monitoringsysteme untersucht: *Unabhängigkeit vom Cloud-Management-System*. Das diese Anforderung von Relevanz ist, zeigt auch die im Zusammenhang mit dieser Dissertation von Rieth [TT-7] durchgeführte Umfrage unter Cloud-Anbietern (für mehr Details siehe Anhang B).

Nachstehend wird zunächst kurz auf den im Rahmen dieser Dissertation anvisierten Zweck für Monitoring eingegangen. Anschließend wird der Systementwurf des zu Grunde liegenden Rahmenwerks präsentiert und ausgeführt, welche Verbesserungen die im Rahmen dieser Dissertation erarbeiteten Monitoring-Bausteine im Vergleich zu verwandten Arbeiten jeweils bieten sollen.

3.2 Betrachteter Monitoringzweck

Monitoring der Funktionalität eines CMS durch Kunden birgt zwei verschiedene Gesichtspunkte in sich. Zum einen die Überprüfung von vertraglichen Vereinbarungen zwischen Kunde und Cloud-Anbieter zur Laufzeit. Hierbei ist wichtig, dass Kunden im Gegensatz zu Zertifizierungen selbst aktiv steuern können, wann und wie oft ein Soll-Ist-Vergleich zwischen vertraglicher Vereinbarung und tatsächlicher Realisierung stattfinden soll. Zum anderen die Aufzeichnung von Monitoringdaten zur späteren Fehler-Ursachen-Analyse. Die so aufgezeichneten Monitoringdaten sollen dann später beispielsweise vor Gericht heranziehbar sein.

In den nachstehenden Abschnitten werden beide Zwecke noch etwas genauer vorgestellt und eine Abgrenzung zu in dieser Dissertation nicht behandelten Aspekten vorgenommen. In Abschnitt 3.4 wird dann näher ausgeführt, wo in diesem Umfeld die Beiträge dieser Dissertation liegen.

3.2.1 Vertragliche Vereinbarungen

Neben SLAs, die Metriken und Schranken für die Qualität der zu erbringenden Dienstleistung beschreiben [88, 189, 5, 146], werden zusätzliche vertragliche Vereinbarungen zwischen Kunde und Cloud-Anbieter festgehalten. Beispielsweise Regelungen zur Durchführung von regelmäßigen Zertifizierungen oder zur Platzierung von VMs auf physischen Cloud-Knoten in Deutschland. Für SLAs existieren dabei zahlreiche Ansätze, die die zu überprüfenden Anforderungen als maschinenlesbare darstellen [128, 111, 177]. Im Fokus dieser Dissertation steht jedoch nicht die Mächtigkeit und das Format einer Sprache zur Beschreibung von vertraglichen Vereinbarungen. Daher wird für den folgenden Verlauf

davon ausgegangen, dass sich Anforderungen des Kunden mit Hilfe einer geeigneten Sprache ausdrücken lassen. Auch ist die Entwicklung von geschickten Algorithmen zur Überprüfung konkreter vertraglicher Vereinbarungen nicht Gegenstand dieser Dissertation.

3.2.2 Aufzeichnung für Fehler-Ursachen-Analysen

Neben der Überprüfung von vertraglichen Vereinbarungen zur Laufzeit, spielt auch die Aufzeichnung von Monitoringdaten eine wesentliche Rolle bei der Überprüfung eines CMS. Während ersteres Informationen über den aktuellen Betrieb liefert, dient letzteres zum Aufbrechen der Beweisasymmetrie zwischen Kunde und Cloud-Anbieter. Typischerweise hat der Kunde keinen Zugriff auf historische Daten über das Cloud-Verhalten und der Cloud-Anbieter kein Interesse daran, diese Daten im Fall einer Pflichtverletzung an den Kunden auszuliefern. Daher ist es sinnvoll, aufgezeichnete Monitoringdaten bei einer vertrauenswürdigen dritten Partei zu hinterlegen. Dort sind die in den Daten enthaltenen Betriebsgeheimnisse gleichermaßen vor dem Zugriff des Kunden und Cloud-Anbieters geschützt. Im Fall von Rechtsstreitigkeiten können diese von der vertrauenswürdigen dritten Partei dann trotzdem einem Experten zur Fehler-Ursachen-Analyse zugänglich gemacht werden. Die komplette Aufzeichnung einer VM während der gesamten Lebensdauer kommt dabei aufgrund der hohen Datenmenge nicht in Frage (vergleiche auch [186, 171]). Von einem Monitoringsystem können daher höchstens Metadaten zum Lebenszyklus von VMs aufgezeichnet werden. Im Fokus dieser Dissertation steht allerdings nicht eine Analyse von relevanten Metadaten oder die Entwicklung geeigneter Verfahren zur Durchführung von Fehler-Ursachen-Analysen selbst. Stattdessen wird zunächst vorausgesetzt, dass die durch das Monitoringsystem erhobenen Monitoringdaten geeignet sind um Fehler-Ursachen-Analysen durchzuführen.

3.3 Grundlegender Systementwurf

Die im Rahmen dieser Dissertation entworfenen Monitoring-Bausteine bauen auf einem agentenbasierten Rahmenwerk zum Monitoring von Cloud-Umgebungen auf (vergleiche zum Beispiel [134]). Der Kategorisierung von Calero und Aguado [29] folgend, ist die Architektur des Rahmenwerks als sogenannte *Extended and Adaptive Internal Monitoring Architecture* einzuordnen: Mit Hilfe von Agenten wird auf den physischen Cloud-Knoten der Zustand der Knoten und der virtuellen Ressourcen überwacht, ohne dass dazu ein Eingriff in VMs notwendig wird. Das hier zur Grunde liegende Rahmenwerk ist also insbesondere nicht-invasiv, da weder aktive Tests durchgeführt noch Anpassungen

von VMs vorgenommen werden. Außerdem erhält das Rahmenwerk Zugriff auf den Kontrollkanal des CMS um bei Bedarf auf Aktionen des CMS reagieren zu können. Im Gegensatz zu beispielsweise Xu et al. [193] ist ein zentraler Controller zur Sammlung von Monitoringdaten und Durchführung von Überprüfungen vorgesehen. Sowohl Kunden als auch der Cloud-Anbieter sollen Zugriff auf das Rahemwerk erhalten.

Das Rahmenwerk wird im weiteren Verlauf als *CloudInspector* bezeichnet (siehe auch [TT-5]). Nachstehend werden zunächst die Komponenten von *CloudInspector* näher vorgestellt. Außerdem wird das interne Kommunikationsmuster zum Abruf von Monitoringdaten skizziert und der Ablauf eines Monitoringprozesses erläutert. Abschließend werden die in dieser Dissertation besprochenen Monitoring-Bausteine in das Gesamtkonzept eingeordnet.

3.3.1 Komponenten

Zur Realisierung des *CloudInspector*-Rahmenwerks werden die Komponenten Monitoring-Controller (Monitoring-Controller), Monitoring-Agent (Monitoring-Agent) und Log-Server (Log-Server) konzipiert. Anhand von Abbildung 3.1 werden diese Komponenten in das im Kapitel 2 vorgestellte architekturelle Rahmenwerk für Cloud-Computing eingeordnet.

Monitoring-Controller: Über den Monitoring-Controller können Kunden auf die Funktionen des *CloudInspectors* zugreifen. Dazu muss der Monitoring-Controller sowohl für die Kunden erreichbar sein, als auch Konnektivität zu den Monitoring-Agenten und dem Log-Server innerhalb der Cloud-Infrastruktur haben.

Monitoring-Agent: Zur lokalen Erfassung von Monitoringdaten wird auf jedem physischen Cloud-Knoten ein Monitoring-Agent ausgebracht. Die zur Erfassung notwendigen Informationen werden aus den vom CMS unabhängigen lokalen Datenquellen (Sensoren) gewonnen. Auf Anfrage des Monitoring-Controllers übermittelt der Monitoring-Agent die erfassten Monitoringdaten oder erhebt neue Monitoringdaten. Darüber hinaus sendet der Monitoring-Agent eigenständig Monitoringdaten an den Log-Server.

Log-Server: Protokollierungskomponente zur beweissicheren Verwahrung von Monitoringdaten bei einer vertrauenswürdigen dritten Partei. Dabei erhält der Log-Server die zu verwahrenden Informationen von den Monitoring-Agenten. Welche vertrauenswürdige dritte Partei zur beweissicheren Verwahrung verwendet werden soll, ist durch den Monitoring-Controller konfigurierbar. Die beweissichere Verwahrung an sich ist allerdings nicht Gegenstand dieser Dissertation.

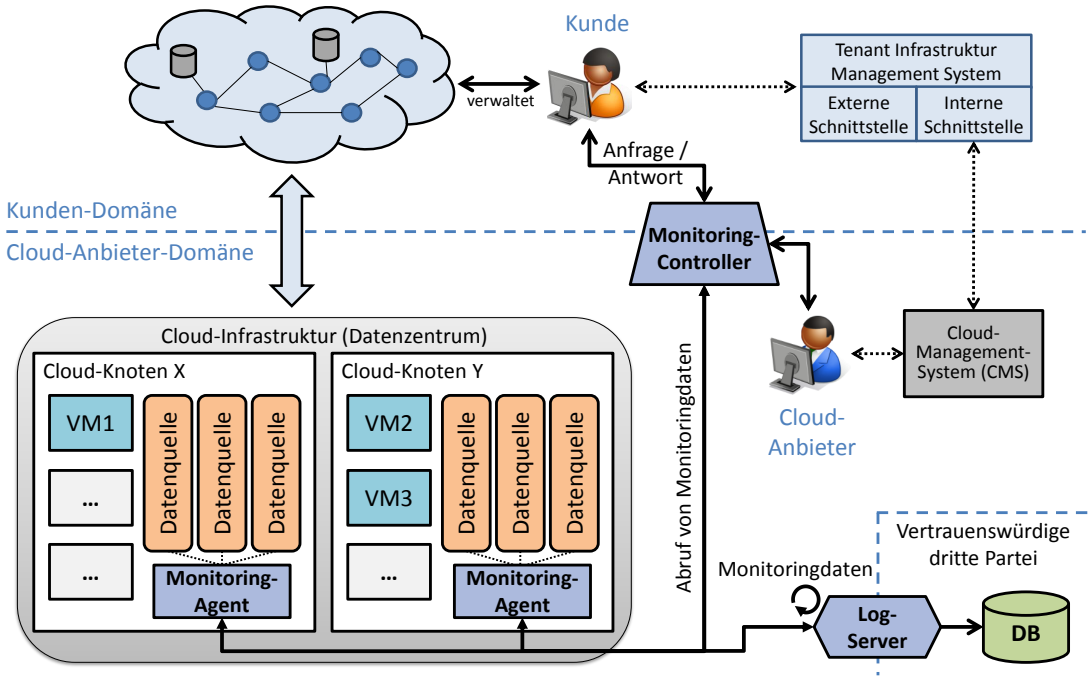

Abbildung 3.1: *Einordnung der Komponenten des CloudInspectors in das verwendete architekturelle Rahmenwerk für Cloud-Computing nach Flittner et al. [TT-5]. Nicht dargestellt: Zugriff des CloudInspectors auf den Kontrollkanal des Cloud-Management-Systems.*

Die Komponenten Monitoring-Agent und Log-Server befinden sich dabei innerhalb der Cloud-Anbieter-Domäne (in der Cloud-Infrastruktur), wohingegen sich der Monitoring-Controller an der Schnittstelle zwischen Kunden- und Cloud-Anbieter-Domäne befindet. Daher ist der Monitoring-Controller auch für Kunden erreichbar, wohingegen der Kunde keinen Zugriff auf die Monitoring-Agenten und den Log-Server hat.

3.3.2 Kommunikation zwischen den Komponenten

Eine Besonderheit des *CloudInspector*-Ansatzes ist die Art und Weise, wie Monitoringdaten vom Monitoring-Controller von den Monitoring-Agenten abgefragt werden. Hier spielt *Skalierbarkeit* eine besondere Rolle.

Eine Cloud-Infrastruktur besteht typischerweise aus mehreren zehntausend physischen Cloud-Knoten [265, 259, 168]. Für die Überprüfung von vertraglichen Vereinbarungen eines einzelnen Kunden sind allerdings nur die Daten einer kleinen Teilmenge der physischen Cloud-Knoten erforderlich – nämlich nur diejenigen, auf denen derzeit virtuelle Ressourcen des Kunden platziert sind.

Um inhärente Skalierbarkeit des *CloudInspector*-Ansatzes zu erreichen, werden Überprüfungen bei *CloudInspector* zielgerichtet auf die physischen Cloud-Knoten beschränkt, die virtuelle Ressourcen für den anfragenden Kunden verwalten. Hierzu wird innerhalb von *CloudInspector* für jeden Kunden eine eigene Monitoringgruppe gebildet. Die Monitoring-Agenten treten einer solchen Kunden-spezifischen Gruppe bei, sobald virtuelle Ressourcen für einen Kunden auf dem lokalen physischen Cloud-Knoten detektiert wurden. Auf diese Art und Weise kann der Monitoring-Controller für Überprüfungen durch Kunden mittels Request-Response-Kommunikation von allen zur Gruppe des Kunden gehörenden Monitoring-Agenten die entsprechenden Monitoringdaten abrufen. Physische Cloud-Knoten beziehungsweise Monitoring-Agenten, die keine virtuellen Ressourcen dieses Kunden verwalten, werden dadurch von der Request-Response-Kommunikation explizit ausgeklammert – diese sind nicht Mitglied in der entsprechenden Kunden-spezifischen Gruppe.

3.3.3 Modellierung des Monitoringprozess

Grundsätzlich lässt sich Prozess eines Monitoring-Mechanismus in einzelne Schritte zerlegen. In Abbildung 3.2 sind die für *CloudInspector* relevanten Prozessschritte *Auslöser*, *Datenerhebung* und *Verarbeitung* vereinfacht dargestellt. An dieser Stelle wird der Prozess zunächst nur im Überblick vorgestellt, weitere Details hierzu finden sich dann in Abschnitt 6.1.

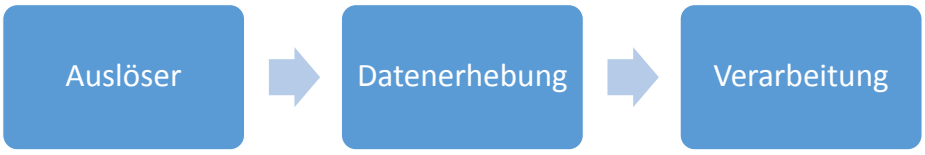

Abbildung 3.2: *Vereinfachte Darstellung eiens Monitoringprozesses.*

3.3.3.1 Auslöser

Auslöser für die Überprüfung von vertraglichen Vereinbarungen zur Laufzeit ist der Kunde. Auf dessen willkürliche Anfrage an den *CloudInspector* wird eine direkte Überprüfung der aktuellen Cloud-Konfiguration vorgenommen. Hierzu werden Monitoringdaten aus der Cloud-Infrastruktur erhoben und mit der gewünschten Spezifikation des Kunden abgeglichen.

Die Aufzeichnung von Monitoringdaten zur späteren Fehler-Ursachen-Analyse hingegen erfolgt kontinuierlich. Hierzu konfiguriert der Kunde den *CloudInspector* entsprechend.

Dieser überwacht dann ob neue Monitoringdaten vorliegen und verwahrt diese anschließend bei einer vertrauenswürdigen dritten Partei.

3.3.3.2 Datenerhebung

Prinzipiell werden die durch *CloudInspector* erhobenen Monitoringdaten aus unterschiedlichen Datenquellen gewonnen. Dabei ist der Zugriff auf *CloudInspector*-externe Komponenten inhärent notwendig. Aus diesen Komponenten werden dann Monitoringdaten über die Cloud-Umgebungen und die Aktivitäten des CMS abgefragt. Hierzu wird nach Möglichkeit auf existierende Schnittstellen zurückgegriffen. Die erhobenen Daten werden anschließend im nächsten Prozessschritt verarbeitet.

3.3.3.3 Verarbeitung

In diesem Prozessschritt wird einerseits abgeglichen, ob die vom Kunden angestoßene Überprüfung von vertraglichen Vereinbarungen erfolgreich war. Hierbei muss sichergestellt werden, dass die ermittelten Monitoringdaten bezüglich der aktiven Cloud-Konfiguration die gewünschte Spezifikation korrekt umsetzen und nicht verletzen.

Zum anderen werden die erhobene Monitoringdaten für beweissichere Verwahrung an eine vertrauenswürdigen dritte Partei übermittelt.

3.4 Monitoring-Bausteine und berücksichtigte Anforderungen

Die in dieser Dissertation behandelten Monitoring-Bausteine sind wie folgt im *CloudInspector*-Rahmenwerk zu verorten. Abbildung 3.3 gibt dabei einen Überblick über die Kapitelstruktur und Einordnung.

In Kapitel 4 wird ein **Adaptives Transportsystem für Request-Response in Gruppen** vorgestellt, welches sich bei *CloudInspector* für den Abruf von Monitoringdaten mehrerer Agenten eignet. Wichtige Anforderungen, die in diesem Kontext berücksichtigt werden, sind: *Anpassungsfähigkeit* (auch als adaptiv bezeichnet), *Skalierbarkeit* und *Effizienz*. Während der Entwurf des Rahmenwerks durch die Bildung von Monitoringgruppen je Kunde bereits von sich aus auf Skalierbarkeit ausgelegt ist, stellt die Ausgestaltung eines effizienten Transportsystems für Gruppen eine besondere Herausforderung dar. Außerdem

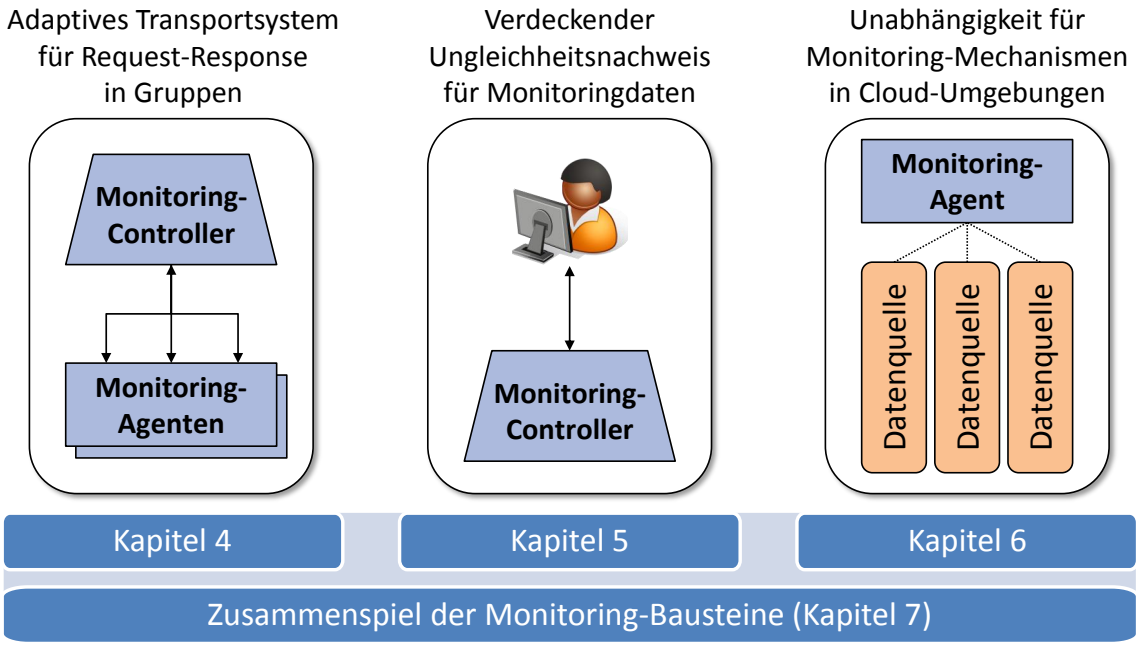

Abbildung 3.3: *Kapitelstruktur zu den behandelten Monitoring-Bausteinen.*

wird in Kapitel 4 dargelegt, wie das Transportsystem adaptiv auf Nachrichtenverlust reagiert.

Kapitel 5 beleuchtet das bei Überprüfungen zur Laufzeit unvermeidliche Spannungsverhältnis zwischen Geheimhaltung von Betriebsgeheimnissen des Cloud-Anbieters und der Nutzung von (geheimen) Monitoringdaten zur Überprüfung. Zur Laufzeit ist der Kunde einerseits nur an dem Ergebnis der Überprüfung interessiert, zur Fehler-Ursachen-Analyse anderseits sind die konkret überprüften Monitoringdaten relevant. In dem Kapitel wird daher ein **Verdeckender Ungleichheitsnachweis für Monitoringdaten** vorgestellt, der beide Anforderungen miteinander vereint: die Monitoringdaten bleiben während der Überprüfung geheim und sind gleichzeitig für spätere Fehler-Ursachen-Analysen eindeutig aufdeckbar. Besonders herausfordernd ist hier die sichere Ausgestaltung des Verfahrens. Eingesetzt werden soll das Verfahren dabei an der Schnittstelle zwischen Kunde und Monitoring-Controller.

Kapitel 6 widmet sich der im Vergleich zum Stand der Wissenschaft neu identifizierten Anforderung nach Unabhängigkeit vom CMS. Hier ist die **Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen** von besonderem Interesse. Dazu werden verschiedene Datenquellen hinsichtlich ihrer Unabhängigkeit vom CMS analysiert. Außerdem wird geklärt, unter welchen Bedingungen Unabhängigkeit vom CMS in realen Cloud-Umgebungen erreichbar ist und es wird eine passende Taxonomie entwickelt. Außerdem

wird aufgezeigt, wie die für Cloud-Umgebungen typische *Elastizität* Monitoringergebnisse beeinflusst. Abschließend erfolgt eine Zusammenfassung des durch unabhängige Monitoring-Mechanismen entstehenden Overheads.

Schließlich wird in Kapitel 7 kurz auf das **Zusammenspiel der Monitoring-Bausteine** innerhalb des hier vorgestellten Rahmenwerks für unabhängiges Cloud-Monitoring eingegangen.

3.5 Zusammenfassung

Cloud-Monitoring ist im aktuellen wissenschaftlichen Diskurs ein relevantes Thema. Allerdings wird der Forderung nach Unabhängigkeit vom CMS nach bestem Wissen bisher keine Aufmerksamkeit geschenkt. In dieser Dissertation hingegen wird der Aspekt der Unabhängigkeit – neben weiteren relevanten Anforderungen wie *Anpassungsfähigkeit*, *Elastizität* und *Skalierbarkeit* – als zentraler Bestandteil mitberücksichtigt.

Im Rahmen dieses Kapitels wird das der Dissertation zu Grunde liegende agentenbasierte Monitoring-Rahmenwerk mit dem Bezeichner *CloudInspector* vorgestellt. Dieses besteht aus den Komponenten Monitoring-Controller, Monitoring-Agenten und Log-Server.

Außerdem wird eine erste Einordnung der im Weiteren besprochenen Monitoring-Bausteine in das vorgestellte Rahmenwerk vorgenommen.

Kapitel 4

Adaptives Transportsystem für Request-Response in Gruppen

Um Monitoringdaten von mehreren physischen Cloud-Knoten einzusammeln, wird Request-Response-Kommunikation für Gruppen benötigt. Der Schwerpunkt dieses Kapitels liegt daher auf der Erweiterung der Request-Response-Dienst-Definition für Gruppen (vergleiche Definition 2.1 in Abschnitt 2.4). Eine besondere Herausforderunge im Kontext von gruppenbasierten Request-Response-Diensten ist die effiziente Nutzung von Netzressourcen. Hinzu kommt, dass der Begriff *Vollständigkeit* im Bezug auf die von den Gruppen erwarteten Response-Nachrichten neu aufgefasst werden muss.

Das in diesem Kapitel beschriebene Transportsystem trägt den Bezeichner *gAUDIT* und soll die Lücke von fehlender Unterstützung für *Request-Response* in dezentral verwalteten Gruppen schließen, indem eine effiziente gruppenbasierte Variante dieses Dienstes zur Verfügung gestellt wird.

Diese Kapitel widmet sich der folgenden Forschungsfrage:

> *Durch welche Mechanismen kann ein Request-Response-Dienst für Gruppen Netzressourcen effizient nutzen? Wie wirkt sich die erwartete Antwort-Vollständigkeit hierauf aus?*

Insbesondere wird besprochen, wann der Versand von Request-Nachrichten per Multicast effizienter als per Unicast ist. Darüber hinaus wird ein ARQ-Verfahren zur Fehlerbehandlung präsentiert und analysiert, wie Response-Nachrichten hinreichend desynchronisiert werden können, sodass die *Incast-Problematik* nicht auftritt und die Ausführungszeit von *gAUDIT* im akzeptablen Rahmen bleibt. Aspekte von Fluss- und Staukontrolle werden hier

zunächst nicht näher untersucht, da in der Regel nur eine einzelne Nachricht übertragen wird.

Ergebnisse dieses Kapitels basieren unter anderem auf der folgenden Publikation:

- M. Flittner, A. Weigel und M. Zitterbart. „gAUDIT: A group communication-capable request-response middleware for auditing clouds“. In: *2017 International Conference on Networked Systems (NetSys)*. März 2017 [TT-3]

4.1 Umzusetzende Kommunikationsanforderungen

Im folgenden Abschnitt werden am Beispiel des *CloudInspector*-Anwendungsfalls die Kommunikationsanforderungen für die Durchführung von Request-Response in Gruppen untersucht. Die Anpassung des Request-Response-Kommunikationsmusters auf Szenarien mit Gruppen ist von besonderem Interesse, weil bei *CloudInspector* Monitoringdaten von einer Gruppe von physischen Cloud-Knoten gleichzeitig abgefragt werden müssen.

Anhand der Abbildung 4.1 werden die Kommunikationsanforderungen des *CloudInspectors* weiter präzisiert. Gegeben sind ein Monitoring-Controller und mehrere Monitoring-Agenten. Um Überprüfungen durch den Monitoring-Controller auf einen einzelnen Kunden beschränkt durchführen zu können, wird bei *CloudInspector* je Kunde eine eigene Gruppe gebildet (siehe Abschnitt 3.3.2) – hier die Gruppen `A`, `B` und `C`. Außerdem wird in der Abbildung 4.1 ersichtlich, dass im abgebildeten Beispiel die Monitoring-Agenten unterschiedlichen Gruppen angehören – der Monitoring-Agent 1 der Gruppe `A` und `B`; der Monitoring-Agent 2 der Gruppe `A`, `B` und `C`; der Monitoring-Agent 3 der Gruppe `A` und `C`.

Ein Monitoring-Agent tritt einer solchen, zu einem Kunden gehörenden Gruppe, eigenständig bei, sobald virtuelle Ressourcen des Kunden auf dem entsprechenden physischen Cloud-Knoten ausgeführt werden (siehe Abschnitt 3.3.2). Die so je Kunde gebildeten Gruppen werden vom Monitoring-Controller verwendet, um gezielt zu einem Kunden gehörende Monitoringdaten von den verantwortlichen Monitoring-Agenten abzurufen – beispielsweise wenn überprüft wird, ob die VMs eines Kunden vom CMS an der vorgesehenen Stelle in der physischen Cloud-Infrastruktur platziert wurden. Bei einer Request-Nachricht an die Gruppe `A` sollen hier die Monitoring-Agenten 1, 2 und 3 erreicht werden. Für die Gruppe `C` sind dies die Monitoring-Agenten 2 und 3.

Je nach Art der Anfrage an die Gruppe sind unterschiedlich viele Antworten der Gruppenmitglieder notwendig. Wird beispielsweise die Platzierung einer einzelnen VM abgefragt, so genügt die Antwort des Monitoring-Agenten, auf dessen physischen Cloud-Knoten die

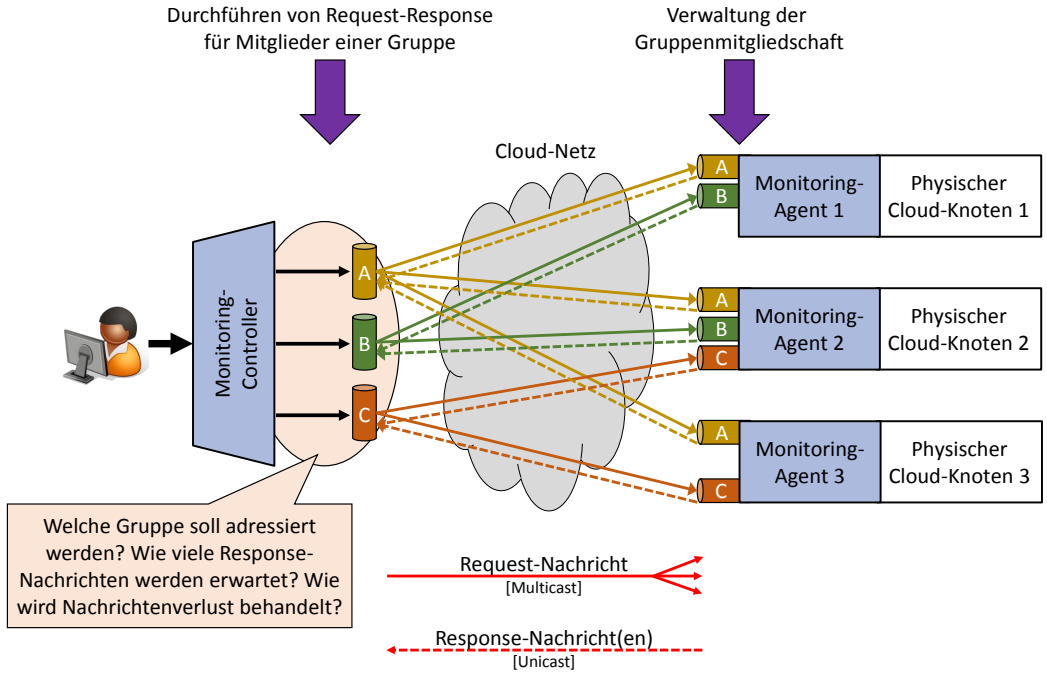

Abbildung 4.1: *High Level Übersicht der CloudInspector-internen Kommunikation und den sich daraus ergebenden Anforderungen.*

VM tatsächlich ausgeführt wird. Hingegen werden im Fall einer Zufallsstichprobe nur die Antworten der zufällig ausgewählten Gruppenmitglieder erwartet.

Hieraus ergeben sich zwei Anforderungen für die *CloudInspector*-interne Kommunikation.

Request-Response für Gruppen: Da bei *CloudInspector* eine Gruppe in der Regel aus mehreren Monitoring-Agenten besteht, muss zunächst die Frage der Durchführung von Request-Response in Gruppen gelöst werden. Hierbei sollen die Kommunikationsbeziehungen zwischen dem Client und mehreren Servern gemeinsam verwaltet und Request-Nachrichten effizient über das Cloud-Netz übertragen werden. Genauso sollen die davon abhängigen Kommunikationsbeziehungen zwischen den Servern und dem Client mit berücksichtigt werden (Bidirektionalität). Des Weiteren soll über Parameter einstellbar sein, wie viele Response-Nachrichten erwartet werden – beispielsweise, ob Response-Nachrichten von allen Mitgliedern erforderlich sind, oder ob bereits eine einzelne Nachricht ausreicht.

Dezentrale Gruppenverwaltung: Die Mitgliedschaft in einer Gruppe soll durch die Monitoring-Agenten frei wählbar sein. Dadurch können Monitoring-Agenten bestimmen, für welche Gruppen Request-Nachrichten empfangen werden. Jedem Kunde des

Cloud-Anbieters wird eindeutig eine Gruppe zugeordnet. Diese eindeutige Zuordnung wird sowohl von den Monitoring-Agenten als auch von dem Monitoring-Controller verwendet.

Die identifizierten Anforderungen werden im Folgenden als Transportsystem umgesetzt. Hierdurch lässt sich die Effizienz von IP- und Ethernet-Multicast für Request-Response in Gruppen nutzen (vergleiche auch Abschnitt 2.2). In Abbildung 4.2 wird dargestellt, welche Dienste die *CloudInspector*-Komponenten dabei von dem Transportsystem benötigen.

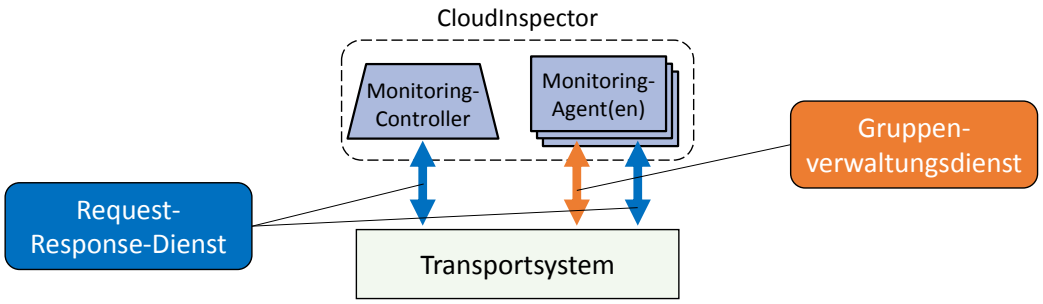

Abbildung 4.2: *Übersicht über die von CloudInspector-Komponenten verwendeten Transportsystem-Dienste.*

Im weiteren Verlauf wird das *gAUDIT*-Transportsystem vorgestellt, welches die oben beschriebene Funktionalität zur Verfügung stellt.

4.2 Request-Response-Dienste für Gruppen

Eine gruppenbasierte Dienstvariante des singulären Request-Response-Diensts (1RRS) (Punkt-zu-Mehrpunkt) wird im Folgenden als Gruppenbasierter 1RRS (g1RRS) bezeichnet und in Definition 4.1 weiter präzisiert. Wie bei 1RRS zwischen Punkt-zu-Punkt (siehe Abschnitt 2.4), spielt Reihenfolgetreue der Nachrichten bei g1RRS im Rahmen dieser Dissertation keine Rolle. Es kommt daher insbesondere nicht darauf an, von welchem Mitglied der Gruppe der Initiator zuerst eine Response-Nachricht erhält.

Definition 4.1 *[Gruppenbasierter 1RRS] Ein Dienst, der von einer vorgegebenen Menge der Mitglieder einer Gruppe Daten korrekt und ohne Duplikate oder Phantom-Daten abruft. Der Dienst ist erfüllt, sobald alle erwarteten Response-Nachrichten empfangen wurden. Die Fehlerbehandlung des Dienstes deckt dabei sowohl den Verlust von Nachrichten in beiden Richtungen zwischen Client- und Server-Instanzen ab (bidirektional), als auch den Verlust von Nachrichten in der Anwendungsebene von Server-Instanzen.*

Ein wichtiges Merkmal von g1RRS ist die Parametrisierung der erwarteten Response-Nachrichten im Bezug zu allen adressierten Gruppenmitgliedern – beispielsweise wenn zwar die gesamte Gruppe angefragt wird, jedoch nur die Antwort von einer Server-Instanz benötigt wird (zum Zeitpunkt der Anfrage ist nicht klar welches Gruppenmitglied antworten muss, sondern nur, dass genau eine Response-Nachricht erwartet wird). Als Begriff für diese Parametrisierung wird der Begriff *Response-Vollständigkeit* verwendet (siehe Definition 4.2).

Definition 4.2 *[Response-Vollständigkeit] Beschreibt bezüglich einer bei Request-Response adressierten Gruppe, von wie vielen Gruppenmitgliedern Response-Nachrichten erwartet werden.*

Typische gruppenbasierte Dienste der Transportschicht (vergleiche beispielsweise [190, 145, 47, 200, 191]) bieten zwar unterschiedliche Zuverlässigkeitsgrade, eignen sich allerdings nicht zur Umsetzung des g1RRS. Die gruppenbasierten Dienste der Transportschicht sind in der Regel auf unidirektionale Datenübertragung ausgelegt. Für einen Request-Response-Dienst bei Gruppen wird jedoch eine bidirektionale Betrachtung der Kommunikation mit Behandlung sowohl von Fehlern auf dem Transportweg als auch in Server-Instanzen vorausgesetzt. Hinzu kommt, dass existierende gruppenbasierte Dienste der Transportschicht keine Parametrisierung von Response-Vollständigkeit erlauben.

Um unterschiedliche Monitoringansätze zu unterstützen sind verschiedene Ausprägungen des g1RRS wünschenswert. In Tabelle 4.2 sind die identifizierten Klassen beschrieben.

Für den Anwendungsfall bei *CloudInspector* kommen insbesondere die g1RRS-Klassen *Statistisch-zuverlässig* und *k-zuverlässig* in Betracht. Die Klasse *Unzuverlässig* dient später zum Vergleich beziehungsweise als Benchmark.

4.3 Herausforderungen

Die wesentliche Herausforderung bei Realisierung des Transportsystems *gAUDIT* ist die Integration eines parametrisierbaren Request-Response-Mechanismus, bei dem die Verwaltung der Gruppenmitgliedschaft den Monitoring-Agenten (Server-Instanzen) obliegt. Weiterhin muss die Effizienz des Datentransports für verschiedene Parameterisierungen von Response-Vollständigkeit mitberücksichtigt werden. Auf beide Fragestellung wird im Folgenden näher eingegangen.

g1RRS-Klasse	Response-Vollständigkeit	Anwendungsfall
Unzuverlässig	keine	Monitoring von nicht kritischen Zuständen. Beispielsweise zur Überprüfung der Systemzeit der physischen Cloud-Knoten.
Statistisch-zuverlässig	Angabe der Auswahlwahrscheinlichkeit a (%), mit deren Hilfe alle Gruppenmitglieder bestimmen, ob sie Teil der Stichprobe sind	Monitoring für Zwecke bei denen eine Totalerhebung zu teuer und zeitaufwändig wäre (z.B. Zufallsstichprobe aus Lastgründen). Bei Verwendung von $a = 100\%$ werden Information von allen Gruppenmitgliedern angefordert.
k-zuverlässig	Absolute Anzahl k der erwarteten Response-Nachrichten	Monitoring für Anwendungsfälle bei denen a priori bekannt ist, von wie vielen Gruppenmitgliedern eine Antwort erwartet wird, aber nicht feststeht welche genau antworten müssen. Es muss sichergestellt sein, dass $k \leq n$ bei Gruppengröße n gewählt wird.

Tabelle 4.2: *Übersicht der Klassen von gruppenbasierten Request-Response-Diensten.*

Request-Response im Kontext von Gruppen: Wird eine Request-Nachricht an eine Gruppe von Server-Instanzen versendet, so muss geklärt werden, wie viele Response-Nachrichten erwartet werden und wie Fehler behandelt werden. Dabei soll das *gAUDIT*-Transportsystem unterschiedliche parametrisierbare Klassen von g1RRS für den Monitoring-Controller bereitstellen, so dass dieser auswählen kann, von wie vielen Gruppenmitgliedern eine Antwort erwartet wird. Des Weiteren muss geklärt werden, wie sich ein g1RRS verhalten soll, wenn sich die Zusammensetzung einer Gruppe während einer laufenden Abfrage ändert. Darüber hinaus stellt die Ausgestaltung eines ARQ-Verfahrens mit Berücksichtigung der parametrisierten Response-Vollständigkeit als Teil der Fehlerkontrolle für den g1RRS eine besondere Herausforderung dar. Hier

muss vor allem geklärt werden, wann Multicast und wann Unicast für Sendewiederholungen geeignet ist.

Effizienz: Der Mechanismus zur Realisierung von gruppenbasierter Request-Response-Kommunikation soll die Ressourcen des Netzes möglichst effizient nutzen. Dies gilt insbesondere für die Umsetzung der verschiedenen g1RRS-Klassen. Hierzu muss insbesondere geklärt werden, auf welche Funktionen der transportorientierten Schichten des Internet Modells (Schicht 1 bis 3) von *gAUDIT* zurückgegriffen werden kann – beispielsweise zur effizienten Realisierung von Multicast.

Die aufgeführten Herausforderungen stellen zusammen mit den Kommunikationsanforderungen von *CloudInspector* zugleich das Hauptunterscheidungsmerkmal zu verwandten Arbeiten dar: Parametrisierbarer g1RRS für effiziente Request-Response Kommunikation in Empfänger-verwalteten Gruppen (vergleiche Abschnitt 4.8).

4.4 Konzept

Bei *CloudInspector* wird die Anwendung in Monitoring-Controller (Client-Instanz) und Monitoring-Agenten (Server-Instanzen) unterschieden. Dabei soll die Client-Instanz mittels des *gAUDIT*-Transportsystems Monitoringdaten von Gruppen abfragen und die dafür erwartete Response-Vollständigkeit parametrisieren können. Eine Server-Instanz tritt mittels des *gAUDIT*-Transportsystems Gruppen bei oder verlässt diese. Außerdem empfangen beziehungsweise beantworten Server-Instanzen eingehende Request-Nachrichten einer Client-Instanz.

Nachstehend wird das *gAUDIT*-Transportsystem detailliert vorgestellt.

4.4.1 Transportsystem-Komponenten

Das *gAUDIT*-Transportsystem besteht aus drei Komponenten: 1) dem *Initiator*, 2) dem *Respondent* und 3) der *Gruppe*. Die Komponenten werden im Folgenden kurz vorgestellt.

Gruppe: Bei einer *Gruppe* handelt es sich um eine logische Abstraktion innerhalb von *gAUDIT*. Es können mehrere *Gruppen* gebildet werden und eine *Server-Instanz* kann Mitglied in mehreren *Gruppen* gleichzeitig sein. Eine *Gruppe* lässt sich durch die sogenannte Gruppen-ID (GID) eindeutig identifizieren.

Initiator: Eine Client-Instanz kann über den *Initiator* auf den g1RRS zur Durchführung von Request-Response in Gruppen zurückgreifen. Die Request-Response-Kommunikation ist dabei auf eine einzelne *Gruppe* beschränkt und muss durch Angabe der Response-Vollständigkeit parametrisiert werden. Die Komponente *Initiator* verwendet die *Initiator-interne* Datenbank *Member* zur Ermittlung der Mitglieder einer *Gruppe*. Der Aufbau und die Verwendung dieser Datenbank wird in Abschnitt 4.5.4 näher ausgeführt.

Respondent: Bietet einer Server-Instanz einen Dienst zur Verwaltung von Gruppenmitgliedschaften an. Außerdem empfängt die Komponente *Respondent* eingehende Request-Nachrichten und reicht diese gegebenenfalls an die Server-Instanz weiter. Der *Respondent* übernimmt außerdem das Versenden von Response-Nachrichten zurück zur Client-Instanz. Intern enthalten die *Respondents* einen Zwischenspeicher zur Verwaltung von Request-Response-Kommunikation. Der Aufbau und die Verwendung dieses Zwischenspeichers wird in Abschnitt 4.6.7 näher ausgeführt.

Für den Einsatz bei *CloudInspector* wird in der Regel eine *Initiator* für den Monitoring-Controller und mehrere *Respondents* für die Monitoring-Agenten verwendet. Auch die *Gruppe* kommt bei *CloudInspector* mehrfach zum Einsatz und zwar so, dass für jeden Kunden der Cloud-Umgebung eine eigene *Gruppe* verwendet wird.

4.4.2 Einordnung in den Protokollstapel

Im Folgenden wird das Zusammenspiel zwischen dem *gAUDIT*-Transportsystem und den Transportsystembenutzern – hier am Beispiel *CloudInspector* mit dem Monitoring-Controller (Client-Instanz) und den Monitoring-Agenten (Server-Instanzen) – ausgeführt. Als Beispiel wird die gruppenbasierte Request-Response-Kommunikation für die *Gruppe* X gezeigt.

Zur Realisierung der *gAUDIT*-Komponenten und Funktionalität bringt das Transportsystem die sogenannte *gAUDIT*-Schicht mit sich, die sich nahtlos zwischen der Anwendungs- und Transportschicht des Internet Modells integriert. Abbildung 4.3 zeigt, dass sich die *gAUDIT*-Schicht aus den *gAUDIT*-Komponenten *Initiator* und *Respondent* zusammensetzt, wobei die Initiator-Komponente vom Monitoring-Controller (Client-Instanz) und die Respondent-Komponente von den Monitoring-Agenten (Server-Instanz) verwendet wird. Außerdem wird deutlich, dass Request-Response-Kommunikation auf eine *Gruppe* beschränkt ist – hier im Beispiel ist dies die *Gruppe X*.

Des Weiteren sind Dienstzugangspunkte für Anwendungsinstanzen eingezeichnet. Das *gAUDIT*-Transportsystem stellt auf Seiten des *Initiators* genauso wie auf Seiten des *Respondents* einen *Request-Response-Dienstzugangspunkt* zur Verfügung. Auf Seiten des

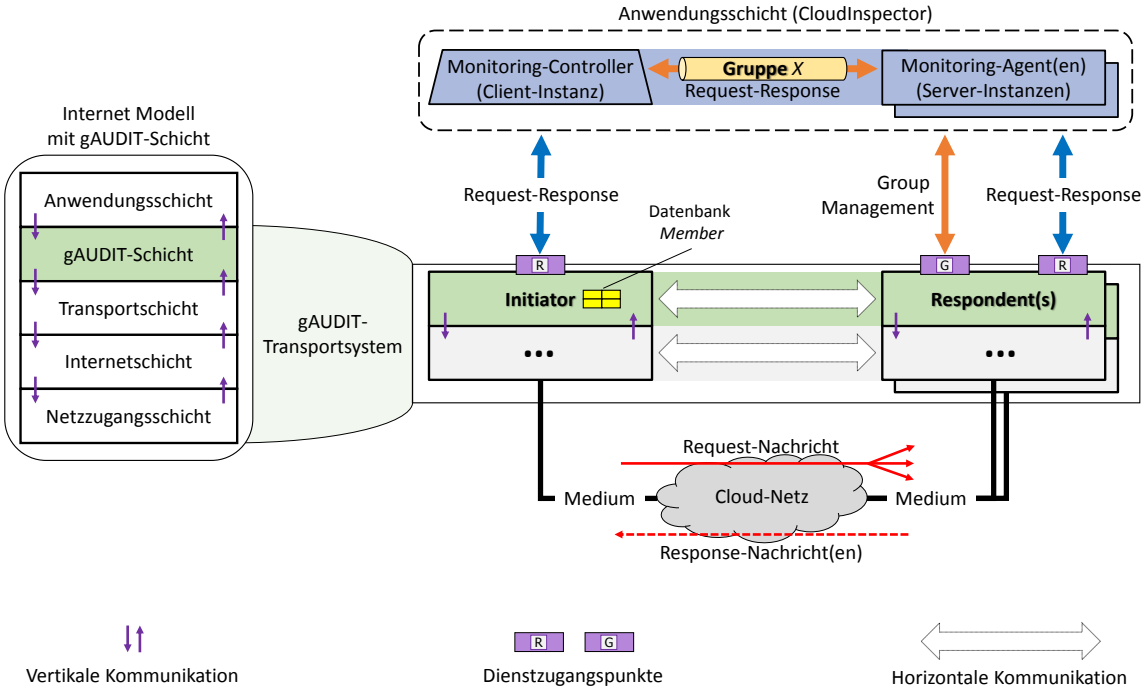

Abbildung 4.3: *Zusammenspiel des gAUDIT-Transportsystems mit dem Monitoring-Controller und den Monitoring-Agenten des CloudInspectors. Einordnung der Komponenten in den Protokollstapel des Internet Modells.*

Initiators kann der *Request-Response-Dienstzugangspunkt* sowohl zum Verwalten der Request-Response-Kommunikation verwendet werden (Starten, Abbrechen, Rückmeldungen) als auch zum Empfang von Response-Nachrichten. Auf Seiten des *Respondents* steht der *Request-Response-Dienstzugangspunkt* zum Empfang von Request-Nachrichten und für das Versenden von Response-Nachrichten zur Verfügung. Zusätzlich wird auf Seiten des *Respondents* ein *Group-Management-Dienstzugangspunkt* angeboten, über den Server-Instanzen *Gruppen* beitreten oder verlassen können.

4.4.3 Dienstangebot und Nachrichtenformat

Jeder Dienst des *gAUDIT*-Transportsystems besteht aus einer Dienstdefinition und Protokollspezifikation (siehe auch [115, 169]). Das *gAUDIT*-Transportsystem bietet die Dienste Group Management Service (GMS) und Request-Response Service (RRS) an. Die unterschiedlichen Dienstzugangspunkte werden aus Abbildung 4.3 ersichtlich. Darüber hinaus finden sich weitere Details im Sequenzdiagramm in Abbildung 4.4.

Im oberen Teil der Abbildung 4.4 ist dargestellt, wie Monitoring-Agenten den GMS von *gAUDIT* zum Gruppenbeitritt verwenden können. Hierzu wird eine Nachricht vom GMS auf Seiten der *Respondents* an den GMS auf Seiten des *Initiators* gesendet. Details hierzu werden in Abschnitt 4.4.3.1 erläutert. Im unteren Teil der Abbildung 4.4 wird dargestellt, wie nach Anfrage durch einen Kunden der Monitoring-Controller des *CloudInspectors* den RRS von *gAUDIT* zum Abrufen von Monitoringdaten nutzt. Hierzu wird eine Request-Nachricht an eine Gruppe von *Respondents* gesendet und Antworten in Form von Response-Nachrichten eingesammelt. Weitere Details hierzu werden in Abschnitt 4.6 erläutert.

In den nachstehenden Unterabschnitten werden die von *gAUDIT* für Anwendungsinstanzen zur Verfügung stehenden Dienste und deren Definition kurz beschrieben. Darüber hinaus wird das von den Diensten eingesetzte Nachrichtenformat präsentiert. Anknüpfend an die Übersicht in diesem Abschnitt werden in den nächsten Abschnitten (Abschnitt 4.5 und 4.6) die Dienstprimitive, Parameter, Dienstelemente und Protokollspezifikationen der Dienste detailliert ausgeführt.

4.4.3.1 Group Management Service (GMS)

Der GMS des *gAUDIT*-Transportsystems wird von Server-Instanzen (bei *CloudInspector* von den Monitoring-Agenten) verwendet, um einer bestimmten *Gruppe* beizutreten oder diese zu verlassen. Auf diese Art und Weise können Server-Instanzen steuern, für welche *Gruppen* Request-Nachrichten empfangen werden sollen und für welche nicht. Aus diesem Grund steht eine Schnittstelle auf Seiten des *Respondents* als Dienstzugangspunkt zum GMS für Anwendungsinstanzen zur Verfügung. Auf Seiten des *Initiators* hingegen wird keine Schnittstelle angeboten, sondern die Dienstelemente des GMS werden vor der Client-Instanz verborgen. Hauptaufgabe des GMS ist die Synchronisierung der Gruppenmitgliedschaften zwischen *Respondent* und *Initiator*. Die Information über die aktuelle Gruppenzusammensetzung wird dann vom RRS zur Umsetzung der angeforderten g1RRS-Klasse benutzt (siehe nächster Abschnitt 4.4.3.2).

Zur Realisierung der Funktionalität des GMS wird ein Mechanismus zur Gruppenverwaltung eingesetzt. Dieser Mechanismus basiert auf den folgenden Dienstelementen – nähere Angaben zu Interna und Aufgaben des GMS finden sich in Abschnitt 4.5.

Datenbank *Member*: Datenbank auf Seiten des Initiators, in der die Mitgliedschaft von Server-Instanzen in *Gruppen* festgehalten wird. Die vom GMS in der Datenbank *Member* hinterlegten Information werden weiter zur Realisierung des RRS benötigt.

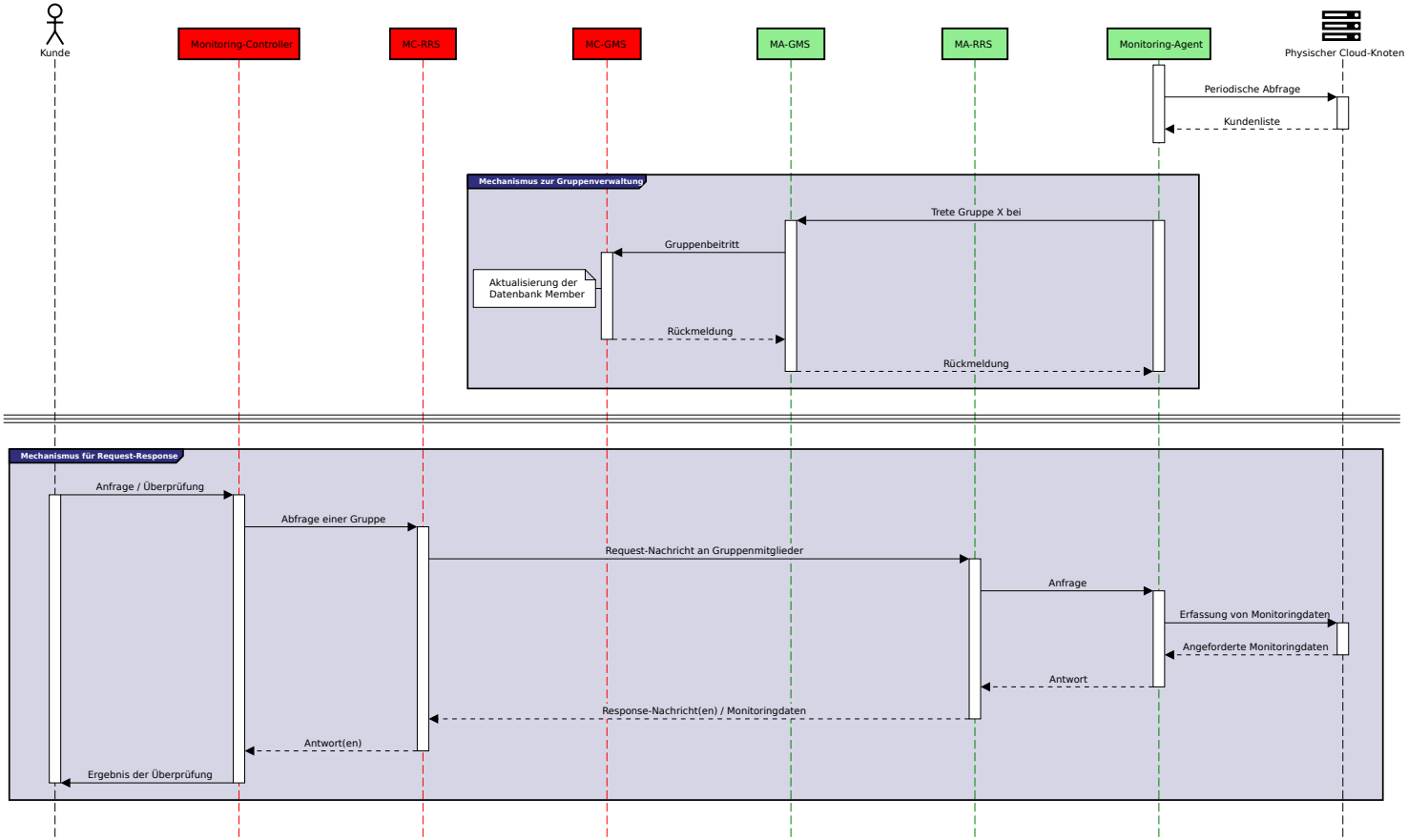

Abbildung 4.4: *Übersicht der Dienstmechanismen des gAUDIT-Transportsystems. Dargestellt sind der Group Management Service (GMS) und der Request-Response Service (RRS). Der obere Teil zeigt den Beitritt zur Gruppe mit dem Bezeichner X. Der untere Teil zeigt Request-Response-Kommunikation in der Gruppe X. MC steht dabei für den Monitoring-Controller und MA für den Monitoring-Agent.*

Management-Nachricht $\mathbb{G}_{Mgmt}$: Dieser Nachrichtentyp wird verwendet, wenn Server-Instanzen einer *Gruppe* beitreten oder diese verlassen wollen. In beiden Fällen versendet der *Respondent* eine Nachricht des Typs $\mathbb{G}_{Mgmt}$ an den *Initiator*.

Approval-Nachricht $\mathbb{G}_{Apvl}$: Nach Empfang und Verarbeitung einer Nachricht des Typs $\mathbb{G}_{Mgmt}$ durch den *Initiator* wird durch den *Initiator* eine Bestätigung des Vorgangs an den *Respondent* zurückgesendet. Hierzu versendet der *Initiator* die Nachricht des Typs $\mathbb{G}_{Apvl}$ an den *Respondent*.

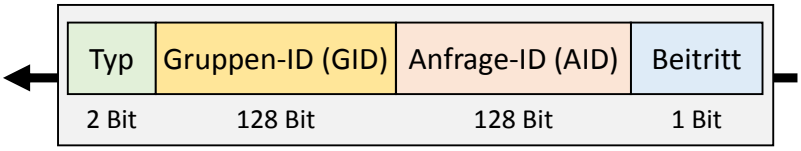

Abbildung 4.5: *Format der Nachrichten des Typs* $\mathbb{G}_{Mgmt}$ *und* $\mathbb{G}_{Apvl}$.

In Abbildung 4.5 ist das Format der Nachrichten des Typs $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$ dargestellt. Das Format beider Nachrichten ist in vier Felder unterteilt: 2 Bit für den Typ, 128 Bit für die GID, 128 Bit für die Anfrage-ID (AID) und 1 Bit für das Feld *Beitritt*. Das Typ-Feld ist hier wie folgt codiert: `10` für eine Nachricht des Typs $\mathbb{G}_{Mgmt}$; `11` für eine Nachricht des Typs $\mathbb{G}_{Apvl}$– die Werte `00` und `01` werden für die Nachrichten des RRS benötigt (siehe Abschnitt 4.6). Das Feld GID enthält die 128 Bit-Kennung der *Gruppe* auf die sich die jeweilige Nachricht bezieht. Das Feld AID wird zur Zuordnung von Anfrage und Bestätigung verwendet. Außerdem dient es zur Erkennung von Duplikaten im Falle von Sendewiederholungen. Über das Beitritt-Feld wird signalisiert, ob ein Beitritt zu der angegebenen *Gruppe* erfolgen soll, oder ob diese verlassen wird. Dabei wird ein Wert von `0` für Beitritt und ein Wert von `1` für Verlassen verwendet.

4.4.3.2 Request-Response Service (RRS)

Der RRS des *gAUDIT*-Transportsystems wird von einer Client-Instanz (bei *CloudInspector* von dem Monitoring-Controller) verwendet, um Daten von einer Gruppe von Server-Instanzen (bei *CloudInspector* Monitoringdaten von Monitoring-Agenten) mittels Request-Response-Kommunikation abzurufen. Die Client-Instanz wählt dazu die g1RRS-Klasse aus und gibt Parameter für die erwartete Response-Vollständigkeit mit an – für mehr Details siehe Abschnitt 4.6. Der RRS bietet sowohl auf Seiten des *Initiators* als auch auf der von *Respondents* eine Schnittstelle als Dienstzugangspunkt an. Auf Seiten des *Initiators* wird diese Schnittstelle von einer Client-Instanz genutzt, um eine Request-Nachricht an eine Gruppe von Server-Instanzen zu versenden und eingehende Response-Nachrichten zu

empfangen. Analog wir der Dienstzugangspunkt des RRS auf Seiten des *Respondents* von Server-Instanzen genutzt, um eingehende Request-Nachrichten zu empfangen und Response-Nachrichten mit den angeforderten Monitoringdaten an die Client-Instanz zurückzusenden. Außerdem steht der RRS auf Seiten des *Respondents* in direkter Abhängigkeit zum GMS: Request-Nachrichten werden nur für *Gruppen* empfangen, für die die Server-Instanz zuvor einen Beitritt veranlasst hat (für mehr Details siehe Abschnitt 4.6).

Da in den unterliegenden Schichten des Internet Modells (Transport-, Internet- und Netzzugangsschicht) nicht entschieden werden kann, ob eine Response-Nachricht ausbleibt weil es auf dem Übertragungsweg zu einem Fehler gekommen ist, oder ob die Server-Instanz noch mit der Bearbeitung der Request-Nachricht beschäftigt ist, sorgt der Request-Response-Mechanismus des RRS bei Ausbleiben von Response-Nachrichten mittels eines speziellen ARQ-Verfahrens für Sendewiederholungen von Request-Nachrichten. Hierdurch wird die von der Client-Instanz angeforderte g1RRS-Klasse und Response-Vollständigkeit umgesetzt. Dieser Mechanismus basiert auf den folgenden Dienstelementen – nähere Angaben zu Interna und Aufgaben des RRS finden sich in Abschnitt 4.6.

Request-Nachricht $\mathbb{G}_{Req}$: Dieser Nachrichtentyp wird vom *Initiator* verwendet, um die Anfrage einer Client-Instanz an eine Gruppe von Server-Instanzen zu übertragen. Hierzu versendet der *Initiator* die Nachricht des Typs $\mathbb{G}_{Req}$ an die entsprechenden *Respondents*, die Mitglied der angegeben *Gruppe* sind.

Response-Nachricht $\mathbb{G}_{Res}$: Dieser Nachrichtentyp wird von einem *Respondent* verwendet, um Antworten von Server-Instanzen zu der anfragenden Client-Instanz zu übertragen. Hierzu versendet ein *Respondent* die Nachricht des Typs $\mathbb{G}_{Res}$ an den *Initiator*, der die Nachricht des Typs $\mathbb{G}_{Req}$ ausgelöst hat.

Anhand Abbildung 4.6 wird das Format der beiden Nachrichten des RRS erläutert. Diese teilen sich dabei die Felder Typ und GID mit den Nachrichten $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$ des Gruppenverwaltungs-Mechanismus (siehe Abschnitt 4.4.3.1). Zusätzlich sind 128 Bit für die AID und 8 Bit zur Angabe der gewählten g1RRS-Klasse vorgesehen. Hiernach folgt ein Feld für Nutzdaten, die eigentliche Monitoring-Abfrage. Das Typ-Feld ist hier wie folgt codiert: `0` für eine Nachricht des Typs $\mathbb{G}_{Req}$; `1` für eine Nachricht des Typs $\mathbb{G}_{Res}$. Das Feld GID enthält wie bei den Nachrichten zur Gruppenverwaltung die 128 Bit-Kennung der *Gruppe* auf den sich die jeweilige Nachricht bezieht. Die AID ist eine eindeutige Identifikationsnummer, um die Nachrichten $\mathbb{G}_{Req}$ und $\mathbb{G}_{Res}$ einander zuordnen zu können. Dabei steht die hier verwendete AID nicht in Bezug mit der beim GMS verwendeten AID – die Unterscheidung erfolgt anhand des Typ-Feldes. Mit Hilfe des Felds `g1RRS-Klasse` teilt der *Initiator* den *Respondents* mit, welche g1RRS-Klasse von der Client-Instanz gewählt wurde und welche *Response-Vollständigkeit* erwartet wird. Dabei signalisiert das erste Bit die Klasse, bei einem Wert von `0` die Klasse *Statistisch-zuverlässig* und bei einem Wert von `1` die Klasse *k-zuverlässig*. Die restlichen sieben Bit des Feldes geben

die erwartete *Response-Vollständigkeit* an. Für die Klasse *statistisch-zuverlässig* wird die parametrisierte Auswahlwahrscheinlichkeit hinterlegt. Bei der Klasse *k-zuverlässig* wird die Anzahl der erwarteten Nachrichten codiert. Bei den Nutzdaten wird in einer Nachricht des Typs $\mathbb{G}_{Req}$ die Anfrage der Client-Instanz hinterlegt (bei *CloudInspector* beispielsweise "Liste aller VMs des Kunden A"). Bei einer Nachricht des Typs $\mathbb{G}_{Res}$ sind in den Nutzdaten die angefragten Monitoringdaten enthalten (bei *CloudInspector* beispielsweise "auf dem physischen Cloud-Knoten Z werden die VMs 1, 2 und 3 des Kunden A ausgeführt").

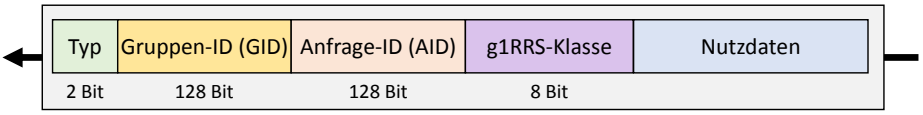

Abbildung 4.6: *Format der Nachrichten $\mathbb{G}_{Req}$ und $\mathbb{G}_{Res}$.*

4.4.4 Nutzung von unterliegenden Schichten

Nachdem die Dienste und die Nachrichtenformate von *gAUDIT* vorgestellt wurden, wird nachstehend geklärt, wie sich die Datenströme des *gAUDIT*-Transportsystems effizient durch die unter der *gAUDIT*-Schicht liegenden Schichten des Internet Modells realisieren lassen (vergleiche Abschnitt 4.4.2 und Abschnitt 4.3) – dies sind die Schichten Netzzugang, Internet und Transport.

Die Kommunikation, die die unterliegenden Schichten unterstützen soll, lässt sich anhand der Dienste des *gAUDIT*-Transportsystems unterscheiden (siehe Abschnitt 4.4.3). Bei Nachrichten des Typs $\mathbb{G}_{Req}$ handelt es sich um Gruppenkommunikation (Multicast) zwischen dem *Initiator* und den *Respondents*. Als Antwort werden Unicast-Nachrichten des Typs $\mathbb{G}_{Res}$ von den *Respondents* zu dem *Initiator* erwartet. Je nach Art der gewählten g1RRS-Klasse werden bei Ausbleiben von Nachrichten des Typs $\mathbb{G}_{Res}$ vom *Initiator* durch ein ARQ-Verfahren Sendewiederholungen der Nachricht des Typs $\mathbb{G}_{Req}$ durchgeführt (siehe Abschnitt 4.4.3.2 und Abschnitt 4.6). Auf diese Art und Weise sollen nicht nur Fehler auf dem Übertragungsweg zwischen *Initiator* und *Respondent* abgedeckt werden, sondern auch temporäre Fehler auf Seiten des *Respondents* – etwa Nachrichtenverlust in Server-Instanzen durch kurzfristige Überlastung des Arbeitsspeichers. Der Fehlerbehandlungsmechanismus deckt Fehler der Art *crash failure* [153] nicht mit ab, hier schlägt auch der RRS von *gAUDIT* fehl – hingegen werden Fehler abgedeckt, bei denen Monitoring-Agenten, beispielsweise bei hoher Auslastung, einige Request-Nachrichten verwerfen. Bei den Nachrichten des Typs $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$ handelt es sich hingegen um Unicast-Nachrichten zwischen den *Respondents* und dem *Initiator* (beziehungsweise in umgekehrter Richtung). Die Nachricht des Typs $\mathbb{G}_{Apvl}$ dient dabei als Bestätigung für den Gruppenbeitritt oder das Verlassen – siehe Abschnitt 4.4.3.1 und Abschnitt 4.5.

Als grundlegendes Merkmal der *gAUDIT*-internen Kommunikation lässt sich ableiten, dass sowohl der Mechanismus des GMS als auch der des RRS die Zuverlässigkeit der Nachrichtenübertragung mitbetrachten. Dies liegt vor allem daran, dass der *Initiator* und die *Respondents* auf explizites Feedback bezüglich der bidirektionalen-Zuverlässigkeit angewiesen sind. Im Falle des Request-Response-Mechanismus ist jeder Empfang einer Nachricht des Typs $\mathbb{G}_{Res}$ zugleich eine implizite Bestätigung, dass kein Fehler auf dem Transportweg und auch nicht in der Anwendungsebene der Server-Instanz auf Seiten des Respondents vorliegt. Im gleichen Sinne nutzt der Mechanismus zur Gruppenverwaltung die Nachricht des Typs $\mathbb{G}_{Apvl}$ als Bestätigung für den zuverlässigen Transport zwischen *Respondent* und *Initiator*. Semantisch kann es sich dabei zusätzlich um eine Bestätigung der Gruppenverwaltungsanfrage oder um eine Ablehnung handeln.

Da beide Mechanismen explizites Feedback für die Kommunikation vorsehen, wird auf ein zuverlässiges Transportprotokoll verzichtet. Stattdessen baut *gAUDIT* auf *UDP* [237] als Transportprotokoll auf. Daher werden die Nachrichten zum Versand in UDP-Segmente verpackt. Um die für die Request-Response-Kommunikation erforderliche Gruppenkommunikation effizient umzusetzen, soll sowohl in der Internet- als auch in der Netzzugangs-Schicht auf Multicast-Mechanismen zurückgegriffen werden. Auf diese Art und Weise kann das Netz Endsysteme bei der effizienten Gruppenkommunikation unterstützen, indem Nachrichten nur an Stellen im Netz dupliziert werden, an denen dies tatsächlich notwendig ist. Für *gAUDIT* wurde als Protokoll der Internetschicht *IPv6* [220] gewählt und die UDP-Segmente werden in IPv6-Datagramme verpackt. Weiter wird für den Netzzugang *Ethernet* [135] (für aktuelle Entwicklung siehe [93] und [269]) eingesetzt (Netzzugangsschicht). Somit werden die IPv6-Datagramme wiederum in Ethernet-Rahmen verpackt. Abbildung 4.7 zeigt, wie die Nachrichten des Typs $\mathbb{G}_{Req}$, $\mathbb{G}_{Res}$, $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$ in die unterliegenden Schichten integriert werden. Weiterhin muss das Netz zwischen *Initiator* und *Respondents* das MLD-Protokoll [216] beziehungsweise Multicast-Routing (vergleiche [225]) unterstützen, um Nachrichten des Typs $\mathbb{G}_{Req}$ effizient übertragen zu können.

Für die Nachricht des Typs $\mathbb{G}_{Req}$ wird auf die Mechanismen zur Gruppenkommunikation des Ethernet- und IPv6-Protokolls zurückgegriffen. Hierfür muss insbesondere die Empfängeradresse im *Kopf* des entsprechenden Protokolls jeweils passend gewählt werden. Durch passende Adressierung kann hier eine Gruppe von Empfängersystemen ausgewählt werden. Nachstehend wird beschrieben, wie die innerhalb von *gAUDIT* zur Adressierung einer *Gruppe* verwendete GID dabei als Empfängeradresse umgesetzt wird. Abbildung 4.8 zeigt den Abbildungsvorgang.

Eine Zieladresse im IPv6-Kopf ist 128 Bit lang (siehe [220]). Soll Gruppenkommunikation angewandt werden, können die letzten 112 Bit frei vergeben werden, die restlichen 16 Bit sind vorgegeben. So beginnt eine IPv6-Multicast-Zieladresse immer mit `ff` gefolgt von

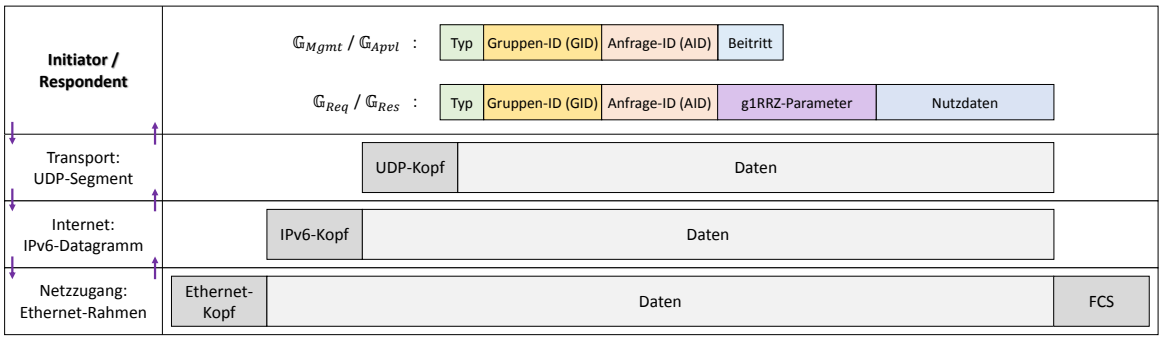

Abbildung 4.7: *gAUDIT-internen Nachrichten und Kapslung in unterliegende Schichten des Internet Modells.*

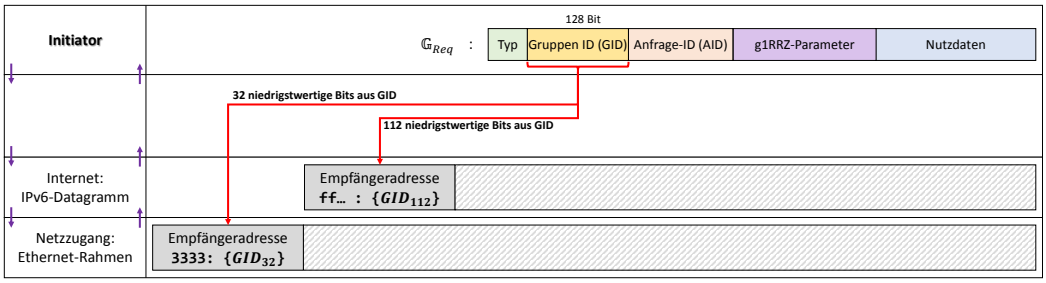

Abbildung 4.8: *Umsetzung der GID der Nachricht des Typs $\mathbb{G}_{Req}$ als Empfängeradresse in unterliegenden Schichten zur Gruppenkommunikation.*

4 Bit für *Flags* und 4 Bit zur Angabe des *Scopes* (vergleiche auch [234]). Da das GID-Feld einer *gAUDIT*-Nachricht des Typs $\mathbb{G}_{Req}$ 128 Bit enthält (typischerweise werden in Cloud-Umgebungen 128 Bit als Kunden-ID verwendet), wird eine Abbildung auf die im IPv6-Kopf zur Verfügung stehenden 112 Bit benötigt. Hier werden die 112 niedrigstwertigen Bit aus GID verwendet.

Weiter erlaubt auch die Zieladresse im Ethernet-Kopf die Adressierung einer *Gruppe*. Hier ist eine Zieladresse 48 Bit lang, wobei wiederum die ersten 16 Bit für eine Gruppenadresse fest vorgeben sind. So beginnt eine Ethernet-Multicast-Zieladresse immer mit *3333* gefolgt von 32 Bit frei zu vergebendem Adressteil. Hier werden die 32 niedrigstwertigen Bit aus GID verwendet. Dies ist analog zur Spezifikation in [229, 209] zur Abbildung von IPv6 auf Ethernet-Multicast-Zieladressen.

Neben dieser Art der Adressierung wird vom *Respondent* verlangt, dass er neben Synchronisierung der Gruppenmitgliedschaften mit dem *Initiator*, dem Netz mittels des MLD-Protokolls [216] mitteilt, für welche *Gruppe* Nachrichten empfangen werden sollen (mehr Details dazu in Abschnitt 4.5.3). Allerdings kann es durch diese Art der Adressierung auch dazu kommen, dass ein *Respondent* eine Nachricht des Typs $\mathbb{G}_{Req}$ erhält,

obwohl dieser nicht zuvor der im GID-Feld angegebenen *Gruppe* (mittels des GMS) beigetreten ist. Ursache dieses Problems ist, dass die *128* Bit GID in den unterliegenden Schichten nur in komprimierter Form vom *Initiator* als Zieladresse verwendet wird – bei IPv6 sind dies die hinteren *112* Bit, bei Ethernet die hinteren *32* Bit. Somit kommt es unweigerlich zu Kollisionen bei der Adressierung. Das *gAUDIT*-Transportsystem löst diese Problematik durch das in einer Nachricht des Typs $\mathbb{G}_{Req}$ enthaltene Feld GID. Anhand dieses Felds identifizieren *Respondents* die betroffene *Gruppe* eindeutig. Nachrichten die für eine *Gruppe* empfangen werden, für die ein Monitoring-Agent zuvor nicht beigetreten ist, werden vom *Respondent* verworfen.

Weiter wird davon ausgegangen, dass die von *gAUDIT* verwendeten Multicast-Gruppen (IPv6 und Ethernet) von keiner anderen Anwendung verwendet werden. Werden trotzdem nicht protokollkonforme Nachrichten, beispielsweise mit anderem Nachrichtenformat, von anderen Anwendungen an die Multicast-Adressen versendet, so werden diese vom RRS erkannt und verworfen. Gleiches gilt für bereits bearbeitete Duplikate von Nachrichten. Für Details zur Absicherung gegen Angreifer siehe Abschnitt 4.7.

4.5 Details des Group Management Service (GMS)

Im nächsten Abschnitt wird der GMS des *gAUDIT*-Transportsystems im Detail erörtert. Dieser stellt einen Dienst (Punkt-zu-Punkt) für Server-Instanzen (bei *CloudInspector* die Monitoring-Agenten) zur Gruppenverwaltung bereit. Des Weiteren besteht eine direkte Abhängigkeit zwischen GMS und RRS, da auf Seiten des *Respondents* nur für zuvor per GMS beigetretenen *Gruppen*, Nachrichten des Typs $\mathbb{G}_{Req}$ empfangen und Monitoringdaten mittels Nachrichten des Typs $\mathbb{G}_{Res}$ versendet werden können. Als wesentliche Funktion realisiert der GMS einen Mechanismus zur Gruppenverwaltung, der die Mitgliedschaft in den *Gruppen* zwischen *Respondents* und *Initiator* synchronisiert.

Anhand Abbildung 4.9 wird der GMS zur Verwaltung der Mitgliedschaft in *Gruppen* vorgestellt. Der Dienstzugangspunkt des GMS bietet auf Seiten des *Respondents* die folgenden zwei Dienstprimitive an. Als Namensschema wird die X.210-Konvention [102] verwendet.

gAUDIT-G.REQ: Diese Dienstprimitiv erwarteten das Tupel `(Initiator, GID, JOIN/LEAVE)` als Eingabeparameter. Eine Server-Instanz kann sich hiermit bei dem angegebenen Initiator für die *Gruppe* mit der GID für den Empfang von Nachrichten des Typs $\mathbb{G}_{Req}$ registrieren (*JOIN*) oder deregistrieren (*LEAVE*). Bei Erfolg der Registrierung (Gruppenbeitritt) wird ein neuer Kommunikationsendpunkt für die Kennung

GID bei dem RRS geöffnet. Bei erfolgreicher Deregistrierung wird der Kommunikationsendpunkt mit der Kennung GID geschlossen.

gAUDIT-G.CNF: Als Bestätigung für das Dienstprimitiv `gAUDIT-G.REQ` verwendet *gAUDIT* das Dienstprimitiv *gAUDIT-G.CNF* um der Server-Instanz mit den Parametern *SUCCESS* oder *FAILURE* zu signalisieren, ob der angeforderte Verwaltungsvorgang (Beitritt oder Verlassen) erfolgreich abgeschlossen werden konnte.

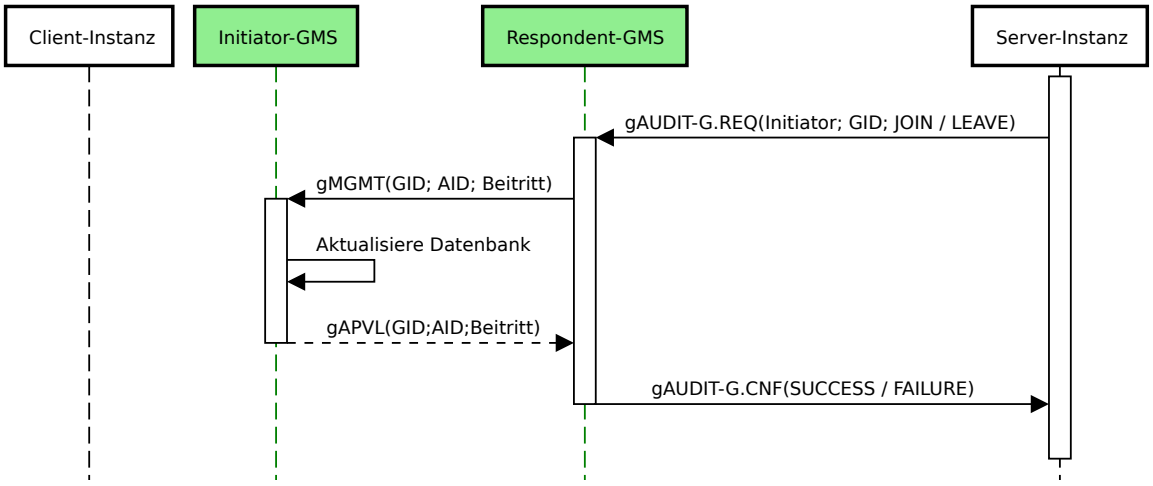

Abbildung 4.9: *Übersicht über den Dienstzugangspunkt und Nachrichten des Group Management Service.*

Auf Seiten einer Client-Instanz werden vom GMS keine Dienstprimitive angeboten. Intern greift der GMS auf die Nachrichten $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$ zurück, um die Anfrage einer Server-Instanz an den Initiator zu übermitteln. Ausführliche Details zur Protokollspezifikation und Fehlerkontrolle folgen in den nächsten Abschnitten.

4.5.1 Fehlerkontrolle des GMS

Die Fehlerkontrolle des GMS lässt sich in drei Teilbereiche untergliedern. Zum einen die Erkennung von Duplikaten, Phantom-Daten und Korrektheit von Nachrichten des Typs $\mathbb{G}_{Mgmt}$ zwischen *Respondent* und *Initiator*. Gleiches gilt für die Gegenrichtung für Nachrichten des Typs $\mathbb{G}_{Apvl}$ zwischen *Initiator* und *Respondent*. Hinzu kommt auf Seiten des *Initiators* die Verlustbehandlung. Da die Nachricht des Typs $\mathbb{G}_{Apvl}$ als implizite Bestätigung für den durch die Nachricht des Typs $\mathbb{G}_{Mgmt}$ ausgelösten Gruppenverwaltungsvorgang aufgefasst werden kann, wird kein zuverlässiger Dienst der Transportschicht benötigt. Vielmehr obliegt es der *gAUDIT*-Schicht die parametrisierte g1RRS-Klasse umzusetzen und

eventuellen Nachrichtenverlust zu behandeln. Darüber hinaus findet die Fehlerkontrolle dem Ende-zu-Ende-Argument [159] folgend im *Respondent* und *Initiator* statt.

Je Kommunikationsrichtung wird auf das Transportprotokoll *UDP* [237] zurückgegriffen (siehe Abschnitt 4.4.4). Wird wie bei *gAUDIT* das Transportprotokoll *UDP* zusammen mit Internetprotokoll *IPv6* eingesetzt, so ist es verpflichtend, dass der Sender für das *UDP*-Feld *Checksum* die sogenannte Internetprüfsumme [201] auf Basis des sogenannten *IPv6 Pseudo-headers* und Inhalt des *UDP-Segments* berechnet [242]. *IPv6* selbst enthält dann keine weitere Prüfsumme. Der Empfänger einer *gAUDIT*-Nachricht überprüft anhand des *UDP*-Felds *Checksum*, ob die Nachricht fehlerfrei übertragen wurde und ob das gewünschte Ziel laut dem Feld *Destination Address* aus dem IPv6-Kopf der Empfänger ist.

gAUDIT bietet mit dieser Variante nur minimale Funktionalität zur Erkennung von Fehlern bei Nachrichtenübertragung (vergleiche auch [217]). Allerdings ist bei dem Einsatz von *gAUDIT* innerhalb von Cloud-Umgebungen auch nicht mit hohen Fehlerraten bei der Übertragungen zu rechnen (vergleiche [198]). Reicht der durch die bei *UDP* eingesetzte 16 Bit Einerkomplement Prüfsumme zur Verfügung gestellte Schutz nicht aus, so muss auf Anwendungsebene zusätzlich mindestens eine 32 Bit CRC-Prüfsumme (vergleiche zum Beispiel [96]) zur Fehlererkennung eingesetzt werden [241, 217]. Hierfür kann das Transportsystem *gAUDIT* entsprechend um CRC-Prüfsummen in den Nachrichten erweitert werden.

Für die Erkennung von Duplikaten und Nachrichtenverlust kommt ein spezielles ARQ-Verfahren beim *Respondent* und *Initiator* zum Einsatz. Reihenfolgetreue spielt hier keine Rolle, da die Übertragung bereits nach einer Nachricht je Kommunikationsrichtung abgeschlossen ist. Abbildung 4.10 beschriebt das ARQ-Verfahren des GMS als Pseudo-Code.

Wird der GMS von einer Server-Instanz mit dem Beitritt oder dem Verlassen einer *Gruppe* beauftragt, so wählt die *gAUDIT*-Komponente *Respondent* zunächst die AID für die Nachricht zufällig – die AID dient in erster Linie dazu, Duplikate bei Sendewiederholungen zu erkennen. Die Wahl der AID muss dabei zufällig erfolgen, da die verwendeten AIDs nicht zwischen allen *Respondents* synchronisiert werden können. Die Lebensdauer einer gewählten AID wird hier [241] folgend auf zwei Minuten beschränkt. Innerhalb dieser Zeitspanne können Duplikate sicher erkannt werden, da die Wahrscheinlichkeit, dass zwei *Respondents* innerhalb eines Zeitfensters von zwei Minuten zufällig die gleiche *128* Bit AID wählen vernachlässigbar gering ist. Zur Abschätzung kann die in der Literatur bekannte *birthday bound* herangezogen werden (siehe zum Beispiel [70, 118]). Hiernach müssten für *128* Bit $\sqrt{2^{128}} \approx 1.8 \cdot 10^{19}$ zufällige AID-Werte innerhalb von zwei Minuten

```
Initiator:
Function EmpfangeGMSNachricht():
    if AID der Nachricht ist nicht im Zwischenspeicher then
        Sende Nachricht des Typs G_Apvl mit AID an anfragenden Respondent
        Aktualisiere Datenbank Member für GID
        Lege versendete Nachricht unter AID im Zwischenspeicher ab
    else
        Sende Nachricht des Typs G_Apvl mit AID erneut an anfragenden Respondent
    end
    Entferne alle Einträge aus Zwischenspeicher die älter als 2 Minuten sind
end

Respondent:
begin
    Wähle AID für Nachricht des Typs G_Mgmt zufällig
    N = 0
    for N < maximale Anzahl für Sendewiederholung do
        Sende Nachricht des Typs G_Mgmt mit AID und GID an den Initiator
        Zeitgeber.start()
        while Zeitgeber.active() do
            if Nachricht des Typs G_Apvl zu AID empfangen then
                return Gruppenverwaltungsvorgang erfolgreich an aufrufende
                 Server-Instanz
            end
            Zeitgeber.tick()
        end
        N=N+1
    end
    return Gruppenverwaltungsvorgang nicht erfolgreich an aufrufende
     Server-Instanz
end
```

Abbildung 4.10: *Pseudo-Code des ARQ-Verfahrens zur Gruppenverwaltung.*

gezogen werden, bevor eine Kollisionen wahrscheinlich ist. Für den *CloudInspector*-Anwendungsfall ist allerdings nicht mit einem so hohem Registrierungsaufkommen innerhalb eines zwei Minuten Zeitfensters zu rechnen.

Anschließend wird eine Nachricht des Typs $\mathbb{G}_{Mgmt}$ an die Komponente *Initiator* versendet. Zeitgleich wird ein Zeitgeber gestartet. Wird vor Ablauf des Zeitgebers die Antwort des *Initiators* in Form der Nachricht des Typs $\mathbb{G}_{Apvl}$ empfangen, so meldet der *Respondent* dies

an die aufrufende Server-Instanz zurück. Wird keine passende Nachricht des Typs $\mathbb{G}_{Apvl}$ vor Ablauf des Zeitgebers empfangen, so wird eine Sendewiederholung der Nachricht des Typs $\mathbb{G}_{Mgmt}$ durchgeführt und der Zeitgeber erneut gestartet. Dieser Vorgang wird so lange wiederholt, bis die konfigurierte Anzahl an maximalen Wiederholungen erreicht ist.

Auf Seiten des Initiators überwacht der GMS kontinuierlich, ob Nachrichten des Typs $\mathbb{G}_{Mgmt}$ eintreffen. Ist dies der Fall, so wird zunächst überprüft, ob zur angegebenen AID bereits eine Nachricht des Typs $\mathbb{G}_{Apvl}$ versendet wurde. Hierzu wird ein Zwischenspeicher eingesetzt, der versendete Nachrichten des GMS des Initiators kurzfristig zwischenspeichert (in der Regel zwei Minuten). Wurde bereits eine Nachricht des Typs $\mathbb{G}_{Apvl}$ zur eingehenden Nachricht des Typs $\mathbb{G}_{Mgmt}$ versendet, so wird diese aus dem Zwischenspeicher ausgelesen und erneut versendet. Findet sich im Zwischenspeicher keine passende Nachricht des Typs $\mathbb{G}_{Apvl}$, so wird der gewünschte Gruppenverwaltungsvorgang durchgeführt (ein neues Gruppenmitglied zur *Gruppe* hinzugefügt oder ein existierendes gelöscht). Anschließend wird die Nachricht des Typs $\mathbb{G}_{Apvl}$ an den anfragenden *Respondent* gesendet und für zwei Minuten im Zwischenspeicher vorgehalten. Als AID der Nachricht des Typs $\mathbb{G}_{Apvl}$ wird die gleiche AID der eingegangenen Nachricht des Typs $\mathbb{G}_{Mgmt}$ verwendet.

Anhand der Zustandsautomaten in den nächsten Abschnitten wird dieses ARQ-Verfahren detaillierter ausgeführt.

4.5.2 Zustandsautomat des Respondents

Anhand des endlichen Automaten in Abbildung 4.11 wird das Protokoll zur Gruppenverwaltung näher beschrieben. Dabei setzt der Zustandsautomat des *Respondents* die folgenden Anforderungen um.

G01 Management der Zugehörigkeit einer Server-Instanz zu einer *Gruppe*.

G02 Synchronisation von Gruppenmitgliedschaften zwischen *Respondent* und *Initiator*.

G03 Durchführen von Sendewiederholungen falls eine $\mathbb{G}_{Mgmt}$- oder $\mathbb{G}_{Apvl}$-Nachricht im Netz verloren geht.

G04 Rückmeldung an die Server-Instanz über den Erfolg oder Misserfolg bei Beitritt oder Verlassen einer *Gruppe*.

Intern verwendet das Protokoll des GMS hierzu die folgenden drei Dienstelemente: 1) `gRTO` als Zeitgeber in Sekunden nach dessen Ablauf eine Sendewiederholung erforderlich ist, 2) `gCNT` als Zähler für die Anzahl der bereits durchgeführten Sendewiederholungen und 3) `gMAX` für die maximale Anzahl von Sendewiederholungen. Sobald der GMS im

Zustand Ruhe per *JOIN* oder *LEAVE* von einer Server-Instanz aufgerufen wird, wird der Zustand `Warten-auf-Bestätigung` erreicht.

Beim Übergang in den Zustand `Warten-auf-Bestätigung` versendet der *Respondent* eine Nachricht des Typs $\mathbb{G}_{Mgmt}$ an den *Initiator*, startet den Zeitgeber `gRTO` und setzt den Zähler `gCNT` für die Anzahl der durchgeführten Sendewiederholung auf `0`. In der Nachricht des Typs $\mathbb{G}_{Mgmt}$ wird das Typ-Feld auf `2` und das Feld GID auf den in dem Dienstprimitiv angegebenen Parameter für GID gesetzt. Das Feld AID wird vom *Respondent* zufällig gewählt. Außerdem wird das Feld Beitritt entsprechend gesetzt – `0` bei *JOIN* und `1` bei *LEAVE*. Als IPv6-Zieladresse der Nachricht des Typs $\mathbb{G}_{Mgmt}$ wird die in dem Dienstprimitiv zum *Initiator* gehörende IPv6-Adresse verwendet.

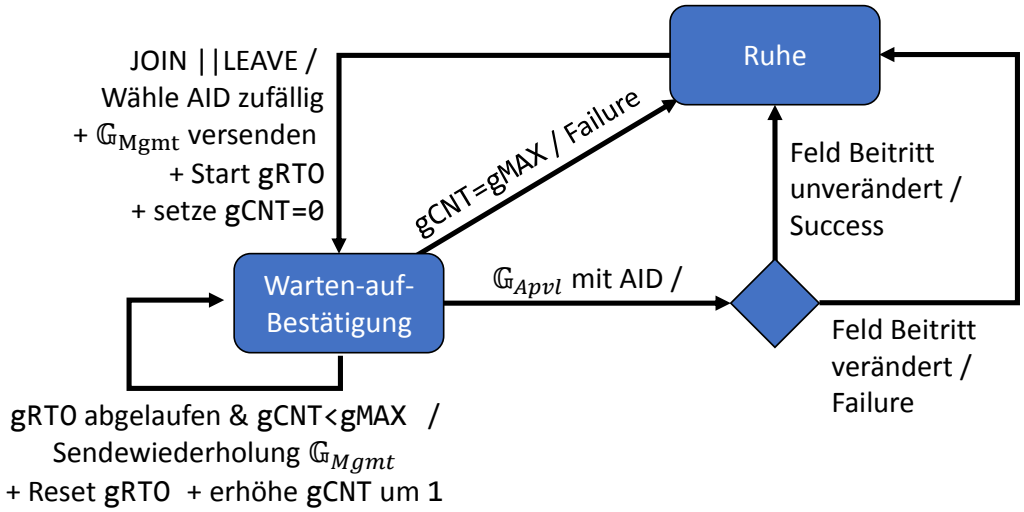

Abbildung 4.11: *Registrierung / Deregistrierung einer Gruppenmitgliedschaft aus Sicht eines Respondents.*

Im Zustand `Warten-auf-Bestätigung` wartet der *Respondent* auf den Empfang einer Nachricht des Typs $\mathbb{G}_{Apvl}$ von dem *Initiator* – hierzu wird die Wartezeit mit Zeitgeber `gRTO` zeitüberwacht. Der Zeitgeber `gRTO` wird dabei nach den Empfehlungen von [230, 241] auf 1 Sekunde eingestellt. In Abschnitt 4.5.4 wird geklärt, wann und wie die Nachricht des Typs $\mathbb{G}_{Apvl}$ von dem *Initiator* versendet wird. Läuft der `gRTO` ohne Empfang der Nachricht des Typs $\mathbb{G}_{Apvl}$ ab und ist die maximale Anzahl von Sendewiederholungen noch nicht erreicht (`gCNT<gMAX`), so wird von dem *Respondent* eine Sendewiederholung der Nachricht des Typs $\mathbb{G}_{Mgmt}$ durchgeführt. Hierbei wird der Zeitgeber `gRTO` neu gestartet und die Variable `gCNT` um 1 erhöht. Dieser Vorgang kann sich so lange wiederholen, bis die maximale Anzahl von Sendewiederholungen erreicht ist (`gCNT=gMAX`) oder die Nachricht des Typs $\mathbb{G}_{Apvl}$ eintrifft. Bei Erreichen der maximalen Anzahl von Sendewiederholungen

wird der Rückgabewert `FAILURE` an die Server-Instanz gemeldet. Zur genauen Wahl von `gMAX` siehe Abschnitt 4.9.

Trifft die Nachricht des Typs $\mathbb{G}_{Apvl}$ mit passender AID vor Erreichen der maximalen Anzahl an Sendewiederholungen ein, so wird überprüft, ob das Feld Beitritt unverändert ist. Entspricht das Feld Beitritt nicht mehr dem ursprünglich in der Nachricht des Typs $\mathbb{G}_{Mgmt}$ gesetzten Wert – wird der Rückgabewert `FAILURE` an die Server-Instanz gemeldet. Durch Verändern des Beitritt-Felds kann der *Initiator* einen Beitritt oder das Verlassen der *Gruppe* ablehnen. Im anderen Fall – das Feld Beitritt entspricht dem in der Nachricht des Typs $\mathbb{G}_{Mgmt}$ gesetzten Wert – wird der Rückgabewert `Success` an die Server-Instanz gemeldet und der zur GID gehörende Kommunikationsendpunkt am RRS geöffnet (*JOIN*) oder geschlossen (*LEAVE*).

4.5.3 Transportsystem des Respondents

Neben dem Ablauf zur Gruppenverwaltung innerhalb des *Respondents*, müssen von *gAUDIT* die Konfigurationen unterliegender Schichten (Transport-, Internet- und Netzzugangsschicht) vorgenommen werden, so dass Gruppenkommunikation für Request-Response-Kommunikation des RRS ermöglicht wird (vergleiche Abschnitt 4.4.4). Die Konfiguration von unterliegenden Schichten ist insbesondere notwendig, wenn das Dienstprimitiv `gAUDIT-G.CNF` den Erfolg (*SUCCESS*) des Verwaltungsvorgangs an die Server-Instanz meldet, da ansonsten per IPv6-Multicast versendete Nachrichten nicht empfangen werden können. Im Folgenden wird dargelegt, welche Änderungen erforderlich sind.

Konfiguration bei *JOIN*: Wurde mit dem Dienstprimitiv `gAUDIT-G.REQ` der Beitritt zu einer *Gruppe* veranlasst, so konfiguriert der *Respondent* in der Internetschicht die zur *Gruppe* gehörende IPv6-Multicast-Adresse [234, 220]. Hierzu wird die in Abschnitt 4.4.4 dargelegte Abbildung von GID auf IPv6-Multicast-Adresse benutzt. Auf diese Art und Weise können die per Multicast versendeten Nachrichten des Typs $\mathbb{G}_{Req}$ vom Respondent empfangen werden.

Konfiguration bei *LEAVE*: Wurde mit dem Dienstprimitiv `gAUDIT-G.REQ` das Verlassen einer *Gruppe* veranlasst, so konfiguriert der *Respondent* die Internetschicht, so dass die zur GID gehörende IPv6-Multicast-Adresse der *Gruppe* nicht mehr verwendet wird. Hierzu wird die in Abschnitt 4.4.4 dargelegte Abbildung von GID auf IPv6-Adresse benutzt.

Im Falle von *JOIN* wird die zur IPv6-Multicast-Adresse gehörende Ethernet-Multicast-Adresse eingetragen, im Falle von *LEAVE* entfernt (vergleiche Abschnitt 4.4.4). Außerdem versendet der Netztreiber anschließend einen *Membership-Report* an den Router im Netz, um die Änderung in der Internet- und Netzzugangsschicht im Netz bekannt zu geben

(vergleiche [222]). Im Fall von *JOIN* wird *JOIN* und im Fall von *LEAVE* wird *LEAVE* im *Membership-Report* versendet. Auf diese Art und Weise kann das Netz die Nachrichten des Typs $\mathbb{G}_{Req}$ an den entsprechenden Stellen duplizieren (Effizienz) und den zur *Gruppe* gehörenden *Respondents* zustellen (siehe auch Abschnitt 4.6).

4.5.4 Zustandsautomat des Initiators

Anhand des endlichen Automaten in Abbildung 4.12 wird das Protokoll zur Gruppenverwaltung aus Sicht des *Initiators* näher beschrieben. Dabei setzt der Zustandsautomat die folgenden Anforderungen um.

G05 Synchronisation der Zugehörigkeit zu einer *Gruppe* zwischen *Respondent* und *Initiator*.

G06 Aktualisierung der Datenbank *Member* bezüglich der Mitgliedschaft von *Respondents* in *Gruppen*.

G07 Ablehnen oder Bestätigen der Nachricht des Typs $\mathbb{G}_{Mgmt}$ eines *Respondents*.

Bei Empfang der Nachricht des Typs $\mathbb{G}_{Mgmt}$ prüft der *Initiator* zunächst, ob dem Wunsch des *Respondents* entsprochen werden kann – dem Beitritt oder dem Verlassen einer *Gruppe*. *gAUDIT* überlässt es dabei der Client-Instanz bei Nutzung des RRS zu entscheiden, ob Änderungen an der Gruppenzusammensetzung während laufender Request-Response-Kommunikation zulässig sind oder nicht. Mehr Details hierzu finden sich bei der Beschreibung des RRS in Abschnitt 4.6. Soll der Gruppenverwaltungsvorgang abgelehnt werden, so kippt der *Initiator* das in dem Feld Beitritt enthaltene Bit für die Nachricht des Typs $\mathbb{G}_{Apvl}$. Wird der Gruppenverwaltungsvorgang angenommen, so nimmt der *Initiator* keine Änderung an dem Feld Beitritt vor.

Wird die Änderung der Gruppenmitgliedschaft akzeptiert, so führt der *Initiator* eine Fallunterscheidung durch. Dabei wird die entsprechende *Gruppe* über das in der Nachricht des Typs $\mathbb{G}_{Mgmt}$ angegebene Feld GID identifiziert und der *Respondent* über die verwendete IPv6-Absenderadresse der Nachricht. Im Fall des Beitritts eines *Respondents* zu einer *Gruppe* (das Feld Beitritt ist auf den Wert `0` gesetzt), fügt der *Initiator* in der Datenbank *Member* einen entsprechenden Eintrag für den *Respondent* hinzu. Im Fall, dass die angegebene *Gruppe* verlassen wird (das Feld Beitritt ist auf den Wert `1` gesetzt), entfernt der *Initiator* in der Datenbank *Member* den entsprechenden Eintrag für den *Respondent*. Bei Akzeptanz der Änderung der Gruppenmitgliedschaft wird das in dem Feld Beitritt enthaltene Bit für die Nachricht des Typs $\mathbb{G}_{Apvl}$ nicht gekippt.

Hiernach wird die Nachricht des Typs $\mathbb{G}_{Apvl}$ an den *Respondent* gesendet. Dabei wird das Feld Beitritt auf den in der Fallunterscheidung ermittelten Wert gesetzt. Das Feld AID

wird dabei auf den in der Nachricht des Typs $\mathbb{G}_{Mgmt}$ verwendeten Werts des Feld AID gesetzt, so dass der *Respondent* die Antwort zur Anfrage zuordnen kann.

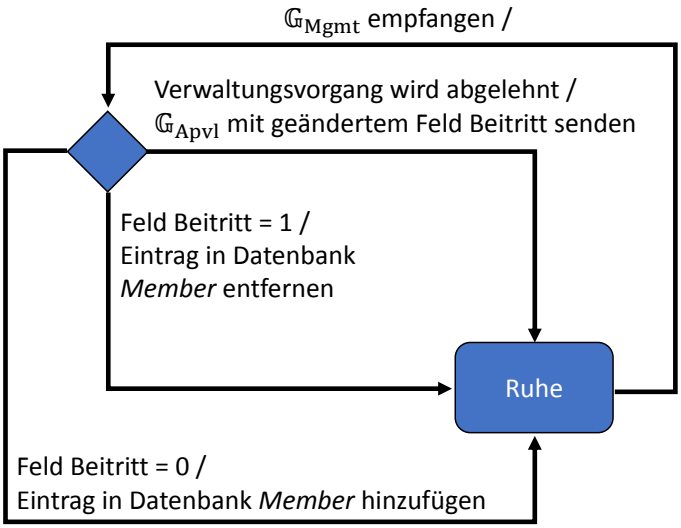

Abbildung 4.12: *Beitritt / Verlassen einer Gruppe aus Sicht des Initiators.*

Ein etwaiger Verlust einer Nachricht des Typs $\mathbb{G}_{Mgmt}$ beim Transport zum *Initiator* oder einer Nachricht des Typs $\mathbb{G}_{Apvl}$ wird durch das eingesetzte ARQ-Verfahren abgefangen.

4.6 Details des Request-Response Service (RRS)

In diesem Unterabschnitt wird der RRS des *gAUDIT*-Transportsystems im Detail erörtert. Dabei wird der RRS von einer Client-Instanz genutzt um Monitoringdaten von einer Gruppe von Server-Instanzen mittels Request-Response-Kommunikation abzufragen (bei *CloudInspector* der Monitoring-Controller für Kommunikation mit den Monitoring-Agenten).

Wie in Abschnitt 4.2 erarbeitet, soll der RRS zwei verschiedene g1RRS-Klassen und die Parametrisierung von Response-Vollständigkeit erlauben. Einerseits wird die g1RRS-Klasse *Statistisch-zuverlässig* mit dem Parameter a zur Angabe der Auswahlwahrscheinlichkeit je Gruppenmitglied der adressierten *Gruppe* vom RRS unterstützt. Hieraus soll der Initiator die Response-Vollständigkeit einer Anfrage deterministisch errechnen können. Anderseits unterstüzt der RRS die g1RRS-Klasse *k-zuverlässig* mit dem Parameter k zur Angabe der erwarteten Response-Nachrichten. In diesem Fall entspricht die Response-Vollständigkeit genau dem Parameter k.

Aufgabe des RRS ist es nun, die Gruppenmitglieder mit der Nachricht des Typs $\mathbb{G}_{Req}$ zu versorgen und der Response-Vollständigkeit entsprechend viele Antworten der Gruppenmitglieder in Form der Nachricht des Typs $\mathbb{G}_{Res}$ einzusammeln. Wesentlicher Schwerpunkt ist hier, neben der Unterstützung beider g1RRS-Klassen, die jeweils effiziente Realisierung im Bezug auf über das Netz versendete Nachrichten in Gegenwart von Nachrichtenverlust. Hierfür kommt ein spezielles ARQ-Verfahren zum Einsatz, welches entscheidet, ob Request-Nachrichten per Multicast und Unicast versendet werden.

Anhand Abbildung 4.13 wird der RRS weiter vorgestellt. Hierzu bietet der Dienstzugangspunkt des RRS die folgenden Dienstprimitive und Parameter an.

Zunächst wird der Dienstzugangspunkt auf Seiten des *Initiators* beschrieben. Als Namensschema wird dabei die X.210-Konvention [102] verwendet.

gAUDIT-R.REQ: Dieses Dienstprimitiv wird von einer Client-Instanz zum Aufrufen des RRS genutzt. Als Eingabeparameter wird das Tupel `(GID, Anfrage, g1RRS-Parameter)` erwartet. Eine Client-Instanz kann hiermit eine Anfrage an alle bei der *Gruppe* GID registrierten Server-Instanzen versenden und Parameter für die erwartete Response-Vollständigkeit für die einzusammelnden Monitoringdaten mit angeben (siehe Abschnitt 4.6.1). Die *Gruppe* wird dabei über die angegebene GID eindeutig identifiziert. Nach Nutzung des Dienstprimitivs wird ein neuer Kommunikationsendpunkt unter dem Bezeichner GID geöffnet, über den die eingegangen Monitoringdaten von der Client-Instanz empfangen werden können (in der Abbildung 4.13 durch den Zylinder mit `X` gekennzeichnet, wobei diese für den Kommunikationsendpunkt für die *Gruppe* mit der GID `X` steht).

gAUDIT-R.CNF: Dieses Dienstprimitiv wird vom *Initiator* verwendet um den Abschluss des Request-Response-Mechanismus an die aufrufende Client-Instanz zu signalisieren. Bei Erfolg der Abfrage werden alle empfangen Monitoringdaten an die Client-Instanz auf einmal übergeben. Dazu wird der zur *Gruppe* gehörende Kommunikationsendpunkt genutzt. Werden mehrere Anfragen für die gleiche *Gruppe* von der Client-Instanz parallel durchgeführt, so werden unterschiedliche Kommunikationsendpunkte verwendet. Je AID wird dabei ein separater Kommunikationsendpunkt verwendet. Mit diesem Dienstprimitiv wird zusätzlich ein Parameter an die aufrufende Client-Instanz übermittelt, der Aufschluss über den Erfolg des vom RRS durchgeführten Request-Response-Mechanismus gibt. Bei Empfang der letzten erforderlichen Nachricht des Typs $\mathbb{G}_{Res}$, die erforderlich war, um das parameterisierte Ziel für Response-Vollständigkeit zu erreichen, wird *Success* an den Initiator zurückgemeldet. Ist dies nicht der Fall und die angeforderte *Response-Vollständigkeit* kann von *gAUDIT* nicht erreicht werden, so wird *FAILURE* an die Client-Instanz zurückgegeben. In diesem Fall werden allerdings keine Monitoringdaten an die Client-Instanz ausgeliefert. In beiden Fällen wird der

zugehörige Kommunikationsendpunkt geschlossen – in dem abgebildeten Beispiel wäre dies der Kommunikationsendpunkt für die *Gruppe* mit der GID `X`.

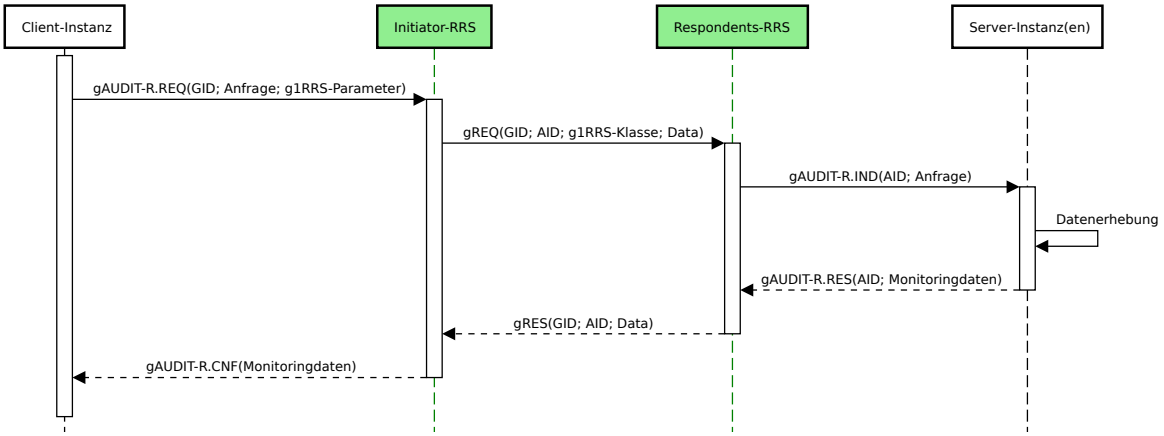

Abbildung 4.13: *Übersicht über die Dienstzugangspunkte und Nachrichten des Request-Response Service.*

Folgende Dienstprimitive stehen am Dienstzugangspunkt auf Seiten des *Respondents* für Server-Instanzen zur Verfügung.

gAUDIT-R.IND: Mit diesem Dienstprimitiv übergibt der *Respondent* in Nachrichten des Typs $\mathbb{G}_{Req}$ eingehende Anfragen an die Server-Instanz. Hierzu wird der per GMS geöffnete und zur *Gruppe* gehörende Kommunikationsendpunkt genutzt. Zusätzlich wird die zur Nachricht gehörende Anfrage-ID (AID) mit an die Server-Instanz übermittelt, so dass Antworten der Server-Instanz sich mit Anfragen verknüpfen lassen. Das Dienstprimitiv verwendet daher das Tupel `(AID, Anfrage)` als Parameter.

gAUDIT-R.RES: Mit diesem Dienstprimitiv sendet eine Server-Instanz angefragte Monitoringdaten an die Client-Instanz. Hierzu muss die bei der Anfrage verwendete AID mit angegeben werden, sodass Antworten der Server-Instanz sich mit Anfragen verknüpfen lassen. Dabei verlässt sich der *Respondent* darauf, dass die Server-Instanz die AID für eine Antwort passend zur Anfrage wählt (sprich unverändert zurück an den *Respondent* übergibt). Genutzt wird der per GMS geöffnete und zur *Gruppe* gehörende Kommunikationsendpunkt. Als Eingabeparameter für das Dienstprimitiv ergibt sich so das Tupel `(AID, Monitoringdaten)`.

Intern greift der RRS des *gAUDIT*-Transportsystems auf die Nachrichten $\mathbb{G}_{Req}$ und $\mathbb{G}_{Res}$ zurück, um die Anfrage einer Client-Instanz an die *Respondents* zu übermitteln. Ausführliche Details zur Protokollspezifikation folgen in den nächsten Abschnitten.

4.6.1 g1RRS-Parameter

gAUDIT erlaubt der Client-Instanz eine g1RRS-Klasse zu wählen und Parameter für die Durchführung der Request-Response-Kommunikation mit anzugeben. Hierzu wird das Dienstprimitiv `gAUDIT-R.REQ` am Dienstzugangspunkt des *Initiators* verwendet. Im Folgenden wird ausgeführt, welche Parameter von *gAUDIT* unterstützt werden.

RClass: Mit Hilfe dieses Parameters wählt eine Client-Instanz eine von zwei g1RRS-Klassen aus und spezifiziert die angeforderte Response-Vollständigkeit. Zur Auswahl stehen 1) *Statistisch-zuverlässig* mit Auswahlwahrscheinlichkeit a oder 2) *k-zuverlässig*, genau die Anzahl k an erwarteten Antworten. Auf Basis dieses Dienstelements und der Datenbank *Member* werden dann vom *Initiator* bei ausbleibenden Response-Nachrichten gegebenenfalls Sendewiederholungen von Nachrichten des Typs $\mathbb{G}_{Req}$ durchgeführt.

MaxGap: Legt die maximale Dauer für die Durchführung einer Request-Response-Kommunikation durch *gAUDIT* fest. So wird sichergestellt, dass die Ausführung von *gAUDIT* in jedem Fall nach der per `MaxGap` eingestellten Dauer terminiert.

MergeSw: Regelt, ob Änderungen an der Zusammensetzung der zur Request-Response-Kommunikation gehörenden *Gruppe* in die laufende Abarbeitung des Request-Response-Mechanismus vom *Initiator* mit einbezogen werden sollen. Sollen Änderungen mit einbezogen werden, so wird dies für `RClass` berücksichtigt. Wenn nicht, wird die Änderung der Zusammensetzung der *Gruppe* vom GMS bis zum Abschluss der Request-Response-Kommunikation abgelehnt oder ein Fehler gemeldet.

Die Parameter müssen passend zum jeweiligen Monitoring-Anwendungsfall gewählt werden.

4.6.2 Umsetzung der Response-Vollständigkeit

Bevor in den folgenden Abschnitten die Fehlerkontrolle des RRS und die Zustandsautomaten der Komponenten besprochen werden, wird an dieser Stelle zunächst auf Details zur Berechnung von Response-Vollständigkeit durch den RRS eingegangen. Die Client-Instanz wählt bei Verwendung des RRS durch Angabe der g1RRS-Parameter aus, welche g1RRS-Klasse vom Dienst umgesetzt werden soll. Entweder *Statistisch-zuverlässig* oder *k-zuverlässig*. Für beide Klassen wird vom Initiator die Response-Vollständigkeit ermittelt, die durch den RRS angestrebt wird. Diese Information wird unter anderem auch für das zum Einsatz kommende ARQ-Verfahren benötigt, so dass entschieden werden kann, von wie vielen Gruppenmitgliedern eine Response-Nachricht erwartet wird. Außerdem

benötigt das ARQ-Verfahren Angaben zur erwarteten Response-Vollständigkeit um zu entschieden, ob der Versand der Request-Nachricht per Multicast effizienter als per Unicast ist.

Der *Initiator* signalisiert dabei den adressierten *Respondents* über das in der Nachricht des Typs $\mathbb{G}_{Req}$ enthaltene Feld `g1RRS-Klasse`, ob die Klasse *Statistisch-zuverlässig* (Wert `0`) oder *k-zuverlässig* (Wert `1`) von der Client-Instanz ausgewählt wurde.

4.6.2.1 Response-Vollständigkeit für die Klasse Statistisch-zuverlässig

Für diese g1RRS-Klasse gibt die Client-Instanz als g1RRS-Parameter an, welche Auswahlwahrscheinlichkeit a für jedes Gruppenmitglied gewählt werden soll.

Zur Berechnung der erwarteten *Response-Vollständigkeit* ruft der Initiator zunächst alle Gruppenmitglieder der adressierten *Gruppe* aus der Datenbank *Member* ab. Insbesondere die IPv6-Adressen der Gruppenmitglieder werden für die Berechnung benötigt. Anschließend berechnet der Initiator mit Hilfe eines Zufallszahlengenerators `PRNG` für jedes Gruppenmitglied, ob eine Antwort erwartet wird. Hierzu erhält der Zufallszahlengenerator `PRNG` als Eingabe die AID der Anfrage und die IPv6-Adresse des jeweiligen Gruppenmitglieds. Als Ergebnis zieht der Zufallszahlengenerator einen gleichverteilten Wert aus dem Intervall `[0;100]`. Ist der zufällig gezogene Wert $\leq a$ so wird eine Antwort von dem *Respondent* erwartet. Auf diese Art und Weise lässt sich durch den *Initiator* die Response-Vollständigkeit ermitteln.

Auf Seiten der adressierten *Respondents* erfolgt eine analoge Berechnung, ob der *Respondent* die Anfrage an die zugehörige Server-Instanz weiterreichen soll. Hierzu wird der gleiche Zufallszahlengenerator `PRNG` wie beim Initiator verwendet. Liegt auf Seiten eines *Respondents* der zufällig gezogene Wert $\leq a$, so wird die Anfrage an die zugehörige Server-Instanz weitergereicht. Ist dies nicht der Fall, so wird die Nachricht des Typs $\mathbb{G}_{Req}$ verworfen.

4.6.2.2 Response-Vollständigkeit für die Klasse k-zuverlässig

Für diese g1RRS-Klasse gibt die Client-Instanz als g1RRS-Parameter direkt an, von wie vielen Gruppenmitgliedern eine Response-Nachricht erwartet wird.

Hierdurch ergibt sich die *Response-Vollständigkeit* direkt aus dem angegebenen Parameter k. Lediglich muss der Initiator sicherstellen, dass der gewählte Parameter $k \leq$ der Anzahl aller zur *Gruppe* gehörenden Mitglieder ist. Andernfalls ist die anvisierte *Response-Vollständigkeit* trivialerweise nie zu erreichen.

Die *Respondents* reichen Nachrichten des Typs $\mathbb{G}_{Req}$, die zur g1RRS-Klasse *k-zuverlässig* gehören, direkt an die entsprechenden Server-Instanzen weiter. Dabei muss der Dienstbenutzer von *gAUDIT* darauf achten, dass in der per *gAUDIT* von der Client-Instanz an Server-Instanzen transportierten Anfrage für die Server-Instanzen ersichtlich wird, dass nur im "positiven" Fall eine Antwort benötigt wird. Wird beispielsweise bei *CloudInspector* die Lokalität einer einzelnen VM abgefragt, so muss für die Monitoring-Agenten ersichtlich werden, dass nur die Antwort des Monitoring-Agenten benötigt wird, auf dem die VM derzeit ausgeführt wird – alle anderen Monitoring-Agenten müssen keine Antwort zurücksenden.

4.6.3 Request-Nachrichten per Multicast oder Unicast

Ein weiterer Bestandteil von *gAUDIT* ist die Entscheidung, Nachrichten des Typs $\mathbb{G}_{Req}$ per Multicast oder Unicast zu versenden. Zur Entscheidungsfindung wird durch *gAUDIT* ermittelt, welche Variante die anvisierte *Response-Vollständigkeit* effizienter erreichen kann.

Wird beispielsweise im Fall der g1RRS-Klasse *Statistisch-zuverlässig* nur von einem Gruppenmitglied bei *1.000* registrierten Gruppenmitgliedern eine Antwort benötigt, so ist die Wahl von Unicast für den Versand der Nachricht des Typs $\mathbb{G}_{Req}$ sinnvoll. Wird allerdings von jedem Gruppenmitglied eine Nachricht benötigt ($a = 1$), so bietet sich der Versand mittels Multicast an.

Während für die g1RRS-Klasse *Statistisch-zuverlässig* bereits für die erste Request-Nachricht entschieden werden kann, ob der Versand per Multicast effizenter als per Unicast ist, ist dies für die g1RRS-Klasse *k-zuverlässig* nicht möglich. Hier ist zum Zeitpunkt der Anfrage nicht geklärt, welches Gruppenmitglied antworten muss, sondern lediglich die Anzahl der erwarteten Antworten ist bekannt. Allerdings kann auch hier die Entscheidung zwischen Multicast und Unicast sinnvoll sein, insbesondere dann, wenn Sendewiederholungen durchgeführt werden müssen. Beispielsweise hat eine *Gruppe 100* Mitglieder wobei $k = 99$ Antworten von Gruppenmitgliedern von der Client-Instanz erwartet werden. Sind nun schon *98* Antworten eingetroffen, so scheint es für Sendewiederholungen sinnvoll, die restlichen zwei verbleibenden Gruppenmitglieder (die, die bisher noch keine Antwort an den Initiator gesendet haben) per Unicast zu adressieren. Würde in diesem Beispiel Multicast für die Sendewiederholung verwendet, so würden alle *100* Gruppenmitglieder erneut die Nachricht des Typs $\mathbb{G}_{Req}$ erhalten und so auch erneut alle bereits erhaltenen Antworten übertragen, was die Ressourcen des Netzes sicher nicht effizient ausnutzt.

Die genaue Untersuchung, in welchen Szenarien der Versand via Unicast Vorteile gegenüber Multicast hat, ist Gegenstand der Evaluation (siehe Abschnitt 4.9). Zunächst werden allerdings die Interna des RRS weiter ausgeführt.

4.6.4 Zeitgeber für den RRS

Essentieller Bestandteil des speziellen ARQ-Verfahrens für bidirektionale Request-Response-Kommunikation in Gruppen sind die Zeitgeber `RTO` und `Backoff`. Der Zeitgeber `RTO` bestimmt dabei, ab wann erwartete Response-Nachrichten auf Seiten des *Initiators* als verloren gelten und daher Sendewiederholungen der Nachricht des Typs $\mathbb{G}_{Req}$ erforderlich sind. Der Zeitgeber `Backoff` hingegen regelt auf Seiten der *Respondents*, wann bei Empfang einer bereits beantworteten Nachricht des Typs $\mathbb{G}_{Req}$ (identifiziert anhand der AID) eine erneute Übertragung der Nachricht des Typs $\mathbb{G}_{Res}$ durchgeführt werden soll.

Für die Wahl des Zeitgebers `Backoff` kommt ein Backoff-Algorithmus ähnlich wie in [93] zum Einsatz. Der Zeitgeber `RTO` hingegen wird aus der gemessenen Umlaufzeit zwischen Initiator und Respondents als Exponentially Weighted Moving Average (EWMA) berechnet.

In Kombination bestimmen diese beiden Größen maßgeblich die Dauer, bis *gAUDIT* in Szenarien mit Nachrichtenverlust die anvisierte *Response-Vollständigkeit* erreichen kann. Werden beispielsweise größere Zeitgeber verwendet und muss eine verlorene Nachricht erneut übertragen werden, so erhöht sich automatisch die Dauer bis der Verlust der Nachricht durch den Initiator erkannt wird (Zeitgeber `RTO`). Folglich erhöht sich die Dauer bis der RRS die Kommunikation erfolgreich abschließen kann.

4.6.4.1 Zeitgeber für Sendewiederholungen (RTO)

Der RRS misst kontinuierlich die Umlaufzeit zwischen *Initiator* und *Respondents*, in dem die Zeit zwischen Versand von Nachrichten des Typs $\mathbb{G}_{Req}$ und dem jeweiligen Empfang von Nachrichten des Typs $\mathbb{G}_{Res}$ ermittelt wird. Der *Initiator* legt so für jeden *Respondent* die ermittelten Werte in der Datenbank *Member* ab. Dort werden diese als Zeitgeber $\texttt{RTO}_x$ gepflegt, wobei x den jeweiligen *Respondent* eindeutig identifiziert. Für Multicast-Anfragen wird dann jeweils der größte Zeitgeber $\texttt{RTO}_{\texttt{max}}$ aller adressierten Gruppenmitglieder verwendet, um zu entscheiden, wann Sendewiederholungen erforderlich sind. Für Unicast-Anfragen werden jeweils die individuellen Zeitgeber $\texttt{RTO}_x$ verwendet.

Die Berechnung mittels EWMA stellt ein Filter mit unendlicher Impulsantwort dar, der den exponentiellen Mittelwert der Umlaufzeit zwischen *Initiator* und *Respondent* ermittelt, wobei jüngere Datenpunkte stärker gewichtet werden. Die Berechnung des EWMA kann als folgende rekursive Funktion angegeben werden, wobei U_i die aktuell gemessene Umlaufzeit darstellt und der Faktor α darüber entscheidet, wie stark neue Messergebnisse gewichtet werden.

$$\texttt{EWMA}(U_i) = \alpha \cdot U_i + (1-\alpha) \cdot \texttt{EWMA}(U_{i-1})$$

Für die Zeitgeber ergibt sich somit $\texttt{RTO}_x = \texttt{EWMA}(U_x)$ wobei U_x die aktuelle, zwischen *Initiator* und *Respondent x* gemessene Umlaufzeit darstellt.

Welcher Wert für α bei *gAUDIT* sinnvoll erscheint, wird in Abschnitt 4.9 näher untersucht.

4.6.4.2 Zeitgeber für Backoff-Algorithmus (Backoff)

Auf Seiten von *Respondents* wird pro versendeter Nachricht des Typs $\mathbb{G}_{Res}$ mit Hilfe eines Backoff-Algorithmus eine zufällig Verzögerung gewählt. Der gewählte Wert wird dann in dem Zeitgeber `Backoff` gespeichert, nach dessen Ablauf die Nachricht des Typs $\mathbb{G}_{Res}$ an den Initiator versendet wird. Durch die zufällige Wahl der Verzögerung auf Seiten von *Respondents* sollen Incast-Situationen auf Seiten des Initiators bei Empfang von Nachrichten des Typs $\mathbb{G}_{Res}$ vermieden werden – diese treffen dann je nach der von *Respondents* gewählten Verzögerung versetzt beim *Initiator* ein.

Der Backoff-Algorithmus wählt für den Zeitgeber `Backoff` zufällig einen Wert aus dem folgenden Intervall.

$$\texttt{Backoff} \in [0; s \cdot 2^{c}\texttt{ms}]$$

Über die Variable `c` wird angegeben, wie viele Übertragungen der Nachricht des Typs $\mathbb{G}_{Res}$ vom *Respondent* bereits veranlasst wurden – für die erste Nachricht steht die Variable `c` auf dem Wert `0` und für die erste Sendewiederholung auf dem Wert `1`. Die Konstante `s` bestimmt den Startwert und ist bei *CloudInspector* standardmäßig auf `1` eingestellt. Dadurch wird der Zeitgeber `Backoff` mindestens aus dem Intervall `[0; 1ms]` zufällig gewählt. Für andere Szenarien, insbesondere für solche, bei denen *gAUDIT* nicht innerhalb von Datenzentren eingesetzt wird, bei denen erfahrungsgemäß mit niedrigen Latenzen zu rechnen ist, muss der Startwert entsprechend angepasst werden.

Durch die Erhöhung der Variablen `c` je erneutem Sendeversuch der Nachricht des Typs $\mathbb{G}_{Res}$, vergrößert sich automatisch das Intervall, aus dem der Backoff-Algorithmus den

Wert für den Zeitgeber `Backoff` wählt. Falls eine Nachricht des Typ $\mathbb{G}_{Res}$, aufgrund einer durch zu viele Nachrichten des Typs $\mathbb{G}_{Res}$ verursachten Überlastsituation verloren geht, soll der bei Sendewiederholungen eingesetzte Backoff-Algorithmus das erneute Auftreten der Überlastsituation verhindern.

Damit der Backoff-Algorithmus unterscheiden kann, ob es sich bei einer zu versendenden Nachricht des Typ $\mathbb{G}_{Res}$ um eine Sendewiederholung handelt oder um die erste Nachricht, wird ein Zwischenspeicher verwendet. In diesem Zwischenspeicher werden alle empfangenen Nachrichten des Typs $\mathbb{G}_{Req}$ und alle versendeten Nachrichten des Typs $\mathbb{G}_{Res}$ für zwei Minuten vorgehalten. Der Wert von zwei Minuten für den Zwischenspeicher wurde gewählt, da dies die maximale mögliche Dauer für die Durchführung von Request-Response bei *gAUDIT* darstellt und an die Empfehlung aus [241] angelehnt ist. Alle Einträge, die älter als zwei Minuten sind, werden von dem Backoff-Algorithmus aus dem Zwischenspeicher gelöscht.

Trifft nun eine Nachricht des Typs $\mathbb{G}_{Req}$ erneut ein (ein Eintrag ist bereits im Zwischenspeicher vorhanden; eindeutig identifiziert anhand der AID und GID der Nachricht), dann wird die Variable `c` um eins erhöht, bevor die im Zwischenspeicher abgelegte Nachricht des Typs $\mathbb{G}_{Res}$ erneut vom *Respondent* an den *Initiator* versendet wird. Weiterhin wird der Wert der Variablen `c` bei dem Eintrag für die jeweilige Nachricht des Typs $\mathbb{G}_{Req}$ hinterlegt.

Zur näheren Analyse zu Auswirkungen des Zeitgebers `Backoff` siehe Abschnitt 4.9.

4.6.5 Fehlerkontrolle des RRS

Die Fehlerkontrolle des RRS lässt sich in drei Teilbereiche untergliedern: 1) Erkennung von Phantom-Daten und Korrektheit für die Übertragung von Nachrichten des Typs $\mathbb{G}_{Req}$ vom *Initiator* an Gruppenmitglieder (*Respondents*), 2) Erkennung von Phantom-Daten und Korrektheit für die Übertragung von Nachrichten des Typs $\mathbb{G}_{Res}$ von *Respondents* an den *Initiator* und 3) Verlustbehandlung zur Realisierung der parametrisierten g1RRS-Klasse.

Für die Erkennung von Phantom-Daten und Korrektheit wird vom RRS wie bei der Fehlerkontrolle des GMS (siehe Abschnitt 4.5.1) auf das eingesetzte Transportprotokoll *UDP* zurückgegriffen. Dabei wird, aufgrund der in Cloud-Infrastrukturen zu erwartenden niedrigen Fehlerrate, auf zusätzliche CRC-Prüfsummen in *gAUDIT*-Nachrichten verzichtet. Wird *gAUDIT* für andere Szenarien eingesetzt, so sollten die Nachrichten um entsprechende Prüfsummen erweitert werden. Für die Verlustbehandlung dienen die Nachrichten des Typs $\mathbb{G}_{Res}$ als implizite Bestätigung für die fehlerfreie Übertragung der Nachricht des Typs $\mathbb{G}_{Req}$ an das jeweilige Gruppenmitglied.

Für die Erkennung von Duplikaten und Nachrichtenverlust kommt das folgende bidirektionale ARQ-Verfahren beim *Respondent* und *Initiator* zum Einsatz. Reihenfolgetreue spielt hier keine Rolle, da die Kommunikation bereits mit der Übertragung einer Nachricht je Kommunikationsrichtung abgeschlossen ist.

Auf Seiten von *Respondents* kommt der in Abbildung 4.14 beschriebene Pseudo-Code zum Einsatz. Zunächst wartet der *Respondent* auf eintreffende Nachrichten des Typs $\mathbb{G}_{Req}$. Sobald eine solche Nachricht empfangen wurde, wird überprüft, ob der *Respondent* Mitglied in der per GID angegebenen *Gruppe* ist und ob eine Antwort vom *Respondent* erwartet wird. Ob eine Antwort erforderlich ist, hängt von der gewählten g1RRS-Klasse und der verwendeten *Response-Vollständigkeit* ab. Beispielsweise für die g1RRS-Klasse *Statistisch-zuverlässig*, ob der *Respondent* mit zur angeforderten Stichprobe gehört.

```
Function EmpfangeRRSNachricht():
    Entferne alle Einträge aus Zwischenspeicher die älter als zwei Minuten sind
    if Respondent ist Mitglied in der Gruppe mit der GID && g1RRS-Klasse und
     Response-Vollständigkeit erfordern eine Antwort des Respondents then
        Führe Backoff-Algorithmus aus
        if Nachricht des Typs G_Res mit AID nicht im lokalen Zwischenspeicher then
            Übergebe Inhalt der Nachricht per gAUDIT-R.IND an Server-Instanz
            Setze Zeitgeber timeout für Rückmeldung durch Server-Instanz
            while timeout.active() do
                if gAUDIT-R.RES von Server-Instanz ausgelöst then
                    Warte Zeitgeber Backoff
                    Sende Nachricht des Typs G_Res an Initiator
                    Lege versendete Nachricht unter AID im Zwischenspeicher ab
                end
                timeout.tick()
            end
        else
            Warte Zeitgeber Backoff
            Sende Nachricht des Typs G_Res mit AID aus Zwischenspeicher erneut
        end
    end
end
```

Abbildung 4.14: *Pseudo-Code des ARQ-Verfahrens für Request-Response-Kommunikation in Gruppen aus Sicht von Respondents.*

Soll der Respondent eine Antwort auf die eingegangene Nachricht des Typs $\mathbb{G}_{Req}$ versenden, so wird zunächst der Backoff-Algorithmus ausgeführt und der Zeitgeber `Backoff`

gesetzt (vergleiche Abschnitt 4.6.4.2). Anschließend wird überprüft, ob bereits eine passende Nachricht des Typs $\mathbb{G}_{Res}$ im internen Zwischenspeicher hinterlegt ist (eindeutig identifiziert anhand der in den Nachrichten angegebenen GID und AID). Wurde bereits eine Nachricht des Typs $\mathbb{G}_{Res}$ im internen Zwischenspeicher hinterlegt, so wird nach Ablauf des Zeitgebers `Backoff` die im internen Zwischenspeicher hinterlegte Nachricht des Typs $\mathbb{G}_{Res}$ erneut an den Initiator übertragen. Ist die Nachricht noch nicht im internen Zwischenspeicher hinterlegt, so übergibt der *Respondent* den Nachrichteninhalt an die entsprechende Server-Instanz. Anschließend wartet der *Respondent* auf eine Antwort der Server-Instanz. Trifft diese Antwort ein, so wird eine Nachricht des Typs $\mathbb{G}_{Res}$ mit der von der Server-Instanz erhaltenen Antwort an die Client-Instanz gesendet und die Nachricht anschließend im Zwischenspeicher hinterlegt. Generell werden alle Einträge des Zwischenspeichers, die älter als zwei Minuten sind, automatisch gelöscht.

Auf der Seite des Initiators kommt der in Abbildung 4.15 beschriebene Pseudo-Code zur Fehlerkontrolle des RRS zum Einsatz. Wird durch eine Client-Instanz eine Abfrage gestartet, so wählt der *Initiator* zunächst die AID zufällig. Danach wird der Zeitgeber `MaxGap` gestartet, der sicherstellt, dass die Request-Response-Kommunikation spätestens nach der von der Client-Instanz gewählten Zeitdauer terminiert. Anschließend findet eine Fallunterscheidung statt, ob die erwarteten Nachrichten des Typs $\mathbb{G}_{Res}$ mittels Unicast- oder Multicast-Kommunikation angefordert werden. Hierbei wählt der Initiator je nach g1RRS-Klasse und erwarteter *Response-Vollständigkeit* die effizientere Variante. Für mehr Details hierzu siehe Abschnitt 4.9.

Soll Unicast-Kommunikation verwendet werden um die erwarteten Nachrichten des Typs $\mathbb{G}_{Res}$ anzufordern, so setzt der *Initiator* je Gruppenmitglied x welches angefragt wird einen eigenen Zeitgeber $\texttt{RTO}_x$ auf den in der Datenbank *Member* hinterlegten Wert (siehe Abschnitt 4.6.4.1). Anschließend wird die Nachricht des Typs $\mathbb{G}_{Req}$ jeweils per Unicast an die Gruppenmitglieder versendet. Je empfangener Nachricht des Typs $\mathbb{G}_{Res}$ aktualisiert der *Initiator* den für das Gruppenmitglied hinterlegten Wert für `RTO` in der Datenbank *Member*. Außerdem überprüft der *Initiator*, ob die von der Client-Instanz angeforderte *Response-Vollständigkeit* erreicht wurde. Ist dies der Fall, so werden die eingesammelten Monitoringdaten an die Client-Instanz übergeben. Ist dies nicht der Fall, so wird so lange weiter auf Nachrichten des Typs $\mathbb{G}_{Res}$ gewartet, bis einer der Zeitgeber $\texttt{RTO}_x$ abgelaufen ist. In diesem Fall wird eine Sendewiederholung per Unicast vorgenommen und der Zeitgeber erneut gestartet. Dieser Vorgang wiederholt sich so lange, bis entweder alle erwarteten Nachrichten von den Gruppenmitgliedern eingesammelt wurden oder der Zeitgeber `MaxGap` abgelaufen ist. Läuft der Zeitgeber ab, bevor alle Nachrichten eingesammelt werden konnten, so wird die nicht erfolgreiche Request-Response-Kommunikation an die Client-Instanz zurückgemeldet.

```
begin
    Wähle AID für Nachricht des Typs G_Req zufällig
    MaxGap.start()
    if Versand mittels Unicast effizienter then
        Start ► Label für Sprunganweisung, siehe weiter unten
        Setze je Gruppenmitglied den RTO-Wert auf Wert aus der Datenbank
        Starte je Gruppenmitglied x Zeitgeber RTO_x
        Sende Nachricht des Typs G_Req jeweils per Unicast an Gruppenmitglied
        while MaxGap.active() do
            if Nachricht des Typs G_Res empfangen then
                Speichere neuen RTO des Respondents in Datenbank Member
                Speichere empfangene Monitoringdaten
                if Angeforderte Response-Vollständigkeit erreicht then
                    return Request-Response erfolgreich an Client-Instanz
                end
            end
            forall RTOs die abgelaufen sind do
                Setze RTO für Gruppenmitglied erneut
                Sende Nachricht des Typs G_Req erneut an Gruppenmitglied
            end
            MaxGap.tick()
        end
    else
        Setze Zeitgeber RTO auf den größtern Wert der Gruppenmitglieder
        Starte Zeitgeber RTO
        Sende Nachricht des Typs G_Req per Multicast an Gruppenmitglieder
        while MaxGap.active() do
            if Nachricht des Typs G_Res empfangen then
                Speichere neuen RTO des Respondents in Datenbank Member
                Speichere empfangene Monitoringdaten
                if Angeforderte Response-Vollständigkeit erreicht then
                    return Request-Response erfolgreich an Client-Instanz
                end
            end
            if RTO abgelaufen && Versand mittels Unicast effizienter then
                goto Start ► Sprunganweisung
            else
                Setze Zeitgeber RTO erneut
            end
            MaxGap.tick()
        end
    end
    return Request-Response nicht erfolgreich an Client-Instanz
end
```

Abbildung 4.15: *Pseudo-Code des ARQ-Verfahrens für Request-Response-Kommunikation in Gruppen aus Sicht des Initiators.*

Soll Multicast-Kommunikation verwendet werden, so setzt der *Initiator* lediglich einen einzelnen Zeitgeber RTO, auf den für alle Gruppenmitglieder höchsten in der Datenbank *Member* hinterlegten Wert (siehe Abschnitt 4.6.4.1) – somit ist sichergestellt, dass selbst das Gruppenmitglied mit der höchsten Umlaufzeit zwischen *Initiator* und *Respondent* vor Ablauf des Zeitgebers RTO antworten kann. Anschließend wird die Nachricht des Typs $\mathbb{G}_{Req}$ mittels Multicast an die Gruppenmitglieder versendet. Ähnlich wie bei der Unicast-Kommunikation überprüft nun der *Initiator* laufend, ob die von der Client-Instanz gewählte Response-Vollständigkeit erreicht wurde und meldet gegebenenfalls den Erfolg der Request-Response-Kommunikation zurück. Außerdem wird der Verlust von Nachrichten des Typs $\mathbb{G}_{Res}$ mit Hilfe des Zeitgebers RTO erkannt, wonach erneut eine Fallunterscheidung stattfindet, ob Multicast- oder Unicast-Kommunikation effizienter ist. Der Ablauf hiernach erfolgt dann wie oben beschrieben. Bei Ablauf des Zeitgebers MaxGap, ohne dass die angefragte *Response-Vollständigkeit* erreicht wurde, wird vom *Initiator* der Client-Instanz die nicht erfolgreiche Abarbeitung der Request-Response-Kommunikation signalisiert.

4.6.6 Zustandsautomat des Initiators

In diesem Unterabschnitt wird auf die Details des *Initiators* zur Realisierung des g1RRS näher eingegangen. Wesentlicher Bestandteil ist das ARQ-Verfahren zur Verlustbehandlung.

Abbildung 4.16 zeigt den Überblick über den Zustandsautomat des *Initiators*. Der Zustandsautomat lässt sich in drei Teile gliedern: Empfang von Nachrichten des Typs $\mathbb{G}_{Res}$, Verlustbehandlung und Gruppenverwaltung während Request-Response-Kommunikation. Außerdem wird per GMS auf die Datenbank *Member* zurückgegriffen. Darüber hinaus verwendet der Zustandsautomat die Zeitgeber GapClk und RTO.

Bei Nutzung von gAUDIT-R.REQ durch die Client-Instanz setzt der *Initiator* zunächst die Dienstelemente GapClk und RTO zurück.

Anschließend findet eine Fallunterscheidung statt, ob Multicast- oder Unicast-Kommunikation für das Versenden der Nachricht des Typs $\mathbb{G}_{Req}$ verwendet werden soll. Hierzu wird ermittelt, wie viele Nachrichten des Typs $\mathbb{G}_{Res}$ vom Initiator erwartet werden. Für mehr Details siehe Abschnitt 4.6.3 und Abschnitt 4.9.

Zum Versand von Nachrichten des Typs $\mathbb{G}_{Req}$ wird die AID vom *Initiator* zufällig erzeugt. Für eine Nachricht des Typs $\mathbb{G}_{Req}$ wird das Typ-Feld auf 0 gesetzt (siehe Abschnitt 4.4.3.2). Des Weiteren startet der *Initiator* den Zeitgeber RTO, der dazu dient, gegebenenfalls Sendewiederholungen auszulösen (siehe Abschnitt 4.6.6.2) – im Fall von Unicast existiert je erwarteter Nachricht des Typs $\mathbb{G}_{Res}$ ein eigener Zeitgeber RTO, im Fall von Multicast

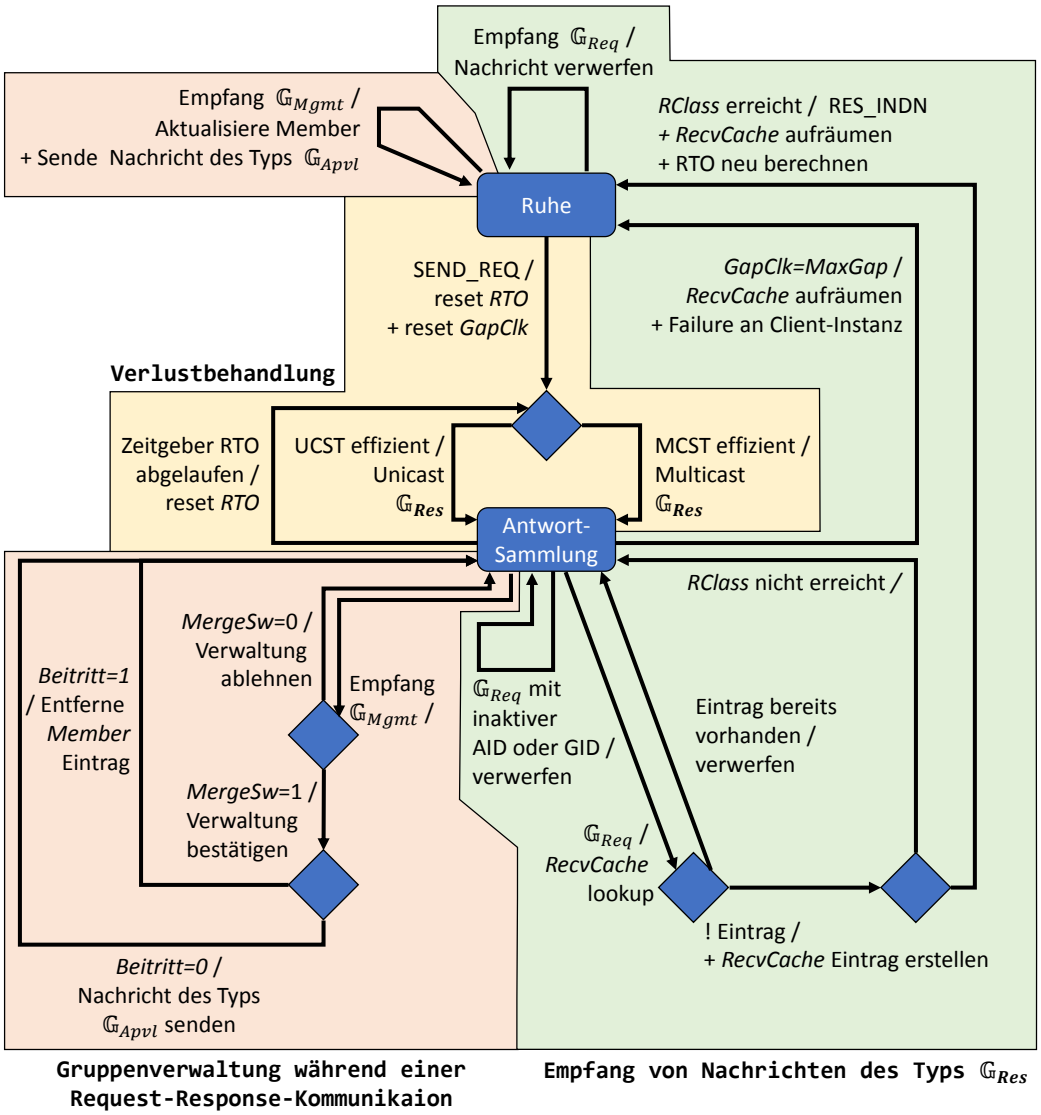

Abbildung 4.16: *Gesamtansicht des RRS-Zustandsautomaten des Initiators. Unterteilt in die Bereiche Verlustbehandlung, Empfang von Nachrichten des Typs $\mathbb{G}_{Res}$ und Gruppenverwaltung während einer Request-Response-Kommunikation. UCST steht hier für Unicast und MCST für Multicast.*

wird ein Zeitgeber für alle erwarteten Nachrichten verwendet. Mehr Details zur Wahl des Zeitgebers `RTO` finden sich in Abschnitt 4.6.4.1 und Abschnitt 4.9. Außerdem startet der *Initiator* den Zeitgeber `GapClk`, der die gesamte Ausführungszeit der Request-Response-Kommunikation misst. Anschließend geht der *Initiator* in den Zustand `Antwort-Sammlung` über.

Vom Zustand `Antwort-Sammlung` aus können nun drei Teile des Zustandsautomaten erreicht werden. Diese werden im Folgenden dargelegt.

4.6.6.1 Empfang von Nachrichten des Typs $\mathbb{G}_{Res}$

Abbildung 4.17 zeigt den Teil des Zustandsautomaten des *Initiators*, der für die Verarbeitung (den Empfang) von Nachrichten des Typs $\mathbb{G}_{Res}$ verantwortlich ist. Zur Realisierung der Funktionalität wird der Zwischenspeicher `RecvCache` verwendet. Dieser Teil des Zustandsautomaten setzt die folgenden Anforderungen um.

R01 Aktualisierung des Zwischenspeichers `RecvCache` bei Erhalt neuer Nachrichten des Typs $\mathbb{G}_{Res}$.

R02 Erkennung von doppelter Auslieferung einer Nachricht des Typs $\mathbb{G}_{Res}$ an die Client-Instanz. Hier besonders relevant, da es durch Sendewiederholung der Nachricht des Typs $\mathbb{G}_{Req}$ an eine *Gruppe* zum mehrfachen Empfang von Nachrichten des Typs $\mathbb{G}_{Req}$ kommen kann.

R03 Überprüfen, ob die von der Client-Instanz gewählte g1RRS-Klasse `RClass` erreicht wurde. Hierzu wird auf den Zwischenspeicher `RecvCache` zurückgegriffen.

R04 Gewährleisten, dass eine Request-Response-Kommunikation nach maximal der Zeitdauer von `MaxGap` terminiert.

R05 Sicherstellen, dass veraltete Einträge in dem Zwischenspeicher `RecvCache` wieder gelöscht werden.

Empfängt der *Initiator* im Zustand `Antwort-Sammlung` eine Nachricht des Typs $\mathbb{G}_{Res}$ eines *Respondents*, so wird zunächst ermittelt, zu welcher *Gruppe* die Nachricht (anhand der GID) gehört und ob die in der Nachricht angegebene AID sich auf die laufende Request-Response-Kommunikation bezieht. Ist dies nicht der Fall, so wird die Nachricht verworfen. Handelt es sich hingegen um eine erwartete Antwort wird anhand des Zwischenspeichers `RecvCache` überprüft, ob die Nachricht bereits vom *Initiator* empfangen wurde. Dabei wird die Nachricht eindeutig anhand der verwendeten IPv6-Absenderadresse und der enthaltenen AID identifiziert. Ist dies der Fall, so wird keine weitere Aktion getätigt und direkt in den Zustand `Antwort-Sammlung` zurückgesprungen – dann handelt es sich hierbei um eine überflüssige Sendewiederholung eines *Respondents*, da die Monitoringdaten mit der ersten empfangenen Nachricht des Typs $\mathbb{G}_{Res}$ bereits im `RecvCache` abgelegt wurden. Wurde die Nachricht noch nicht empfangen, so wird der Nachrichteninhalt in dem Zwischenspeicher `RecvCache` bei dem *Respondent* vermerkt. Durch den Eintrag in dem Zwischenspeicher `RecvCache` wird sichergestellt, dass keine Nachrichten des Typs $\mathbb{G}_{Res}$ doppelt an die Client-Instanz ausgeliefert werden. Außerdem wird je empfangener

Nachricht des Typs $\mathbb{G}_{Res}$ von einem *Respondent* die Umlaufzeit zwischen *Initiator* und *Respondent* ermittelt und in der Datenbank *Member* hinterlegt.

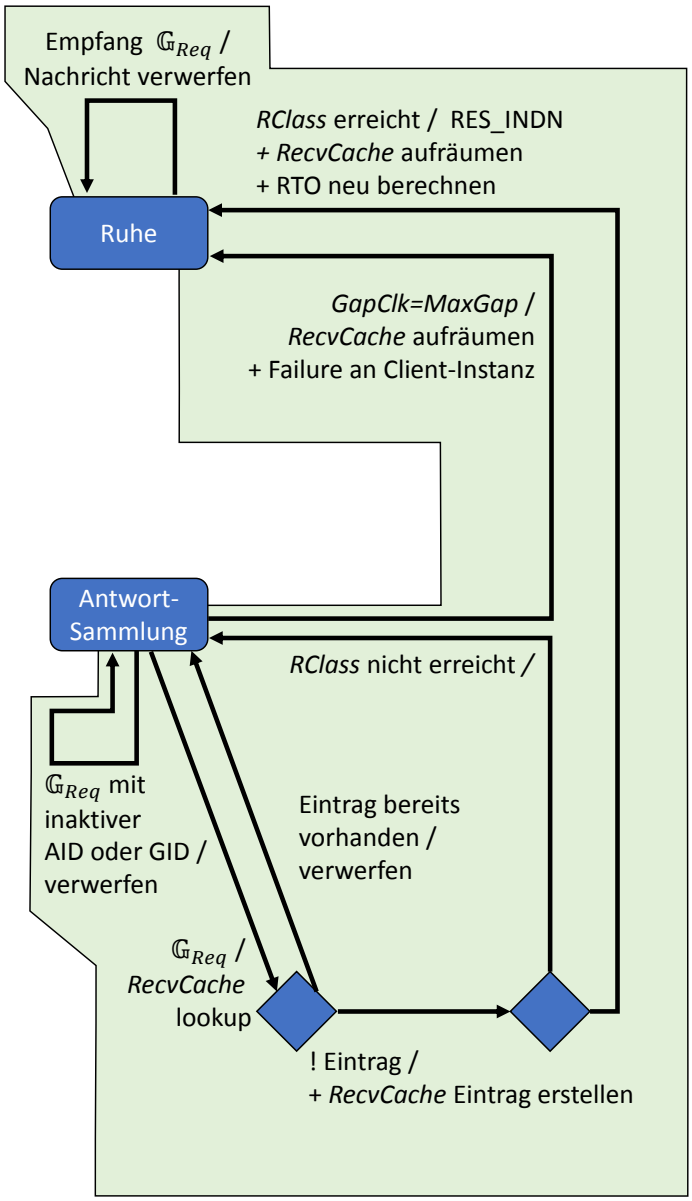

Abbildung 4.17: *Detailansicht des Initiator-Zustandsautomaten für den Empfang von Nachrichten des Typs* $\mathbb{G}_{Res}$.

Nach Aktualisierung des Zwischenspeichers `RecvCache` wird geprüft, ob die g1RRS-Klasse `RClass` erreicht wurde (siehe Abschnitt 4.6.2).

Um dies zu überprüfen, wird anhand der Datenbank *Member* des GMS und des Zwischenspeichers `RecvCache` überprüft wie viele erwartete *Respondents* noch keine Nachricht

des Typs $\mathbb{G}_{Res}$ gesendet haben. Ist diese Anzahl größer als der Wert 0, so liegen noch nicht alle Monitoringdaten (Antworten) vor und es wird in den Zustand `Antwort-Sammlung` gewechselt. Liegen alle Antworten vor (der Wert ist 0), so wird die Request-Response-Kommunikation beendet, die Antworten im `RecvCache` werden an die Client-Instanz ausgeliefert und der zu GID gehörige Kommunikationsendpunkt geschlossen. Außerdem werden beim Beenden alle zum abgelaufenen Request-Response-Mechanismus gehörende Einträge aus dem Zwischenspeicher `RecvCache` gelöscht – identifiziert werden diese anhand der GID und AID.

Während sich der Automat im Zustand `Antwort-Sammlung` befindet, wird laufend überprüft, ob der Zeitgeber `GapClk` bereits den Wert des Parameters `MaxGap` erreicht hat. Ist dies der Fall, so wird die Request-Response-Kommunikation mit einer Fehlermeldung beendet und der zu GID gehörige Kommunikationsendpunkt geschlossen. Beim Beenden werden zusätzlich alle zur Request-Response-Kommunikation gehörenden Einträge aus dem Zwischenspeicher `RecvCache` gelöscht.

4.6.6.2 Verlustbehandlung

Abbildung 4.18 zeigt den Teil des Zustandsautomaten des *Initiators*, der bei Nachrichtenverlust Sendewiederholungen durchführt. Dabei setzt dieser Teil des Zustandsautomaten die folgende Anforderung um.

R06 Überwachung des Zeitgebers `RTO` und gegebenenfalls wiederholtes Senden der Nachricht des Typs $\mathbb{G}_{Req}$ bei Nachrichtenverlust an eine *Gruppe* (per Unicast oder Multicast).

Um den fehlenden Empfang einer Nachricht des Typs $\mathbb{G}_{Res}$ zu detektieren, werden alle Zeitgeber `RTO` überwacht. Für Multicast muss lediglich ein einzelner Zeitgeber überwacht werden, für Unicast muss für jeden adressierten *Respondent* ein Zeitgeber `RTO` überwacht werden. Ist dessen Laufzeit abgelaufen, so wird von einem Nachrichtenverlust ausgegangen. Zur Verlustbehandlung werden Sendewiederholungen durchgeführt.

Durch das erneute Senden der Nachrichten des Typs $\mathbb{G}_{Req}$ per Multicast an eine *Gruppe* können *Respondents* eine bereits zuvor empfangene $\mathbb{G}_{Req}$ mehrfach erhalten und zwar unabhängig davon, ob sie diese bereits beantwortet haben. Aus diesem Grund muss dieser Fall auf Seiten des *Respondents* behandelt werden (siehe Abschnitt 4.6.7). Falls eine Nachricht des Typs $\mathbb{G}_{Res}$ ein zweites Mal bei der Client-Instanz eintrifft (beispielsweise weil der *Respondent* diese ein zweites Mal versendet hat), so wird dies anhand der AID erkannt (siehe Abschnitt 4.6.6.1) und nicht erneut in den Zwischenspeicher `RecvCache` aufgenommen.

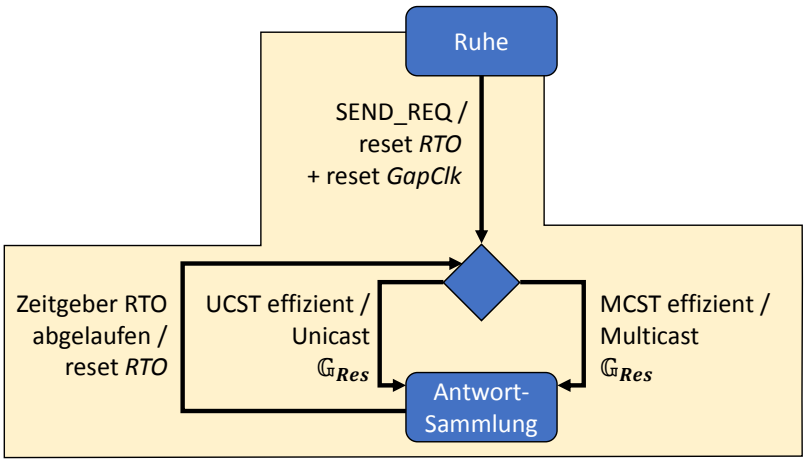

Abbildung 4.18: *Detailansicht des Initiator-Zustandsautomaten für die Behandlung von Nachrichtenverlust.*

4.6.6.3 Gruppenverwaltung während Request-Response-Kommunikation

Anhand der Abbildung 4.19 wird das Verhalten des *Initiators* bei Änderung der Gruppenmitgliedschaften während einer Request-Response-Kommunikation erörtert. Dabei setzt dieser Teil des Zustandsautomaten die folgenden Anforderungen um.

R07 Integration von Änderungen in der Gruppenzusammensetzung in eine laufende Request-Response-Kommunikation. Je nach gesetztem Wert für den Parameter `MergeSw`.

R08 Gewährleisten, dass neu zur *Gruppe* beigetretene *Respondents* mit der aktuellen Nachricht des Typs $\mathbb{G}_{Req}$ versorgt werden.

R09 Sicherstellen, dass die *Respondents* die die *Gruppe* verlassen haben, nicht mehr länger bei der Überprüfung der g1RRS-Klasse `RClass` mit berücksichtigt werden.

Erreicht eine Nachricht des Typs $\mathbb{G}_{Mgmt}$ den *Initiator*, wobei die GID identisch zu der laufenden Request-Response-Kommunikation ist (gleiche *Gruppe* betroffen), so wird eine Fallunterscheidung durchgeführt.

Wurde der Parameter `MergeSw` von der Client-Instanz bei Aufruf mittels `gAUDIT-R.REQ` auf `0` gesetzt, so wird die eingehende Änderung an der Gruppenmitgliedschaft vom *Initiator* abgelehnt. Hierzu wird die Nachricht des Typs $\mathbb{G}_{Apvl}$ mit invertiertem Feld Beitritt (im Bezug zum Wert in der Nachricht des Typs $\mathbb{G}_{Mgmt}$) an den anfragenden *Respondent* zurückgesendet (vergleiche Abschnitt 4.5.4).

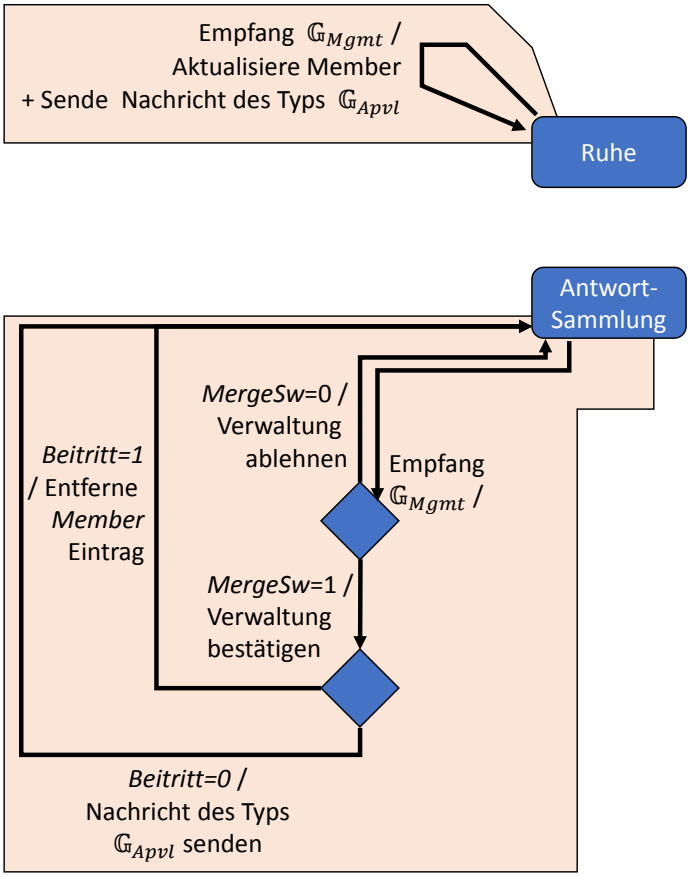

Abbildung 4.19: *Darstellung der Gruppenverwaltung durch den Initiator.*

Ist der Parameter `MergeSw` hingegen auf den Wert `1` gesetzt, so integriert der *Initiator* die neue Gruppenzusammensetzung in die laufende Request-Response-Kommunikation und versendet eine positive $\mathbb{G}_{Apvl}$ an den anfragenden *Respondent* (siehe Abschnitt 4.5.4). Hierzu wird zunächst ermittelt, ob es sich bei der Nachricht des Typs $\mathbb{G}_{Mgmt}$ um einen Beitritt oder das Verlassen einer *Gruppe* handelt. Handelt es sich um einen Beitritt – das Feld Beitritt steht auf `0` – so ermittelt der *Initiator*, ob der neue *Respondent* Teil der g1RRS-Klasse ist und versendet gegebenenfalls die zuvor an die *Gruppe* gesendete Nachricht des Typs $\mathbb{G}_{Req}$ mittels Unicast an den neuregistrierten *Respondent*. Handelt es sich um den Austritt aus einer *Gruppe*, so wird der entsprechende Eintrag für den *Respondent* und den GID aus der Datenbank `Member` entfernt. Außerdem werden bereits erhaltene Nachrichten des Respondents anhand der AID und der IPv6-Absenderadresse des *Respondents* identifiziert und aus dem Zwischenspeicher `RecvCache` gelöscht.

4.6.7 Zustandsautomat des Respondents

Abbildung 4.20 zeigt den Zustandsautomaten des *Respondent* bei Erhalt einer Nachricht des Typs $\mathbb{G}_{Req}$. Dabei setzt dieser Abschnitt des Zustandsautomaten die folgenden Anforderungen um.

R10 Überprüfen, ob eingehende Nachrichten des Typs $\mathbb{G}_{Req}$ bereits von der Server-Instanz behandelt wurden. Verhindert Mehraufwand auf Seiten von Server-Instanzen bei Sendewiederholungen des Initiators.

R11 Übermittlung von noch nicht bearbeiteten Nachrichten des Typs $\mathbb{G}_{Req}$ an die Server-Instanz.

R12 Aktualisierung des Zwischenspeichers `RecvCache` auf Basis der von der Server-Instanz erhaltenen Nachrichten des Typs $\mathbb{G}_{Res}$. Außerdem wird eine entsprechende Nachricht des Typs $\mathbb{G}_{Res}$ an den *Initiator* versendet.

R13 Erkennen, dass die mit dem *Respondent* verbundene Server-Instanz aufgrund eines Fehlers die Anfrage nicht bearbeitet.

Nach Erhalt einer Nachricht des Typs $\mathbb{G}_{Req}$ liest der *Respondent* die Nachrichtenfelder aus. Anschließend wird in dem Zwischenspeicher `RecvCache` geprüft, ob für die vorliegende GID bereits eine Nachricht des Typs $\mathbb{G}_{Res}$ an den *Initiator* versendet wurde. Ist dies der Fall (Eintrag im Zwischenspeicher vorhanden), so führt der *Respondent* die Sendewiederholung der gespeicherten Nachrichten des Typs $\mathbb{G}_{Res}$ durch, ohne noch einmal die Server-Instanz aufzurufen. Hierzu wird ein Backoff-Algorithmus verwendet. Für jede durchzuführende Sendewiederholung wird dabei das Fenster zur zufälligen Wahl des Backoffs exponentiell vergrößert. Mehr Details zur Bewertung des Backoff-Algorithmus, insbesondere in Kombination mit der Berechnung von `RTO` auf Seiten des *Initiators*, finden sich in Abschnitt 4.6.4.2 und Abschnitt 4.9.

Ist die AID nicht bekannt (kein Eintrag in der Liste vorhanden), so löst der *Respondent* `gAUDIT-R.IND` aus. Hierfür wird der Kommunikationsendpunkt der Server-Instanz gewählt, der zur GID gehört. Dieser wurde beim Beitritt zur *Gruppe* geöffnet (siehe Details des GMS in Abschnitt 4.5). Neben der Anfrage wird mittels `gAUDIT-R.IND` auch die AID über den Kommunikationsendpunkt an die Server-Instanz übermittelt. Hiernach startet der *Respondent* einen Zeitgeber von einer Minute, um zu überwachen, ob die Server-Instanz antwortet. Ohne Beschränkung der Allgemeinheit wird dieser Wert als maximale Verarbeitungszeit der Server-Instanz für eine Anfrage angenommen. Bei anderen Anwendungsfällen als dem *CloudInspector* kann diese Konstante angepasst werden.

Läuft dieser Timeout ab, ohne dass die Server-Instanz `gAUDIT-R.RES` ausgelöst hat, so beendet der *Respondent* die lokale Abarbeitung des Request-Response-Mechanismus und

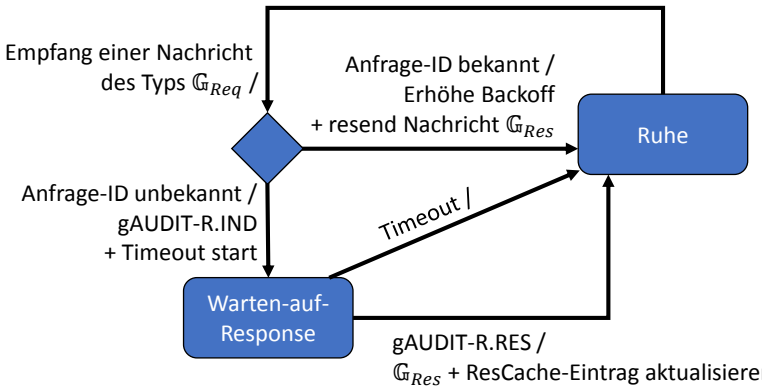

Abbildung 4.20: *Zustandsautomat auf Seiten des Respondents zur Durchführung von vom Initiator ausgehender Request-Response-Kommunikation.*

geht in den Zustand `Ruhe` über. Wird `gAUDIT-R.RES` von der Server-Instanz vor Ablauf des Timeouts ausgelöst, so wird eine Nachrichten des Typs $\mathbb{G}_{Res}$ an den *Initiator* versendet. Zusätzlich werden die mittels der Nachrichten des Typs $\mathbb{G}_{Res}$ zu transportierenden Monitoringdaten der Server-Instanz in dem Zwischenspeicher `RecvCache` hinterlegt. Hierzu wird die GID und AID verwendet, so dass der *Respondent* eine Sendewiederholung durchführen kann, ohne die Server-Instanz erneut zu involvieren. Dabei löscht der Zwischenspeicher jeden Eintrag zwei Minuten nach Hinzufügen. Dies ist der maximale Wert, den eine Client-Instanz bei dem Parameter `MaxGap` einstellen kann und somit die maximale Zeit, in der Sendewiederholungen innerhalb eines Request-Response-Mechanismus vom *Initiator* durchgeführt werden können (siehe Abschnitt 4.6.1) und eine Sendewiederholung von bereits versendeten Nachrichten des Typs $\mathbb{G}_{Res}$ vom *Respondent* erforderlich sein können.

4.7 Sicherheitsaspekte

Zwar wird in dieser Dissertation prinzipiell von einem ehrlichen Cloud-Anbieter ausgegangen (vergleiche Kapitel 1), dennoch kann es sinnvoll sein, die Kommunikation des Transportsystems zumindest gegen passive Angreifer zu schützen. Beispielsweise in dem Fall, dass ein Angreifer Zugriff auf das Cloud-Netz erhält, aber keinen Zugriff auf die physischen Cloud-Knoten oder den Monitoring-Controller hat. Als Angreifer-Modell wird im Folgenden das in Abbildung 4.21 dargestellte *Dolev-Yao-Modell* [49] angenommen. Der Angreifer hat insbesondere keinen Zugriff auf die Komponenten von *gAUDIT* (*Initiator* und *Respondent*) auf den Endsystemen.

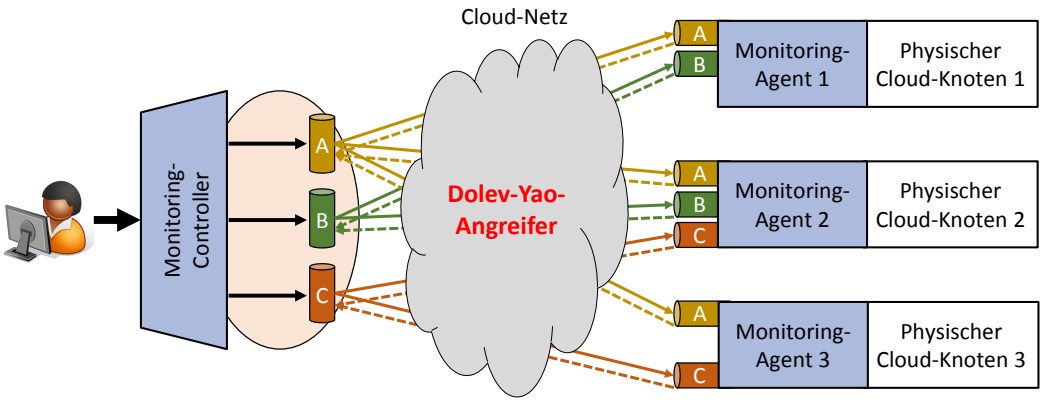

Abbildung 4.21: *Einsatz von gAUDIT in Gegenwart eines Dolev-Yao-Angreifers.*

Als Schutzziel soll vertrauliche und integere Kommunikation zwischen *Initiator* und *Respondents* in Gegenwart von Dolev-Yao-Angreifern gewährleistet werden. Die Erweiterung für *gAUDIT* ist in die folgenden drei Teilbereiche unterteilt:

Schlüsselverteilung und Erneuerung: Bei der Installation des *gAUDIT* Transportsystems wird Schlüsselmaterial auf den *Initiator* beziehungsweise die *Respondents* verteilt. Je *Respondent* wird dabei ein eigenes symmetrisches Schlüsselpaar R_K mit dem *Initiator* gebildet und in einer eigenen *Schlüssel-Datenbank* hinterlegt. Des Weiteren wird für den *Initiator* mit Hilfe eines asymmetrischen Verfahrens ein öffentlicher I_{PK} und privater Schlüssel I_{SK} erzeugt. Zusätzlich erhalten die *Respondents* als Vertrauensanker den öffentlichen Schlüssel C_{PK}. Der öffentliche Schlüssel C_{PK} dient zur Überprüfung für Zertifikate auf den öffentlichen Schlüssel I_{PK} des *Initiators*, so dass der *Initiator* einerseits eindeutig identifiziert wird und anderseits der öffentliche Schlüssel I_{PK} gewechselt werden kann (beispielsweise bei Erneuerung des Schlüsselmaterials). Das symmetrische Schlüsselmaterial der *Respondents* wird in regelmäßigen Abständen mit Hilfe eines geeigneten Protokollmechanismus erneuert.

Schutz von Unicast-Nachrichten: Neben der Verteilung von Schlüsselmaterial bei der Installation wird auch der Ablauf von *gAUDIT* angepasst. So werden alle Unicast-Nachrichten (Nachrichten des Typs $\mathbb{G}_{Res}$, $\mathbb{G}_{Mgmt}$ und $\mathbb{G}_{Apvl}$) mit dem im Voraus verteilten symmetrischen Schlüssel mit der Funktion `EncSign(Schlüssel, Nachricht)` verschlüsselt und signiert. Beispielsweise $\texttt{EncSign}(R_K, \mathbb{G}_{Res}) = C$. Anhand der Abbildung von Quelladresse des Respondents auf symmetrischen Schlüssel R_K in der Schlüssel-Datenbank entschlüsselt der *Initiator* diese Nachrichten dann mit der Funktion `DecVerify(Schlüssel, Chiffrat)` und prüft, ob diese unverändert empfangen wurde. Beispielsweise $\texttt{DecVerify}(R_K, C) = \mathbb{G}_{Res}$. Die Unicast-Nachricht des Typs $\mathbb{G}_{Apvl}$ von dem *Initiator* zu einem *Respondent* ist ebenfalls mit dem entsprechenden symmetrischen Schlüssel R_K zwischen *Initiator* und *Respondent* verschlüsselt. Zur Inte-

gritätssicherung wird allerdings eine asymmetrische digitale Signatur des *Initiators* mit dem Schlüssel $I_{\mathtt{SK}}$ verwendet. Hierzu wird der Funktion ein Tupel als Schlüssel übergeben `EncSign((Verschlüsselungsschlüssel, Signierschlüssel), Nachricht)`. Außerdem wird der Nachricht $\mathbb{G}_{Apvl}$ der öffentliche Schlüssel $I_{\mathtt{PK}}$ und das Zertifikat des *Initiator* $I_{\mathtt{Cert}}$ angehängt. Jeder *Respondent* überprüft dann anhand des im Vorfeld verteilten öffentlichen Schlüssels $C_{\mathtt{PK}}$, ob die Signatur auf das Zertifikat $I_{\mathtt{Cert}}$ des Initiators gültig ist. Bei erfolgreicher Überprüfung wird die Nachricht mit Hilfe der Funktion `DecVerify((Verschlüsselungsschlüssel, Signierschlüssel), Chiffrat)` entschlüsselt. Als Signierschlüssel wird hier der öffentliche Schlüssel $I_{\mathtt{PK}}$ des Initiators verwendet. Hierdurch wird sichergestellt, dass sich kein *Respondent* als *Initiator* ausgeben kann und gleichzeitig bleiben Unicast-Nachrichten zwischen *Initiator* und *Respondent* vertraulich. Als Verschlüsselungsverfahren kommt ein Verfahren aus der Familie *Authenticated encryption* zum Einsatz, zum Beispiel der *Galois Counter Mode (GCM)* [132, 133] oder *Counter with CBC-MAC (CCM)* [215].

Schutz der Gruppenkommunikation: Zur Absicherung der Nachricht des Typs $\mathbb{G}_{Req}$, die als Multicast-Nachricht an eine *Gruppe* versendet wird, wird je *Gruppe* ein spezifischer Schlüssel G_K verwendet (Gruppenschlüssel). Der *Initiator* hinterlegt je *Gruppe* diesen Gruppenschlüssel G_K in der Datenbank `Member`. Dem *Respondent* wird dieser Gruppenschlüssel bei Beitritt zu einer *Gruppe* in der $\mathbb{G}_{Apvl}$ Nachricht mitgeteilt. Die Nachricht des Typs $\mathbb{G}_{Apvl}$ ist wiederum mit dem auf der *Respondent* im Voraus verteilten symmetrischen Schlüssel R_K verschlüsselt und durch die digitale Signatur des *Initiators* integritätsgesichert. Der *Respondent* speichert für jede *Gruppe*, bei der er beigetreten ist, den zugehörigen Gruppenschlüssel G_K ab. Der *Initiator* verschlüsselt nun jede Nachricht des Typs $\mathbb{G}_{Req}$ an die *Gruppe* mit dem entsprechenden in der Datenbank `Member` hinterlegten Gruppenschlüssel G_K und sichert die Integrität mit Hilfe seiner digitalen Signatur unter Verwendung des Schlüssels $I_{\mathtt{SK}}$. Durch die asymmetrische digitale Signatur des *Initiators* wird gewährleistet, dass kein *Respondent* eine gültige Nachricht des Typs $\mathbb{G}_{Req}$ an einen *Gruppe* senden kann, obwohl diese im Besitz des Gruppenschlüssels sind. Das Verlassen einer *Gruppe* erfolgt analog, wobei der *Respondent* den entsprechenden Gruppenschlüssel G_K löschen muss. Das Löschen des Gruppenschlüssels G_K dient dabei als "Vorsichtsmaßnahme", so dass bei der eventuellen Übernahme eines *Respondents* durch einen stärkeren Angreifer nicht mehr benötigtes Schlüsselmaterial nicht abhandenkommen kann. Außerdem wird je *Gruppe* der Gruppenschlüssel in regelmäßigen Abständen mit Hilfe eines geeigneten Protokollmechanismus erneuert, beispielsweise mit Hilfe von [206].

Mit dem hier beschriebenen Verfahren kann Vertraulichkeit und Integrität gegenüber passiven Angreifern im Netz des Cloud-Anbieters erreicht werden. Außerdem wird sichergestellt, dass nur ein legitimer *Initiator* Nachrichten des Typs $\mathbb{G}_{Req}$ an eine *Gruppe* senden kann.

Gelingt es allerdings einem stärkeren Angreifer, einen physischen Cloud-Knoten und in dessen Folge einen *Respondent* zu übernehmen, so hat dieser direkten Zugriff auf das im Vorfeld verteilte Schlüsselmaterial zwischen *Respondent* und *Initiator*. Allerdings hat der Angreifer keinen Zugriff auf die im Zertifikat beglaubigte Identität des *Initiators* (insbesondere nicht auf den privaten Schlüssel I_{SK}) und kann somit auch keine gültige Nachricht des Typs $\mathbb{G}_{Req}$ an eine *Gruppe* senden.

Jedoch kann der Angreifer bei Übernahme eines *Respondents* beliebigen *Gruppen* beitreten, Nachrichten mitlesen oder falsche Monitoringdaten an den *Initiator* zurücksenden. Hierbei ist er allerdings limitiert auf den übernommenen *Respondent*. Einen anderen *Respondent* kann der Angreifer nicht impersonifizieren, da er keinen Zugriff auf dessen individuelles Schlüsselmaterial R_K mit dem *Initiator* hat. Nichtsdestotrotz würde dieser Angriff die Funktionalität von Anwendungsinstanzen des *gAUDIT*-Transportsystems wie *CloudInspector* massiv beeinflussen. Daher muss durch andere Maßnahmen die Übernahme eines physischen Cloud-Knotens durch den Angreifer verhindert werden. Üblicherweise hat hier der Cloud-Anbieter selber ein immenses Interesse daran, sicherzustellen, dass unautorisierte Nutzer keinen Zugriff auf physische Cloud-Knoten haben, und setzt daher bereits entsprechende Sicherheitsmaßnahmen ein.

4.8 Verwandte Ansätze

Der Umsetzung von adaptiver Request-Response-Kommunikation in Szenarien mit Gruppen wird bisher insgesamt wenig Aufmerksamkeit gewidmet. Dies liegt vor allem daran, das Request-Response typischerweise oberhalb der Transportschicht angesiedelt ist und daher typischerweise anwendungsfallspezifisch realisiert wird. Ein wesentliches Abgrenzungsmerkmale von *gAUDIT* gegenüber dem Stand der Forschung ist die effiziente Umsetzung von *Response-Vollständigkeit* für verschiedene g1RRS-Klassen durch geschickte Auswahl zwischen Multicast und Unicast. Hinzu kommt, dass *gAUDIT* nach bestem Wissen derzeit das einzige Verfahren ist, dass überhaupt statistische Parameterisierung der Response-Vollständigkeit erlaubt und hierzu passend Multicast oder Unicast einsetzt und Response-Nachrichten in Abhängigkeit der erwarteten Antworten desynchronisiert.

4.8.1 Robuste Übertragung von Response-Nachrichten

Um Response-Nachrichten vom Initiator an eine Gruppe von Respondents zuverlässig zu übertragen, kommen Multicastprotokolle in Betracht. Unter den zahlreichen Varianten [145, 47, 190, 119, 191, 90] existieren mitunter auch zuverlässige Varianten für die Transportschicht [211, 221, 228, 231].

Zwar unterstützen einige Varianten die Verwaltung der Gruppenmitgliedschaft durch Empfänger, allerdings sind zuverlässige Multicastprotokolle nicht auf Request-Response-Kommunikation ausgelegt und unterstützen somit auch keine *Response-Vollständigkeit*, die sich parametrisieren ließe. Dies liegt vor allem daran, dass die Dienstdefinitionen von Protokollen der Transportschicht sich auf unidirektionale Datenübertragung von Sender zu Empfänger beziehen. Bidirektionale Garantien, wie die Anforderung einer Response-Nachricht werden in der Regel nicht mit abgedeckt. In der Konsequenz können bei *gAUDIT* durch Einsatz eines zuverlässigen Multicastprotokolls lediglich Nachrichten des Typs $\mathbb{G}_{Req}$ vom *Initiator* zu den *Respondents* übertragen werden, aber keine Nachrichtenverluste innerhalb der Server-Instanz oder auf dem Rückweg zum *Initiator* behandelt werden. Genauso wenig unterstützen zuverlässige Multicastprotokolle die vorgestellten g1RRS-Klassen. Aus diesen Gründen können Multicastprotokolle für den Einsatz bei *CloudInspector* ausgeschlossen werden. Das *gAUDIT* Transportsystem bietet hier wesentlich bessere Eigenschaften (siehe auch Tabelle 4.3).

Weiter sind Arbeiten zu *Publish-Subscribe* durch den Stand der Forschung gut abgedeckt [53, 126, 11]. Ein typisches Charaktermerkmal des *Publish-Subscribe* Kommunikationsmusters ist die Entkopplung von *Raum*, *Zeit* und *Synchronisation* zwischen Sender und Empfänger (vergleiche [53]). Bei *gAUDIT* hingegen, ist die Kopplung von *Zeit* und *Synchronisation* zwischen *Initiator* und *Respondents* zwingend erforderlich, um Monitoringdaten zur Überprüfung an *CloudInspector* auszuliefern. Sowohl die Client-Instanz als auch die Server-Instanz müssen bei *gAUDIT* zur gleichen Zeit ausgeführt werden, denn der Initiator blockiert nach Versenden einer Request-Nachricht und sammelt bis zum Ablauf eines Zeitgebers eintreffende Response-Nachrichten ein. Aufgrund dieser kategorischen Unterschiede können Verfahren für *Publish-Subscribe* für den Einsatz bei *CloudInspector* im Vorhinein ausgeschlossen werden. Darüber hinaus unterstützt nach derzeitigem Stand lediglich ein einziges *Publish-Subscribe*-Verfahren die Abfrage von Daten [43, 84], allerdings ohne dass dabei die Client-Instanz aus verschiedenen g1RRS-Klassen wählen könnte oder die gewünschte *Response-Vollständigkeit* parametrisieren kann. Außerdem verwendet dieses *Publish-Subscribe*-Verfahren kein effizientes ARQ-Verfahren welches geschickt zwischen Multicast und Unicast wählen würde. Aus diesen Gründen stellt *gAUDIT* eine bessere Alternative zur Realisierung von parametrisierbarer Request-Response-Kommunikation in Gruppen dar.

4.8.2 Multi-RPC

Wie in Abschnitt 2.4 angedeutet, sind Forschungsarbeiten zu RPC [232, 226, 21, 173, 82, 162] mit *gAUDIT* verwandt. Insbesondere solche, die RPC in Szenarien mit Gruppen einsetzen. Diese werden in der Literatur meist als sogenannte Middlewares realisiert

und stellen Anwendungen einen gruppenbasierten RPC-Dienst auf Anwendungsebene zur Verfügung. Allerdings sind die zur Verfügung gestellten Dienste oft sehr generisch und müssen um entsprechende Anwendungslogik erweitert werden, um beispielsweise verschiedene g1RRS-Klassen oder parametrisierbare *Response-Vollständigkeit* zu unterstützen. Auch ist keines der untersuchten Verfahren von sich aus in der Lage, geschickt zwischen Multicast oder Unicast für den Nachrichtenversand auszuwählen. Anhand von *CORBA* wird außerdem deutlich, dass Middlewares zwar sehr flexibel sind, aber in der Regel selbst eine hohe Komplexität bei der Verwendung in Anwendungen aufweisen [83]. So kommt beispielsweise auch einer der Hauptentwickler von *CORBA Electra* (S. Maffeis) [262] zu folgender Einsicht:

> *Even though the inclusion of fault-tolerance features was successful, my 6+ years of CORBA research & development led me to the conclusion that CORBA does not provide an adequate abstraction for mission critical applications, which need to be highly available, scalable, and extensible.*

Die derzeit bekannten Multi-RPC-Verfahren scheinen daher nicht für den Einsatz bei *CloudInspector* geeignet. Vielmehr ist ein Transportsystem wie *gAUDIT* erforderlich, welches der Anwendung die Parameterisierung von g1RRS erlaubt und geschickt zwischen Unicast und Multicast auswählt. Vergleiche auch Tabelle 4.3.

4.8.3 Zusammenfassung

In Tabelle 4.3 ist das Ergebnis eines Abgleichs der *gAUDIT*-Charakteristika mit derzeit verwandten Arbeiten aufgetragen.

In der Spalte *Zuverlässigkeit* ist aufgeführt, wie der jeweilige Ansatz für Zuverlässigkeit in der Nachrichtenübertragung sorgt.

Die Spalte *Auswahl* führt auf, ob und wie eine geschickte Auswahl zwischen Unicast und Multicast erfolgt. Der Eintrag *fest* bei *LRBM* [87] bedeutet, dass *LRBM* ab einer festen Anzahl von fehlenden Antworten von Multicast auf Unicast umschaltet. Die Einträge *Subcasting* stehen bei Verfahren, die neue Multicastgruppen für die Durchführung von Sendewiederholungen bilden und somit nicht erneut die ursprüngliche Multicastgruppe adressieren. Ähnlich funktioniert dass Verfahren *TRAM* [36] mit dem Eintrag *Baum* – hier werden Sendewiederholungen nur in einem Teilbaum wiederholt. *RMTP* [147] hingegen nutzt für Sendewiederholungen je Region sogenannte *designated receiver*, weswegen dort in der Tabelle 4.3 der Eintrag *Verteilt* zu finden ist.

Die Spalte *Antworten* gibt an, ob das jeweilige Verfahren eine Semantik zur Parameterisierung der Response-Vollständigkeit anbietet. Wie aus der dargestellten Analyse

	Ansatz	Zuverlässigkeit	Auswahl	Antworten	Incast	Protokoll
Multicastprotokolle	LRBM [87]	Empfänger	fest	–	–	proprietär
	MFTP [137, 138]	SNAK	Anwendung	–	Aggregation	UDP
	MTP [204]	SNAK	–	–	–	IP
	OTRES [120]	NAK	Subcasting	–	–	(IP)
	PGM [67, 211]	NAK	Unicast	–	Backoff	IP
	RMTP [147]	ACK	Verteilt	–	Aggregation	UDP
	SRM [62, 109]	Subgruppen	Subcasting	–	–	IP
	TRAM [36]	SNAK	Baum	–	Baum	IP
Publish-Subscribe	JGroups pub-sub [13, 274]	Anwendung	Anwendung	Anzahl	–	IP
	JMS pub-sub [86, 273, 80]	ACK	–	–	–	TCP/IP
	Kafka pub-sub [66, 247]	Anwendung	–	–	–	TCP
	Pub-Sub with reply [43, 84, 42]	–	–	Nur alle	Middleware	Middleware
	ZeroMQ pub-sub [315, 313]	–	–	–	–	Multicast
	gAUDIT [TT-3]	Sender	Parameter s	g1RRS-Klassen	Zufallsverzögerung	UDP/IPv6

Tabellarischer Abgleich der Charakteristika von gAUDIT mit verwandten Ansätzen (Fortsetzung nächste Seite).

	Ansatz	Zuverlässigkeit	Auswahl	Antworten	Incast	Protokoll
Multi-RPC	CORBA [129, 182, 56, 57, 10]	Broker	Broker	Anzahl / Mehrheit	–	–
	Amoeba [108, 107]	SNAK	–	Anwendung	–	–
	DCOM [260]	Anwendung	–	–	Anwendung	–
	Horus [156]	NAK	–	Anwendung	Anwendung	–
	MPI [75, 74, 39]	Anwendung	Anwendung	Anwendung	Anwendung	Anwendung / TCP
	Nanomsg survey [284, 283]	–	–	Zeitgeber	–	TCP
	Nimbus [285]	Anwendung	–	Zeitgeber / Genau eine	–	Service Bus
	gAUDIT [TT-3]	Sender	Parameter *s*	g1RRS-Klassen	Zufallsverzögerung	UDP/IPv6

Tabelle 4.3: *Tabellarischer Abgleich der Charakteristika von gAUDIT mit verwandten Ansätzen (Fortsetzung vorherige Seite).*

ersichtlich wird, ist *gAUDIT* nach derzeitigem Wissen das einzige Verfahren, was die Parameterisierung einer statistischen Vollständigkeitsklasse erlaubt.

In der Spalte *Incast* wird dargestellt, ob die analysierten Verfahren einen Mechanismus zur Desynchronisierung von Response-Nachrichten vorsehen, damit es nicht zum Incast-Problem kommt. Einige Verfahren setzen hier auf *Aggregation* von Nachrichten in Richtung *Initiator*, was bei *gAUDIT* aber nicht in Frage kommt, da für Cloud-Infrastrukturen nicht vorausgesetzt werden kann, dass Zwischensysteme die Aggregation von Response-Nachrichten übernehmen. Andere sind in diesem Gesichtspunkt von der Implementierung der Anwendung abhängig – Eintrag *Anwendung*.

Die Spalte *Protokoll* zeigt, welche Voraussetzungen die einzelnen Verfahren an unterliegende Schichten haben. Verfahren bei denen hier das Symbol "–" aufgeführt ist, bringen keine Voraussetzungen mit sich.

Im Ergebnis bleibt festzuhalten, dass der Entwurf von *gAUDIT* einen Mehrwert für bidirektionale Kommunikation zwischen einer Client-Instanz und einer Gruppe Server-Instanzen bietet: Server-Instanz gesteuerte Gruppenverwaltung, Synchronisation der Gruppenmitgliedschaft zwischen *Respondent* und *Initiator*, geschickte Wahl zwischen Multicast und Unicast, die Nutzung von Sendewiederholung, um die gewünschte *Response-Vollständigkeit* zu realisieren und Desynchronisierung von Response-Nachrichten zur Vorbeugung von Incast-Szenarien.

4.9 Evaluation

Die Evaluation des Transportsystems ist in drei Teile strukturiert. Im ersten Teil wird das bei *gAUDIT* zur Fehlerbehandlung eingesetzte ARQ-Verfahren näher untersucht. Der zweite Teil widmet sich der Untersuchung der durch Auswahl zwischen Multicast und Unicast erreichbaren Effizienz. Im dritten Teil wird aufgezeigt, mit welchem Verfahren sich Response-Nachrichten so desynchronisieren lassen, dass es nicht zur *Incast-Problematik* kommt und trotzdem eine geringe Ausführungszeit von *gAUDIT* erreicht wird.

Die hier vorgestellten Evaluationsergebnisse basieren auf einem simulativen Ansatz. Hierfür werden verschiedene Parameterstudien mit Hilfe des diskreten und ereignisorientierten Simulationswerkzeugs *OMNeT++* [179, 180, 289] und dem Modellierungswerkzeug *INET* [270] durchgeführt. Details zu den jeweiligen Simulationsparametern der Studien finden sich im Anhang C.

4.9.1 ARQ-Verfahren

Im Folgenden wird anhand einer simulativen Studie belegt, dass das ARQ-Verfahren von *gAUDIT* durch den Einsatz von Sendewiederholungen eine größere Response-Vollständigkeit im Gegensatz zu verwandten Arbeiten, wie zum Beispiel *Survey* [284, 283], erreicht. Dies liegt vor allem daran, dass vergleichbare Arbeiten die hier vorgestellten Klassen für Response-Vollständigkeit nicht mit berücksichtigen.

4.9.1.1 Simulationsaufbau

Das zu *OMNeT++* gehörende Modellierungswerkzeug *INET* [270] unterstützt in der Standardvariante derzeit keine Modellierung von Schicht-2 spezifischen Multicastgruppen auf Netz-Ebene, wie etwa die Unterstützung für das *MLD*-Protokoll [216, 224] oder *IGMP* [212] für Switches, so dass *MLD snooping* oder *IGMP snooping* [223] nicht realisierbar sind. Da *gAUDIT* für die effiziente Realisierung von Multicast allerdings auf solche Funktionalität im Netz angewiesen ist, wurde zur Durchführung der Simulation die existierende *INET*-Variante entsprechend erweitert – hier für IPv6-Multicast (siehe auch [TT-3, TT-8]).

Des Weiteren wurde in *OMNeT++* die in Abbildung 4.22 dargestellte Cloud-Infrastruktur modelliert. Zur Modellierung der Netztopologie einer Cloud-Infrastruktur wurde sich an der *Clos-artigen* [40] Topologie von *Google Jupiter* [168] orientiert. Die modellierte Topologie besteht aus *100* physischen Cloud-Knoten mit jeweils einem simulierten Monitoring-Agent des *CloudInspectors* (insgesamt *100* Monitoring-Agenten). Die 100 physischen Cloud-Knoten sind dabei gleichmäßig auf *20* Serverracks abgebildet. Zusätzlich wurde der Monitoring-Controller des *CloudInspectors* in *OMNeT++* modelliert. Somit ergeben sich für *gAUDIT* eine Initiator-Komponente (für den Monitoring-Controller) und *100* Empfänger-Komponenten (für die Monitoring-Agenten auf den physischen Cloud-Knoten).

Für die Modellierung von Cloud-Ereignissne wurde ein CMS in *OMNeT++* mit der Basis-Funktionalität von Starten, Stoppen und Migration von VMs abgebildet. Zur Modellierung des Ausbleibens von Response-Nachrichten auf Request-Nachrichten – sei es durch Verlust auf Anwendungsebene innerhalb eines Monitoring-Agenten, durch Paketverlust auf dem Transportweg zwischen Empfänger- und Initiator-Komponente oder durch Verlust einer Response-Nachricht in umgekehrter Richtung – wurde das *Gilbert-Elliott Fehlermodel* [50, 81] mit einstellbarer zufälliger Fehlerrate in *OMNeT++* umgesetzt. Um die entstehenden Effekte besser analysieren zu können, werden jeweils nur die Links direkt zwischen *Respondents* und Top-of-Rack-Switches mit der Fehlerrate belegt. Alle anderen Links arbeiten in der Simulation fehlerfrei.

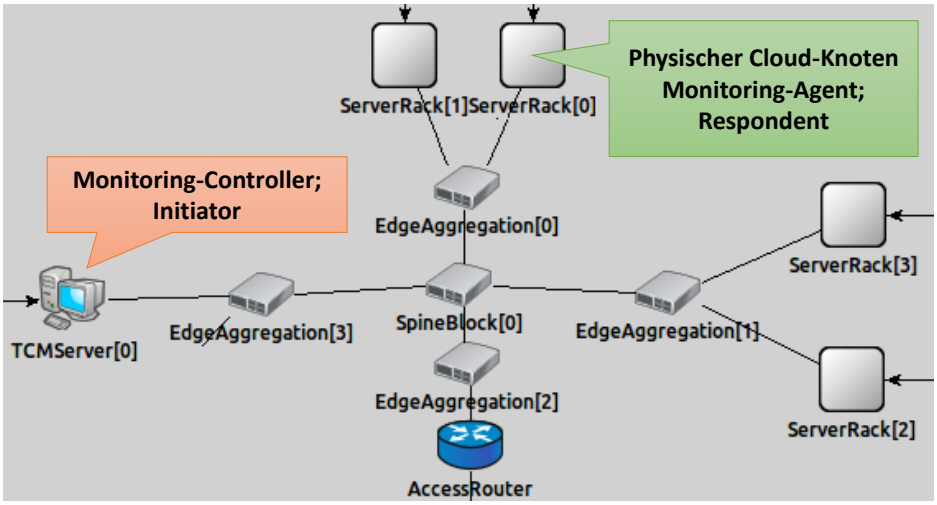

Abbildung 4.22: *Modellierung einer Cloud-Infrastruktur mit installiertem CloudInspector. Quelle [TT-3].*

Um die simulativ erzeugten Messergebnisse zu validieren, wird zusätzlich der Versuchsaufbau mit *Mininet* [281] nachgestellt.

4.9.1.2 Durchführung

Folgende Ergebnisse beziehen sich auf die g1RRS-Klasse *Statistisch-zuverlässig* mit $a = 1$, das heißt 100%-Auswahlwahrscheinlichkeit je Gruppenmitglied. Dabei wird die Performance des *gAUDIT*-ARQ-Verfahrens mit dem *Survey*-Dienst von *Nanomsg* [284, 283] verglichen. Für die Erfolgsquote von *gAUDIT* unter Einsatz einer verschieden großen Anzahl an Sendewiederholungen wird zusätzlich mit Hilfe von Bernoulli-Experimenten der Erwartungswert der Simulation mit angegeben (vergleiche [117] S. 63).

Für eine Bernoulli-Kette wird bei N-maliger Wiederholung eines Bernoulli-Experiments mit Wahrscheinlichkeit p die Eintrittswahrscheinlichkeit für k-Treffer wie folgt berechnet:

$$B(N;p;k) = \binom{N}{k} \cdot p^k (1-p)^{N-k} = \frac{N!}{(N-k)!k!} \cdot p^k (1-p)^{N-k}$$

Soll, wie zur Evaluation von *gAUDIT*, berechnet werden, wie hoch die Wahrscheinlichkeit ist, dass bei einer Fehlerrate von 1% sowohl die Request-Nachricht als auch die Response-Nachricht zuverlässig übertragen werden, so ergibt sich eine Bernoulli-Kette der Länge $N = 2$. Allerdings muss für p in die oben stehende Formel zur Berechnung der Erfolgswahrscheinlichkeit die Gegenwahrscheinlichkeit zur Fehlerrate eingesetzt werden. Die beträgt hier $p = 1 - 0.01$. Dann kann durch Setzen der Variablen $k = 2$ berechnet werden, wie hoch die Erfolgswahrscheinlichkeit ist, dass sowohl die Request- als auch

die Response-Nachricht erfolgreich übertragen wird. Für die Fehlerrate r ergibt sich die folgende Formel zur Berechnung, wobei $p = 1 - r$ ist:

$$B(2; 1-r; 2) = \binom{2}{2} \cdot p^2 (1-p)^{2-2} = \frac{2!}{(2-2)!2!} \cdot p^2 = p^2$$

Da in dieser Studie jeweils 100 Respondents abgefragt werden, muss diese Bernoulli-Kette 100-mal ausgeführt werden, wodurch sich die folgende Formel zur Berechnung der Erfolgswahrscheinlichkeit für einen erfolgreichen Durchgang bei Fehlerrate r ergibt:

$$P(r) = \prod^{100} (1-r)^2 = (1-r)^{200}$$

Soll nun die Erfolgswahrscheinlichkeit S bei Einsatz einer gewissen Anzahl an Sendewiederholungen t für m Respondents bei Fehlerrate r berechnet werden $S(m; t; r) = \prod^m P(r; t)$, so wird die folgende rekursive Rechenvorschrift $P(r; t)$ angewandt:

$$P(r; t) = \begin{cases} (1-r)^2 + (1-(1-r))^2 \cdot (1-r)^2 & t = 1 \\ P(r; t-1) + (1 - P(r; t-1)) \cdot (1-r)^2 & sonst \end{cases}$$

Durch die Rechenvorschrift wird für jede Sendewiederholung jeweils die Wahrscheinlichkeit, dass kein Fehler auftritt $(1-r)^2$ mit der Wahrscheinlichkeit dafür, dass im Fehlerfall $(1-(1-r))^2$ durch Einsatz einer Sendewiederholung die Nachrichten erfolgreich übertragen werden können, $(1-(1-r))^2 \cdot (1-r)^2$ addiert. Durch Rekursion lässt sich die Erfolgswahrscheinlichkeit für eine beliebige Anzahl von Sendewiederholungen berechnen.

In Abbildung 4.23 wird der Einfluss von Sendewiederholungen auf die Erfolgswahrscheinlichkeit von *gAUDIT* bei unterschiedlicher Fehlerquote für Response-Nachrichten untersucht. Im Vergleich mit *Survey* zeigt sich deutlich, dass bereits der Einsatz einer einzelnen Sendewiederholung in der Simulation wie zu erwarten zu einer deutlich höheren Zuverlässigkeit bei Nachrichtenverlust führt. Während beispielsweise bei *Survey* bereits bei einer durchschnittlichen Fehlerquote von 0.5% nur knapp *50%* der *1.000* durchgeführten Request-Response-Vorgänge erfolgreich sind, sind es bei *gAUDIT* mit einer Sendewiederholung nahezu *100%*. Durch Einsatz von maximal vier Wiederholungen lässt sich so bereits eine durchschnittliche Fehlerquote von *2%* abdecken.

Zwar ist in Datenzentren nicht mit einer so hohen Fehlerquote auf dem Übertragungsweg zu rechnen, dennoch kann es durch kurzfristige Lastspitzen auch zu Verlust von Response-Nachrichten im Monitoring-Agent kommen.

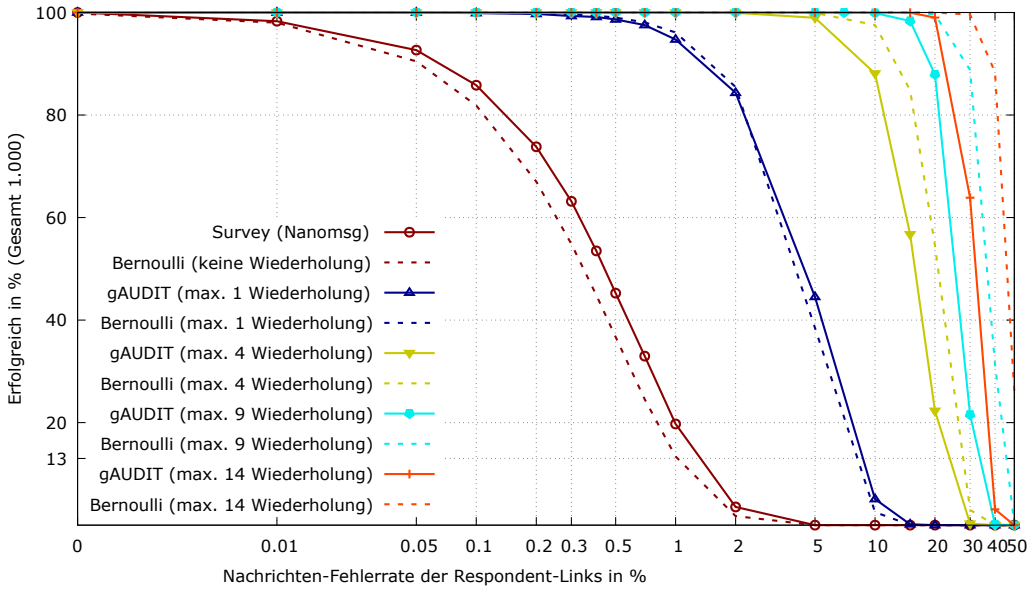

Abbildung 4.23: *Auswirkung von ausbleibenden Audit-Antworten bei Einsatz von unterschiedlich vielen Sendewiederholungen. Quelle [TT-3].*

Weiter zeigen die Evaluationsergebnisse, dass der Einsatz von Sendewiederholungen die Gesamtausführungszeit von *gAUDIT* erhöht. Da in dieser Simulation der Zeitgeber für Sendewiederholungen fest auf *100ms* gesetzt wurde, erhöht sich die Ausführungszeit von *gAUDIT* je nach zum Einsatz kommender Anzahl von Sendewiederholungen entsprechend. In Abbildung 4.24 werden die Ergebnisse der Studie für unterschiedliche Anzahl an Sendewiederholungen dargestellt. Hierbei werden nur Szenarien berücksichtigt, bei denen *gAUDIT* erfolgreich den Request-Response-Vorgang abschließen konnte, weswegen nicht für alle Fehlerquoten Messwerte vorliegen. Beispielsweise ist bei einer durchschnittlichen Fehlerquote von *20%* kein einziger Request-Response-Vorgang erfolgreich gewesen.

4.9.1.3 Ergebnisse

Die Fehlerbehandlung von *gAUDIT* ist im Gegensatz zu verwandten Arbeiten in der Lage, die angestrebte g1RRS-Klasse (hier *Statistisch-zuverlässig*) zu erreichen. Beim Einsatz im Datenzentrum sind aufgrund der erwarteten geringen Verlustwahrscheinlichkeit von Response-Nachrichten dafür in der Regel nur wenige Sendewiederholungen notwendig.

So wirkt sich einerseits das zum Einsatz kommende ARQ-Verfahren positiv auf Realisierung der eingestellten Response-Vollständigkeit aus. Andererseits wirkt sich die erhöhte Ausführungszeit von *gAUDIT* bei Abdeckung fehlender Response-Nachrichten negativ auf die durch das Monitoring erreichbare Konsistenz aus Abschnitt 6.6.1 aus. Da allerdings

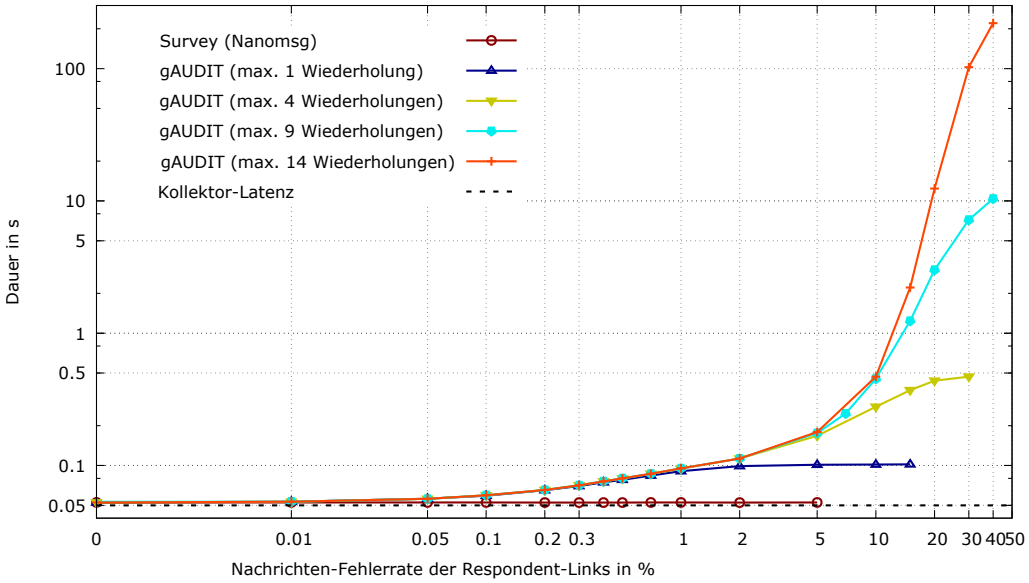

Abbildung 4.24: *Auswirkung von ausbleibenden Response-Nachrichten auf die gAUDIT-Ausführungsdauer bei Einsatz unterschiedlich vieler Sendewiederholungen. Quelle [TT-3].*

im Datenzentrum durch die geringe Verlustwahrscheinlichkeit nur wenige Sendewiederholungen notwendig sind, sind keine gravierenden Auswirkungen auf die Konsistenz der Messergebnisse zu erwarten.

4.9.2 Auswahl zwischen Multicast und Unicast

Um zu untersuchen, wann die Wahl von Multicast gegenüber Unicast überwiegt (vergleiche Abschnitt 4.6.3), wird die folgende Parameterstudie durchgeführt. Als Metrik für die Effizienz eines Ansatzes werden die benötigten Netzressourcen gegenübergestellt. Hierfür werden die übertragenen UDP-Segmente gemessen, wobei auf jedem Übertragungsabschnitt (beispielsweise zwischen Aggregate- und ToR-Switch) übertragene UDP-Segmente mitgezählt werden.

Ziel der folgenden Parameterstudie ist es, die geeignete Wahl eines Parameters s zu ermitteln, der festlegt, ab welchem Verhältnis von fehlenden Antworten $|\mathbb{M}|$ im Bezug zur Gesamtgruppengröße $|\mathbb{G}|$ der Einsatz von Multicast sinnvoll ist. Der für s ermittelte Wert beeinflusst dann das Verhalten von *gAUDIT* sowohl initial als auch bei der Durchführung

von Sendewiederholungen. In Abhängigkeit von s ergibt sich die folgende Funktion zur Auswahl zwischen Multicast und Unicast:

$$f(s) = \begin{cases} \texttt{Multicast} & |\mathbb{M}| \geq \frac{|\mathbb{G}| \cdot s}{100} \\ \texttt{Unicast} & \texttt{sonst} \end{cases}$$

Wenn beispielsweise *gAUDIT* mit der g1RRS-Klasse *Statistisch-zuverlässig* mit $a = 0.5$ verwendet wird, so ist für eine Gesamtgruppengröße von $|\mathbb{G}| = 100$ zu erwarten, dass initial die Anzahl fehlender Antworten im Mittel $|\mathbb{M}| = 50$ beträgt. Wird nun der Parameter s auf den Wert $s = 10$ gesetzt, so bedeutet dies, dass sobald mehr Antworten als *10%* der Gesamtgruppengröße fehlen, Multicast verwendet werden soll. Dies ist in diesem Beispiel der Fall, da $|\mathbb{M}| \geq \frac{|\mathbb{G}| \cdot 10}{100}$ gilt. Wird hier allerdings der Parameter s auf einen Wert größer 50 gesetzt, so wird in diesem Beispiel Unicast verwendet, da $|\mathbb{M}| \geq \frac{|\mathbb{G}| \cdot s}{100}$ nicht mehr gilt. An dieser Stelle sei noch einmal darauf hingewiesen, dass die Auswahl zwischen Multicast und Unicast bei *gAUDIT* sowohl initial als auch für Sendewiederholungen durchgeführt wird. Wenn beispielsweise der Parameter s auf den Wert 10 gesetzt ist und zunächst Multicast verwendet wird, aber durch Nachrichtenverlust eine einzelne Response-Nachricht verloren geht, so wird die Sendewiederholung per Unicast durchgeführt, da die Größe der Menge der fehlenden Antworten $\mathbb{M}$ im Verhältnis zur Gesamtgruppengröße $|\mathbb{G}|$ zum Zeitpunkt der Sendewiederholung weniger als 10% beträgt – hier fehlt dann eine einzelne Nachricht, was in diesem Fall 1% entspricht.

4.9.2.1 Simulationsaufbau

Dem Stand der Wissenschaft folgend wird die Netztopologie der Cloud-Infrastruktur hier als *Clos-artige* [40] Topologie modelliert [89, 192, 185, 130, 168, 104]. Als Vereinfachung wird davon ausgegangen, dass der Initiator immer über einen *Spine-Switch* seine Anfragen an die *Respondents* senden muss.

In Abbildung 4.25 wird ein Ausschnitt der Netztopologie eines Datenzentrums zur Verdeutlichung des Simulationsaufbaus gezeigt. In dem Beispiel wird eine *Gruppe* von vier *Respondents* dargestellt, die auf drei Serverschränke (*Racks*) des Datenzentrums aufgeteilt ist, wobei zwei *Respondents* sich im selben Serverschrank befinden. Sowohl zwischen den *Spine-Switches* und *Aggregate-Switches* als auch zwischen *Top-of-Rack-Switch* und physischen Cloud-Knoten existieren redundante Verbindungen im Datenzentrum (vergleiche beispielsweise [168]).

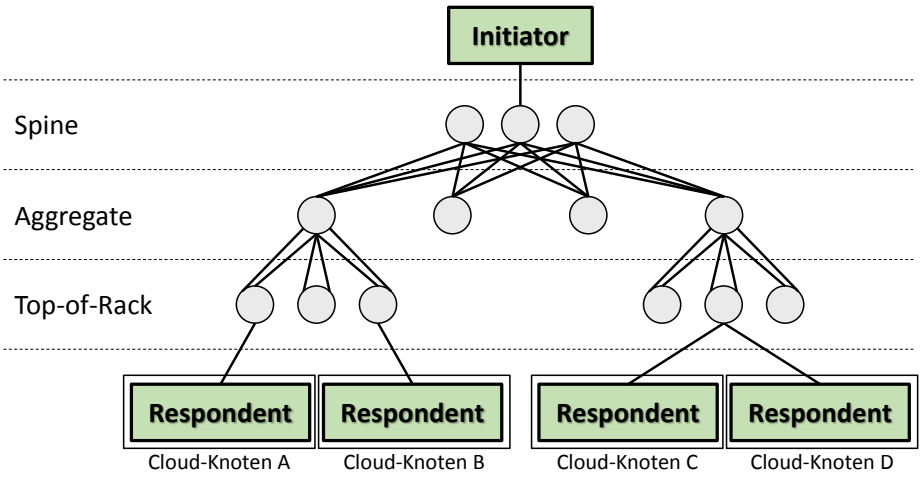

Abbildung 4.25: *Ausschnitt der Netztopologie eines Datenzentrums mit gAUDIT-Komponenten.*

4.9.2.1.1 Multicastverteilbäume

Neben der Topologie des Datenzentrums ist die Struktur des darin je *Gruppe* erzeugten Multicastverteilbaums entscheidend. Prinzipiell wird mit Hilfe eines Multicast-Routing-Protokolls [238, 207, 218, 205, 202, 90] ein Multicastverteilbaum mit kürzesten Pfaden zu den Gruppenmitgliedern ermittelt [136, 90]. Da auf der untersten Ebene eines Multicastverteilbaums (auf Ebene der physischen Cloud-Knoten) kein Einsparungspotential für Multicast vorhanden ist, wird für den Simulationsaufbau von *Top-of-Rack-Switches* abstrahiert. In Abbildung 4.26 wird beispielhaft gezeigt, wie der sich in der Netztopologie ergebende Multicastverteilbaum in der Simulation modelliert wird. Bei dem sich so ergebenden Multicastverteilbaum befinden sich die Respondents immer an den Blättern mit Tiefe $\omega = 3$ (für Multicastverteilbaum ab *Spine-Switch*).

Die Berechnung eines optimalen Multicastverteilbaums für eine gegebene Netztopologie lässt sich auf das *Steinerbaumproblem* zurückführen, was als NP-vollständiges Problem in der Literatur bekannt ist [92]. Aus diesem Grund wird in dieser Dissertation die Berechnung von Multicastverteilbäumen für gegebene Netztopologien explizit ausgeklammert. Stattdessen werden die in der Parameterstudie verwendeten Multicastverteilbäume so erzeugt, dass die Bandbreite zwischen *Best-* und *Worst-Case-Szenarien* für Multicast abgedeckt ist. Ein Multicastverteilbaum wird als *Best-Case-Szenario* bezeichnet, sofern sich durch den Einsatz von Multicast im Vergleich zu Unicast maximal viele Nachrichten einsparen lassen. Analog dazu wird ein Multicastverteilbaum als *Worst-Case-Szenario* bezeichnet, sofern sich durch den Einsatz von Multicast keine Nachrichten mehr einsparen lassen. Hierfür wird ähnlich wie in [38, 152] angenommen, dass die sich für Multicast

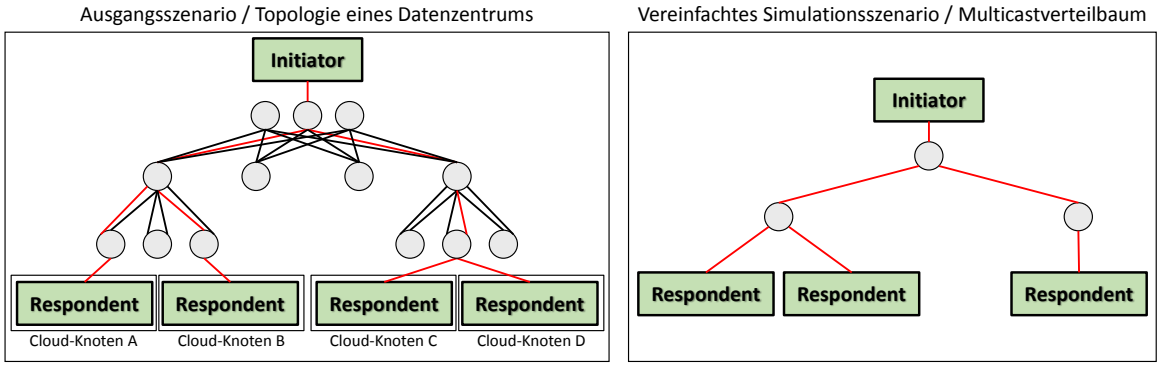

Abbildung 4.26: *Beispielhafte Gegenüberstellung der Netztopologie und des simulierten Multicastverteilbaums.*

ergebenden kürzesten Pfade deckungsgleich mit denen für Unicast sind und dass das Routing jeweils symmetrisch ist. In Abbildung 4.27 wird für drei Gruppenteilnehmer beispielhaft das Best- dem Worst-Case-Szenario gegenübergestellt.

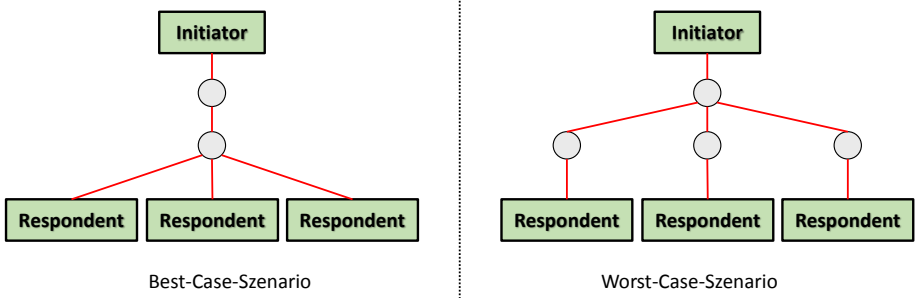

Abbildung 4.27: *Beispielhafte Gegenüberstellung der Multicastverteilbäume für das Best- und Worst-Case-Szenario mit drei Gruppenteilnehmern.*

Bei einer Auswahlwahrscheinlichkeit von 100% und keinerlei Paketverlust können im Best-Case-Szenario im Vergleich mit Unicast bei Multicast acht UDP-Segmente eingespart werden. Im Worst-Case-Szenario ist die für Multicast benötigte Anzahl an UDP-Segmenten deckungsgleich mit der für Unicast. In diesem Beispiel werden 18 UDP-Segmente für den gesamten Request-Response-Vorgang, von Versand der Request-Nachricht an drei Gruppenmitglieder über drei Übertragungsabschnitte (sechs UDP-Segmente) und je Gruppenmitglied eine Response-Nachricht über je drei Übertragungsabschnitte (sechs UDP-Segmente), benötigt. Entscheidend für das Einsparungspotential bei Multicast ist die Pfadüberdeckung von der Wurzel zu den Blättern [31, 32, 178]: je mehr Respondents gebündelt sind, desto höhere Pfadüberdeckung ergibt sich.

Zur Beurteilung der in der Parameterstudie verwendeten Multicastverteilbäume wird der Floyd-Warshall-Algorithmus [94, 60, 187] herangezogen (siehe Abbildung 4.28). Dieser

berechnet in $\Theta(|V|^3)$ auf Basis einer Adjazenzmatrix A eines Graphen G die kürzesten Pfade zwischen allen Paaren von Knoten (*All-Pairs Shortest Paths*).

```
for k in |V| do
    for i in |V| do
        for j in |V| do
            amx[i,j]=min(amx[i,j], amx[i,k]+amx[i,j])
        end
    end
end
```

Abbildung 4.28: *Floyd-Warshall-Algorithmus zur Berechnung der kürzesten Pfade zwischen allen Paaren von Knoten eines Graphen (All-Pairs Shortest Paths); amx für Adjazenzmatrix; V für Knotenmenge.*

Zur Berechnung der kürzesten Pfade wird die Adjazenzmatrix eines erzeugten Multicastverteilbaums mit Pfadkosten von eins je benachbartem Knoten vorbelegt. Alle anderen Verbindungen werden mit ∞ instantiiert. Zur Beurteilung des Multicastverteilbaums ist nach Ablauf des Floyd-Warshall-Algorithmus insbesondere die Summe der Pfadkosten aller Blätter untereinander interessant. Diese gibt Aufschluss darüber, wie gebündelt die *Respondents* zueinander sind. Beispielsweise ist für den in Abbildung 4.29 dargestellten Graphen G die Summe der Pfadkosten aller Blätter untereinander aus der Blätter-Teilmatrix hier 12. Im Folgenden wird `lmx` als Bezeichner für diese Blätter-Teilmatrix verwendet.

Im Allgemeinen wird die Summe der Pfadkosten K aller Blätter L nach Ablauf des Floyd-Warshall-Algorithmus wie folgt berechnet.

$$K = \sum_{i=0}^{|L|} \sum_{j=0}^{|L|} \texttt{lmx}[i,j]$$

Anhand dieser Summe wird nun beurteilt, ob ein Best- oder Worst-Case-Szenario für Multicast vorliegt. Hierzu wird die so ermittelte Summe K ins Verhältnis zur Summe der Pfadkosten für den Fall, dass alle Blätter gebündelt sind, gesetzt. Sind alle $|L|$ Blätter gebündelt, so ergibt sich die Summe für die Pfadkosten untereinander von $2 \cdot |L| \cdot (|L|-1)$. Durch die vordefinierten Pfadkosten von *1* erreicht jedes Blatt $l \in L$ alle anderen Blätter $k \in L \backslash l$ mit Pfadkosten von 2. Mit Hilfe der Beurteilungsfunktion $E()$ kann nun für

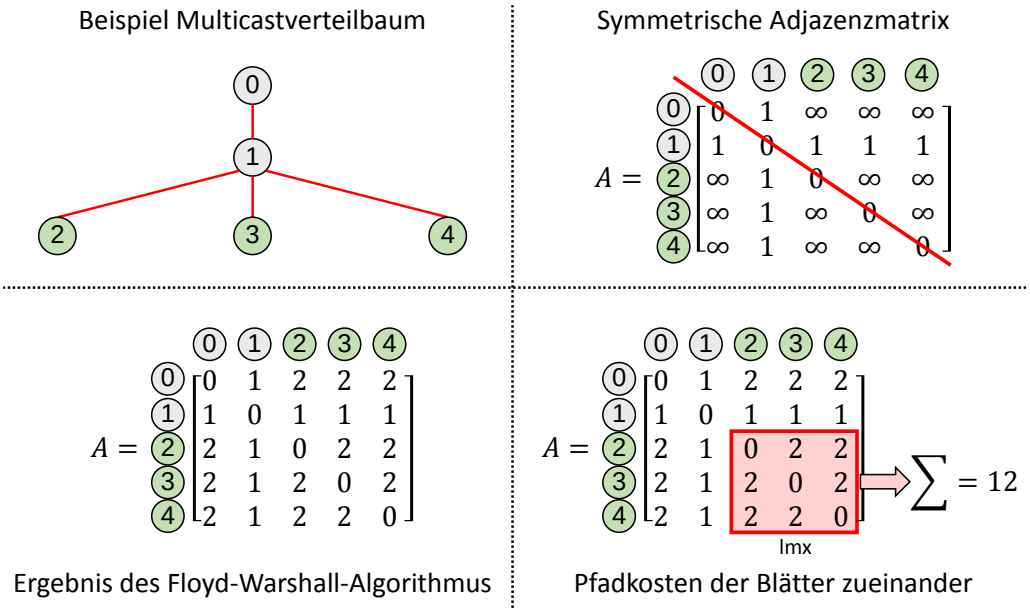

Abbildung 4.29: *Erläuterung, wie für die Simulation verwendete Multicastverteilbäume mit Hilfe des Floyd-Warshall-Algorithmus und Berechnung der Pfadkosten aller Blätter untereinander beurteilt werden.*

Graphen ermittelt werden, wie sehr diese vom Best-Case-Szenario (alle Blätter sind gebündelt) abweichen.

$$E(G) = \frac{\sum_{i=0}^{|L|} \sum_{j=0}^{|L|} \texttt{lmx}[i,j]}{2 \cdot |L| \cdot (|L|-1)}$$

Im Best-Case-Szenario ergibt sich für $E()$ ein Wert von 1.0, im Worst-Case-Szenario ein Wert von 2.0. Der Wert von 2.0 ergibt sich für Worst-Case-Szenarien, da durch die Vorgabe, dass alle Blätter eine Tiefe von $\omega = 3$ aufweisen müssen, die maximalen Pfadkosten zu jedem anderem Blatt 4 betragen. Somit ergibt sich die Summe der Pfadkosten von $4 \cdot |L| \cdot (|L|-1)$, was gerade dem doppelten der Summe der Pfadkosten, in dem Fall dass alle Blätter gebündelt sind, entspricht $(2 \cdot |L| \cdot (|L|-1))$.

Damit die für die Parameterstudie verwendeten Multicastverteilbäume verschiedene Szenarien von Best- bis Worst-Case abdecken, wird der folgenden Algorithmus `gen_mdt(N,W)` zur Erzeugung angewandt. Als Eingabeparameter wird eingestellt, wie viele Respondents (Blätter) N im Multicastverteilbaum enthalten sein sollen und ob eher Best- oder Worst-Case-Szenarien angestrebt werden sollen. Für letzteres wird der Parameter W im Intervall $[0, 1.0]$ verwendet. Bei Wahl von $W = 0$ wird das Worst-Case-Szenario erzeugt, für $W = 1.0$ das Best-Case-Szenario. Bei den erzeugten Multicastverteilbäumen haben alle Blätter eine Tiefe von $\omega = 3$. Der in Abbildung 4.30 beschriebene Algorithmus

`gen_mdt(N,W)` ermittelt mit Hilfe des Wertes für W in jedem Durchgang, wie viele Blätter von der Gesamtanzahl noch nicht hinzugefügter Blätter zu bündeln sind (Variablen `local` und `remote`).

```
Function gen_mdt(N,W):
    G=Graph() ► Erzeugt Graph-Objekt
    G.add_node(0) ► Erzeugt Root

    aggregate = 0
    remote = N

    while remote != 0 do
        aggregate += 1
        G.add_node(aggregate) ► Erzeugt neuen Aggregate-Switch
        G.add_edge(0,aggregate) ► Kante zwischen Root und neuem
         Aggregate-Switch
        local = int(remote · W) ► Ermittelt Anzahl zu bündelnder Blätter
        if local == 0 then
            local = 1
        end
        for i=0; i <= local; i++ do
            G.add_node(N+aggregate+i+2)
            G.add_edge(aggregate,N+aggregate+i+2)
        end
        remote -= local
    end

    return G
end
```

Abbildung 4.30: *Für die Simulation verwendeter Algorithmus zur Erzeugung von Multicastverteilbäumen. Parameter N bestimmt Anzahl der im Baum zu platzierenden Respondents als Blätter mit Tiefe $\omega = 3$. Parameter W bestimmt ob generierter Multicastverteilbaum sich dem Best- oder Worst-Case-Szenario annähern soll.*

Die durch den in Abbildung 4.30 beschriebenen Algorithmus `gen_mdt(N,W)` erzeugten Multicastverteilbäume decken bei fester Anzahl von Blättern N durch Varianz des Parameters W alle Fälle zwischen Best- und Worst-Case-Szenarien ab. In Abbildung 4.31 wird beispielhaft das Ergebnis des Algorithmus `gen_mdt(N,W)` für $N = 50$ und $W = [0.25, 0.5, 0.75]$ dargestellt. Außerdem ist die Ausgabe der Beurteilungsfunktion $E()$ für die erzeugten Multicastverteilbäume mit angegeben.

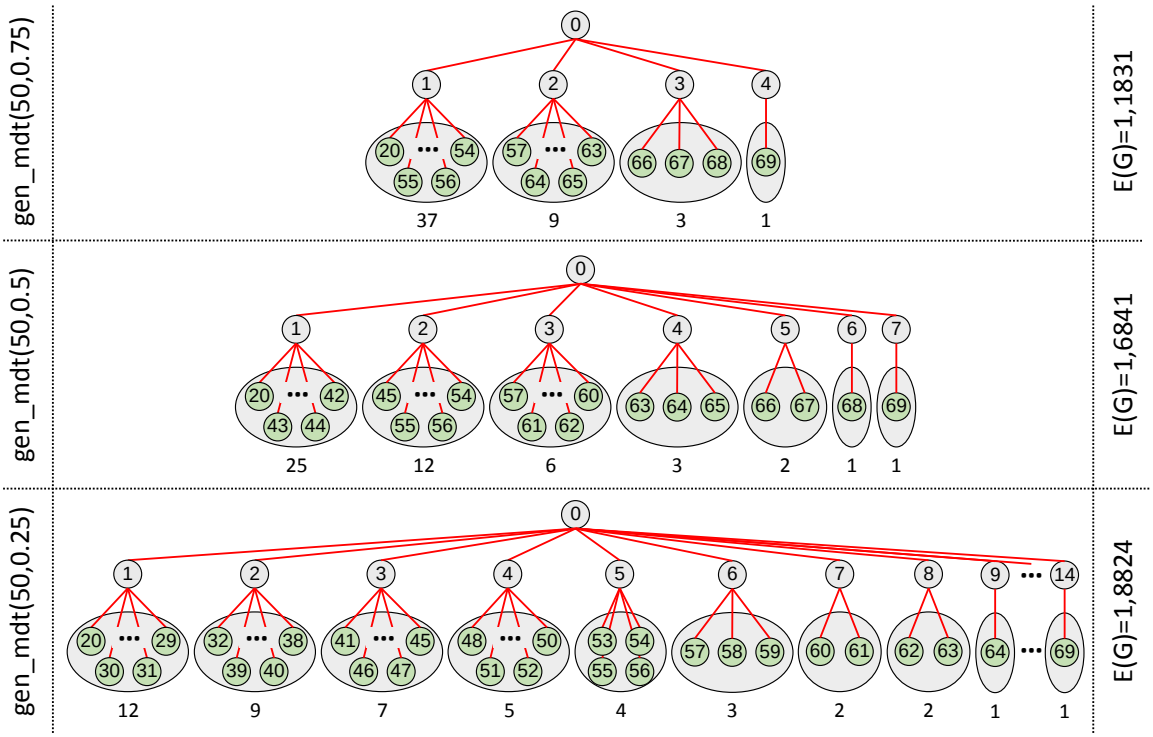

Abbildung 4.31: *Beispiel der durch* `gen_mdt` *erzeugten Multicastverteilbäume für die Simulation bei fester Anzahl von Blättern und unterschiedlichem W. Zusätzlich ist das Ergebnis der Beurteilungsfunktion $E()$ mit angegeben.*

Für die durchgeführte Simulation werden Szenarien mit zwei bis 1.002 Blättern untersucht bei jeweiliger Varianz von W im Bereich zwischen 0 und 1.0 in 0.1-Schritten. Die so erzeugten Multicastverteilbäume werden anschließend mit der Funktion $E()$ beurteilt. In Abbildung 4.32 ist auf der linken Seite das Histogramm für die erzeugten Szenarien dargestellt, aufgeteilt in die Gruppen $[1.0 \leq 1.125, 1.125 \leq 1.25, 1.25 \leq 1.375, 1.375 \leq 1.5, 1.5 \leq 1.625, 1.625 \leq 1.75, 1.75 \leq 1.875, 1.875 \leq 2.0]$. Da durch den gewählten Ansatz ein Ungleichgewicht an erzeugten Szenarien im Bereich von 1.8 bis 2.0 vorliegt, mussten in diesem Bereich einige Multicastverteilbäume für die Simulation ausgeschlossen werden. Es hat sich gezeigt, dass durch Ausschluss der Szenarien mit W im Intervall $[70;90]$ eine akzeptable Verteilung der Multicastverteilbäume erreicht werden kann (siehe rechte Seite der Abbildung 4.32).

Aus diesem Grund beziehen sich die hier vorgestellten Ergebnisse der Parameterstudie nur auf Szenarien mit Multicastverteilbäumen, die nicht mit $W = [70;90]$ erzeugt wurden.

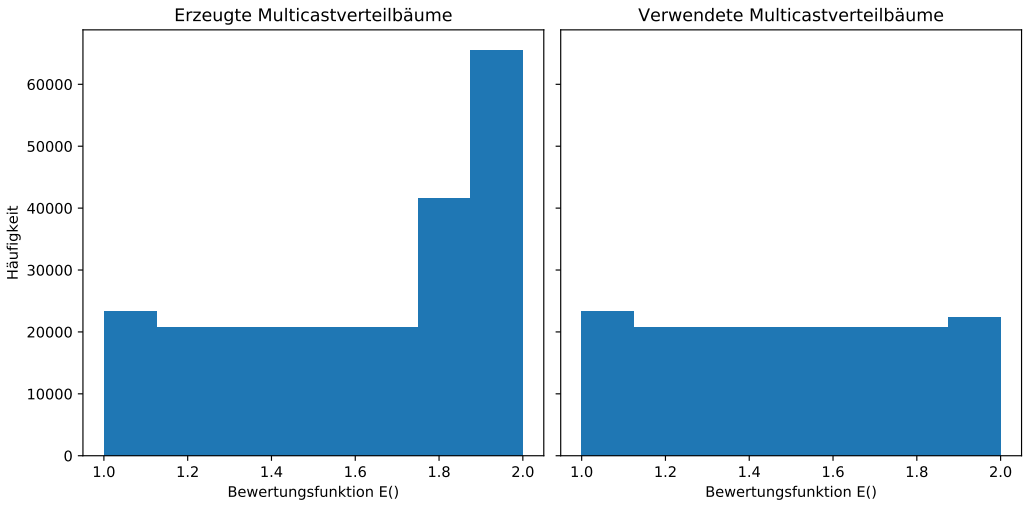

Abbildung 4.32: *Histogramme für erzeugte und in der Parameterstudie verwendete Multicastverteilbäume.*

4.9.2.1.2 Baseline-Szenario

Um die Leistungsfähigkeit der Auswahl zwischen Multicast und Unicast bei Request-Response in Gruppen für verschiedene g1RRS-Klassen beurteilen zu können, wird ein Baseline-Szenario zum Vergleich benötigt. Dabei werden die in der jeweiligen Variante von *gAUDIT* benötigten UDP-Segmente dem Baseline-Szenario gegenübergestellt. Im Folgenden wird beschrieben, wie das verwendete Baseline-Szenario aufgebaut ist.

In Szenarien ohne Nachrichtenverlust kann je nach Szenario durch den Einsatz von Multicast die Übertragung von UDP-Segmenten eingespart werden. Die für Unicast benötigte Menge an UDP-Segmenten hingegen hängt in dieser Parameterstudie alleine von der Anzahl der zu erreichenden Respondents (Blätter) ab. Folglich gelten die folgenden zwei Gleichungen, wobei $|MC|$ die für Multicast benötigten UPD-Segmente beschreibt und $|UC|$ die für Unicast.

$$|MC| \leq |UC|$$
$$|UC| \propto |L|$$

Dadurch, dass die in der Parameterstudie verwendeten Multicastverteilbäume wie oben beschrieben und in Abbildung 4.27 dargestellt strukturiert aufgebaut sind, ergibt sich für $|UC|$ hier ein Wert von $|UC| = 6 \cdot |L|$. In diesem Szenario werden für ein einzelnes Gruppenmitglied drei UDP-Segmente für die Anfrage und drei für die Antwort benötigt.

Das für den Vergleich mit *gAUDIT* herangezogene Baseline-Szenario macht sich diese Eigenschaft zu Nutze. So wird für jedes Szenario anhand der Gesamtgruppengröße ermittelt, wie viele UDP-Segmente nötig wären, um alle Gruppenmitglieder zu erreichen. Dies kann unabhängig von dem konkreten Multicastverteilbaum und dem gewählten Parameter W und g1RRS-Klasse geschehen. Anschließend werden die von *gAUDIT* benötigten UDP-Segmente dem gegenübergestellt. So kann für jedes untersuchte Szenario ein *Einsparungsfaktor* berechnet werden, der im Verhältnis zum Baseline-Szenario angibt, um welchen Faktor Netzressourcen durch den Einsatz von *gAUDIT* eingespart werden können. Der Einsparungsfaktor berechnet sich daher wie folgt:

$$\texttt{Einsparungsfaktor} = \frac{|L| \cdot 6}{\texttt{Gemessene UDP} - \texttt{Segmente}}$$

Da allerdings zur Berechnung des Einsparungsfaktors weder Nachrichtenverlust noch g1RRS-Parameter berücksichtigt werden, können sich auch Werte im Intervall *[0;1]* ergeben. Hier schneidet *gAUDIT* dann "schlechter" als das Baseline-Szenario ab, weil beispielsweise fehlende Nachrichten per Unicast nachgefordert werden.

4.9.2.1.3 Primitive Varianten

Zusätzlich zur Studie zur Parameterisierung von *s* für *gAUDIT* werden zwei primitive Varianten untersucht. Diese dienen als weiteres Vergleichskriterium für die Leistungsfähigkeit von *gAUDIT*. Dabei handelt es sich um die primitiven Varianten *Multicast* und *Unicast*.

Die primitive Multicast-Variante verwendet zum Versand von Request-Nachrichten sowohl initial als auch im Fall von Sendewiederholungen immer Multicast. Die Variante verhält sich also wie *gAUDIT* mit der Parameterisierung *s=0*. Die primitive Unicast-Variante verwendet zum Versand von Request-Nachrichten sowohl initial als auch im Fall von Sendewiederholungen immer Unicast. Die Variante verhält sich also wie *gAUDIT* mit der Parameterisierung *s=100*.

4.9.2.2 Durchführung

In der Parameterstudie werden unterschiedliche Multicastverteilbäume, Parameter für die g1RRS-Klasse *Statistisch-zuverlässig*, Anzahl an *Respondents* und Szenarien mit Nachrichtenverlust untersucht. Für Details zu den Simulationsparametern siehe Anhang C. Ziel der Parameterstudie ist es einen geeigneten Parameter für *s* zu finden.

Im linken Teil der Abbildung 4.33 ist die empirische Verteilungsfunktion für den Einsparungsfaktor gegenüber dem Baseline-Szenario über alle 170.560 untersuchten Szenarien für *gAUDIT* mit unterschiedlichem Parameter s dargestellt. Zusätzlich ist der Einsparungsfaktor für die zwei primitiven Varianten aufgetragen. Dabei gilt, je mehr Szenarien durch einen größeren Einsparungsfaktor abgedeckt sind, umso besser.

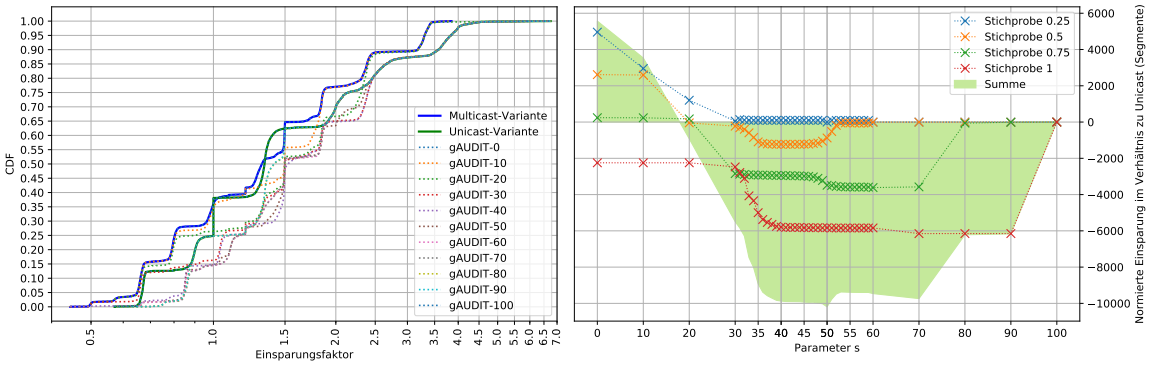

Abbildung 4.33: *Ergebnisse der Parameterstudie bei unterschiedlicher Wahl des Parameters s. Links: Aggregierte Ergebnisse über alle untersuchten Szenarien. Rechts: Normierte Einsparung von gAUDIT.*

Im rechten Teil der Abbildung 4.33 erfolgt eine detaillierte Betrachtung für unterschiedliche Wahl des Parameters s und Auswahlwahrscheinlichkeit für die g1RRS-Klasse *Statistisch-zuverlässig* (Stichprobe). Der Vorteil von *gAUDIT* wird hier im Vergleich zur primitiven Unicast-Variante normiert. Zur Normierung wird der in [38, 152] ermittelte Faktor $N^{0.8}$ für N gleich der Anzahl der Gruppenmitglieder verwendet. Dargestellt ist das nach der Normierung mit Hilfe der Trapezregel bestimmte Integral $\int_a^b f(x)\mathrm{d}x$ im Intervall $[0, 7]$ (vergleiche linke Seite) je Stichprobengröße. Negative Werte bedeuten hier eingesparte UDP-Segmente. Das heißt die *gAUDIT*-Variante ist effizenter als die primitive Unicast-Variante. Zusätzlich dargestellt ist eine detailliertere Abstufung für s im Intervall $[30; 60]$, so dass insgesamt 414.592 Szenarien untersucht werden.

Wird eine Response-Nachricht von jedem Gruppenmitglied benötigt (Stichprobengröße 1), so zeigt sich, dass durch jede Belegung des Parameters s UDP-Segmente im Vergleich zur primitiven Unicast-Variante eingespart werden können. Für andere Stichprobengrößen ergibt sich ein etwas differenzierteres Bild. Beispielsweise kann es bei Stichprobengrößen von 0.5 und 0.25 durch ungeschickte Wahl des Parameters s zu erhöhter Nutzung von Netzressourcen kommen (hier für $s = 0$ oder $s = 10$). Insgesamt zeigt der rechte Teil der Abbildung 4.33, dass die Summe der Einsparungen für verschiedene Stichprobengrößen bei Belegung von $s = 50$ am Niedrigsten ist (größtes Einsparungspotential).

Die Detailansicht in Abbildung 4.34 zeigt, dass während die primitive Unicast-Variante in rund 38% aller untersuchten Szenarien und die primitive Multicast-Variante in rund 35% aller Fälle schlechtere Ergebnisse als das Baseline-Szenario liefert, *gAUDIT* mit Parameter $s = 50$ nur in rund 15% der Fälle schlechtere Ergebnisse liefert.

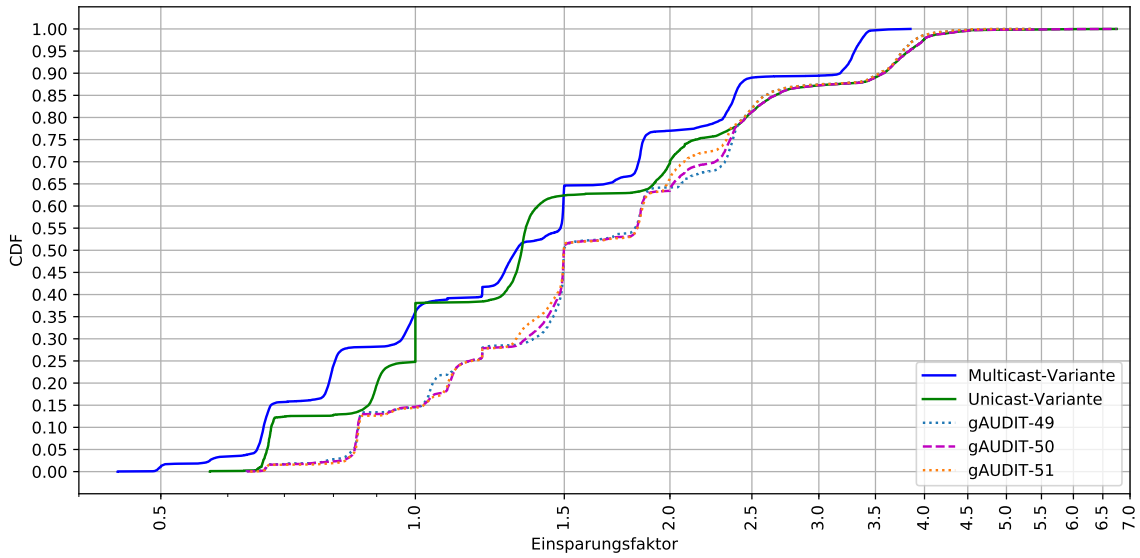

Abbildung 4.34: *Detaillierte Darstellung für Wahl des Parameters s von* 49*,* 50 *und* 51 *im Vergleich zur primitiven Unicast- und Multicast-Variante.*

Außerdem wird in Abbildung 4.34 deutlich, dass während die primitive Multicast-Variante nur in der Lage ist, einen maximalen Einsparungsfaktor von knapp 3.8 zu erreichen, *gAUDIT* mit Parameter $s = 50$ einen Einsparungsfaktor von über 6.6 erreichen kann. Außerdem fällt auf, dass die Kurve für *gAUDIT* ab einem Einsparungsfaktor von 2.5 deckungsgleich mit der primitiven Unicast-Variante ist. Zum Vergleich sind in der Abbildung außerdem die Werte für die Parameterisierung von s mit 49 und 51 aufgetragen.

Der hohe maximale Einsparungsfaktor für Unicast und *gAUDIT* gegenüber der primitiven Multicast-Variante und dem Baseline-Szenario ergibt sich durch die Szenarien, bei denen nur eine kleine Stichprobe aus der Gesamtmenge der Gruppenteilnehmer gezogen wird. Sowohl *gAUDIT* als auch die primitive Unicast-Variante sind hier effizienter, da gezielt nur die *Respondents* befragt werden, die Teil der Stichprobe sind und nicht wie im Baseline-Szenario alle Gruppenmitglieder. Der Vorteil gegenüber der primitiven Multicast-Variante ergibt sich zusätzlich wenn es zu Nachrichtenverlust kommt. Während *gAUDIT* und die primitive Unicast-Variante gezielt fehlende Nachrichten nachfordern können, erfolgt in der primitiven Multicast-Variante die Sendewiederholung auch per Multicast, was unnötige Wiederholungen bereits erhaltener Response-Nachrichten und somit zusätzliche Nutzung von Netzressourcen zur Folge hat.

4.9.2.3 Ergebnisse

Die hier durchgeführte Parameterstudie zeigt, dass im Allgemeinen der Parameter *s* beim Einsatz von *gAUDIT* auf 50 gesetzt werden sollte, da hier die größten Effizienzgewinne zu erwarten sind. Allerdings zeigt die Betrachtung auch, dass der Vorteil von *gAUDIT* gegenüber den klassischen Verfahren auch von der zu ermittelnden Stichprobengröße abhängt. Eine ungünstige Wahl des Parameters *s* (zum Beispiel kleiner 30) kann im Fall einer Stichprobengröße von 0.5 oder 0.25 zu schlechterem Verhalten als Unicast führen. Auch bei zu großer Wahl des Parameter *s* verschwindet das Einsparungspotential von *gAUDIT*, beispielsweise bei den Stichprobengrößen von 0.25 bis 0.75 und einem Parameter größer 80.

Aus diesem Grund lässt sich aus der hier durchgeführten Parameterstudie ableiten, dass der Parameter *s* im Bereich zwischen 40 und 70 gewählt werden sollte, wobei sich bei der Wahl von 50 das größte Einsparungspotential gegenüber dem Baseline-Szenario ergibt.

4.9.3 Incast-Verhalten

Ein weiteres wichtiges Evaluationskriterium für den Einsatz von *gAUDIT* in Datenzentren ist die Untersuchung von *Incast-Szenarien* [192, 78]. In der Literatur wird als *Incast-Problematik* das nahezu synchronisierte Versenden von Antwort-Nachrichten einer großen Gruppe von Servern an einen einzelnen Client bezeichnet (n:1-Kommunikation). Durch die Synchronisation der Antwort-Nachrichten kommt es auf dem Weg zum Empfänger durch übergelaufene Warteschlangen in Zwischensystemen zu Nachrichtenverlust [6, 33]. Wenn verlorene Nachrichten erneut übertragen werden müssen und nicht für Desynchronisierung gesorgt wird, kommt es zum sogenannten Netzkollaps: die Nachrichten können aufgrund der immer wieder überlaufenden Warteschlangen nie zum Empfänger übertragen werden. Aus diesem Grund ist *gAUDIT* so auszulegen, dass Antwort-Nachrichten nicht synchronisiert von den *Respondents* zum *Initiator* übertragen werden und dass gleichzeitig eine niedrige Gesamtausführungszeit erreicht wird.

Die folgende Parameterstudie untersucht, wie viel Zufallsverzögerung [6, 61] auf Seiten der *Respondents* zur Desynchronisierung der Response-Nachrichten erforderlich ist, so dass Nachrichtenverlust durch übergelaufene Warteschlangen vermieden wird und gleichzeitig eine niedrige Gesamtausführungszeit erreicht wird. Die Untersuchung beschränkt sich dabei auf die oben als optimal ermittelte *gAUDIT*-Variante mit Parameter $s = 50$. Insgesamt werden *191.879* Szenarien untersucht.

4.9.3.1 Simulationsaufbau

Im Gegensatz zu der in Abschnitt 4.9.2 beschriebenen Analyse werden hier nur Multicastverteilbäumen mit gebündelten *Respondents* untersucht. In diesen wird die Synchronisierung von Response-Nachrichten am ehesten zum *Incast-Problem*, da alle *Respondents* über den selben Top-of-Rack-Switch und Aggregate-Switch an den *Initiator* senden. Für diese gebündelten Szenarien wird die Anzahl der Respondents im Multicastverteilbaum variiert. Abbildung 4.35 zeigt die für diesen Teil der Parameterstudie ausgewählte Art der Multicastverteilbäume.

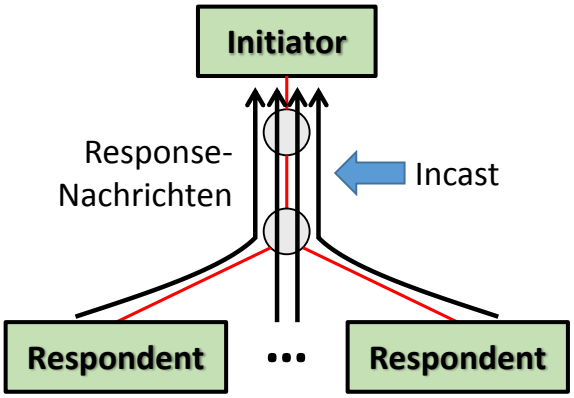

Abbildung 4.35: *Multicastverteilbaum mit variabler Anzahl an gebündelten Respondents zur Untersuchung von Incast-Verhalten.*

Zusätzlich zur vorangegangenen Simulation wird für diese Parameterstudie die Ausführungszeit von *gAUDIT* und die Anzahl an durch *Incast* in Warteschlangen verlorener *gAUDIT*-Nachrichten ermittelt. Außerdem werden hier verschiedene Strategien zur Desynchronisierung von Response-Nachrichten gegenübergestellt und deren Auswirkung auf die Gesamtausführungszeit analysiert. Der Zeitgeber für Sendewiederholungen auf Seiten des *Initiators* wurde hier fest auf einen relativ hohen Wert von einer Sekunde gestellt, um das Incast-Verhalten von *gAUDIT* bei unterschiedlicher Zufallsverzögerung gut untersuchen zu können. Interferenzen durch einen adaptiven Zeitgeber für Sendewiederholungen sind so ausgeschlossen und die Wahl der Zufallsverzögerung zur Desynchronisierung lässt sich isoliert analysieren. Darüber hinaus wurde die Warteschlangenlänge aller Zwischensysteme wie im *Basic Incast* Szenario aus [6] auf 100 Rahmen eingestellt (für Cisco Router ein Standardwert).

Prinzipiell wird danach unterschieden, welche Verteilungsfunktion in welcher konkreten Parameterisierung auf Seiten der *Respondents* zur Berechnung der Zufallsverzögerung zum Einsatz kommt. Als Verteilungsfunktion werden die *gAUDIT*-unabhängige Exponential- und Gleichverteilung analysiert. Zusätzlich wird eine *gAUDIT*-abhängige

Variante der beiden Verteilungsfunktionen untersucht. Abhängig sind die Verteilungsfunktionen deshalb, da zusätzlich zu den regulären Parametern die Anzahl der erwarteten Antworten mit in die Berechnung der Zufallsverzögerung einfließt. Prinzipiell soll so ermittelt werden, ob sich Incast-Szenarien besser vermeiden lassen, wenn die Anzahl der erwarteten Response-Nachrichten bei Wahl der Zufallsverzögerung mitberücksichtigt wird. Außerdem soll die Parameterstudie zeigen, welche Verteilungsfunktion geeignet ist.

Es ergeben sich die in Tabelle 4.5 beschriebenen Strategien zur Berechnung der Zufallsverzögerung. Die Variable $|M|$ steht dabei für die durch *gAUDIT* ermittelte Anzahl an fehlenden (erwarteten) Response-Nachrichten.

Strategie	**Parameter**
Exponentialverteilung	Den Erwartungswert $E(X) = \int_0^\infty \lambda x e^{-\lambda x} \mathrm{dx} = \frac{1}{\lambda}$
Gleichverteilung	Grenze für halboffenes Intervall $[0, \texttt{Grenze})$
gAUDIT-abhängige Exponentialverteilung	Den Erwartungswert $E(X) \cdot \lvert M \rvert = \int_0^\infty \lambda x e^{-\lambda x} \mathrm{dx} = \frac{1}{\lambda}$
Gleichverteilung	Grenze für halboffenes Intervall $[0, \texttt{Grenze} \cdot \lvert M \rvert)$

Tabelle 4.5: *Strategien zur Berechnung der Zufallsverzögerung.*

4.9.3.2 Durchführung

In Abbildung 4.36 werden die in Zwischensystem-Warteschlangen verworfenen *gAUDIT*-Nachrichten in Abhängigkeit unterschiedlicher Parameterisierungen der Verteilungsfunktionen untersucht. Tritt kein Nachrichtenverlust auf, so sind die Nachrichten hinreichend desynchronisiert und es kommt nicht zum *Incast-Problem*. Aus der Untersuchung in Abbildung 4.36 wird ersichtlich, dass in den analysierten Szenarien für die *gAUDIT*-Varianten ab einem Parameterwert größer 100*ns* keine Nachrichten mehr verloren gehen. Für die

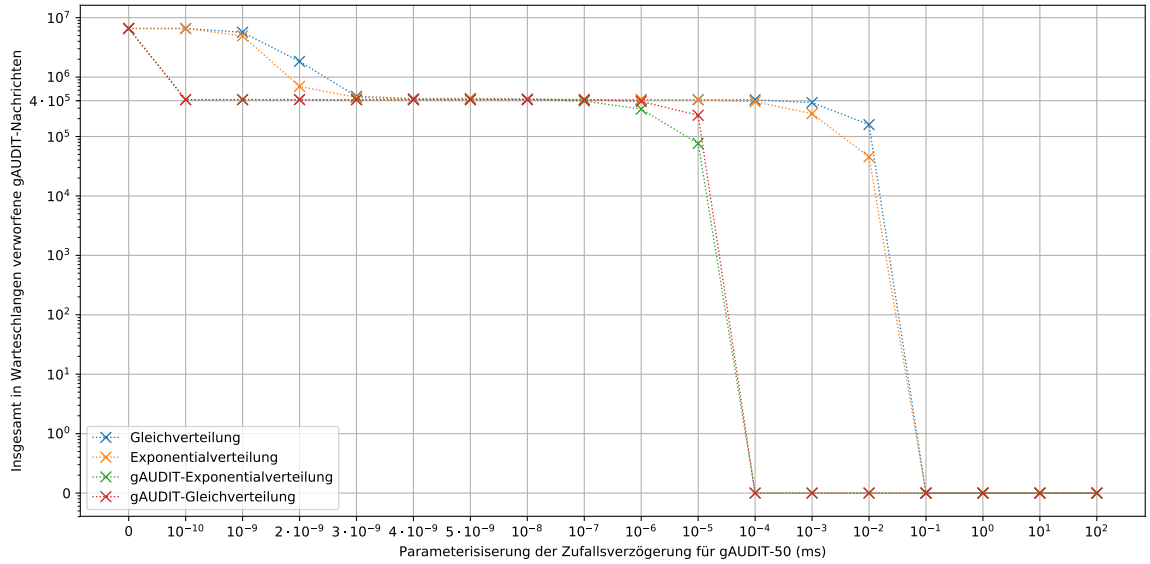

Abbildung 4.36: *Analyse der absolut in Warteschlangen durch Incast verworfenen gAUDIT-Nachrichten für unterschiedliche Wahl des Parameter zur Bestimmung der Zufallsverzögerung.*

beiden anderen Verteilungsfunktion wird hingegen mindestens ein Parameterwert von 100μs zur Desynchronisierung benötigt – Unterschied von Faktor 1.000.

Allerdings wirkt sich eine höhere Parameterisierung für die Verteilungsfunktion auch negativ auf die Ausführungszeit aus. Dies führt automatisch zu einer größeren Zufallsverzögerung und längerer Ausführungszeit. Daher erfolgt in Abbildung 4.37 eine Analyse der durchschnittlichen Ausführungszeit bei unterschiedlicher Parameterwahl. Hier wird auch der eigentliche Vorteil der *gAUDIT*-Variante sichtbar.

In nicht allen Fällen der Parameterstudie konnte die Request-Response-Kommunikation innerhalb der maximalen Simulationszeit von 100s abgeschlossen werden. Diese Fälle werden mit einer Ausführungszeit von 100s in den Durchschnitt in Abbildung 4.37 mit einberechnet. Für die Parameterwahl zwischen 10μs und 1ms schlagen allerdings keine *gAUDIT*-Ausführungen fehl.

Der obere Teil von Abbildung 4.37 unterstreicht, dass die beiden *gAUDIT*-Varianten für eine Parameterwahl im Bereich von 100ns zu einer deutlich kürzereren durchschnittlichen Ausführungszeit führen als die beiden anderen Varianten (die schwarzen Pfeile in der Abbildung 4.37 zeigen die zu vergleichenden Kurven). Im direkten Vergleich ergibt sich in beiden Fällen ein Verbesserungsfaktor von über ~ 1.8, für die Exponentialverteilungen eine Verbesserung von $\sim 653\mu s$ zu $\sim 359\mu s$ und für die Gleichverteilungen von $\sim 100\mu s$ zu $\sim 51\mu s$. Insgesamt schneiden die Gleichverteilungsvarianten hier besser ab. Die An-

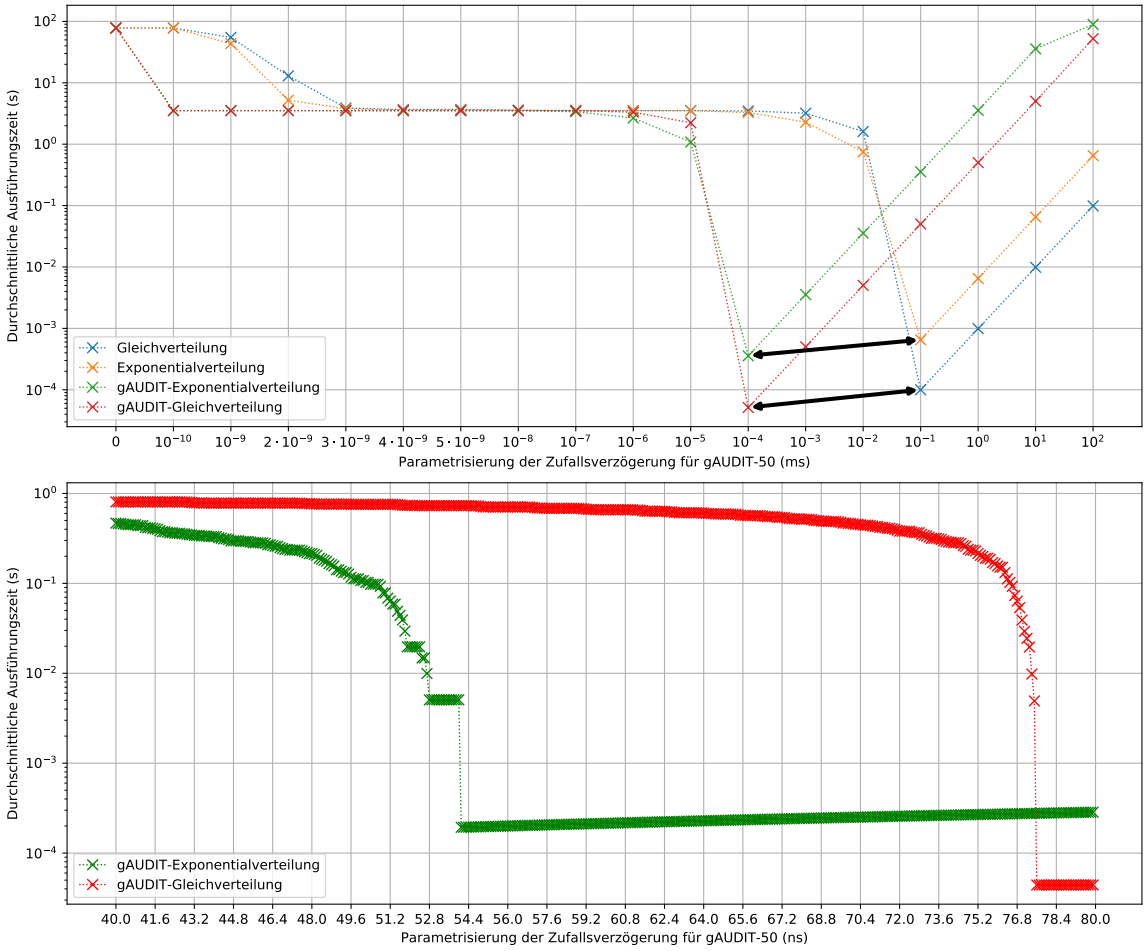

Abbildung 4.37: *Auswirkung der Parameterwahl und Wahl der Verteilungsfunktion auf die durchschnittliche Ausführungszeit von gAUDIT. Die schwarzen Pfeile im oberen Teil der Abbildung zeigen die zu vergleichenden Kurven. Im unteren Teil ist eine detailliertere Betrachtung dargestellt.*

stiege im rechten Teil der oberen Abbildung lassen sich einerseits darauf zurückführen, dass die durch die Parameterisierung erreichte Zufallsverzögerung so groß wird, dass die Response-Nachrichten den Initiator nicht vor Ablauf des Zeitgebers für Sendewiederholungen erreichen, weswegen Sendewiederholungen notwendig werden und sich die Ausführungszeit erhöht. Anderseits ist bei den *gAUDIT*-Varianten der starke Anstieg auf die Einbeziehung der erwarteten Response-Nachrichten $|M|$ zurückzuführen.

Im unteren Teil der Abbildung 4.37 erfolgt eine detailliertere Betrachtung des Parameterraums, um die optimalen Werte für *gAUDIT* zu ermitteln. Hierfür werden weitere Messungen im Bereich für die Parameterisierungen der Verteilungsfunktionen im Bereich von 50*ns* bis 150*ns* durchgeführt. Im unteren Teil der Abbildung ist der Bereich im

Intervall $[40, 80)$ dargestellt. Die detailliertere Analyse zeigt, dass der optimale Wert für *gAUDIT*-abhängige Gleichverteilungsvarianten bei $77.6ns$ liegt. Der optimale Wert für die *gAUDIT*-abhängige Exponentialverteilungsvariante liegt hingegen bei $54.1ns$. Die Verwendung einer *gAUDIT*-abhängigen Gleichverteilungsfunktion zur Berechnung der Zufallsverzögerung bringt einen rund 4.4-fachen Vorteil im Bezug zur Ausführungszeit der *gAUDIT*-abhängigen Exponentialverteilungsvariante mit sich (von $\sim 193\mu s$ zu $44\mu s$).

In Abbildung 4.38 zeigt sich, dass bei optimaler Parameterisierung der Verteilungsfunktionen keine Sendewiederholungen zur Durchführung von Request-Response benötigt werden. Außerdem zeigt sich, dass die *gAUDIT*-Exponentialverteilungsvariante bereits bei $100\mu s$ zu Sendewiederholungen führt, weil im Durchschnitt die Response-Nachrichten den Initiator nicht vor Ablauf des Zeitgebers für Sendewiederholungen (hier $1s$) erreichen. In Warteschlangen werden keine *gAUDIT*-Nachrichten verworfen, siehe Abbildung 4.36.

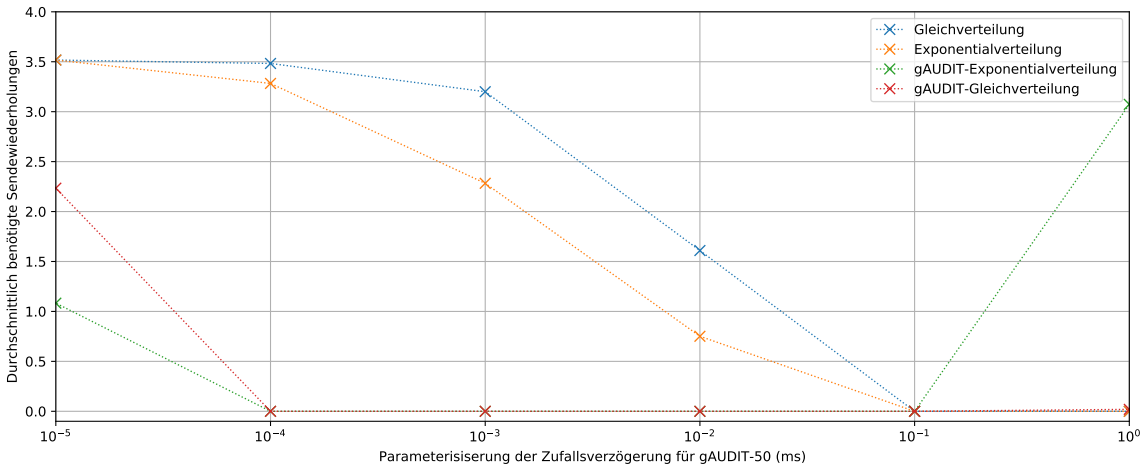

Abbildung 4.38: *Detailausschnitt für durchschnittlich benötigte Sendewiederholungen im Bereich von* $10ps$ *bis* $1ms$*. Keine gAUDIT-Ausführung schlägt fehl.*

4.9.3.3 Ergebnisse

Als Schlussfolgerung aus der Parameterstudie zum Incast-Verhalten von *gAUDIT* lässt sich ableiten, dass die Ausführungszeit von *gAUDIT* sich bei Einbeziehung von erwarteten Response-Nachrichten für die Berechnung der Zufallsverzögerung zur Desynchronisierung von Nachrichten um den Faktor ~ 1.8, im Verhältnis zu Varianten die die erwarteten Response-Nachrichten nicht mitberücksichtigen, verbessert. Außerdem zeigt die Parameterstudie, dass für das untersuchte Incast-Szenario bei *gAUDIT*-50 ($s = 50$) als Verteilungsfunktion eine *gAUDIT*-abhängige Gleichverteilungsfunktion mit $77.6ns$ je erwarteter Response-Nachricht gewählt werden sollte. Diese sorgt für das *Basic Incast*

Szenario aus [6] für hinreichend viel Zufallsverzögerung zur Desynchronisierung von Nachrichten bei gleichzeitiger optimaler Ausführungszeit.

4.10 Zusammenfassung

In diesem Kapitel wird das *gAUDIT*-Transportsystem mit dem adaptiven Request-Response-Dienst für Gruppen und dem Gruppenverwaltungsdienst vorgestellt. Zur Durchführung von Request-Response in Gruppen ermöglicht *gAUDIT* Anwendungsinstanzen die erwartete Response-Vollständigkeit zu parametrisieren. Beim Versand von Request-Nachrichten wählt *gAUDIT* geschickt zwischen Multicast und Unicast aus, so dass weniger Netzressourcen benötigt werden als in vergleichbaren Arbeiten. Außerdem wird auftretender Nachrichtenverlust behandelt und durch Hinzufügen von Zufallsverzögerung die Incast-Problematik vermieden.

Die durchgeführten Parameterstudien belegen, dass der Versand von Request-Nachrichten per Multicast an eine *Gruppe* von *Respondents* dann effizienter als die Nutzung von Unicast ist, wenn mehr als 50% der Response-Nachrichten fehlen. Außerdem decken die Parameterstudien auf, dass die Nutzung einer Gleichverteilungsfunktion für *gAUDIT* in der Studie am Besten das Incast-Problem vermeidet. Konkret sollte hier auf Seiten der Respondents die obere Schranke der Gleichverteilungsfunktion pro fehlender Response-Nachrichten um $\sim 77.6ns$ erhöht werden. Schließlich belegt die Evaluation, dass die Verwendung von Sendewiederholungen wie erwartet zu einer höheren Response-Vollständigkeit führt als bei Verfahren, die keine Sendewiederholungen nutzen. Darüber hinaus zeigt sich aber auch, dass in Datenzentren aufgrund der geringen Verlustwahrscheinlichkeit (in der Regel $< 10^{-8}$) meistens keine, allenfalls eine einzelne Sendewiederholung benötigt wird. Diese eine gegebenenfalls fehlende Nachricht sollte dann per Unicast vom fehlenden Respondent nachgefordert werden.

In zukünftigen Arbeiten können noch Aspekte wie Stau- und Flusskontrolle oder komplexere Incast-Szenarien mit Hintergrundverkehr untersucht werden.

Kapitel 5

Verdeckender Ungleichheitsnachweis für Monitoringdaten

Der Schutz vor Offenlegung von Monitoringdaten des Cloud-Anbieters und die Nachvollziehbarkeit im Nachhinein stehen bei Monitoring von Cloud-Umgebungen im Spannungsfeld zueinander. Einerseits sollen vertragliche Vereinbarungen zur Laufzeit überprüft werden können, ohne dass dabei Monitoringdaten des Cloud-Anbieters offen gelegt werden müssen. Andererseits müssen die zur Überprüfung verwendeten Monitoringdaten des Cloud-Anbieters im Nachhinein nachvollziehbar sein, beispielsweise zur Fehler-Ursachen-Analyse vor Gericht. Als Monitoringdaten werden hier insbesondere die von *CloudInspector* ermittelten Monitoringdaten aufgefasst.

In diesem Kapitel wird ein Verfahren für den Monitoring-Controller vorgestellt, mit dessen Hilfe Kunden die Einhaltung von vertragliche Vereinbarungen mit dem Cloud-Anbieter zur Laufzeit überprüfen können, bei gleichzeitiger Geheimhaltung der zur Überprüfung verwendeten Monitoringdaten. Im weiteren Verlauf wird dieses Verfahren als *CoCoPAS* (Corporate Confidentiality Preserving Auditing Scheme) bezeichnet. Das hier vorgestellte Verfahren eignet sich insbesondere zur Überprüfung auf Gleichheit oder Ungleichheit von Monitoringdaten – beispielsweise wenn überprüft werden soll, dass zwei VMs sich nicht auf dem gleichen physischen Cloud-Knoten befinden. Zentrales Element bei *CoCoPAS* sind sogenannte *Auditwerte* (siehe Abschnitt 5.3). In einem Auditwert wird von *CoCoPAS* ein konkretes Monitoringdatenelement, wie zum Beispiel die IP-Adresse eines physischen Cloud-Knotens oder die aktuelle Betriebssystemversionsnummer, hinterlegt. Die Begriffe Monitoringdaten und Monitoringdatenelement sind dabei als Synonym zu verstehen. Der Ausdruck Monitoringdatenelement wird lediglich verwendet, um zu verdeutlichen, dass es sich um ein einzelnes Element handelt. Mit Hilfe von *CoCoPAS* überzeugt sich der Kunde

dann, dass die in Auditwerten hinterlegten Monitoringdaten ungleich beziehungsweise gleich sind, ohne dass dazu die Monitoringdaten selbst offengelegt werden müssten. Im Rahmen von *CoCoPAS* wird außerdem eine vertrauenswürdige dritte Partei zur Absicherung des zeitlichen Bezugs von Auditwerten eingesetzt. Neu ist hier insbesondere die verdeckende Überprüfung auf Ungleichheit. Eine Übersicht der in diesem Kapitel verwendeten Variablen findet sich in Anhang D.

Folgende Forschungsfrage wird behandelt:

> *Wie lassen sich Monitoringdaten auf Ungleichheit überprüfen, so dass die Überprüfung verdeckend und im Nachhinein nachvollziehbar ist? Welche Sicherheitseigenschaften können dabei erreicht werden?*

Ergebnisse dieses Kapitels basieren unter anderem auf der folgenden Publikation:

- R. Bless und M. Flittner. „Towards corporate confidentiality preserving auditing mechanisms for Clouds“. In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. Okt. 2014, S. 381–387. DOI: 10.1109/CloudNet.2014.6969025 [TT-13]

5.1 Anforderungen an einen verdeckenden Ungleichheitsnachweis

Im Rahmen dieser Dissertation wurden die folgenden Anforderungen an einen verdeckenden Ungleichheitsnachweis für Monitoringdaten identifiziert.

Geheimhaltung: Der Kunde soll das in einem Auditwert hinterlegte Monitoringdatenelement nicht ermitteln können. Wenn beispielsweise der Auditwert A_1 das Monitoringdatenelement S_1 enthält, so darf der Kunde S_1 nicht erfahren. Allerdings gilt diese Geheimhaltung nicht gegenüber einer vertrauenswürdigen dritten Partei. Diese soll mit Hilfe einer Geheimtür in der Lage sein, das in dem Auditwert A_1 hinterlegte Monitoringdatenelement S_1 aufzudecken.

Eindeutigkeit: Dem Cloud-Anbieter darf es nicht möglich sein, das in einem Auditwert enthaltene Monitoringdatenelement nach der Hinterlegung zu verändern. Daraus folgt, dass sich jedes in einem Auditwert hinterlegte Monitoringdatenelement eindeutig aufdecken lassen muss. Wenn der Cloud-Anbieter beispielsweise das Monitoringdatenelement S_1 in dem Auditwert A_1 hinterlegt hat, darf es ihm nicht möglich sein, das Monitoringdatenelement S_2 mit $S_2 \neq S_1$ zu dem Auditwert A_1 aufzudecken. Gleiches gilt für die vertrauenswürdige dritte Partei.

Zurechenbarkeit: Auditwerte sollen sich nicht in einen anderen zeitlichen oder inhaltlichen Zusammenhang bringen lassen, als sie es bei ihrer Erstellung waren. Darüber hinaus soll jeder Auditwert eindeutig der Anfrage des Kunden, aus der dieser hervorgegangen ist, zuordenbar sein. Außerdem soll erkennbar sein, welche Entität einen Auditwert erzeugt hat. Darüber hinaus soll der Zeitpunkt der Erzeugung eines Auditwertes eindeutig festgelegt und nachvollziehbar sein. Hierdurch soll es Kunden nicht möglich sein, selbst Auditwerte zu erzeugen, die den Anschein erwecken, als seien sie vom Cloud-Anbieter ausgestellt worden. Gleichermaßen soll es dem Cloud-Anbieter nicht möglich sein, den zeitlichen oder inhaltlichen Zusammenhang eines Auditwertes zu verändern.

Nichtabstreitbarkeit: Durch den Cloud-Anbieter erzeugte Auditwerte sollen nicht abstreitbar sein. Hierdurch soll insbesondere sichergestellt werden, dass der Cloud-Anbieter im Nachhinein nicht behaupten kann, die vorliegenden Auditwerte nicht erzeugt zu haben. Auf diese Art und Weise soll gewährleistet sein, dass bei Streitigkeiten vor Gericht der Cloud-Anbieter für die durch Auditwerte belegten Sachverhalte haftbar gemacht werden kann. Beispielsweise wenn die in Auditwerten enthaltenen Monitoringdaten nachweisen, dass zwei VMs, im Gegensatz zur vertraglichen Vereinbarung, auf dem gleichen physischen Cloud-Knoten ausgeführt wurden.

5.2 Angreifermodell

Im Folgenden wird das Angreifermodell für *CoCoPAS* weiter präzisiert. Grundlegend wird im Rahmen dieser Dissertation zunächst von einem ehrlichen Cloud-Anbieter ausgegangen (vergleiche Kapitel 1). Das heißt zum Zeitpunkt einer Abfrage werden vom Cloud-Anbieter keine Manipulationen vorgenommen, insbesondere nicht an den erhobenen Monitoringdaten oder an der Durchführung einer Überprüfung. Daher ist der Cloud-Anbieter hier zunächst nicht als Angreifer zu sehen.

Allerdings wird diese Annahme im Fall von Fehler-Ursachen-Analysen vor Gericht nicht mehr aufrechterhalten. Nun wird davon ausgegangen, dass der Cloud-Anbieter Ergebnisse durchgeführter Überprüfungen anzweifelt oder versucht, Verfahren zu seinen Gunsten zu beeinflussen. Genau hierfür soll *CoCoPAS* eine Absicherung bieten: gegen einen Cloud-Anbieter der im Nachhinein erzeugte Auditwerte abstreiten oder verändern, neue erzeugt oder bei Überprüfungen entstandene Nachweise manipuliert.

Auch Kunden werden bei einer Fehler-Ursachen-Analyse vor Gericht als Angreifer auf das hier entwickelte *CoCoPAS*-Verfahren gesehen. So soll das Verfahren Schutz gegen

Manipulationen durch Kunden bieten. Beispielsweise sollen Kunden vor Gericht keine unauthentischen Nachweise für durchgeführte Überprüfungen vorlegen können.

Außerdem werden für das Angreifermodell die Fähigkeiten der an *CoCoPAS* beteiligten Entitäten festgelegt. Hierfür wird angenommen, dass deren Rechenleistung auf Polynomialzeit beschränkt ist. Dies gilt insbesondere auch für den Angreifer. Eine detaillierte Beschreibung der Angriffsanalyse folgt in Abschnitt 5.8.1.

Angriffe durch Dritte auf das entwickelte Verfahren, beispielsweise durch Konkurrenten des Cloud-Anbieters, werden hier ausgeklammert. Hierfür müssen andere geeignete Maßnahmen, wie beispielsweise die Etablierung eines sicheren Kanals zwischen Kunde und Cloud-Anbieter mittels TLS [244], getroffen werden, welche nicht Untersuchungsgegenstand dieser Dissertation sind.

5.3 Konzeption des Verfahrens

Mittels *CoCoPAS* können Kunden die zu vertraglichen Vereinbarungen gehörenden Monitoringdaten auf Gleichheit oder Ungleichheit überprüfen, ohne dass diese dazu vom Cloud-Anbieter offengelegt werden müssen. Beispielsweise wenn überprüft werden soll, auf welchem physischen Cloud-Knoten zwei VMs eines Kunden vom CMS platziert wurden, ohne dass dabei die genutzten physischen Cloud-Knoten offengelegt werden müssten – hierbei sind die konkreten physischen Cloud-Knoten als Monitoringdaten anzusehen, deren Offenlegung der Cloud-Anbieter gegenüber dem Kunden verhindern möchte. Außerdem sollen sowohl der Cloud-Anbieter als auch eine vertrauenswürdige dritte Partei im Nachhinein in der Lage sein, die zur Überprüfung verwendeten Monitoringdaten aufzudecken – zum Beispiel soll im Streitfall vor Gericht nachvollziehbar sein, welche konkreten physischen Cloud-Knoten zur Abbildung von VMs verwendet wurden. Da *CoCoPAS* nicht nur auf den Anwendungsfall bei *CloudInspector* beschränkt ist, wird im Folgenden anstatt des Ablaufs zwischen Kunde und Monitoring-Controller des *CloudInspectors* der Ablauf zwischen Kunde und Cloud-Anbieter erläutert. Durch welche Komponente der Cloud-Anbieter dabei die *CoCoPAS*-Funktionalität bereitstellt, bleibt dabei ihm überlassen.

Bei *CoCoPAS* lassen sich demnach drei Entitäten identifizieren:

Kunde: Entität, welche Überprüfung mittels des *CoCoPAS* veranlassen oder eine Aufdeckung von Monitoringdaten durch eine dritte Partei vor Gericht anfordern kann.

Cloud-Anbieter: Entität, welche Kontrolle über die Cloud-Infrastruktur hat und Informationen für das *CoCoPAS* liefert. Übermittelt auf Anfrage des Kunden Auditwerte oder assistiert bei der Aufdeckung von Monitoringdaten.

Eine vertrauenswürdige dritte Partei: Entität, welche bei der Aufdeckung von Monitoringdaten assistieren kann. Stellt zusätzlich einen vertrauenswürdigen Zeitstempeldienst zur Verfügung (zum Beispiel mittels [210]).

Ein Kunde kann mit Hilfe des *CoCoPAS* eine *Anfrage* nach zu einer vertraglichen Vereinbarung gehörenden Monitoringdatenelement an den Cloud-Anbieter stellen. Als Antwort übermittelt der Cloud-Anbieter an den Kunden einen sogenannten *Auditwert*, der sich als abschließbarer blickdichter *Kasten* veranschaulichen lässt, wobei nur der Cloud-Anbieter im Besitz des Schlüssels ist. In dem Kasten hinterlegt der Cloud-Anbieter das angefragte Monitoringdatenelement und verschließt diesen. Anschließend wird der blickdichte Kasten an den Kunden gesandt. Auf diese Art und Weise erhält der Kunde keinerlei Informationen über das Monitoringdatenelement und für den Cloud-Anbieter ist es nach Übermittlung an den Kunden nicht mehr möglich, das Monitoringdatenelement im Kasten zu verändern.

Die in zwei verschiedenen Anfragen übermittelten Auditwerten können dann vom Kunden mit Hilfe des *CoCoPAS* auf Gleichheit oder Ungleichheit überprüft werden, ohne dass dazu die Monitoringdaten vom Cloud-Anbieter aufgedeckt werden müssen – d.h. zur Überprüfung auf Ungleichheit muss der blickdichte Kasten nicht geöffnet werden. Das *Resultat* einer Überprüfung auf Gleichheit oder Ungleichheit ist ein binärer Wert, der widerspiegelt ob die Überprüfung erfolgreich war. Hieraus lässt sich dann das Ergebnis der Überprüfung und somit die Erfüllung der vertraglichen Vereinbarung ableiten.

Weiterhin ermöglicht *CoCoPAS*, die in Auditwerten hinterlegten Monitoringdaten des Cloud-Anbieters im Nachhinein aufzudecken. Dies ist beispielsweise zur Fehler-Ursachen-Analyse vor Gericht verwendbar. Allerdings führt *CoCoPAS* selbst keine Fehler-Ursachen-Analyse durch, sondern liefert vielmehr eine Datenbasis, auf deren Grundlage ein technischer Experte eine Fehler-Ursachen-Analyse durchführen kann. Die Nachvollziehbarkeit von Monitoringdaten kann dabei auf zwei Arten realisiert werden. Einerseits ist es möglich, dass der Cloud-Anbieter selbst die Monitoringdaten, die zur Erstellung von Auditwerten benötigt wurden, aufdeckt. Alternativ kann eine vertrauenswürdige dritte Partei die in Auditwerten enthaltenen Monitoringdaten ohne Zutun des Cloud-Anbieters aufdecken.

Weiterhin können bei *CoCoPAS* vier Phasen unterschieden werden:

Vorbereitungsphase: Initialisierungsphase von *CoCoPAS*. Läuft in der Regel zwischen jedem Kunde und Cloud-Anbieter ein einziges Mal zur Vorbereitung des Verfahrens ab.

Hinterlegungsphase: Der Cloud-Anbieter kann zur Laufzeit Monitoringdaten in Auditwerten hinterlegen. Der Kunde kann die in der Hinterlegungsphase hinterlegten Monitoringdaten jederzeit vom Cloud-Anbieter anfordern.

Überprüfungsphase: Kunden ist es jederzeit möglich, die in Auditwerten hinterlegten Monitoringdaten auf Gleichheit oder Ungleichheit zu überprüfen, ohne dass dazu die hinterlegten Monitoringdaten selbst aufgedeckt werden müssten. Vorausgesetzt natürlich, der Cloud-Anbieter kooperiert mit dem Kunden, was bei einem ehrlichem Cloud-Anbieter implizit gegeben ist.

Aufdeckungsphase: Das Aufdecken der in Auditwerten hinterlegten Monitoringdaten ist im Nachhinein möglich. Entweder mit Hilfe des Cloud-Anbieters oder einer vertrauenswürdigen dritten Partei.

Für den weiteren Verlauf wird zunächst davon ausgegangen, dass alle Monitoringdatenelemente sich in einer nummerischen Form ausdrücken lassen. Dies ist beispielsweise für die IP-Adresse oder Betriebssystemversionsnummern von physische Cloud-Knoten trivialerweise der Fall. Für alle anderen Monitoringdatenelemente wird dann zusätzlich ein geeignetes Codierungsverfahren benötigt, welches das Element in nummerischer Form beschreibt. Die Wahl eines geeigneten Codierungsverfahrens ist jedoch nicht Gegenstand dieser Dissertation.

5.3.1 Beispielablauf

Für das Beispiel wird angenommen, dass ein Kunde mit einem Cloud-Anbieter vereinbart hat, dass die beiden VMs V_1 und V_2 nicht auf dem gleichen physischen Cloud-Knoten ausgeführt werden dürfen. Das CMS des Cloud-Anbieters ist verantwortlich, diese Vereinbarungen innerhalb der Cloud-Infrastruktur umzusetzen. Derzeit wird die VM V_1 auf dem physischen Cloud-Knoten N_1 und die VM V_2 auf dem physischen Cloud-Knoten N_2 ausgeführt (mit $N_1 \neq N_2$). Welche VM auf welchem Cloud-Knoten ausgeführt wird soll jedoch vor dem Kunden geheim gehalten werden. Dem Kunden sind daher nur die VMs V_1 und V_2 bekannt.

Möchte der Kunde überprüfen, ob sich die beiden VMs V_1 und V_2 auf unterschiedlichen physischen Cloud-Knoten befinden, so kann er in der Hinterlegungsphase zunächst eine Anfrage bezüglich der Abbildung der VMs auf physischen Cloud-Knoten stellen. Der Kunde sendet dazu, wie in Abbildung 5.1 dargestellt, eine Anfrage bezüglich der Abbildung der VMs V_1 und V_2 an den Cloud-Anbieter. *CoCoPAS* liefert dann je VM einen separaten Auditwert als Antwort an den Kunden zurück. Dabei handelt es sich hier konkret um die an den Kunden übermittelten Auditwerte A_1 und A_2. In jedem Auditwert ist in diesem Fall der physische Cloud-Knoten der angefragten VM als Monitoringdatenelement hinterlegt. Konkret enthält der Auditwert A_1 das Monitoringdatenelement S_1 und der Auditwert A_2 das Monitoringdatenelement S_2, wobei $S_1 = N_1$ und $S_2 = N_2$ gilt.

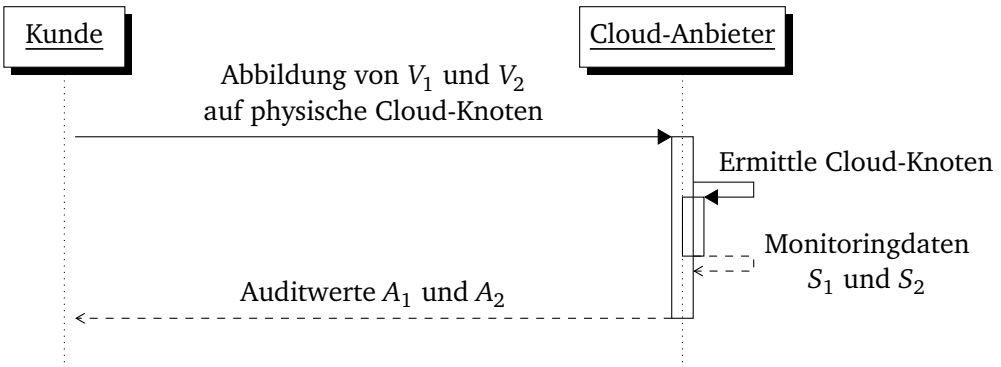

Abbildung 5.1: *Beispiel der Hinterlegungsphase.*

Die Monitoringdaten S_1 und S_2, die in den Auditwerten A_1 und A_2 hinterlegt sind, kann der Kunde dann mit Hilfe des *CoCoPAS* in der Überprüfungsphase auf Ungleichheit überprüfen, also sicherstellen dass die physischen Cloud-Knoten der angefragten VMs V_1 und V_2 unterschiedlich sind. Das Resultat der Überprüfung ist 1, falls die in den beiden Auditwerten hinterlegten Monitoringdaten ungleich sind. Wenn die Überprüfung der hinterlegten Monitoringdaten fehlschlägt, ist das Resultat der Überprüfung hingegen 0. Allerdings kann vom fehlschlagen des Überprüfungsprozess auf Ungleichheit nicht automatisch auf Gleichheit geschlossen werden. Analog gilt dies anders rum. Da von einem ehrlichem Cloud-Anbieter ausgegangen wird, schlägt die Überprüfung nur in dem Fall fehl, in dem die beiden Monitoringdatenelemente ungleich sind. Würde ein betrügerischer Cloud-Anbieter unterstellt, so könnte dieser die Überprüfung auch absichtlich fehlschlagen lassen, was allerdings hier keinen Sinn macht, da der Kunde ja einen Nachweis für die Ungleichheit verlangt und ein Fehlschlagen des Verfahrens zum Nachteil des Cloud-Anbieters ist.

Falls die Überprüfung auf Ungleichheit fehlschlägt, kann es beispielsweise in einem Rechtsstreit zwischen Kunde und Cloud-Anbieter erforderlich sein, die Monitoringdatenelemente zu den Auditwerten aufzudecken. Hierzu kann *CoCoPAS* in der Aufdeckungsphase, beispielsweise durch Einsatz einer vertrauenswürdigen dritten Partei, verwendet werden. Beispielsweise bittet der Kunde dann vor Gericht die vertrauenswürdige dritte Partei, das in einem Auditwert hinterlegte Monitoringdatenelement aufzudecken. Der vertrauenswürdigen dritten Partei muss dazu sowohl vom Cloud-Anbieter als auch vom Kunden vertraut werden. Hierzu sendet der Kunde die Auditwerte A_1 und A_2 an die vertrauenswürdige dritte Partei. Mit Hilfe einer *CoCoPAS*-spezifischen *Geheimtür* ist die vertrauenswürdige dritte Partei dann in der Lage die Monitoringdaten N_1 und N_2 aufzudecken.

5.3.2 Auswahl geeigneter Mechanismen

Für die Gestaltung des *CoCoPAS*-Verfahrens kommen grundsätzlich verschiedene Mechanismen in Betracht. Beispielsweise der Einsatz von Verschlüsselung, Hashfunktionen oder Hinterlegungsverfahren. Allerdings schränkt die spezielle Forderung nach verdeckender Überprüfung auf Ungleichheit mit gleichzeitiger Möglichkeit zur Aufdeckung die Auswahl erheblich ein.

Daher wird im Rahmen dieser Dissertation ein auf Hinterlegungsverfahren basierender Ansatz gewählt. Dieser Mechanismus bietet den Vorteil, dass einerseits die Überprüfung auf Gleichheit und Aufdeckung bereits Bestandteil von Hinterlegungsverfahren ist. Die Ausgestaltung der Überprüfung auf Ungleichheit ist dann Gegenstand des hier vorgestellten Forschungsansatzes.

5.3.3 Voraussetzungen

Das hier vorgestellte *CoCoPAS* basiert auf der Arbeit von Bless und Flittner [TT-13]. Hierfür wird ein auf dem *Encryption Scheme* von ElGamal basierendes Hinterlegungsverfahren mit Geheimtür verwendet [51, 158] (vergleiche auch Kapitel 2, Abschnitt 5 [59]). Dieses *Encryption Scheme* wiederum basiert auf dem *Key Exchange Scheme* von *Diffie-Hellman* [46]. Zur Überprüfung auf Gleichheit wird die *Masking*-Eigenschaft von Pedersen [150] verwendet. Außerdem wird zur Überprüfung auf Ungleichheit eine Kombination des *ID Protocols* von Schnorr [160] mit der *Multiplication Technique* von Abe et al. [2] als *interaktiver Zero-Knowledge-Beweis* verwendet.

Sei G eine multiplikative endliche zyklische Gruppe der Ordnung q mit Generator g, dann wird im Rahmen dieser Dissertation davon ausgegangen, dass die folgenden bekannten Diffie-Hellman-Annahmen gelten:

Annahme 5.1 *Discrete Logarithm Problem (DLOG). Die Wahrscheinlichkeit, dass aus gegebenen G, g und g^a der diskrete Logarithmus a zur Basis g in Polynomialzeit berechnet werden kann, ist vernachlässigbar.*

Annahme 5.2 *Computational-Diffie-Hellman-Problem (CDH). Die Wahrscheinlichkeit, dass aus gegebenen G, g, g^a und g^b das Diffie-Hellman-Triple (g^a, g^b, g^{ab}) in Polynomialzeit berechnet werden kann, ist vernachlässigbar. Diese Annahme hängt direkt von Annahme 5.1 ab.*

Annahme 5.3 *Decisional-Diffie-Hellman-Problem (DDH). Die Wahrscheinlichkeit, dass aus gegebenen G, g, g^a, g^b und g^c in Polynomialzeit bestimmt werden kann, ob es sich um ein Diffie-Hellman-Triple handelt, also ob $c = a \cdot b$ gilt, ist vernachlässigbar. Diese Annahme hängt direkt von Annahme 5.1 ab.*

Im Rahmen dieser Dissertation wird vorausgesetzt, dass eine geeignete multiplikative endliche zyklische Gruppe G existiert, die die oben stehenden Annahmen erfüllt. Beispielsweise kann derzeit nach Katz und Lindell [110] davon ausgegangen werden, dass die Teilgruppe der quadratischen Reste von $\mathbb{Z}_p^*$ mit $p = 2q + 1$ und p und q prim, alle eben eingeführten Anforderungen (*DLOG*, *CDH* und *DDH*) erfüllt. Daher wird bei der folgenden Betrachtung diese Gruppe ohne Beschränkung der Allgemeinheit als Referenz genutzt. Weiterhin liegt der Fokus dieser Dissertation weder darauf, nachzuweisen, dass die eben eingeführten Annahmen tatsächlich nicht in Polynomialzeit zu brechen sind, noch darauf, für geltende Annahmen geeignete Gruppen zu finden.

5.4 Vorbereitungsphase

Die Initialisierung von *CoCoPAS* übernimmt die eingesetzte vertrauenswürdige dritte Partei, sodass diese in den Besitz der Geheimtür kommt und somit in der Lage ist, in Auditwerten hinterlegte Monitoringdaten ohne Zutun des Cloud-Anbieters aufzudecken. Der vertrauenswürdigen dritten Partei muss dabei sowohl vom Cloud-Anbieter als auch vom Kunden vertraut werden.

Zur Initialisierung wählt die vertrauenswürdige dritte Partei die Gruppe G der quadratischen Reste modulo p aus $\mathbb{Z}_p^*$ mit einem Generator g, für die gilt $p := 2q + 1$ mit p und q sind prim. Danach wählt die vertrauenswürdige dritte Partei zufällig eine Zahl $s \in \{1, \ldots, p-1\}$ mit $\text{ggT}(s, p) = 1$ als Geheimnis und erzeugt damit einen weiteren Generator $h := g^s$. Das Geheimnis kann später von der vertrauenswürdigen dritten Partei zum Aufdecken von Auditwerten mittels der Geheimtür verwendet werden. Anschließend übermittelt die vertrauenswürdige dritte Partei sowohl an den Cloud-Anbieter als auch an den Kunden die Werte (G, g, h). Das Geheimnis s wird dabei nicht übermittelt. Abbildung 5.2 veranschaulicht den Ablauf der Vorbereitung von *CoCoPAS*.

Nach Ablauf der Vorbereitung können vom Cloud-Anbieter sowohl Monitoringdatenelemente in Auditwerten hinterlegt werden, als auch Überprüfungen durch Kunden durchgeführt werden. Auch das Aufdecken von in Auditwerten enthaltenen Monitoringdaten wird hierdurch ermöglicht. Die einzelnen Phasen von *CoCoPAS* werden in den nächsten Abschnitten vorgestellt.

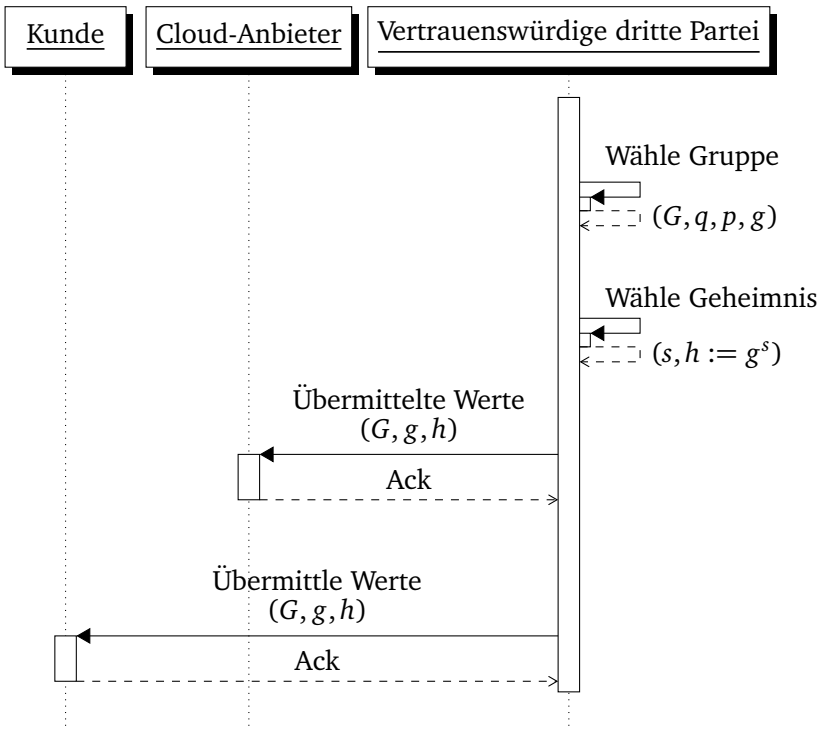

Abbildung 5.2: *Vorbereitung für den Einsatz von CoCoPAS.*

5.5 Hinterlegungsphase

In dieser Phase kann ein Kunde die Hinterlegung von Monitoringdaten in Auditwerten mittels *CoCoPAS* anfordern. Dazu sendet der Kunde eine Anfrage an den Cloud-Anbieter. Dieser ermittelt dann das zugehörige Monitoringdatenelement und hinterlegt dieses in einem Auditwert. Als ein Bestandteil der Hinterlegung von Monitoringdaten in Auditwerten wird die *commit*-Funktion von ElGamal verwendet [51]. Dabei ist die *commit*-Funktion von ElGamal wie folgt definiert: $\texttt{commit}(x, y) := z_2 = g^x \cdot h^y$. Zusätzlich wird dem Auditwert der Wert $z_1 := g^y$ hinzugefügt, der von der Parameterisierung der *commit*-Funktion abhängt. Hieraus ergibt sich die *hinterlege*-Funktion, die zur Hinterlegung von Monitoringdaten in Auditwerten genutzt wird:

$$\texttt{hinterlege}(x, y) := (z_1, z_2) = (g^y, \texttt{commit}(x, y)) = (g^y, g^x \cdot h^y)$$

Sei Q_1 die Anfrage des Kunden, S_1 das dazugehörige Monitoringdatenelement des Cloud-Anbieters und A_1 der Auditwert in dem das Monitoringdatenelement mittels *CoCoPAS* hinterlegt werden soll, dann läuft die Hinterlegungsphase wie folgt ab. Der Cloud-Anbieter wählt zunächst eine zufällige Zahl $r_1 \in \{1, \ldots, p-1\}$ mit $\text{ggT}(r_1, p) = 1$ und übergibt diese

zusammen mit dem Monitoringdatenelement S_1 als Eingabe an die *hinterlege*-Funktion. Die Ausgabe der *hinterlege*-Funktion definiert den Auditwert A_1. Hieraus folgt:

$$A_1 := \texttt{hinterlege}(S_1, r_1) = (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$$

Ein auf diese Art und Weise vom Cloud-Anbieter erzeugter Auditwert erfüllt die in Abschnitt 5.1 als Geheimhaltung formulierte Anforderung. Nachweis durch Satz 5.1.

Satz 5.1 *Einem Kunden ist es nicht möglich aus einem Auditwert A_1 das Monitoringdatenelement S_1 zu extrahieren. Gegeben sind die Parameter (G, g, h) und der Auditwert $A_1 := (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$.*

Beweis. Der Nachweis der Geheimhaltung von Auditwerten folgt direkt aus dem hier verwendeten Hinterlegungsverfahren von ElGamal [51]. Dieses garantiert bereits Geheimhaltung unter der Annahme 5.1 (DLOG), der Annahme 5.2 (CDH) und der Annahme 5.3 (DDH). Weiterhin wird dazu vorausgesetzt, dass der Kunde in seiner Rechenleistung auf Polynomialzeit beschränkt ist (vergleiche Abschnitt 5.1).

Daraus folgt, dass es einem Kunden weder möglich ist das Geheimnis s aus $h := g^s$ für die Verwendung der Geheimtür, noch den verwendeten Zufall r_1 aus $z_1 = g^{r_1}$ zu berechnen. Gleiches gilt für die Extrahierung des Monitoringdatenelement S_1 aus $z_2 := g^{S_1} \cdot h^{r_1}$. Hieraus folgt: Geheimhaltung für einen Auditwert ist gegeben. □

Nachdem der Cloud-Anbieter auf diese Art und Weise in der Lage ist einen Auditwert A_1 zu erzeugen, der Geheimhaltung bietet, muss zusätzlich die in Abschnitt 5.1 als Zurechenbarkeit formulierte Anforderung eingehalten werden. Dazu muss zum einen die Zugehörigkeit von Anfrage Q_1 zu Auditwert A_1 eindeutig gewährleistet werden. Zum anderen muss der zeitliche Bezug der Ausstellung und die Identität des Ausstellers (Cloud-Anbieters) sichergestellt werden. Zur Sicherstellung des zeitlichen Bezugs wird der Zeitstempeldienst der vertrauenswürdigen dritten Partei genutzt, beispielsweise mittels [210].

Dazu wird vom Cloud-Anbieter der Auditwert A_1 mit der Anfrage Q_1 des Kunden zu dem Element (Q_1, A_1) verknüpft. Dieses Element wird anschließend vom Cloud-Anbieter signiert, sodass die Signatur `Csign` entsteht. Dazu kann jedes sichere asymmetrische Signaturverfahren verwendet werden, es muss lediglich sichergestellt werden, dass der Kunde den öffentlichen Schlüssel des Cloud-Anbieters als authentisch anerkennt und auf diese Weise in der Lage ist, die Signatur zu verifizieren. Zusätzlich lässt sich der Cloud-Anbieter den Entstehungszeitpunkt der Signatur `Csign` von der vertrauenswürdigen dritten Partei für Kunden überprüfbar attestieren. Hierzu wird die Signatur `Csign`

vom Cloud-Anbieter zur vertrauenswürdigen dritten Partei übertragen. Dort wird die aktuelle Systemzeit `Ttime` ausgelesen und mit der Signatur `Csign` des Cloud-Anbieters zu dem Element (`Csign`, `Ttime`) verknüpft. Dieses Element wird anschließend durch die vertrauenswürdige dritte Partei signiert, sodass die Signatur `Tsign` entsteht. Als Signaturverfahren kann jedes sichere asymmetrische Verfahren verwendet werden. Es muss lediglich sichergestellt werden, dass sowohl der Kunde als auch der Cloud-Anbieter den öffentlichen Schlüssel der vertrauenswürdigen dritten Partei als authentisch anerkennt und auf diese Art und Weise in der Lage ist, die Signatur zu verifizieren. Abschließend wird die Signatur `Tsign` von der vertrauenswürdigen dritten Partei zum Cloud-Anbieter zurückgesandt. Der Kunde erhält vom Cloud-Anbieter als Antwort auf seine Anfrage Q_1 die folgenden Werte: den Auditwert A_1 und das Element (`Csign`, `Ttime`) mit der Signatur `Tsign` der vertrauenswürdigen dritten Partei. Die Signatur der vertrauenswürdigen dritten Partei wird anschließend vom Kunden überprüft.

Mit Hilfe der Signatur `Csign` des Cloud-Anbieters ist die Anfrage Q_1 eindeutig dem Auditwert A_1 zugeordnet und gleichzeitig die Identität des Cloud-Anbieters belegt. Außerdem wird durch den Zeitstempel `Ttime` und die Signatur `Tsign` der vertrauenswürdigen dritten Partei der zeitliche Bezug der Anfrage sichergestellt. Hieraus ergibt sich die in Abschnitt 5.1 als Zurechenbarkeit formulierte Anforderung. Nachweis durch Satz 5.2.

Satz 5.2 *Einem Kunden ist es nicht möglich eine gültige Signatur* `Csign` *des Cloud-Anbieters zu erzeugen. Gleichermaßen ist es weder dem Kunden noch dem Cloud-Anbieter möglich, eine Signatur* `Tsign` *der vertrauenswürdigen dritten Partei für beliebige Werte zu erzeugen.*

Beweis. Der Cloud-Anbieter und die vertrauenswürdige dritte Partei verwenden ein sicheres asymmetrischen Signaturverfahren zum Erstellen von `Csign` und `Tsign`. Aus der Geheimhaltung der privaten Schlüssel und der Überprüfung der Signaturen folgt daher direkt die Zurechenbarkeit von Auditwerten.

Weiterhin wird durch die Beschränkung der Parteien auf Polynomialzeit sichergestellt, dass weder der Cloud-Anbieter noch der Kunde gültige Signaturen der vertrauenswürdigen dritten Partei erstellen können. □

5.6 Überprüfungsphase

In der Überprüfungsphase von *CoCoPAS* können jeweils zwei in Auditwerten hinterlegte Monitoringdaten auf Gleichheit oder auf Ungleichheit überprüft werden, ohne dass dazu die zu überprüfenden Monitoringdatenelemente dem Kunden bekannt werden. Da nach

Satz 5.1 Auditwerte bezüglich der darin hinterlegten Monitoringdaten Vertraulichkeit bieten, ist dem Kunden eine direkte Überprüfung ohne weiteres nicht möglich. Vielmehr braucht es einen weiteren Nachrichtenaustausch zwischen Kunde und Cloud-Anbieter. Dazu sendet der Kunde zwei in der Hinterlegungsphase erhaltene Auditwerte und die Art der gewünschten Überprüfung, Gleichheit oder Ungleichheit, an den Cloud-Anbieter. Der Cloud-Anbieter erzeugt daraufhin einen *Beweis*, der die zu überprüfende Eigenschaft der in den Auditwerten hinterlegten Monitoringdaten belegt und sendet diesen Beweis an den Kunden zurück. Die Überprüfung auf Gleichheit oder Ungleichheit zweier in Auditwerten hinterlegten Monitoringdaten ist erfolgreich, wenn der Kunde den Beweis des Cloud-Anbieters akzeptiert. Sie ist fehlgeschlagen, falls der Beweis vom Kunden nicht akzeptiert wird. Ein Fehlschlagen des Beweises verursacht durch den Cloud-Anbieter ist an dieser Stelle auszuschließen, da hierzu der Cloud-Anbieter vom vorgegebenen Protokollablauf abweichen müsste, was nicht in seinem Interesse ist – der Cloud-Anbieter möchte ja gerade den Kunden von der Gleichheit oder Ungleichheit der in Auditwerten hinterlegten Monitoringdaten überzeugen.

Im Folgenden wird zunächst erläutert, wie die in zwei Auditwerten hinterlegten Monitoringdaten von einem Kunden auf Gleichheit überprüft werden können. Anschließend wird darauf eingegangen, wie die Überprüfung auf Ungleichheit abläuft.

5.6.1 Gleichheit

Der erste Auditwert, der aus einer Anfrage des Kunden zu einem bestimmten Monitoringdatenelement hervorgeht, wird zunächst vom Cloud-Anbieter regulär erzeugt (vergleiche Abschnitt 5.5). Sei A_1 der erste Auditwert zu dem Monitoringdatenelement S_1, so wählt der Cloud-Anbieter einen Zufallswert r_1 und ruft die *hinterlege*-Funktion zur Hinterlegung auf: $A_1 = \texttt{hinterlege}(S_1, r_1) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$.

Soll das Monitoringdatenelement S_1 zu einem späteren Zeitpunkt in dem Auditwert A_2 hinterlegt werden, so wird vom Cloud-Anbieter der zweite Auditwert A_2 in Abhängigkeit vom ersten Auditwert A_1 erzeugt. Dazu wählt der Cloud-Anbieter einen Zufallswert r_* und addiert diesen auf den für A_1 gewählten Zufallswert r_1. Somit ist $r_2 = r_1 + r_*$. Anschließend ruft der Cloud-Anbieter die *hinterlege*-Funktion zur Hinterlegung mit der Eingabe r_2 und S_1 auf: $A_2 = \texttt{hinterlege}(S_1, r_2) = (g^{r_2}, g^{S_1} \cdot h^{r_2}) = (g^{r_1+r_*}, g^{S_1} \cdot h^{r_1+r_*})$.

Dieses Vorgehen ist äquivalent zur *Masking*-Eigenschaft von Pedersen [150]. Der Auditwert A_2 kann auch ohne Aufruf der *hinterlege*-Funktion direkt aus dem Auditwert A_1

berechnet werden, in dem der Zufallswert r_* gemäß folgender Rechenvorschrift hinzugefügt wird:

$$A_2 = A_1 \circledast r_* := (z_1 \cdot g^{r_*}, z_2 \cdot h^{r_*}) = (g^{r_1} \cdot g^{r_*}, g^{S_1} \cdot h^{r_1} \cdot h^{r_*}) = (g^{r_1+r_*}, g^{S_1} \cdot h^{r_1+r_*})$$

Dabei beschreibt die Operation $\circledast$ die Erweiterung eines bestehenden Auditwertes mit neuem Zufall, sodass ein neuer Auditwert mit dem gleichen hinterlegten Monitoringdatenelement entsteht.

Sobald ein Kunde zwei Auditwerte $A_1 = (g^{r_1}, g^{S_1} \cdot h^{r_1})$ und $A_2 = (g^{r_1+r_*}, g^{S_1} \cdot h^{r_1+r_*})$ vom Cloud-Anbieter erhalten hat, kann der Kunde mittels *CoCoPAS* überprüfen, ob die in den Auditwerten hinterlegten Monitoringdaten gleich sind. Dazu fordert der Kunde vom Cloud-Anbieter einen Beweis für die Gleichheit (=) der Monitoringdaten an. Der Cloud-Anbieter übermittelt dazu den Wert r_* als Beweis an den Kunden. Der Kunde prüft dann, ob $A_2 \stackrel{?}{=} A_1 \circledast r_*$ gilt. Dies ist für die *Masking*-Eigenschaft von Pedersen genau dann der Fall, wenn das in A_1 und A_2 enthaltene Monitoringdatenelement S_1 das Gleiche ist und A_2 unter Verwendung von r_* in Abhängigkeit von A_1 erstellt wurde. Andernfalls schlägt die Überprüfung auf Gleichheit fehl und der Beweis wird vom Kunden nicht akzeptiert.

Auf diese Art und Weise kann die Gleichheit der in zwei Auditwerten hinterlegten Monitoringdaten gezeigt werden. Das Monitoringdatenelement S_1 selber muss dazu nicht vom Cloud-Anbieter aufgedeckt werden, sodass die in Abschnitt 5.1 als Geheimhaltung formulierte Anforderung gilt. Lediglich der zu dem Auditwert A_1 hinzugefügte Zufallswert r_* wird dem Kunden als Beweis für Gleichheit bekannt. Außerdem erfährt der Kunde, dass in dem Auditwert A_1 und in dem Auditwert A_2 das gleiche Monitoringdatenelement vom Cloud-Anbieter hinterlegt wurde, was dem Ziel von *CoCoPAS* in der Überprüfungsphase auf Gleichheit entspricht. Der Umkehrschluss, dass, wenn sich keine Gleichheit zeigen lässt, automatisch Ungleichheit gilt, ist nicht zulässig. Vielmehr könnte der Cloud-Anbieter ein nicht passendes r_* liefern, obwohl die in den Auditwerten A_1 und A_2 hinterlegten Monitoringdaten S_1 und S_2 gleich sind.

5.6.2 Ungleichheit

Zur Überprüfung auf Ungleichheit von zwei in Auditwerten hinterlegten Monitoringdaten legt der Cloud-Anbieter dem Kunden auf Anfrage einen entsprechenden Beweis vor. Anhand dieses Beweises kann der Kunde überprüfen, ob die hinterlegten Monitoringdaten unterschiedlich sind. Dabei wird die Ungleichheit durch den Cloud-Anbieter indirekt nachgewiesen. Der Cloud-Anbieter beweist dem Kunden, dass er ein Geheimnis i kennt, welches genau dann existiert und dem Cloud-Anbieter nur dann bekannt ist, wenn die in zwei Auditwerten hinterlegten Monitoringdaten ungleich sind.

Innerhalb von *CoCoPAS* wird für diesen Beweis eine eigens entwickelte Kombination des *ID Protocols* von Schnorr [160] mit der *Multiplication Technique* von Abe et al. [2] als *interaktiver Zero-Knowledge-Beweis* verwendet. Für den Zero-Knowledge-Beweis müssen dem Kunden daher weder das Geheimnis i, noch die in den Auditwerten hinterlegten Monitoringdaten vom Cloud-Anbieter mitgeteilt werden. Allerdings müssen dazu mehrere Nachrichten zwischen Kunde und Cloud-Anbieter ausgetauscht werden. Bei dem Protokollablauf des Zero-Knowledge-Beweises nimmt der Kunde die Rolle des *Verifizierers* und der Cloud-Anbieter die Rolle des *Beweisers* ein. Zusätzlich müssen von *CoCoPAS* während der Überprüfung auf Ungleichheit folgende, für Zero-Knowledge-Beweise, typische Anforderungen eingehalten werden:

Vollständigkeit: Bei Ungleichheit der in den beiden Auditwerten hinterlegten Monitoringdaten soll der Verifizierer nach Ablauf des Protokolls den Zero-Knowledge-Beweis akzeptieren.

Zuverlässigkeit: Bei Gleichheit der in den beiden Auditwerten hinterlegten Monitoringdaten soll es einem Beweiser nicht möglich sein, dem Verifizierer einen Beweis vorzulegen, den dieser akzeptiert.

Zero-Knowledge-Eigenschaft: Diese Eigenschaft kann als eine abgewandelte Form der in Abschnitt 5.1 als Geheimhaltung formulierten Anforderung betrachtet werden. Aus der Interaktion zwischen Beweiser und Verifizierer dürfen neben dem Beweis auf Ungleichheit keine Informationen über die in Auditwerten hinterlegten Monitoringdaten bekannt werden.

Gegeben ist der Auditwert $A_1 = (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$ und der Auditwert $A_2 = (z_3, z_4) = (g^{r_2}, g^{S_2} \cdot h^{r_2})$ mit $S_1 \neq S_2$. In diesem Fall ist das multiplikativ inverse Element $i = (S_1 - S_2)^{-1}$ zu der Differenz der hinterlegten Monitoringdaten $S_1 - S_2$ das Geheimnis, dessen Existenz der Cloud-Anbieter dem Kunden mit Hilfe des Zero-Knowledge-Beweises belegen möchte. Ein solches multiplikativ inverses Element i existiert genau dann, wenn die Differenz aus $S_1 - S_2 \neq 0$ ist. Zu dem Wert 0 (bei Gleichheit von S_1 und S_2) kann kein multiplikativ inverses Element existieren.

Zunächst wird der Quotient D aus den in den Auditwerten A_1 und A_2 hinterlegten Werten $z_2 := \texttt{commit}(S_1, r_1)$ und $z_4 := \texttt{commit}(S_2, r_2)$ gebildet, welche ihrerseits aus der Verwendung der *commit*-Funktion von ElGamal hervorgehen.

$$D := \frac{z_2}{z_4} = \frac{\texttt{commit}(S_1, r_1)}{\texttt{commit}(S_2, r_2)} = \frac{g^{S_1} \cdot h^{r_1}}{g^{S_2} \cdot h^{r_2}} = g^{S_1 - S_2} \cdot h^{r_1 - r_2}$$

Der Quotient D enthält im Exponenten von g die Differenz $S_1 - S_2$ der hinterlegten Monitoringdaten. Der Cloud-Anbieter kann den Kunden davon überzeugen, dass er ein multiplikativ inverses Element i zur Differenz $S_1 - S_2$ der hinterlegten Monitoringdaten

kennt, indem er den Quotienten D zu $g^1 \cdot h^{(r_1-r_2)\cdot i}$ überführt, ohne jedoch dabei dem Kunden das Geheimnis i mitzuteilen.

$$D^i = (g^{S_1-S_2})^i \cdot (h^{r_1-r_2})^i = g^1 \cdot h^{(r_1-r_2)\cdot i}$$

Eine solche Überführung von D zu $g^1 \cdot h^{(r_1-r_2)\cdot i}$ ist nur dann möglich, wenn dem Cloud-Anbieter das multiplikativ inverse Element i bekannt ist. Dazu muss die Differenz $S_1 - S_2$ der Monitoringdaten ungleich dem Wert 0 sein, woraus folgt, dass die Monitoringdaten S_1 und S_2 verschieden zueinander sind. Bei Gleichheit der Monitoringdaten $S_1 = S_2$ ist eine Überführung des Quotienten D zu $g^1 \cdot h^{(r_1-r_2)\cdot i}$ nicht möglich (für $g \neq 1$):

$$D^i = (g^{S_1-S_2} \cdot h^{r_1-r_2})^i = (g^0 \cdot h^{r_1-r_2})^i = 1 \cdot h^{(r_1-r_2)\cdot i} \neq g^1 \cdot h^{(r_1-r_2)\cdot i}$$

Für den Zero-Knowledge-Beweis muss die in der Hinterlegungsphase verwendete *commit*-Funktion von ElGamal angepasst werden (vergleiche auch *Multiplication Technique* von Abe et al. [2]). Diese wird in Abschnitt 5.5 definiert als: $\texttt{commit}(x, y) := g^x \cdot h^y$. Bei dem Zero-Knowledge-Beweis wird im Folgenden anstelle des Generators g der Quotient D verwendet. Aus diesem Grund erhält die *commit*-Funktion von ElGamal einen weiteren Eingabeparameter z, der spezifiziert, welcher Generator für den ersten Faktor genutzt werden soll: $\texttt{commit}(z, x, y)$. Wird, wie bei der *commit*-Funktion von ElGamal der Generator g verwendet, so muss die abgewandelte *commit*-Funktion nun wie folgt aufgerufen werden: $\texttt{commit}(g, x, y) := g^x \cdot h^y$. Soll anstatt dessen der Quotient D anstelle des Generators g verwendet werden, so wird die abgewandelte *commit*-Funktion wie folgt aufgerufen:

$$\texttt{commit}(D, x, y) := D^x \cdot h^y$$

5.6.2.1 Vorbereitung der Zero-Knowledge-Beweise

Im Folgenden wird näher auf den Protokollablauf des Zero-Knowledge-Beweises eingegangen. Dieser Zero-Knowledge-Beweis soll belegen, dass ein multiplikativ inverses Element i zur Differenz $S_1 - S_2$ der in den Auditwerten A_1 und A_2 hinterlegten Monitoringdaten existiert und damit, dass $S_1 \neq S_2$ gilt.

Der Kunde startet die Überprüfung auf Ungleichheit der hinterlegten Monitoringdaten, indem er A_1 und A_2 und die Art der Überprüfung, d.h. Ungleichheit ($\neq$) an den Cloud-Anbieter sendet. Abbildung 5.3 visualisiert die Vorbereitungen zur Durchführung des Zero-Knowledge-Beweises. Hiernach berechnen sowohl der Cloud-Anbieter als auch der Kunde aus den gegeben Auditwerten $A_1 = (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$ und $A_2 = (z_3, z_4) = (g^{r_2}, g^{S_2} \cdot h^{r_2})$ den Quotienten $D = \frac{z_2}{z_4} = g^{S_1-S_2} \cdot h^{r_1-r_2}$.

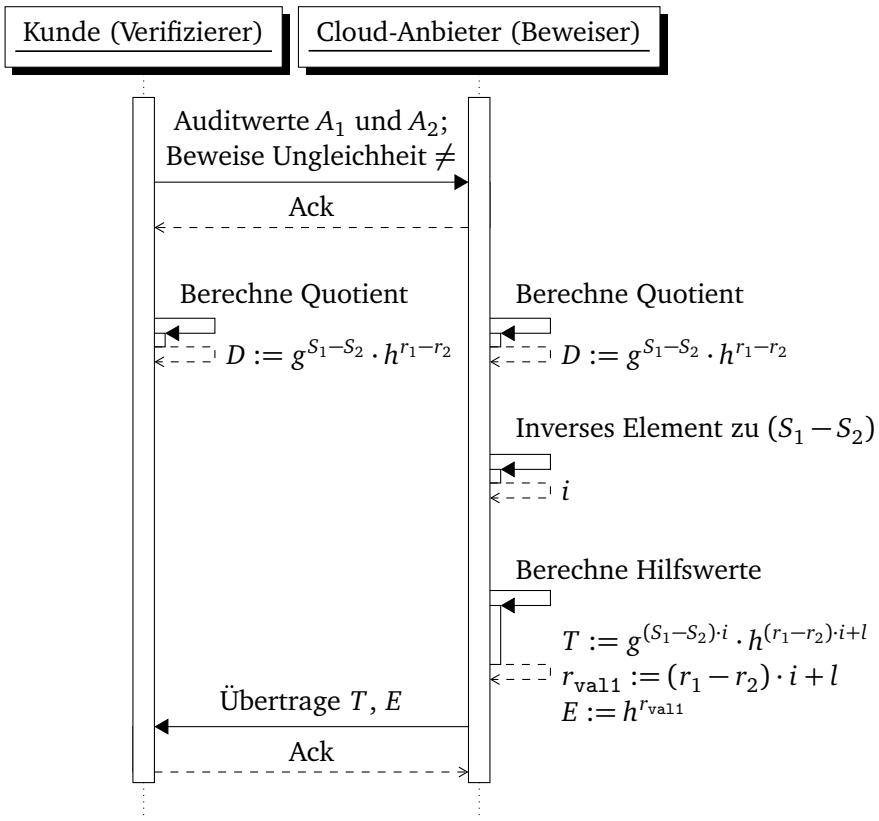

Abbildung 5.3: *Vorbereitung der Zero-Knowledge-Beweise für Ungleichheit der hinterlegten Monitoringdaten S_1 und S_2.*

Anschließend berechnet der Cloud-Anbieter das multiplikativ inverse Element i zur Differenz $S_1 - S_2$, beispielsweise mit Hilfe des erweiterten euklidischen Algorithmus (vergleiche [72] Seite 103f). Für den Zero-Knowledge-Beweis muss der Cloud-Anbieter außerdem zwei Hilfswerte T und E berechnen, so dass gilt:

$$\frac{T}{E} = g^{(S_1-S_2)\cdot i}$$

Zur Erzeugung des Hilfswerts T wählt der Cloud-Anbieter den Zufallswert $l \in \{1, \ldots, p-1\}$ mit ggT$(l, p) = 1$ und ruft die angepasste *commit*-Funktion wie folgt auf:

$$T := \texttt{commit}(D, i, l) = D^i \cdot h^l = g^{(S_1-S_2)\cdot i} \cdot h^{(r_1-r_2)\cdot i} \cdot h^l = g^{(S_1-S_2)\cdot i} \cdot h^{(r_1-r_2)\cdot i+l}$$

Weiter erzeugt der Cloud-Anbieter den Hilfswert E, indem er $r_{\texttt{val1}} := (r_1 - r_2) \cdot i + l$ wählt und h damit potenziert:

$$E := h^{r_{\texttt{val1}}} = h^{(r_1-r_2)\cdot i+l}$$

Die Hilfswerte T und E sind dabei vom Cloud-Anbieter so gewählt worden, dass sich durch Multiplikation von E mit g^1 der Hilfswert T ergibt. Dies ist genau dann der Fall, wenn der Wert i das multiplikativ inverse Element zu $S_1 - S_2$ ist. Trifft dies zu, dann gilt $(S_1 - S_2) \cdot i = 1$, woraus $S_1 \neq S_2$ folgt.

$$\begin{aligned} g^1 \cdot E &\equiv T \\ \texttt{commit}(g, 1, r_{\texttt{val1}}) &\equiv \texttt{commit}(D, i, l) \\ g^1 \cdot h^{(r_1 - r_2) \cdot i + l} &\equiv D^i \cdot h^l \\ g^1 \cdot h^{(r_1 - r_2) \cdot i + l} &\equiv \left(g^{S_1 - S_2} \cdot h^{r_1 - r_2}\right)^i \cdot h^l \\ g^1 \cdot h^{(r_1 - r_2) \cdot i + l} &\equiv g^{(S_1 - S_2) \cdot i} \cdot h^{(r_1 - r_2) \cdot i + l} \end{aligned}$$

5.6.2.2 Durchführung der Zero-Knowledge-Beweise

Nach Wahl der Hilfswerte E und T durch den Cloud-Anbieter kann der Protokollablauf der Zero-Knowledge-Beweise zur Existenz des multiplikativ inversen Elements i starten. Dazu müssen zwei Zero-Knowledge-Beweise durchgeführt werden. Dabei wird das *ID Protocol* von Schnorr [160] mehrfach ausgeführt.

Der Cloud-Anbieter beweist dem Kunden, dass er die Werte i und l kennt, die zur Erzeugung des Hilfswertes $T = D^i \cdot h^l$ notwendig waren. Außerdem beweist der Cloud-Anbieter dem Kunden, dass er den Wert $r_{\texttt{val1}}$ kennt, der zur Erzeugung des Hilfswertes $E = h^{r_{\texttt{val1}}} = h^{(r_1 - r_2) \cdot i + l}$ notwendig war. Beide Beweise müssen vollständig, zuverlässig und *Zero-Knowledge*-Eigenschaft erfüllen. Weiterhin muss der Kunde verifizieren, dass Folgendes gilt:

$$\begin{aligned} \frac{T}{E} &= g^1 \\ \frac{g^{(S_1 - S_2) \cdot i} \cdot h^{(r_1 - r_2) \cdot i + l}}{h^{(r_1 - r_2) \cdot i + l}} &= g^1 \end{aligned}$$

Dies ist genau dann der Fall, wenn i das multiplikativ inverse Element zu $S_1 - S_2$ ist, woraus $S_1 \neq S_2$ folgt. Durch die Verwendung von Zero-Knowledge-Beweisen erfährt der Kunde weder $r_{\texttt{val1}}$, i, l noch S_1, S_2 oder $S_1 - S_2$. Er lernt lediglich, dass die in den Auditwerten hinterlegten Monitoringdaten unterschiedlich sind.

Abbildung 5.4 visualisiert den Protokollablauf der Zero-Knowledge-Beweise. Für den Nachweis der Kenntnis der Werte i und l erzeugt der Cloud-Anbieter zunächst den

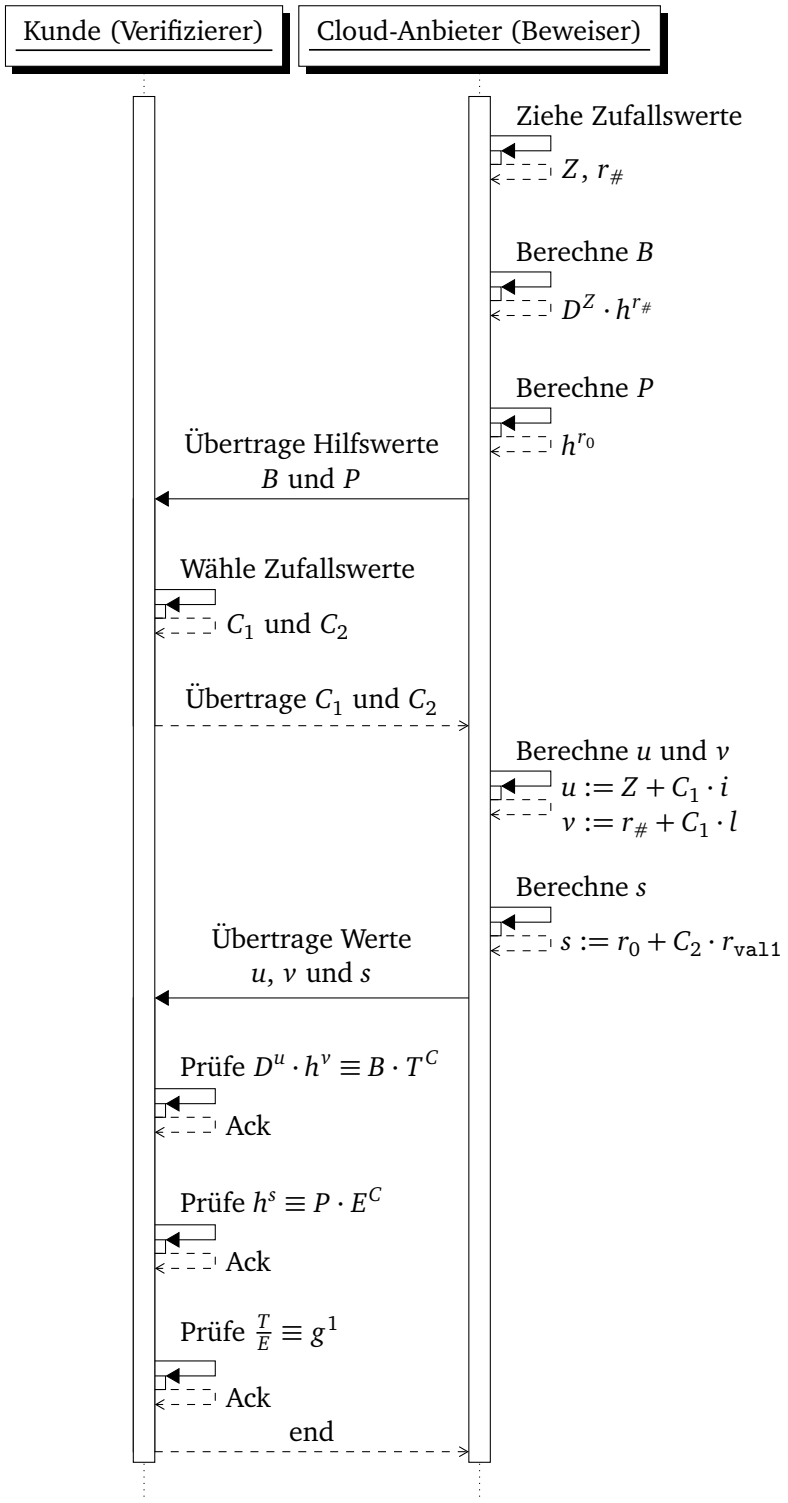

Abbildung 5.4: *Protokollablauf des Zero-Knowledge-Beweises für Nachweis der Ungleichheit.*

Hilfswert B, indem er die angepasste *commit*-Funktion mit zwei zufällig gewählten Werten Z und $r_{\#}$ aufruft.

$$B := \texttt{commit}(D, Z, r_{\#}) = D^{Z} \cdot h^{r_{\#}}$$

Anschließend berechnet der Cloud-Anbieter für den Nachweis der Kenntnis des Wertes $r_{\texttt{val1}}$ den Hilfswert P, indem h mit einem weiteren zufällig gewählten Wert r_0 potenziert. Die Werte B und P werden dem *ID Protocol* von Schnorr [160] folgend vom Cloud-Anbieter zum Kunden übertragen.

Im weiteren Verlauf überzeugt sich der Kunde entweder davon, wie die Werte B und P zustande gekommen sind, oder fährt mit dem weiteren Ablauf des Zero-Knowledge-Protokolls fort. Durch diese Option wird verhindert, dass der Cloud-Anbieter die Werte B und P nicht wie im Zero-Knowledge-Protokoll vorgesehen berechnet hat. Findet eine Überprüfung der *Form* der Hilfswerte statt und das Verfahren wird nicht fortgesetzt, überträgt der Cloud-Anbieter die Werte Z, $r_{\#}$ und r_0 an den Kunden. Hierbei erfährt der Kunde keine zusätzlichen Informationen da es sich um Zufallswerte handelt.

Wird das Verfahren fortgesetzt, wählt der Kunde seinerseits zwei Zufallswerte C_1 und C_2 und sendet diese an den Cloud-Anbieter. Auf Basis des vom Kunden erhaltenen Zufallswertes C_1 berechnet der Cloud-Anbieter die Werte u, v und s, sodass gilt:

$$\begin{aligned} u &:= Z + C_1 \cdot i \\ v &:= r_{\#} + C_1 \cdot l \\ s &:= r_0 + C_2 \cdot r_{\texttt{val1}} \end{aligned}$$

Anhand von u, v und s kann der Cloud-Anbieter dem Kunden später belegen, dass er in der Lage ist, den vom Kunden erhaltenen Zufallswert C_1 im Exponenten mit dem inversen Element i zu multiplizieren. Der Hilfswert B dient hier nur zur Verschleierung des tatsächlich im Hilfswert T enthaltenen Exponenten i und muss daher für jeden Durchgang neu gewählt werden. Der Cloud-Anbieter überträgt die Werte u, v und s an den Kunden.

Der Kunde seinerseits ist nun in der Lage zu überprüfen, ob $D^u \cdot h^v \equiv B \cdot T^{C_1}$ gilt.

$$\begin{aligned} D^u \cdot h^v &= D^{(Z+C\cdot i)} \cdot h^{(r_{\#}+C_1\cdot l)} \\ &= D^Z \cdot D^{C_1\cdot i} \cdot h^{r_{\#}} \cdot h^{C_1\cdot l} \\ &= D^Z \cdot h^{r_{\#}} \cdot (D^i)^{C_1} \cdot (h^l)^{C_1} \\ &= D^Z \cdot h^{r_{\#}} \cdot (D^i \cdot h^l)^{C_1} \\ &\Rightarrow B \cdot T^{C_1} \end{aligned}$$

Ist dies der Fall, so hat der Cloud-Anbieter den Kunden davon überzeugt, dass er die Werte i und l kennt, mit denen der Hilfswert T erzeugt wurde.

Außerdem überprüft der Kunde ob $h^s \equiv P \cdot E^{C_2}$ gilt.

$$\begin{aligned} h^s &= h^{r_0 + C_2 \cdot r_{\texttt{val1}}} \\ &= h^{r_0} \cdot h^{C_2 \cdot r_{\texttt{val1}}} \\ &= h^{r_0} \cdot (h^{r_{\texttt{val1}}})^{C_2} \\ &\Rightarrow P \cdot E^{C_2} \end{aligned}$$

Ist dies der Fall, so hat der Cloud-Anbieter den Kunden davon überzeugt, dass er den Wert $r_{\texttt{val1}}$ kennt, mit dem der Hilfswert E erzeugt wurde.

Zusätzlich muss der Kunde nun überprüfen, ob $\frac{T}{E} \equiv g^1$ gilt.

$$\begin{aligned} \frac{T}{E} &= \frac{D^i \cdot h^l}{h^{(r_1 - r_2) \cdot i + l}} \\ &= \frac{(g^{S_1 - S_2} \cdot h^{r_1 - r_2})^i \cdot h^l}{h^{(r_1 - r_2) \cdot i + l}} \\ &= \frac{g^{(S_1 - S_2) \cdot i} \cdot h^{(r_1 - r_2) \cdot i} \cdot h^l}{h^{(r_1 - r_2) \cdot i + l}} \\ &= \frac{g^{(S_1 - S_2) \cdot i} \cdot h^{(r_1 - r_2) \cdot i + l}}{h^{(r_1 - r_2) \cdot i + l}} \\ &\Rightarrow g^1 \text{ (für } S_1 \neq S_2 \text{ ist } i \text{ multiplikativ inverses Element zu } S_1 - S_2) \end{aligned}$$

Ist dies der Fall, so hat der Cloud-Anbieter den Kunden davon überzeugt, dass der Wert i das multiplikativ inverse Element zu $S_1 - S_2$ ist und somit $S_1 \neq S_2$ gilt. Im Falle $S_1 = S_2$ würde die Überprüfung fehlschlagen, da $g^1 \neq g^0$ gilt (für $g \neq 1$).

Der Kunde kann sich nach erfolgreicher Überprüfung sicher sein, dass für die in den Auditwerten A_1 und A_2 hinterlegten Monitoringdaten Ungleichheit gilt ($S_1 \neq S_2$).

Die Vollständigkeit, Zuverlässigkeit und Zero-Knowledge-Eigenschaft der präsentierten interaktiven Zero-Knowledge-Beweise folgen direkt aus der Vollständigkeit und Zuverlässigkeit des *ID Protocols* von Schnorr [160]. Außerdem wird zur Überprüfung, ob es sich bei i um das multiplikativ inverse Element handelt, die *Multiplication Technique* von Abe et al. [2] angewandt und es gilt daher Vollständigkeit und Zuverlässigkeit.

5.6.2.3 Skizze für einen Simulator

Zum Nachweis der Zero-Knowledge-Eigenschaft von *CoCoPAS* bei Überprüfungen auf Ungleichheit wird im Folgenden ein Simulator skizziert.

Ein Verfahren erfüllt die *Zero-Knowledge*-Eigenschaft, sofern das zwischen Beweiser und Verifizier erzeugte Transkript genauso gut von einem Simulator erzeugt werden könnte. Der Simulator darf hierzu kein Zugriff auf die Geheimnisse des Beweisers, hier des Cloud-Anbieters, erhalten.

Ausgangsbasis für den Zero-Knowledge-Beweis auf Ungleichheit ist der Quotient D. Auf dessen Basis berechnet der Cloud-Anbieter, wie oben beschrieben, die Werte T und E und tauscht diese mit dem Kunden aus. Der Kunde prüft ob $\frac{T}{E} = g^1$ gilt.

Ein Simulator kann die beiden Werte dabei ohne Kenntnis des multiplikativ inversen Elements i berechnen, indem er zunächst T durch Anwendung von $\texttt{commit}(D, n, m)$ berechnet. Die Werte n und m sind dabei zwei vom Simulator frei wählbare Werte. Anschließend berechnet der Simulator den Wert E wie folgt:

$$E := \frac{1}{T^{-1} \cdot g}$$

Hierdurch gilt intuitiv:

$$\frac{T}{E} = T \cdot T^{-1} \cdot g = g$$

Dies gilt unabhängig von den vom Simulator gewählten Werte n und m.

Weiter kann der Simulator das oben erzeugte Transkript zwischen Kunde und Cloud-Anbieter wie folgt erzeugen. Für die Simulation des Falls, dass der Kunde sich von der Echtheit der Werte B und P überzeugen möchte, erzeugt der Simulator die beiden Werte mit Hilfe der Zufallswerte Z, $r_\#$ und r_0 wie vorgeschrieben:

$$B := D^Z \cdot h^{r_\#}$$
$$P := h^{r_0}$$

Für den Fall der Simulation des Transkripts des weiteren Protokollablaufs erzeugt der Simulator die beiden Werte zunächst nicht. Anstatt dessen initiiert der Simulator zunächst die Werte u, v und s mit Zufallswerten. An dieser Stelle sei darauf hingewiesen, dass der Simulator bei Erstellung des Transkripts nicht die oben vorgeschriebene Reihenfolge einhalten muss, weswegen er mit den Werten u, v und s beginnen kann.

Nachdem er diese Werte zufällig belegt hat, wählt er C_1 und C_2 ebenfalls zufällig.

Anschließend berechnet der Simulator die Werte B und P wie folgt:

$$B := \frac{g^u \cdot h^v}{T^{C_1}}$$
$$P := \frac{h^s}{E^{C_2}}$$

Hierdurch ist der Simulator wie gefordert in der Lage, ein gültiges Transkript auszugeben. In der Simulation sendet der Cloud-Anbieter die eben erzeugten Werte B und P an den Kunden. Dieser wählt die vom Simulator vorgegebenen Werte C_1 und C_2 und sendet diese an den Cloud-Anbieter. Der Cloud-Anbieter sendet in der Simulation die eben berechneten Werte u, v und s an den Kunden zurück. Folglich läuft die Überprüfung auf Seiten des Kunden erfolgreich ab:

$$\frac{T}{E} = g^1$$
$$D^u \cdot h^v = B \cdot T^{C_1}$$
$$h^s = P \cdot E^{C_2}$$

Es existiert also ein Simulator, der das durch den Zero-Knowledge-Beweis erzeugte Transkript insbesondere ohne Kenntnis von S_1, S_2 oder i erzeugt. In weiterführenden Arbeiten können auf Grundlage der hier angeführten Skizze weitergehende Untersuchungen, beispielsweise zu Effizienz des Verfahrens, angestellt werden. Diese sind nicht Gegenstand dieser Dissertation.

Während der Simulator zur Erzeugung des Transkripts die Reihenfolge der zwischen Kunde und Cloud-Anbieter ausgetauschten Nachrichten frei wählen kann, ist bei der realen Interaktion zwischen Kunde und Cloud-Anbieter die Einhaltung der Reihenfolge bei *CoCoPAS* verpflichtend. Eben hierdurch kann sich der Kunde versichern, dass zum Quotient tatsächlich ein multiplikativ inverses Element existiert und der Cloud-Anbieter dieses kennt. Insbesondere dadurch, dass erst nachdem der Cloud-Anbieter die Werte P und B an den Kunden übermittelt hat, der Kunde seinerseits die Werte C_1 und C_2 wählt. Hierdurch wird der Cloud-Anbieter *gezwungen* im Exponenten sowohl zu addieren als auch zu multiplizieren, was er nur bei Kenntnis der Exponenten kann.

5.7 Aufdeckungsphase

Ziel dieser Phase ist es, dass das in einem Auditwert vom Cloud-Anbieter hinterlegte Monitoringdatenelement wieder eindeutig aufgedeckt werden kann. Die Aufdeckung von

Monitoringdaten findet dabei immer im Nachhinein statt, das heißt, der Kunde kann zu bereits erhaltenen Auditwerten zu einem späteren Zeitpunkt die darin hinterlegten Monitoringdaten anfordern. Sowohl der Cloud-Anbieter als auch die vertrauenswürdige dritte Partei sind in der Lage, zu einem vom Kunden gegebenen Auditwert das darin hinterlegte Monitoringdatenelement zu ermitteln und aufzudecken. Da mit dem Aufdecken von Auditwerten die zu schützenden Monitoringdaten des Cloud-Anbieters bekannt werden, ist die Aufdeckungsphase von *CoCoPAS* nur im Ausnahmefall vorgesehen. Ein solcher Ausnahmefall liegt vor, wenn sich Kunde und Cloud-Anbieter gerichtlich auseinandersetzen und daher zur Fehler-Ursachen-Analyse Zugriff auf die in Auditwerten hinterlegten Monitoringdaten erforderlich ist. Die in Abschnitt 5.1 als Geheimhaltung formulierte Anforderung muss dazu aufgeben werden, wohingegen Eindeutigkeit und Zurechenbarkeit weiter aufrechterhalten werden sollen. Der zeitliche Bezug eines Auditwertes wird dabei durch die Signatur des Zeitstempeldienstes der vertrauenswürdigen dritten Partei sichergestellt. Die Signatur erstreckt sich dabei über den Auditwert und den vertrauenswürdigen Zeitstempel `Ttime`, der während der Hinterlegungsphase angefordert wurde (vergleiche Abschnitt 5.5).

Anhand des folgenden Beispiels wird der Ablauf der Aufdeckungsphase demonstriert. Für das Beispiel wird angenommen, dass ein Cloud-Anbieter in dem Auditwert $A_1 := (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$ das Monitoringdatenelement S_1 hinterlegt hat. Zunächst wird im Folgenden beschrieben, wie das Aufdecken durch den Cloud-Anbieter abläuft. Alternativ, wenn die Aufdeckungsphase von *CoCoPAS* ohne Hilfe des Cloud-Anbieters genutzt werden soll, wird danach beschrieben, wie das Aufdecken durch eine vertrauenswürdige dritte Partei abläuft. Der Auditwert A_1 gilt als aufgedeckt, sobald vor Gericht das darin hinterlegte Monitoringdatenelement S_1 bekannt wird.

5.7.1 Aufdeckung durch den Cloud-Anbieter

Soll das in einem Auditwert hinterlegte Monitoringdatenelement durch den Cloud-Anbieter aufgedeckt werden, so muss zunächst der entsprechende Auditwert vom Kunden an den Cloud-Anbieter übermittelt werden. Dieser überprüft, ob es sich dabei um einen Auditwert handelt, den er selber ausgestellt hat. Für diese Überprüfung verwendet der Cloud-Anbieter die in der Hinterlegungsphase zum Auditwert hinzugefügte Signatur (vergleiche Abschnitt 5.5). Ist dies der Fall, so ermittelt der Cloud-Anbieter die Werte S_1 und r_1 die er zum Erstellen des Auditwertes A_1 verwendet hat. Diese übermittelt der Cloud-Anbieter an die Instanz (in der Regel der Sachverständige vor Gericht), welche die Aufdeckung veranlasst hat. Diese überprüft, ob $g^{r_1} \equiv z_1$ und ob $g^{S_1} \cdot h^{r_1} \equiv z_2$ gilt. Ist dies der Fall, so ist S_1 das im Auditwert A_1 vom Cloud-Anbieter hinterlegte Monitoringdatenelement.

Die in Abschnitt 5.1 als Eindeutigkeit formulierte Anforderung ist erfüllt. Der Nachweis erfolgt durch Satz 5.3.

Satz 5.3 *Dem Cloud-Anbieter ist es nicht möglich, das in einem Auditwert A_1 hinterlegte Monitoringdatenelement S_1, zu einem Monitoringdatenelement S_2 mit $S_2 \neq S_1$ aufzudecken. Gegeben sind (G, g, h) und der Auditwert $A_1 := (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$.*

Beweis. Der Nachweis der Eindeutigkeit von Auditwerten folgt direkt aus dem hier verwendeten Hinterlegungsverfahren von ElGamal [51]. Dieses garantiert bereits Eindeutigkeit, ohne Zuhilfenahme von weiteren Annahmen. Vielmehr legt der Wert z_1 eines Auditwerts eindeutig fest, welcher Zufall r_1 zum Aufdecken verwendet werden kann und somit auch, welches Monitoringdatenelement S_1 in dem Wert z_2 enthalten ist. Eindeutigkeit gilt also, ohne dass dazu der Cloud-Anbieter in seiner Rechenleistung beschränkt werden müsste (vergleiche Abschnitt 5.1), sondern vielmehr alleine durch die mathematische Konstruktion des Verfahrens.

Hieraus folgt: Eindeutigkeit für einen Auditwert ist gegeben. □

5.7.2 Aufdeckung durch die vertrauenswürdige dritte Partei

Die vertrauenswürdige dritte Partei, welche bei der Vorbereitung von *CoCoPAS* mitgewirkt hat (vergleiche Abschnitt 5.4), kann mit Hilfe der Geheimtür Auditwerte aufdecken. Während der Vorbereitung von *CoCoPAS* hat die vertrauenswürdige dritte Partei ein Geheimnis s gewählt, für welches gilt $g^s = h$.

Wird der vertrauenswürdigen dritten Partei ein Auditwert $A_1 := (z_1, z_2) = (g^{r_1}, g^{S_1} \cdot h^{r_1})$ zur Aufdeckung vorgelegt, so geht diese wie folgt vor. Zunächst dividiert die vertrauenswürdige dritte Partei den Wert z_2 durch den Wert z_1, der zuvor mit dem Geheimnis s potenziert wurde, so dass g^{S_1} entsteht.

$$\frac{z_2}{(z_1)^s} = \frac{g^{S_1} \cdot h^{r_1}}{(g^{r_1})^s} = \frac{g^{S_1} \cdot g^{s \cdot r_1}}{g^{r_1 \cdot s}} = g^{S_1}$$

Da die in Abschnitt 5.3.3 formulierten Annahmen (DLOG, CDH, DDH) weiter gelten, muss die vertrauenswürdige dritte Partei eine Lookup-Tabelle zur Zuordnung von S_1 zu g^{S_1} verwenden. Ohne Beschränkung der Allgemeinheit kann davon ausgegangen werden, dass der Cloud-Anbieter eine solche Lookup-Tabelle, welche sämtliche Monitoringdaten S auf g^S abbildet, der vertrauenswürdigen dritten Partei im Vorfeld zur Verfügung stellt. Satz 5.3 und die in Abschnitt 5.1 als Eindeutigkeit formulierte Anforderung gelten unverändert.

5.8 Evaluation

Die in Abschnitt 5.1 formulierten Anforderungen *Geheimhaltung*, *Eindeutigkeit* und *Zurechenbarkeit* wurden jeweils in den entsprechenden Phasen von *CoCoPAS* betrachtet (siehe Abschnitt 5.5, 5.6 und 5.7). In diesem Abschnitt folgt nun eine Angriffsanalyse, analytische Betrachtung und die Untersuchung des durch *CoCoPAS* entstehenden Overheads. Hier ist insbesondere die Ausführungszeit und die in den einzelnen Phasen entstehende Datenmenge relevant. Weiterhin spielt der verwendete Sicherheitsgrad eine wesentliche Rolle. Dieser wird unterschieden in sogenannte “short-term” (112 bits), “near-term” (128 bits) und “long-term” (192 bits) Sicherheit (vergleiche [52, 144, 24]). Die Angabe in *bits* beim Sicherheitsgrad bezieht sich dabei auf die Schlüssellänge für symmetrische Verfahren. Für Diffie-Hellman-basierte Verfahren werden dazu 3072 bits für den Modulus p gewählt [4, 214]. Für *CoCoPAS* eignet sich “near-term” Sicherheit, da man darunter einen Schutz für die nächsten 10 Jahre versteht. Es kann davon ausgegangen werden, dass in Auditwerten hinterlegte Monitoringdaten des Cloud-Anbieters in 10 Jahren nicht mehr schützenswert sind, da der Cloud-Anbieter bis dahin seine Infrastruktur maßgeblich abgeändert hat und auch keine rechtliche Haftung mehr besteht.

5.8.1 Angriffsanalyse

Bei der Analyse möglicher Angriffe auf *CoCoPAS* muss unterschieden werden, welche Entität den Angriff durchführt und was das Angriffsziel ist. Hier kommt einerseits der Kunde als Angreifer, der die Geheimhaltung oder die Zurechenbarkeit von Auditwerten brechen möchte, in Betracht. Außerdem ist als Angreifer der Cloud-Anbieter, der die Zurechenbarkeit oder die Eindeutigkeit von Auditwerten brechen möchte, denkbar.

Wird beispielsweise vom Kunden als Angreifer ausgegangen, dann kann dieser aufgrund seiner beschränkten Rechenleistung nicht die in Auditwerten enthalten Monitoringdaten ohne Hilfe des Cloud-Anbieters aufdecken. Dies ergibt sich aus den Eigenschaften des verwendeten Hinterlegungsverfahrens (siehe Abschnitt 5.5). Außerdem ist es einem Kunden nicht möglich, Auditwerte zu erstellen, die den Anschein erwecken, als hätte sie der Cloud-Anbieter erzeugt. Dies ergibt sich durch das verwendete Signaturverfahren. Die Monitoringdaten des Cloud-Anbieters werden also geschützt.

Geht man vom Cloud-Anbieter als Angreifer aus, so ist dieser nicht in der Lage, die in einem Auditwert hinterlegten Monitoringdaten im Nachhinein zu ändern. Dies ergibt sich aus den Eigenschaften des verwendeten Hinterlegungsverfahrens (siehe Abschnitt 5.5) und gilt aufgrund des verwendeten Verfahrens unabhängig von der dem Cloud-Anbieter zur Verfügung stehenden Rechenleistung – dies ist eine Eigenschaft von auf ElGamal

basierenden Hinterlegungsverfahren [51]. Der zeitliche Bezug oder die Zuordnung von Auditwerten zu Anfragen des Kunden lässt sich für den Cloud-Anbieter aufgrund der Verwendung des Zeitstempeldienstes der vertrauenswürdigen dritten Partei und deren Signaturen nicht brechen. Hier greift die Annahme das der Cloud-Anbieter in seiner Rechenleistung auf Polynomialzeit beschränkt ist und ein hinreichend sicherer Signaturverfahren verwandt wird. Allerdings kann der Cloud-Anbieter als Angreifer in der Überprüfungsphase falsche Beweise liefern und so erreichen, dass eine Überprüfung auf Gleichheit oder Ungleichheit fehlschlägt, obwohl diese eigentlich erfolgreich ablaufen müsste. Dieser mögliche Angriff ist allerdings nur ein theoretisches Problem, da der Angreifer nur den Misserfolg einer Überprüfung fälschen, aber nicht falsche Beweise senden kann, die den Kunden von Gleichheit oder Ungleichheit überzeugen. Führt der Cloud-Anbieter diesen Angriff dennoch durch und die Überprüfung von vertraglichen Vereinbarungen schlägt aufgrund fehlerhafter Beweise fehl, so gefährdet dies nachhaltig die Vertrauensbeziehung zwischen Cloud-Anbieter und Kunde. Vielmehr liegt es im Interesse des Cloud-Anbieters, dem Kunden die Einhaltung von vertraglichen Vereinbarungen nachzuweisen. Außerdem wird im Rahmen dieser Dissertation zunächst von einem ehrlichen Cloud-Anbieter ausgegangen.

5.8.2 Analytische Betrachtung

Neben den in den entsprechenden Phasen von *CoCoPAS* betrachteten formalen Anforderungen kann *CoCoPAS* auch analytisch evaluiert werden. Besonders im Fokus stehen die Überprüfungs- und die Aufdeckungsphase.

Bei der Überprüfung auf Gleichheit (siehe Abschnitt 5.6.1) ist ein Nachrichtenaustausch zwischen Cloud-Anbieter und Kunde erforderlich (Übertragung von r_*). Auf Seiten des Kunden müssen des Weiteren zwei Rechenoperationen durchgeführt werden: eine Potenzierung (h^{r_*}) und eine Multiplikation. Der Cloud-Anbieter hingegen hat keinen zusätzlichen Rechenaufwand, da alle benötigten Werte (r_*) bereits in der Hinterlegungsphase erzeugt wurden.

Die Überprüfung auf Ungleichheit (siehe Abschnitt 5.6.2) hingegen ist deutlich aufwendiger: es müssen mehrere Nachrichten ausgetauscht werden und es entsteht mehr Rechenaufwand sowohl auf Seiten des Cloud-Anbieters als auch auf Seiten des Kunden. Insgesamt müssen drei Nachrichten vom Cloud-Anbieter zum Kunden und eine Nachricht vom Kunden zum Cloud-Anbieter übertragen werden. Des Weiteren muss der Cloud-Anbieter folgende Rechenoperationen durchführen: eine Subtraktion, vier Additionen, sechs Multiplikationen, eine Division und fünf Potenzierungen. Hinzu kommt die Ausführung des erweiterten euklidischen Algorithmus. Während auf Seiten des Cloud-Anbieters die eben beschriebenen Rechenoperationen erforderlich sind, muss der Kunde

folgende Rechenoperationen durchführen: drei Multiplikationen, zwei Divisionen und fünf Potenzierungen. Die Überprüfung auf Ungleichheit ist daher deutlich aufwendiger, als eine Überprüfung auf Gleichheit, wobei der Mehraufwand deutlich auf Seiten des Cloud-Anbieters entsteht. Der Kunde hingegen muss lediglich zwei Multiplikationen, eine Division und vier Potenzierungen mehr ausführen, als bei der Überprüfung auf Gleichheit.

Die Aufdeckungsphase (siehe Abschnitt 5.7) kann in Aufdeckung durch den Cloud-Anbieter und Aufdeckung durch die vertrauenswürdige dritte Partei unterschieden werden. Während der Cloud-Anbieter die in Auditwerten hinterlegten Monitoringdaten ohne zusätzliche Rechenoperationen direkt aufdecken kann, muss die vertrauenswürdige dritte Partei zwei Rechenoperationen durchführen: eine Division und eine Potenzierung. Aus diesem Grund kann die Aufdeckungsphase als effizient angesehen werden.

5.8.3 Ausführungszeit

Zur näheren Betrachtung des Overheads wurde *CoCoPAS* in *python 2.7* unter Benutzung der *GMP Library (C) 6.0.0a / gmp* und *OpenSSL 1.0.1i / M2Crypto* implementiert. Nachrichten wurden dabei in *ASN1/DER* (Abstract Syntax Notation One / Distinguished Encoding Rules) [103] kodiert. Als Zeitstempeldienst für die vertrauenswürdige dritte Partei wurde RFC3161 [210] umgesetzt. Für Signaturen kommt das in RFC5652 [227] beschriebene Verfahren und ECDSA [65] zum Einsatz. Die prototypische Implementierung von *CoCoPAS* wurde auf einer *2.5 Ghz Intel(R) Core(TM) i7-4710HQ CPU* mit *8 GB RAM* zur Ausführung gebracht. Als Szenario wurde dabei sowohl die Erzeugung von zwei Auditwerten und Überprüfung auf Gleichheit als auch auf Ungleichheit untersucht. Die in diesem Kapitel präsentierten Werte basieren jeweils auf den Mittelwerten von *1.000* durchgeführten Messungen. Die Standardabweichung war maximal 3%.

In Abbildung 5.5 wird die Ausführungszeit von *CoCoPAS* in Millisekunden (ms) für unterschiedliche Sicherheitsgrade untersucht. Außerdem erfolgt die Unterscheidung der Ausführungszeit nach Rollen: Kunde, Cloud-Anbieter und vertrauenswürdige dritte Partei. Auffallend ist hier, dass vor allem die Ausführungszeit des Kunden und die des Cloud-Anbieters mit steigendem Sicherheitsgrad exponentiell wächst. Die Ausführungszeit der vertrauenswürdigen dritten Partei bleibt hingegen nahezu konstant. Tabelle 5.1 (Ausführungszeit für Gleichheit) und Tabelle 5.2 (Ausführungszeit für Ungleichheit) verdeutlichen diese Beobachtung.

Die durch höheren Sicherheitsgrad verursachte Ausführungszeit bei dem Cloud-Anbieter lässt sich auf den Aufwand zur Erstellung der Auditwerte und dem Signieren der selbigen zurückführen. Die Ausführungszeit von *CoCoPAS* korreliert daher mit der Ausführungszeit

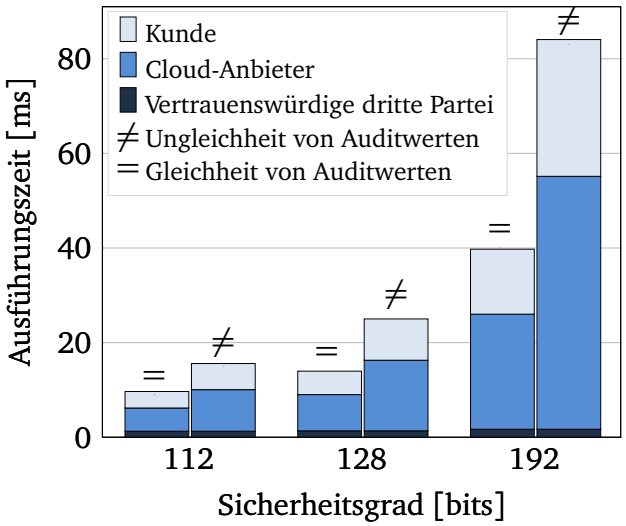

Abbildung 5.5: *Betrachtung der Ausführungszeit von CoCoPAS in Abhängigkeit des Sicherheitsgrads und der Rolle (Kunde, Cloud-Anbieter, vertrauenswürdige dritte Partei).*

des von *CoCoPAS* verwendeten Hinterlegungsverfahrens. Außerdem liegt die Ausführungszeit für Ungleichheit höher als die Ausführungszeit für Gleichheit, da hier zusätzliche Berechnungen für die Durchführung der Zero-Knowledge-Beweise notwendig sind. Der Kunde benötigt mit steigendem Sicherheitsgrad zusätzliche Rechenzeit zur Überprüfung der Signaturen und des Zero-Knowledge-Beweises bei Ungleichheit.

Die Ausführungszeit der vertrauenswürdigen dritten Partei bleibt nahezu konstant, da der Sicherheitsgrad des dort zum Einsatz kommenden Signaturverfahrens nicht variiert wurde. Lediglich ein leichter Anstieg lässt sich feststellen. Dies ist auf die bei *CoCoPAS* unter höheren Sicherheitsgraden entstehende größere Datenmenge zurückzuführen (siehe auch nächster Abschnitt). Die auf Seiten einer vertrauenswürdigen dritten Partei benötigte Signaturzeit ist nämlich abhängig von der zur signierenden Datenmenge.

Sicherheitsgrad	Kunde	Cloud-Anbieter	Vertrauenswürdige dritte Partei
112 bits	3.49 ms	4.91 ms	1.27 ms
128 bits	4.95 ms	7.66 ms	1.35 ms
192 bits	13.75 ms	24.32 ms	1.69 ms

Tabelle 5.1: *Ausführungszeit von CoCoPAS zur Überprüfung auf Gleichheit (=) von in Auditwerten hinterlegten Monitoringdaten.*

Sicherheitsgrad	Kunde	Cloud-Anbieter	Vertrauenswürdige dritte Partei
112 bits	5.53 ms	8.78 ms	1.28 ms
128 bits	8.73 ms	14.93 ms	1.35 ms
192 bits	28.91 ms	53.45 ms	1.70 ms

Tabelle 5.2: *Ausführungszeit von CoCoPAS zur Überprüfung auf Ungleichheit (≠) von in Auditwerten hinterlegten Monitoringdaten.*

5.8.4 Datenmenge

Nachstehend wird die bei *CoCoPAS* entstehende Datenmenge untersucht. In Tabelle 5.3 wird dazu die Menge der bei einer Überprüfung auf Gleichheit entstehenden Daten der Überprüfung auf Ungleichheit gegenübergestellt.

Auffallend ist hier, dass die Datenmenge der Anfrage an die vertrauenswürdige dritte Partei, beziehungsweise deren Antwort, in beiden Fällen identisch ist. Dies liegt an dem Umstand, dass sowohl bei Gleichheit als auch bei Ungleichheit zwei Auditwerte des Cloud-Anbieters von der vertrauenswürdigen dritten Partei mit einem Zeitstempel und einer Signatur versehen werden müssen. Allerdings unterscheidet sich die bei Ungleichheit entstehenden Datenmenge deutlich von der bei Gleichheit entstehende Datenmenge. Dies ist auf die Verwendung der Zero-Knowledge-Beweise und die darin zusätzlich enthaltenen Hilfswerte zurückzuführen.

Außerdem zeigt die Tabelle 5.3 deutlich, dass die je Auditwert zu speichernden Daten hier sehr gering sind (wenige kB). Dominiert wird die Datenmenge durch die Signatur des Cloud-Anbieters.

Ablauf	Gleichheit (=)	Ungleichheit (≠)
Auditwerte und Beweis	1,096 bytes	1,996 bytes
Anfrage an vertrauenswürdige dritte Partei	54 bytes	54 bytes
Antwort an vertrauenswürdige dritte Partei	322 bytes	322 bytes
Signatur des Cloud-Anbieters	1,641 bytes	2,542 bytes

Tabelle 5.3: *Betrachtung der Menge an durch CoCoPAS in der Aufdeckungsphase erzeugten Daten. Sicherheitsgrad 128 bit (near-term security).*

5.9 Verwandte Arbeiten

Da es sich bei *CoCoPAS* um die Lösung eines sehr speziellen Problems handelt, sind für die Überprüfung auf Ungleichheit von hinterlegten Werten derzeit keine verwandten Arbeiten bekannt. Allerdings werden in Teilbereichen von *CoCoPAS* (Hinterlegungsphase, Aufdeckungsphase, Überprüfung auf Gleichheit) existierende Arbeiten verwendet. Bei der Überprüfung auf Ungleichheit wird hier zweimal das *ID Protocol* von Schnorr [160] als Zero-Knowledge-Beweis ausgeführt und die *Multiplication Technique* von Abe et al. [2] eingesetzt. Allerdings werden diese mit dem Ziel verwendet, Ungleichheit von in zwei Auditwerten hinterlegten Monitoringdaten zu zeigen.

In der Literatur existieren hierzu verwandte Beweisverfahren, zum Beispiel für relationale Beziehungen zweier durch ein Hinterlegungsverfahren hinterlegter Werte. So verfolgen Naor und Ziv [141] in Rückgriff auf Camenisch und Shoup [30] ein ganz ähnliches Ziel. Allerdings wird hier ein Zero-Knowledge-Beweis konstruiert, der nachweist, dass ein bestimmter Wert nicht durch ein Hinterlegungsverfahren hinterlegt wurde. Beispielsweise dass in der Hinterlegung von x in $g^x \cdot h^y$ nicht der Wert z hinterlegt wurde. Also ein Nachweis, dass $x \neq z$ gilt ohne dass dabei x offengelegt wird. Dieses Vorgehen eignet sich aber nur zum Ausschluss, nämlich dass ein bestimmter Wert nicht hinterlegt wurde. Zur Überprüfung, ob zwei hinterlegte Werte ungleich sind, eignet sich dieses Verfahren nicht. Dazu müsste das Verfahren von Naor und Ziv [141] in der Lage sein, zu zeigen, dass der Wert 0 nicht in dem Quotienten der Auditwerte A_1 und A_2 hinterlegt ist. Eine Überprüfung auf 0 ist jedoch nicht möglich.

Anstatt eines Hinterlegungsverfahrens könnte auch ein Hashverfahren verwendet werden. Allerdings lassen sich mit Hilfe eines Hashverfahrens nicht ohne weiteres die an *CoCoPAS* gestellten Anforderungen *Geheimhaltung* und *Eindeutigkeit* umsetzen. Das in *CoCoPAS* zum Einsatz kommende Hinterlegungsverfahren von ElGamal [51] bietet hingegen perfekte statistische Eindeutigkeit ohne Beschränkung der Rechenleistung und Geheimhaltung, sofern der Kunde in seiner Rechenleistung beschränkt wird.

In zukünftigen Arbeiten ließe sich die eingesetzte vertrauenswürdige Partei auch auf andere Weise realisieren, beispielsweise durch Einsatz der Blockchain-Technologie. Der Einsatz einer *Blockchain* würde den Einsatz einer zentralen vertrauenswürdigen dritten Partei zur Absicherung der zeitlichen Zusammenhänge bei *CoCoPAS* überflüssig machen. Zusätzliche könnte mit Hilfe von *Smart Contracts* das Aufdecken von Auditwerten vor Gericht gelöst werden. Allerdings ist in beiden Fällen zusätzliche Forschungsarbeit notwendig: Die derzeitigen Blockchain-Technologien sind zum Teil recht langsam (erlauben nur wenige Transaktionen pro Sekunde). Außerdem lassen sich einmal in der

Blockchain hinterlegte Werte nicht einfach löschen, was aber im Fall der Auflösung der Vertragsverhältnisses zwischen Kunde und Cloud-Anbieter erforderlich sein kann.

5.10 Zusammenfassung

In diesem Kapitel wird *CoCoPAS* zum Einsatz an der Kundenschnittstelle von *CloudInspector* vorgestellt. Das Verfahren basiert auf der *Masking*-Eigenschaft von Pedersen, Zero-Knowledge-Beweisen und der *Multiplication Technique* von Abe et al. [2].

Mit *CoCoPAS* wird ein Baustein zur Überprüfung zur Laufzeit vorgestellt, der in der Lage ist Monitoringdaten des Cloud-Anbieters zu schützen. Sind Monitoringdaten während einer Überprüfung durch *CoCoPAS* geschützt, so lassen sich diese, beispielsweise zur Fehler-Ursachen-Analyse, im Nachhinein nachvollziehen. Hierzu stellt eine vertrauenswürdige dritte Partei die Integrität der Daten und den zeitlichen Zusammenhang sicher.

Es wird gezeigt, dass eine Überprüfung zur Laufzeit mit nachträglicher Nachvollziehbarkeit durch die verwendeten kryptographischen Bausteine realisierbar ist. Weiterhin zeigt *CoCoPAS*, dass die zu Beginn des Kapitels formulierten Anforderungen (Geheimhaltung, Eindeutigkeit und Zurechenbarkeit) eingehalten werden können. Hierzu greift *CoCoPAS* im Wesentlichen auf die Eigenschaft der verwendeten kryptographischen Bausteine zurück.

Zur Evaluierung von *CoCoPAS* wird sowohl die entstehende Datenmenge als auch die Ausführungszeit untersucht. Bei der Untersuchung ergab sich, dass *CoCoPAS* in gängigen Cloud-Umgebungen bei Nutzung von “near-term” Sicherheit eingesetzt werden kann.

Kapitel 6

Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen

Im Rahmen dieser Dissertation soll Cloud-Monitoring möglichst unabhängig vom CMS sein. Dieses Kapitel widmet sich dabei der durch Monitoring-Mechanismen prinzipiell erreichbaren Unabhängigkeit. Hierfür muss zunächst geklärt werden, was unter dem Begriff *Monitoring-Unabhängigkeit* verstanden werden soll. Danach wird ein zweistufiger Prozess zur Entwicklung einer Taxonomie für Monitoring-Unabhängigkeit angewandt. Im ersten Schritt werden mögliche Datenquellen bezüglich ihrer Unabhängigkeit vom CMS theoretisch betrachtet. Anschließend findet im zweiten Schritt ein Abgleich zwischen Theorie und Praxis statt und es werden Implikationen von realen Cloud-Umgebungen aufgezeigt. Als Ergebnis des Prozesses wird die entwickelte Taxonomie präsentiert. Außerdem widmet sich dieses Kapitel der Analyse des Einfluss von Elastizität auf Monitoring-Unabhängigkeit und dem beim CMS-unabhängigen Monitoring entstehenden Overhead.

Ergebnisse dieses Kapitels basieren unter anderem auf der folgenden Publikation:

- M. Flittner, S. Balaban und R. Bless. „CloudInspector: A Transparency-as-a-Service Solution for Legal Issues in Cloud Computing“. In: *IEEE International Conference on Cloud Engineering Workshop* (Apr. 2016) [TT-5]

Diese Kapitel widmet sich dabei der folgenden Forschungsfrage:

> *Unter welchen Rahmenbedingungen ist Cloud-Monitoring unabhängig vom Cloud-Management-System? Welche Auswirkungen hat die Forderung nach Unabhängigkeit auf die Konsistenz von Monitoringergebnissen?*

6.1 Monitoring-Unabhängigkeit

Bevor in den nächsten Abschnitten die Theorie hinter Unabhängigkeit von Monitoring-Mechanismen ausgeführt und Implikationen in realen Cloud-Umgebungen aufgezeigt werden, wird in diesem Abschnitt zunächst der Begriff *Monitoring-Unabhängigkeit* erarbeitet. Hierzu wird der Monitoringprozess aus Abschnitt 3.3.3 wieder aufgegriffen und verfeinert. Der Monitoringprozess von Monitoring-Mechanismen ist dabei in die folgenden drei Teile gegliedert: *Auslöser*, *Datenerhebung* und *Verarbeitung*. Zur Einschätzung der erreichbaren *Monitoring-Unabhängigkeit* ist vor allem interessant inwiefern das CMS während eines Monitoringprozesses involviert ist. In Abbildung 6.1 sind die einzelnen Prozessschritte und die mögliche Beteiligung des CMS dargestellt.

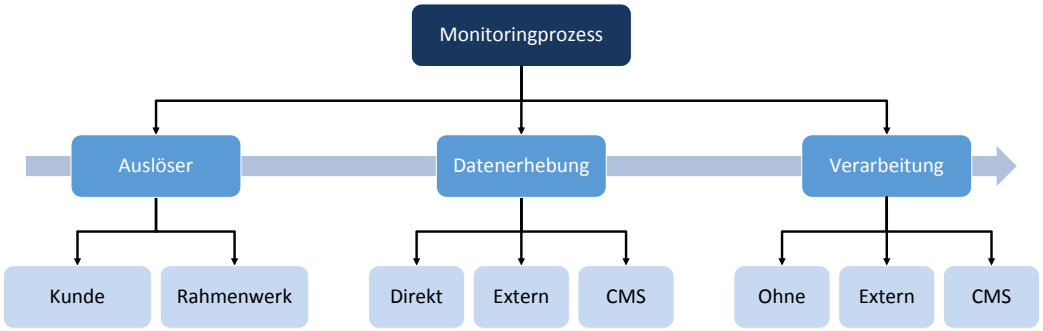

Abbildung 6.1: *Übersicht über die Prozessschritte eines Monitoringprozesses und der möglichen Beteiligung des Cloud-Management-Systems (CMS). Ohne steht hier für keine weiteren benötigten Informationen.*

In dem ersten Prozessschritt *Auslöser* ist keine Beteiligung des CMS vorgesehen. Es wird höchstens unterschieden, ob der Monitoring-Mechanismus vom *Kunden* angestoßen wurde – beispielsweise zur Überprüfung von vertraglichen Vereinbarungen zur Laufzeit – oder ob das *Rahmenwerk* selbst der Auslöser ist – beispielsweise bei kontinuierlicher Erfassung von Monitoringdaten zur späteren Fehler-Ursachen-Analyse. Im Ergebnis lässt sich festhalten, dass der Prozessschritt *Auslöser* keinen Einfluss auf Monitoring-Unabhängigkeit hat.

Der zweite Prozessschritt *Datenerhebung* hingegen hat Einfluss auf die erreichbare Monitoring-Unabhängigkeit. Die Monitoringdaten können vom Monitoring-Mechanismus *Direkt* auf dem physischen Cloud-Knoten erhoben werden. Dies erfolgt in der Regel unabhängig vom CMS. Je nach Monitoring-Mechanismus können Monitoringdaten auch *Extern* von anderen Komponenten als dem physischen Cloud-Knoten erhoben werden. In der Regel handelt es sich dabei nicht um das CMS sondern um weitere in der Cloud-Umgebung bereitgestellte Dienste, wie beispielsweise ein Blockspeicherdienst. Die dort

erhobenen Monitoringdaten sind typischerweise unabhängig vom CMS. Allerdings greift ein Monitoring-Mechanismus unter Umständen während der Datenerhebung auch auf das *CMS* zu. Die dort erhoben Monitoringdaten können selbstverständlich nicht die Eigenschaft Monitoring-Unabhängigkeit erfüllen.

In dem letzten Prozessschritt erfolgt die *Verarbeitung* der erhobenen Monitoringdaten. Je nach Monitoringprozess werden dazu keine weiteren Informationen benötigt – in der Abbildung mit *Ohne* gekennzeichnet. Trivialerweise beeinflusst dies die Monitoring-Unabhängigkeit nicht. Allerdings kann es in diesem Prozessschritt auch erforderlich sein, weitere Monitoringdaten von *Externen* Komponenten anzufordern. Beispielsweise von einem vertrauenswürdigem Zeitstempeldienst. Unter Umständen tangiert dies die Monitoring-Unabhängigkeit. Wenn hingegen für die Verarbeitung der Monitoringdaten auf das CMS zugegriffen wird um weitere Informationen, wie die Zuordnung zwischen virtuellen Ressourcen und Kunde, abzurufen, so ist die Monitoring-Unabhängigkeit direkt davon betroffen.

Im Ergebnis hängt die erreichbare *Monitoring-Unabhängigkeit* also stark von der konkreten Umsetzung eines Monitoringprozesses ab. Aus diesem Grund findet nachstehend eine theoretische Analyse des Zusammenhangs zwischen Monitoring-Unabhängigkeit und Datenquellen statt. Anschließend werden die ermittelten Ergebnisse in den Kontext von realen Cloud-Umgebungen gestellt und eine Taxonomie für Monitoring-Unabhängigkeit präsentiert.

6.2 Forschungsansatz

Um die Rahmenbedingungen für Monitoring-Unabhängigkeit fassen zu können, wird der in Abbildung 6.2 dargestellte Forschungsansatz verfolgt.

Zunächst wird in Abschnitt 6.3 systematisch analysiert, welche Datenquellen in einer Cloud-Umgebung für die Erhebung von Monitoringdaten prinzipiell zur Verfügung stehen. Hierfür wird der Aufbau von Cloud-Umgebungen detailliert studiert. Als Datenquellen werden dabei *Compute*-, *Network*-, *Storage*-Ressourcen und das Betriebssystem beziehungsweise der Kontrollkanal des CMS betrachtet. Für die in Abschnitt 6.3 identifizierten Datenquellen werden dann exemplarischen Implikationen in realen Cloud-Umgebungen in Abschnitt 6.4 untersucht. Im Rahmen dieser Dissertation werden hierfür die gängigen Cloud-Umgebung *OpenStack* und *VMware* betrachtet.

Aus den so ermittelten Erkenntnissen wird dann induktiv eine Taxonomie für Monitoring-Unabhängigkeit in Abschnitt 6.5 etabliert. Als Klassen für Monitoring-Unabhängigkeit werden dabei *Unabhängig*, *Teilunabhängig*, *Teilabhängig* und *Abhängig* identifiziert.

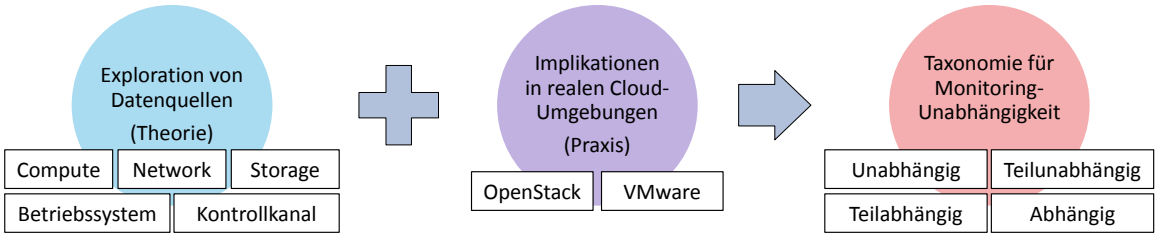

Abbildung 6.2: *Übersicht des gewählten Forschungsansatzes zur Bestimmung einer Taxonomie für Monitoring-Unabhängigkeit.*

Die so ermittelten Ergebnisse sollen als Grundlage zur Klassifizierung von gegenwärtigen und zukünftigen Anwendungsfällen im Bereich von Cloud-Monitoring dienen.

6.3 Exploration von Datenquellen

In diesem Abschnitt wird geklärt, aus welchen Datenquellen Monitoring-Mechanismen prinzipiell Monitoringdaten erheben können. Da im Rahmen dieser Dissertation Unabhängigkeit vom CMS eine entscheidende Rolle spielt, soll generell sichergestellt werden, dass es sich bei den Datenquellen nicht um das CMS selbst handelt. Hierzu muss geklärt werden, wo die Grenze zwischen CMS und externen Datenquellen gezogen wird. Nicht im Fokus dieser Dissertation steht dabei die Vertrauenswürdigkeit der Datenquellen an sich – Gegenstand der Untersuchung sind vielmehr Fehlkonfigurationen des CMS und nicht Angriffe durch den Cloud-Anbieter.

In Abbildung 6.3 ist, der schematische Aufbau einer Cloud-Umgebung dargestellt. Dort abgebildet ist das CMS welches über einen Kontrollkanal mit Hilfe von sogenannten Steuersignalen die durch physische Cloud-Knoten bereitgestellte Cloud-Umgebung steuert.

In einer IaaS Cloud-Umgebung wird dabei grundsätzlich zwischen drei Arten von Ressourcen unterschieden: *Compute*, *Network* und *Storage*. Auf physischen Cloud-Knoten übernimmt der sogenannte Virtual-Machine-Monitor die Verwaltung von *Compute-Ressourcen*. Für *Network-Ressourcen* steht der sogenannte *Network-Hypervisor* zur Verfügung. Die *Storage-Ressourcen* einer VM werden vom sogenannten *Storage-Hypervisor* verwaltet. Außerdem wird auf den physischen Cloud-Knoten ein Betriebssystem zur Verwaltung der verschiedenen Cloud-Komponenten eingesetzt. Für mehr Details zum Aufbau einer Cloud-Umgebung siehe [16, 143] oder Abschnitt 2.1.

Wenn im Rahmen dieser Dissertation von Unabhängigkeit die Rede ist, dann bezieht sich dies zunächst nur auf das CMS an sich, also die Stelle an der die Virtualisierung auf physischen Cloud-Knoten gesteuert wird. In der Abbildung 6.3 durch die graue

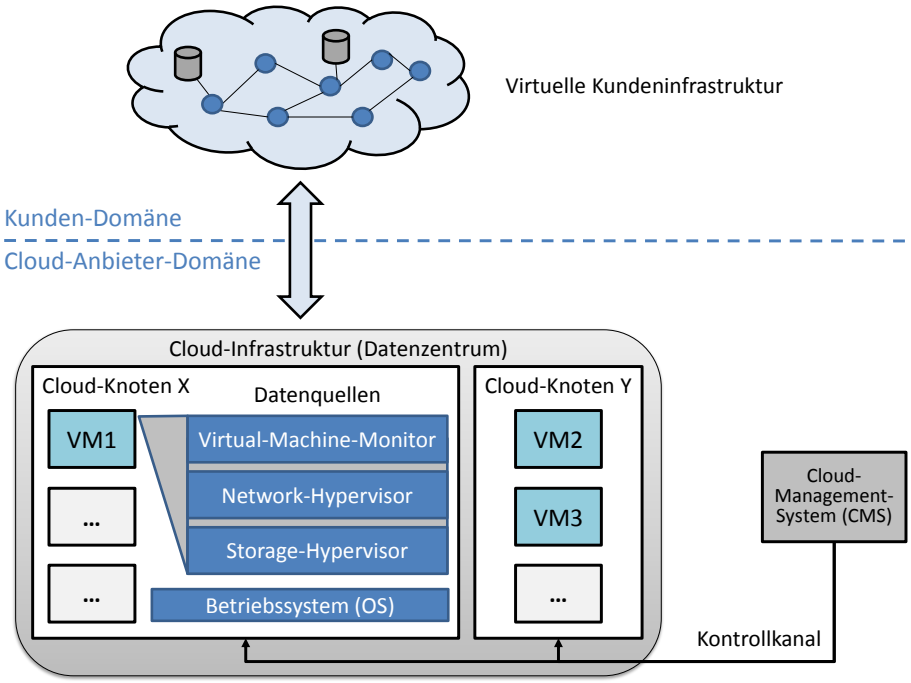

Abbildung 6.3: *High-level Übersicht über den Aufbau einer Cloud-Umgebung.*

Box mit dem Bezeichner Cloud-Management-System gekennzeichnet. Die Komponenten auf dem physischen Cloud-Knoten wie Virtual-Machine-Monitor (VMM), *Storage*- und *Network-Hypervisor* oder das eingesetzte Betriebssystem zählen explizit nicht mit zum CMS. Vergleiche auch mit den Ausführungen in Abschnitt 6.1 weiter oben.

Weiterhin werden im Rahmen dieser Dissertation nur Datenquellen betrachtet, bei denen *direkt* Informationen über die aktuelle Cloud-Konfiguration erhoben werden können. Die Erhebung von Monitoringdaten wird als *direkt* bezeichnet, wenn die ermittelten Informationen von Komponenten stammen, die ihrerseits für die Verwaltung von virtuellen Ressourcen verantwortlich sind. Der Gegensatz dazu wäre die Ermittlung von Monitoringdaten aus Komponenten, die nicht an der Virtualisierung einer Cloud-Umgebung beteiligt sind – etwa von Sensoren anderer Monitoringsysteme oder zusätzlicher Dienste innerhalb der Cloud-Umgebung. Im Rahmen dieser Dissertation liegt der Fokus allerdings auf Komponenten die direkt an der Virtualisierung beteiligt sind und die vom CMS gesteuert werden. Dies schließt insbesondere die fünf in Abbildung 6.3 skizzierten Komponenten ein (VMM, *Storage*- und *Network-Hypervisor*, Betriebssystem und Kontrollkanal des CMS). Diese werden im Folgenden genauer betrachtet. Inwiefern sich die ermittelten Erkenntnisse jeweils in realen Cloud-Umgebungen umsetzen lassen wird dann in Abschnitt 6.4 besprochen.

6.3.1 Virtual-Machine-Manager

Um VMs zur Verfügung stellen zu können, wird von der Cloud-Umgebung auf jedem physischen Cloud-Knoten ein VMM ausgeführt. Im Folgenden wird dabei davon ausgegangen, dass je VM nur ein VMM benötigt wird. Zwar sind Szenarien vorstellbar, in denen die *Compute-Ressource* einer einzelnen VM auf mehreren physischen Cloud-Knoten realisiert wird, jedoch fällt diese Art von Cloud-Computing nicht in den Anwendungsbereich von IaaS, wie es im Rahmen dieser Dissertation aufgefasst wird. Allerdings können selbstverständlich mehrere VMs je VMM realisiert werden.

Durch Abfrage des VMM auf einem physischen Cloud-Knoten lässt sich im Prozessschritt der *Datenerhebung* beispielsweise ermitteln, welche VMs derzeit ausgeführt werden. Darüber hinaus liefert der VMM eines physischen Cloud-Knotens typischerweise wichtige Informationen über die Bestandteile einer VM und ihren derzeitigen Ausführungszustand. Zum Beispiel welche virtuellen Netzschnittstellen von einer VM genutzt werden oder auf welcher Speicherlösung zur VM gehörige Daten abgelegt werden. Aber auch ob eine VM derzeit ausgeführt wird oder pausiert ist, lässt sich durch den VMM erfahren.

Entscheidend ist die Tatsache, dass sich die Monitoringdaten hier direkt (im Sinne der obigen Definition) vom VMM erheben lassen und keine Interaktion mit dem CMS erforderlich ist. In der Theorie sind die erhobenen Monitoringdaten also unabhängig vom CMS.

6.3.2 Storage-Hypervisor

Eine weitere wichtige Quelle für Monitoringdaten stellt der Storage-Hypervisor dar. Dieser wird zur Datenablage für VMs verwendet. Häufig werden dazu sogenannte externe Blockspeicherdienste an den Storage-Hypervisor angebunden. An dieser Stelle sei darauf hingewiesen, dass es sich bei dem durch einen Storage-Hypervisor bereitgestellten Speicher nicht um den Arbeitsspeicher einer VM handelt, sondern um das virtuelle Pendant zu Festspeicher.

Im Prozessschritt der *Datenerhebung* lässt sich durch Abfrage des *Storage-Hypervisors* beispielsweise ermitteln, auf welchem externen Blockspeicherdienst die Daten einer VM abgelegt werden. Dieser externe Blockspeicherdienst ist im Prozessschritt der *Datenerhebung* dann typischerweise Gegenstand weiterer Abfragen. Beispielsweise, wie viele physische Medien zur Speicherung der Daten eingesetzt werden und ob diese redundant ausgelegt sind. Die so ermittelten Monitoringdaten beinhalten dabei lediglich Metainformationen über die im Blockspeicherdienst abgelegten Daten. Keinesfalls wird hier eine Monitoringsystem angestrebt, bei dem alle abgelegten Daten abgerufen werden.

Im Gegensatz zum VMM kann in der Theorie beim Storage-Hypervisor nicht *Direkt* ermittelt werden, wie die aktuelle Konfiguration des Blockspeicherdienstes ist. Hierzu ist eine weitere *Externe* Abfrage von Monitoringdaten notwendig. Dies hat direkten Einfluss auf die erreichbare Monitoring-Unabhängigkeit.

6.3.3 Network-Hypervisor

Für die Bereitstellung von Netzressourcen wird in der Regel der Network-Hypervisor auf physischen Cloud-Knoten an einen virtuellen Switch angebunden. Zur Virtualisierung des Netzes werden weiterhin mehrere Abstraktionsebenen gebildet. Dabei werden Teile des unterliegenden Cloud-Netzes genutzt, um virtuelle Netze für Kunden bereitzustellen.

Durch Abfrage eines Network-Hypervisors im Prozessschritt *Datenerhebung* lässt sich beispielsweise ermitteln, ob die im virtuellen Switch konfigurierten Weiterleitungsregeln so eingerichtet sind, dass sie den Anforderungen des Kunden entsprechen.

Im Rahmen dieser Dissertation wird sich dabei auf die Erhebung von Monitoringdaten von Network-Hypervisoren auf physischen Cloud-Knoten beschränkt. Die Ermittlung eines vollständigen und aktuellen Abbildes der Konfiguration des Cloud-Netzes bleibt Gegenstand weiterführender Arbeiten.

In der Theorie können dabei sowohl Information vom eingesetzten Network-Hypervisor, als auch vom genutzten virtuellen Switch *Direkt* erhoben werden. Die erhoben Monitoringdaten sind also unabhängig vom CMS.

6.3.4 Kontrollkanal des Cloud-Management-Systems

Über den Kontrollkanal zwischen CMS und physischen Cloud-Knoten werden Steuersignale zur Verwaltung der Cloud-Infrastruktur übermittelt. Beispielsweise veranlasst das CMS hierüber das Starten und Stoppen von VMs oder *Live-Migrationen*.

Durch Abhören dieses Kontrollkanals lässt in dem Prozessschritt der *Datenerhebung* beispielsweise ermitteln, ob das CMS gerade eine Konfiguration der Cloud-Umgebung vornimmt. Außerdem können so die geplanten Aktionen des CMS aufgezeichnet werden. Dabei kann der Kontrollkanal in der Regel sowohl direkt am physischen Cloud-Knoten selbst, als auch auf der Komponente die den Kontrollkanal für die Cloud-Umgebung zur Verfügung stellt (zum Beispiel ein *Broker*) abgehört werden. Während in der ersten Variante nur Steuersignale abgehört werden können die an den physischen Cloud-Knoten selbst adressiert sind, kann im zweiten Fall alle Steuersignale der Cloud-Umgebung als Monitoringdaten erfasst werden.

Für die Betrachtung von Monitoring-Unabhängigkeit ergeben sich hier zwei Aspekte. Wird der Kontrollkanal *Direkt* auf dem physischen Cloud-Knoten abgehört, so ist Monitoring-Unabhängigkeit gegeben. Allerdings hat die Nutzung der *Externen* Komponente Einfluss auf die erreichbare Monitoring-Unabhängigkeit.

6.3.5 Betriebssystem eines physischen Cloud-Knotens

Eine weitere Datenquelle, die Informationen über die aktuelle Cloud-Konfiguration liefern kann, ist das Betriebssystem eines physischen Cloud-Knotens.

Typischerweise lässt sich vom Betriebssystem eines physischen Cloud-Knotens in dem Prozessschritt der *Datenerhebung* Information über die CPU-Auslastung, Nutzung des Hauptspeichers oder Umgebungstemperatur abfragen. Anhand dieser Monitoringdaten kann dann beispielsweise überprüft werden, ob wie vom Kunden vereinbart Reservekapazität auf dem physischen Cloud-Knoten zur Verfügung steht.

Da hier die ermittelten Daten *Direkt* erhoben werden, ist die Unabhängigkeit vom CMS trivialerweise gegeben. Daher wird dieses im nächsten Abschnitt nicht mehr weiter betrachtet.

6.4 Implikationen in realen Cloud-Umgebungen

Im vorangegangenen Abschnitt wurden verschiedene Datenquellen für Cloud-Monitoring untersucht, die sich in der Theorie für CMS-unabhängiges Monitoring eigenen. In diesem Kapitel wird nun gezeigt, wie sich die theoretischen Erkenntnisse auf reale Cloud-Umgebungen übertragen lassen. Hierfür werden die Cloud-Umgebungen *OpenStack* [292] und *VMware vSphere* [310] herangezogen.

Zur Ermittlung der nachstehend beschriebenen Aspekte wurden beide Cloud-Umgebungen in einer kleinen Teststellung aufgebaut und der Abruf von Monitoringdaten durch Monitoring-Agenten untersucht. Hierfür wurden exemplarisch der Monitoringprozess für einige Monitoring-Mechanismen implementiert.

6.4.1 Datenquelle Virtual-Machine-Monitor bei OpenStack

Bei der Cloud-Umgebung *OpenStack* [292] übernimmt auf den physischen Cloud-Knoten die CMS-Komponente *Nova* [287] die Verwaltung von VMs für das CMS. Zur Virtualisierung von VMs erlaubt Nova den Einsatz verschiedenster Technologien (siehe [286]).

Typischerweise wird der auf dem physischen Cloud-Knoten eingesetzte VMM mittels der *libvirt* Application Programming Interface (API) [279] an Nova angebunden. Beispielsweise werden so *QEMU* [298] oder *KVM* [277] als VMM bei OpenStack eingesetzt. Da *libvirt* eine einheitliche Programmierschnittstelle als Zugangspunkt zum durch Nova eingesetzten VMM bietet, bietet es sich an, diese Programmierschnittstelle als Datenquelle zu nutzen.

Per *libvirt* lässt sich für den Prozessschritt der Datenerhebung die Liste und Beschaffenheit der derzeit vom VMM auf dem physischen Cloud-Knoten ausgeführten VMs abrufen. Um die Zusammensetzung einer VM zu detektieren, ist dabei neben dem Zugriff auf *libvirt* kein Zugriff auf weitere Datenquellen erforderlich, insbesondere nicht auf das CMS.

Die Abfrage kann dabei sowohl zyklisch als auch ereignisbasiert erfolgen, wobei Untersuchungen gezeigt haben, dass die ereignisbasierte Abfrage von *libvirt* im Vergleich zur zyklischen Abfrage (je nach Abfrageintervall) leicht verzögerte Ergebnisse liefert. Vergleiche dazu Abschnitt 6.2 der Arbeit von Roth [TT-4]. Dies liegt beispielsweise beim Starten von VMs daran, dass das Ereignis erst nach vollständigem Start einer VM ausgelöst wird, wohingegen bei der zyklischen Abfrage *libvirt* die VM schon während des Startvorgangs auflistet, was eine frühere Erkennung ermöglicht.

Als Rückgabe liefert *libvirt* hier ein XML Document Object Model (DOM) an den Monitoring-Agenten zurück. Dieses DOM wird in Abbildung 6.4 ausschnittsweise dargestellt – beschränkt auf die Angaben, die im Folgenden besprochen werden.

Je ausgeführter VM enthält das DOM ein Element des Typs `domain`. Darin enthalten ist unter anderem ein Element des Typs `uuid`, das die hier wichtige Angabe der Universally Unique Identifier (UUID) einer VM angibt. In dem Element des Typs `nova:name` wird der vom Kunden vergebene Name der VM angegeben und somit eine Zuordnung zur UUID hergestellt. Außerdem wird jede erfasste VM mit Hilfe von `nova:user` direkt zu einem Kunden und dessen UUID zugeordnet. Darüber hinaus sind in dem Element des Typs `domain` noch weitere Angaben über die Zusammensetzung einer VM enthalten. Beispielsweise die verwendeten CPU-Kerne, Festplatten und Netzschnittstellen.

Außerdem eigenen sich die ermittelten Information bezüglich der Zusammensetzung einer VM um gezielt weitere Datenquellen abzufragen und weitere Information zu ermitteln. Beispielsweise den Storage-Hypervisor, um zu detektieren, wie die durch die VM verwendete Blockspeicherlösung beschaffen ist.

Neben dem Abruf der Monitoringdaten vom VMs, muss für *CloudInspector* eine Zuordnung von VM zu Kunde erfolgen. Dies ist notwendig, damit der beteiligte Monitoring-Agent anschließend der zum Kunden gehörenden Gruppe beitreten kann (oder prüft ob

```
<domain type='kvm' id='1'>
  <name>instance-00000001</name>
  <uuid>17e8040d-9ad1-4dc1-be52-f7fddd80f8cd</uuid>
  <metadata>
    <nova:instance xmlns:nova="http://openstack.org/xmlns/libvirt/nova/1.0">
      [...]
      <nova:name>MyVirtualMachine</nova:name>
      <nova:owner>
        <nova:user uuid="af8c27ea125348e6b166066f75185aee">admin</nova:user>
        <nova:project uuid="33f8c9a2ecf449278a88b8b927919862">alt_demo</nova:project>
      </nova:owner>
    </nova:instance>
  </metadata>
  [...]
  <memory unit='KiB'>524288</memory>
  <currentMemory unit='KiB'>524288</currentMemory>
  [...]
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='block' device='disk'>
      [...]
    </disk>
    [...]
    <interface type='bridge'>
      <mac address='fa:16:3e:54:2f:89'/>
      [...]
    </interface>
  </devices>
 [...]
</domain>
```

Abbildung 6.4: *Rückgabe von libvirt bei Abfrage des Virtual-Machine-Monitors. Vereinfachte Darstellung. Weggelassene Teile des XML Document Object Models wurden durch [...] ersetzt.*

er schon Mitglied ist) und somit Anfragen des Monitoring-Controllers zu diesem Kundenkontext empfängt. Dabei liefert *libvirt* bei *OpenStack* die Zugehörigkeit zu Kunden, wie oben beschrieben, bereits direkt mit – ein Zugriff auf das CMS ist hierfür nicht nötig.

Zwischenfazit

Für die Cloud-Umgebung *OpenStack* ist der VMM als Datenquelle für CMS-unabhängigem Monitoring geeignet. Das zentrale Problem der Zuordnung von virtuellen Ressourcen hat hier eine triviale Lösung und beeinflusst die Monitoring-Unabhängigkeit nicht. Im nächsten Abschnitt wird allerdings gezeigt, dass dies nicht für alle Cloud-Umgebungen der Fall ist.

6.4.2 Datenquelle Virtual-Machine-Monitor bei VMware vSphere

Für *VMware vSphere* [310] kommt auf den physischen Cloud-Knoten der VMM *VMware ESXi* [309] zum Einsatz. Als Datenquelle für den Zugriff auf den VMM steht dazu beispielsweise das für Python ausgelegte Werkzeug *pyVmomi* [297] zur Verfügung. Daher lässt sich, ähnlich wie bei *libvirt*, der VMM zyklisch mittels *Python* durch einen Monitoring-Mechanismus abfragen.

Bei *VMware ESXi* wird als Antwort bei einer Abfrage des VMMs ein *JSON* codiertes Objekt vom Typ `vim.vm.Summary` an den Monitoring-Agenten zurückgeliefert. Siehe vereinfachte Darstellung der Rückgabe von *pyVmomi* in Abbildung 6.5.

```
(vim.vm.Summary) {
[...]
   vm = 'vim.VirtualMachine:1',
   runtime = (vim.vm.RuntimeInfo) {
      [...]
   },
   guest = (vim.vm.Summary.GuestSummary) {
     [...]
   },
   config = (vim.vm.Summary.ConfigSummary) {
      [...]
      name = 'MyVirtualMachine',
      [...]
      uuid = '564d8470-8845-468f-6abb-fbad7ed322ec',
      instanceUuid = '52bd6e21-2ff9-fa22-b7c0-097225067dda',
      [...]
   },
   storage = (vim.vm.Summary.StorageSummary) {
      [...]
   },
   quickStats = (vim.vm.Summary.QuickStats) {
      [...]
   },
   overallStatus = 'green',
   customValue = (vim.CustomFieldsManager.Value) []
}
```

Abbildung 6.5: *Rückgabe von pyVmomi bei Abfrage durch einen Monitoring-Agent. Vereinfachte Darstellung. Weggelassene Teile des JSON-Objekts wurden durch [...] ersetzt.*

Dabei enthält das JSON-Objekt von *VMware ESXi* ganz ähnliche Angaben wie bei *libvirt*. Beispielsweise wird in dem Element `instanceUuid` der UUID jeder auf dem VMM ausgeführten VM angegeben [308]. Das Element `uuid` enthält einen für den physischen Cloud-Knoten eindeutigen Bezeichner.

Zwar enthält das JSON-Objekt Angaben zur Zusammensetzung der VM (beispielsweise welche Speicherressourcen für die VM verwendet werden), allerdings fehlen Metadaten, die die Zugehörigkeit einer VM zu einem Kunden belegen (siehe Element `nova:user` bei *libvirt*). Dies liegt vor allem daran, dass das CMS bei VMware – etwa das *VMware vCenter* [306] oder im professionellen Umfeld der *VMware vCloud Director* [307] – die Zugehörigkeit von VMs zu Kunden nur zentral verwaltet und diese Information daher auf den physischen Cloud-Knoten nicht vorliegt.

Da die Zuordnung von VMs zu Kunden bei *CloudInspector* aber von Monitoring-Agenten benötigt wird, um der entsprechenden Gruppe des Kunden beizutreten, muss die Zugehörigkeit zusätzlich ermittelt werden. Im Falle von *VMware* ist hierbei ein Zugriff auf das CMS unvermeidlich, da diese Informationen nur dort vorliegen.

Aus diesem Grund muss an dieser Stelle die Forderung nach Unabhängigkeit vom CMS etwas gelockert werden. So wird angenommen, dass bei VMware die Abbildung von UUIDs der VMs auf UUIDs von Kunden authentisch ist. Kommt beispielsweise *VMware vCloud Director* als CMS zum Einsatz, so ist der Kundenkontext einer VM an der REST API über das `/owner` Element abfragbar [305].

Zwischenfazit

Im Vergleich zu *OpenStack* lassen sich folgende Implikationen für *VMware* festhalten. Zwar werden im Prozessschritt der Datenerhebung die eigentlichen Monitoringdaten *Direkt* erhoben (zum Beispiel der Status einer VM), allerdings ist für die Zuordnung zu Kunden im Prozessschritt der *Verarbeitung* der Zugriff auf Informationen des CMS unausweichlich. Vollständige Monitoring-Unabhängigkeit lässt sich in diesem Fall nicht erreichen. In der Taxonomie in Abschnitt 6.5 findet hierfür eine feingranulare Unterscheidung statt.

6.4.3 Datenquelle Storage-Hypervisor bei OpenStack

Als nächstes wird gezeigt, wie der Storage-Hypervisor in realen Cloud-Umgebungen für CMS-unabhängiges Cloud-Monitoring verwendet werden kann. Zur Realisierung von *Storage* Ressourcen existiert eine Vielzahl von verschiedenen Speichertechnologien, die zur Virtualisierung verwendet werden. Bei OpenStack wird für Blockspeicher die Komponente *Cinder* verwendet [253]. Ähnlich wie bei der Komponente *Nova* werden hier verschiedenste Blockspeicherdienste unterstützt [293]. Beispielsweise Logical Volume Manager (LVM) [184] oder *CEPH* [188, 252].

Eine besondere Herausforderung bei der Erfassung von Blockspeicherdiensten im Prozessschritt *Datenerhebung* stellt deren dezentral und redundant aufgebaute Struktur dar. Bei LVM wird beispielsweise zusätzlich in die Abstraktionsebenen *Logical Volumes*, *Volume Groups*, *Physical Volumes* und *Block Devices* unterschieden. Abbildung 6.6 veranschaulicht den Aufbau von LVM. Dabei werden die *Physical Volumes* verschiedener *Block Devices* zu *Volume Groups* zusammengefasst. Ein *Physical Volume* ist jeweils nur Mitglied in einer *Volume Group*. Aus dem gesamt für eine *Volume Group* zur Verfügung stehenden Speicher (der der integrierten *Physical Volumes*) können dann *Logical Volumes* realisiert werden. Ein *Logical Volume* ist dabei fest einer *Volume Group* zugeordnet, nicht jedoch den *Physical Volumes*. So ist es bei *LVM* möglich, dass ein *Logical Volume* von einem *Physical Volume* zu einem anderen unterbrechungsfrei umzieht. Die *Logical Volumes* wiederum werden von OpenStack (*Cinder*) verwendet, um Blockspeicher für VMs zur Verfügung zu stellen. Darüber hinaus kann LVM auch als Cluster betrieben werden, dann können *Block Devices* verschiedener physischer Cloud-Knoten für die Zusammensetzung einer *Volume Group* genutzt werden. Im Folgenden wird davon ausgegangen, dass LVM nicht im Cluster-Modus betrieben wird und somit *Volume Groups* nur durch *Block Devices* auf einem physischen Cloud Knoten realisiert werden. Andere Szenarien werden in der Arbeit von Hoor [TT-6] untersucht.

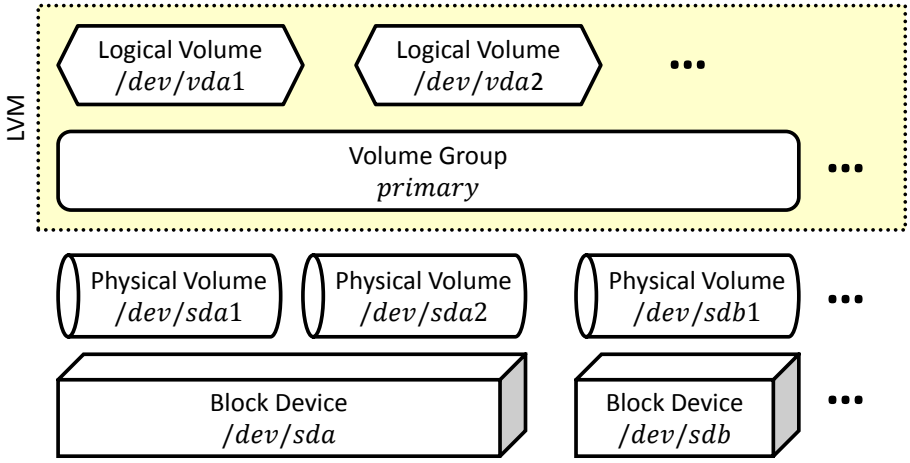

Abbildung 6.6: *Schematische Darstellung des Aufbaus des Logical Volume Managers. Visualisiert den Zusammenhang zwischen Block Devices, Physical Volumes, Volume Groups und Logical Volumes.*

Die Vielzahl an Konfigurationsmöglichkeiten für LVM muss bei der Abfrage von Monitoringdaten berücksichtigt werden. Das heißt, die Datenerhebung muss je nach vorliegender Konfiguration entsprechend angepasst werden. Eine einheitliche API wie *libvirt* im Fall von VMM gibt es hier nicht.

Die Zuordnung von *Logical Volumes* zu VM und somit auch zu einem Kunden erfolgt über die Abfrage des VMMs. Kommt bei OpenStack LVM als Blockspeicherdienst zum Einsatz, so gibt *libvirt* mittels des Elements `disk` für jede VM an, welche *Logical Volumes* für die VM verwendet werden. Abbildung 6.7 zeigt dabei das Ergebnis einer Anfrage an *libvirt*. Zugewiesene *Logical Volumes* werden dabei eindeutig über das Element `serial` innerhalb des Elements `disk` einer VM zugewiesen [278]. Ein Zugriff auf das CMS ist hierzu nicht notwendig.

In diesem Beispiel wird der VM eine virtuelle Festplatte `vda` zur Verfügung gestellt, die durch die *Logical Volume* mit dem UUID `7dae9931-103b-48b2-a2b5-be8032240e82` realisiert wird und auf dem physischen Cloud-Knoten unter `/dev/sdb` erreichbar ist.

```
[...]
  <disk type='block' device='disk'>
    [...]
    <source dev='/dev/sdb'/>
    [...]
    <target dev='vda' bus='virtio'/>
    <serial>7dae9931-103b-48b2-a2b5-be8032240e82</serial>
    [...]
  </disk>
[...]
```

Abbildung 6.7: *Angabe der einer VM zugewiesenen Volume Group von LVM. Auszug aus einer Anfrage an libvirt. Weggelassene Teile wurden durch [...] ersetzt.*

Zur Erfassung der eigentlichen LVM-Konfiguration des Blockspeicherdienstes existieren mehrere Datenquellen. Beispielsweise direkt über das Dateisystem (Betriebssystem) oder bei externer Realisierung über das Verwaltungsprogramm von LVM.

Ziel eines Monitoring-Mechanismus ist die Nachbildung der lokalen Cloud-Konfiguration bezüglich LVM. Im Prozessschritt der *Datenerhebung* wird dazu die LVM-Konfiguration ausgelesen und ein Baum erzeugt, in dem der LVM-Aufbau nachgestellt wird – also die Beziehung zwischen *Logical Volumes*, *Volume Groups*, *Physical Volumes* und *Block Devices*.

So kann beispielsweise mittels des Befehls `vgdisplay -v` ermittelt werden, welche *Logical Volumes* angelegt sind und zu welcher *Volume Group* diese gehören. In Abbildung 6.8 wird ein Auszug der Rückgabe der Datenquelle dargestellt. Dabei kann ein Monitoring-Mechanismus die zur VM gehörende *Logical Volume* anhand des in dem Element `serial` verwendeten UUID wiedererkennen, da die selbe UUID im Element `LV Name` verwendet wird. Auch hierzu ist kein Zugriff auf das CMS notwendig.

Die so ermittelte *Logical Volume* gehört zur übergeordneten *Volume Group* – in diesem Beispiel zu `stack-volumes-lvmdriver-1`. Das zugehörige *Physical Volume* findet sich

in dem zur *Volume Group* gehörenden Teil – in diesem Beispiel wird die *Volume Group* durch das *Physical Volume* `/dev/loop1` versorgt.

```
--- Volume group ---
  VG Name  stack-volumes-lvmdriver-1
  [...]
  VG UUID  YtcKK2-RbJd-PFeH-OONh-d4u1-t2Oi-GjcOih

[...]

  --- Logical volume ---
  LV Path  /dev/stack-volumes-lvmdriver-1/volume-7dae9931-103b-48b2-a2b5-be8032240e82
  LV Name  volume-7dae9931-103b-48b2-a2b5-be8032240e82
  VG Name  stack-volumes-lvmdriver-1
  LV UUID  wbLmP0-DXt8-QyXf-qLiV-4NKW-fLnY-j3FeqW
  [...]

  --- Physical volumes ---
  PV Name  /dev/loop1
  PV UUID  OyWJjt-Lmvd-72Kz-Vxsj-wg2k-YKUV-PUOj3n
  [...]

[...]
```

Abbildung 6.8: *Ausgabe des Befehls `vgdisplay -v` zur Erfassung der LVM-Konfiguration eines physischen Cloud-Knotens durch einen Monitoring-Agenten. Weggelassene Teile wurden durch [...] ersetzt.*

Die so ermittelten Werte können dann vom Monitor-Mechanismus beispielsweise in eine Baumstruktur überführt werden. Siehe Abbildung 6.9. Mit Hilfe dieser Baumstruktur lässt sich dann beispielsweise überprüfen, ob der Blockspeicherdienst einer bestimmten VM wie spezifiziert durch mehrere *Physical Volumes* beziehungsweise redundante *Block Devices* realisiert wird. Für weitere Details hierzu und Designalternativen siehe Arbeit von Hoor [TT-6]. Insbesondere für Angaben dazu, wie verteilte Blockspeicherdienste durch verschiedene Monitoring-Mechanismen erfasst und überprüft werden können. Außerdem finden sich in der Abschlussarbeit ausführliche Evaluationsergebnisse hierzu.

Zwischenfazit

Zwar ist die Konfiguration von Blockspeicherdiensten sehr Cloud-Anbieter spezifisch, allerdings lassen sich Monitoringdaten über die Beschaffenheit in der Regel *Direkt* erheben. Hierfür werden neben Monitoringdaten des Storage-Hypervisors Monitoringdaten des VMM benötigt, um zu ermitteln welcher Blockspeicherdienst von welcher VM verwendet wird. Ein Zugriff auf das CMS ist allerdings nicht notwendig. Wird der Blockspeicherdienst jedoch nicht lokal auf dem physischen Cloud-Knoten realisiert, so ist die *Externe* Erfassung

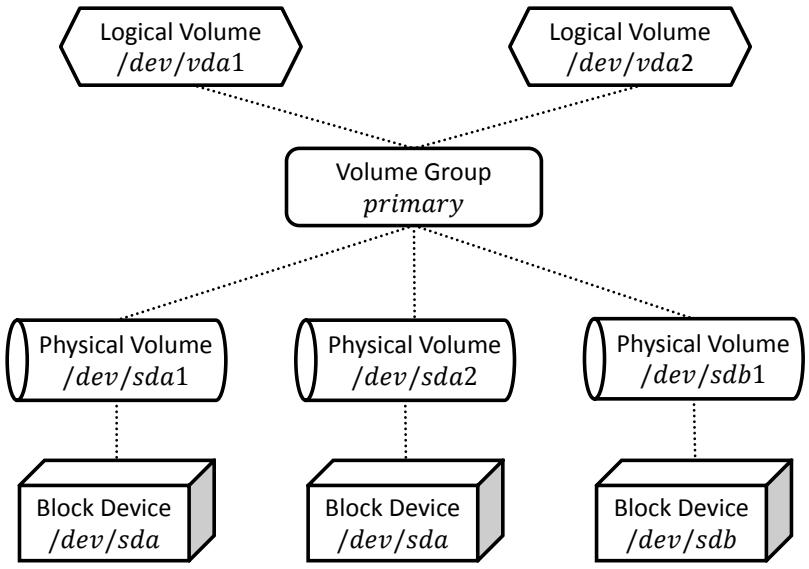

Abbildung 6.9: *Schematische Darstellung des durch den Mechanismus ermittelten Speicherbaums.*

von Monitoringdaten im Prozessschritt *Datenerhebung* erforderlich. Dies hat Auswirkung auf die erreichbare Monitoring-Unabhängigkeit.

6.4.4 Datenquelle Network-Hypervisor bei OpenStack

Nachstehend wird aufgezeigt, ob in realen Cloud-Umgebungen der Network-Hypervisor CMS-unabhängige Monitoringdaten liefern kann.

Die auf einem physischen Cloud-Knoten bereitgestellten *Network* Ressourcen werden dabei durch virtuelle Switches zur Verfügung gestellt, beispielsweise durch den *Open vSwitch* [151, 295]. Dieser virtuelle Switch enthält neben anderen Informationen Flow-Regeln für die Weiterleitung von Paketen in virtuellen Netzen. Die im Switch enthaltenen Informationen lassen sich als Datenquelle im Prozesschritt *Datenerhebung* nutzen.

Die Daten eines virtuellen Switches lassen sich dabei wie folgt erfassen: Zum einen lassen sich alle Einträge in den Flow-Tabellen mittels des Kommandozeilenbefehls `ovs-ofctl dump-tables switch` auslesen. Zum anderen bietet auch der *Open vSwitch* eine Programmierschnittstelle für Python an [296]. In beiden Fällen kann ein Monitoring-Mechanismus allerdings nur zyklisch auf den *Open vSwitch* zugreifen, eine ereignisbasierte Abfrage der Flow-Tabellen wird nach bestem Wissen derzeit von *Open vSwitch* nicht unterstützt. In beiden Fällen können allerdings die Informationen des virtuellen Switches ohne Zutun des CMS abgefragt werden.

Wird zusätzlich Service-Function-Chaining (SFC) [236] eingesetzt, beispielsweise bei der OpenStack-Erweiterung *OPNFV* [154, 290], dann enthält der *Open vSwitch* mehrere Flow-Tabellen und zusätzliche Einträge für die SFCs. Auch diese zusätzlichen Regeln lassen sich durch einen geeigneten Monitoring-Mechanismus erfassen. Allerdings muss dazu die von *OPNFV* angewandte Logik zur Implementierung von SFC berücksichtigt werden.

In Abbildung 6.10 wird schematisch dargestellt, wie *OPNFV* SFCs umsetzt und welche Informationen ein Monitoring-Agent mit Hilfe von Monitoring-Mechanismen aus einem virtuellen Switch als Monitoringdaten gewinnen kann. Die vom Kunden typischerweise vorgenommene Spezifikation einer SFC besteht aus einem sogenannten Match-Kriterium und einem Pfad von *Service Functions*. Als Match-Kriterium gibt der Kunde in Form von Filterregeln vor, welche Pakete durch die SFC laufen sollen. Das CMS konfiguriert die SFC dann als Overlay in der Cloud-Infrastruktur.

Im mittleren Teil der Abbildung 6.10 ist ein Pfad einer solchen SFC durch das Netz der Cloud-Infrastruktur dargestellt. Die entsprechenden Flow-Regeln dafür sind in den entsprechenden virtuellen Switches auf den physischen Cloud-Knoten hinterlegt.

Im unteren Teil der Abbildung 6.10 wird dargestellt, wie SFC auf Ebene eines virtuellen Switches umgesetzt wird. Hierzu lassen sich die Komponenten *Classifier* (Klassifikator) und *Service-Function-Forwarder* (SFF) identifizieren. Dabei ist je Komponente eine separate Flow-Tabelle mit Einträgen vorgesehen. Zusätzlich ist eine Flow-Tabelle für die Weiterleitung des Overlays vorhanden.

Bei Abfrage im Prozessschritt *Datenerhebung* werden alle Flow-Regeln aus den Flow-Tabellen extrahiert. Für SFC werden dabei die Informationen der einzelnen Tabellen so in eine Graph-Struktur überführt, dass später überprüft werden kann, ob die virtuellen Switches entsprechend konfiguriert wurden. Allerdings ist es ohne Hilfe des CMS nicht möglich, festzustellen, welche SFC zu welchem Kunden und zu welcher VM gehört, weswegen hierfür zusätzliche Informationen aus dem CMS benötigt werden. Die eigentlichen Monitoringdaten (hier die Flow-Regeln) können allerdings unabhängig vom CMS direkt aus den virtuellen Switches erhoben werden. Mehr Details zum Monitoring von SFCs finden sich in [TT-2].

Zwischenfazit

Die in einem virtuellen Switch hinterlegten Monitoringdaten lassen sich in der Regel *Direkt* vom CMS erheben. Soll allerdings SFC überwacht werden, werden zur Unterstützung weitere Informationen aus dem *CMS* benötigt, weswegen nicht in jedem Fall Monitoring-Unabhängigkeit zu erreichen ist.

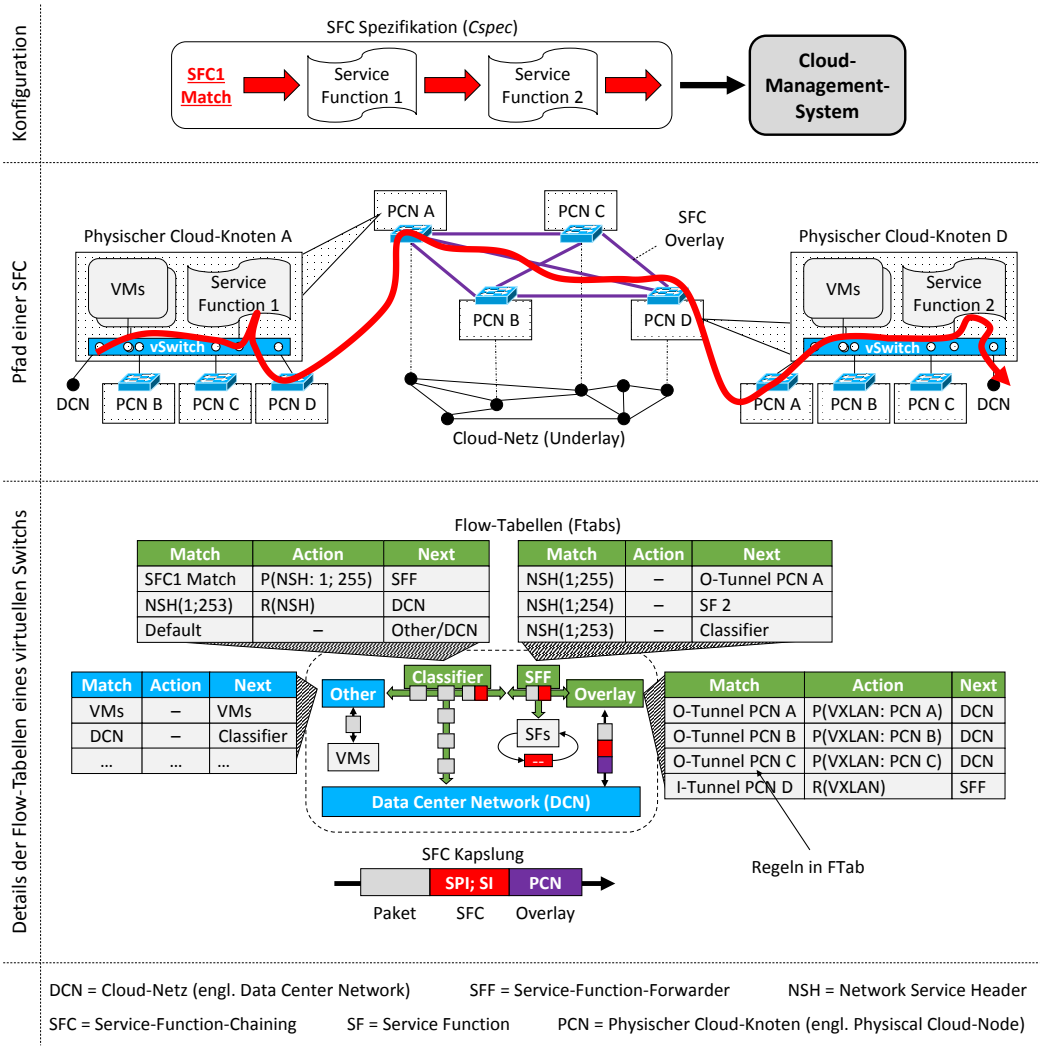

Abbildung 6.10: *Übersicht von Service-Function-Chaining in einer Cloud-Umgebung. Hier OPNFV. Darstellung der Flow-Tabellen des virtuellen Switches nach Flittner et al. [TT-2]. Im oberen Teil ist die Konfiguration einer Service-Function-Chain dargestellt. In der Mitte wird der virtuelle Pfad durch das Netz visualisiert. Im unteren Teil sind die Details eines virtuellen Switches zur Umsetzung von Service-Function-Chaining abgebildet.*

6.4.5 Datenquelle CMS-Kontrollkanal bei OpenStack

Als nächstes wird die Erhebung von Monitoringdaten aus dem CMS-Kontrollkanal genauer betrachtet und aufgezeigt, ob dies unabhängig vom CMS geschehen kann.

Bei *OpenStack* wird der Kontrollkanal zwischen CMS und physischen Cloud-Knoten durch eine auf dem Advanced Message Queuing Protocol (AMQP)-Protokoll [97] aufbau-

ende nachrichtenbasierte Middleware realisiert. Dafür wird bei *OpenStack* der Broker *RabbitMQ* eingesetzt [299, 291].

Soll der Kontrollkanal als Datenquelle für Monitoring-Mechanismen verwendet werden, so können die darüber versandten Steuersignale durch Registrierung als *Subscriber* bei der eingesetzten nachrichtenbasierten Middleware abgehört werden. Eine Auflistung aller Steuersignale des Kontrollkanals die für Cloud-Monitoring genutzt werden könnten findet sich in Beschreibung der Klasse `LegacyValidatingNotifier` in [294].

Zwischenfazit

Zwar würde man bei dem Kontrollkanal des CMS intuitiv eine hohe Abhängigkeit vom CMS erwarten, allerdings wird dieser in realen Cloud-Umgebungen häufig durch *Externe* Komponenten wie Broker realisiert. Die dort als Monitoringdaten erfassten Information sind dann unabhängig vom CMS.

Anders jedoch als bei den bisher vorgestellten Beispielen beziehen sich erhobene Monitoringdaten nicht direkt auf die Cloud-Konfiguration, sondern hier auf die Kommunikation innerhalb des CMS. In der Konsequenz kann zwar eine hohe Monitoring-Unabhängigkeit erreicht werden, allerdings ist die Aussagekraft von erhobenen Monitoringdaten auf CMS-interne Vorgänge beschränkt. Diese Informationen sind im Rahmen von Fehler-Ursachen-Analysen oder der bei Überprüfung von CMS-interne Vorgänge (beispielsweise wenn überprüft werden soll, mit welchem Zeitverzug Aktionen des CMS auf physischen Cloud-Knoten umgesetzt werden) relevant.

6.4.6 Vorläufiges Ergebnis

Als vorläufiges Ergebnis ist festzuhalten, dass verschiedene Datenquellen für Umsetzung von CMS-unabhängigen Monitoring-Mechanismen existieren. Monitoring-Unabhängigkeit lässt sich aber nicht unbedingt in allen Fällen erreichen.

Während beispielsweise bei *OpenStack* die Zuordnung von VMs zu Kunden *Direkt* ohne Zutun des CMS erfolgen kann, werden bei *VMware* Information aus dem *CMS* benötigt. Ähnlich gestaltet es sich für die Überprüfung des Netzzugangs einer VM. Wird kein SFC eingesetzt können Monitoringdaten *Direkt* erhoben werden. Kommt hingegen SFC zum Einsatz so ist ein Zugriff auf das CMS im Prozessschritt *Verarbeitung* unausweichlich. Hinzukommt, dass unter Umständen im Fall von Blockspeicherdiensten von *Externen* Komponenten im Porzessschritt der *Datenerhebung* Monitoringdaten erfasst werden müssen.

Insgesamt zeichnet sich recht deutlich ab, dass die für einen Monitoring-Mechanismus erreichbare Monitoring-Unabhängigkeit stark von dessen Einsatzzweck und Realisierung abhängt. Aus diesem Grund ist eine generelle Aussage für die Erreichbarkeit von Monitoring-Unabhängigkeit unmöglich – existieren doch eine schier unbegrenzte Anzahl von Fällen bei denen Cloud-Monitoring zum Einsatz kommen kann. Aus diesem Grund dient die in diesem Abschnitt vorgestellte Untersuchung auch nur als Indiz für die in der Praxis erreichbare Monitoring-Unabhängigkeit.

Allerdings bietet diese exemplarische Untersuchung von Implikationen zusammen mit der theoretischen Analyse eine gute Ausgangsbasis zur Entwicklung einer Taxonomie für Monitoring-Unabhängigkeit. Diese wird im nächsten Abschnitt vorgestellt.

6.5 Taxonomie für Monitoring-Unabhängigkeit

Aufbauend auf der in den vorherigen Abschnitten vorgestellten Analyse, wird in diesem Abschnitt eine Taxonomie für Monitoring-Unabhängigkeit entwickelt. Damit sich diese Taxonomie nicht ausschließlich auf bestimmte Anwendungsfälle oder ausgewählte Cloud-Umgebungen bezieht, wird als einziges Entscheidungskriterium der *Ort* der Monitoringdatenerhebung verwendet.

Folgende disjunkte Orte zur Erhebung von Monitoringdaten lassen sich ausmachen:

Physischer Cloud-Knoten: Hierunter werden alle Anwendungen subsumiert, von denen direkt auf einem physischen Cloud-Knoten Monitoringdaten abgefragt werden können. Beispielsweise das Betriebssystem des physischen Cloud-Knotens, der VMM und Storage- oder Network-Hypervisor.

Externe Komponenten: In diese Kategorie fällt die Abfrage von Monitoringdaten außerhalb eines physischen Cloud-Knotens. Beispielsweise Informationen von Blockspeicherdiensten. Allerdings fällt hierunter nicht der Abruf von Informationen aus dem CMS.

Cloud-Management-System: Der Abruf von Monitoringdaten direkt aus dem CMS fällt in diese Kategorie. Beispielsweise um zu ermitteln, welche Aktionen das CMS für eine bestimmte VM vorgesehen hat.

Das vorgestellte Entscheidungskriterium dient nun zur Klassifizierung der in einem Monitoringprozess erreichbaren Unabhängigkeit vom CMS. Nach folgenden aufeinander aufbauende Klassen wird dabei klassifiziert:

Unabhängig: In diese Klasse fallen alle Anwendungsfälle, bei denen Monitoring-Mechanismen lediglich Informationen von physischen Cloud-Knoten erheben.

Teilunabhängig: Wird durch Monitoring-Mechanismen auf *Externe* Komponenten zugegriffen, so fallen diese Anwendungsfälle in die Klasse *Teilunabhängig*.

Teilabhängig: In diese Klasse fallen Anwendungsfälle, bei denen durch Monitoring-Mechanismen im Prozessschritt *Verarbeitung* auf Informationen des CMS zurückgegriffen wird. In den Anwendungsfällen stehen dabei nicht die Informationen aus dem CMS im Vordergrund, sondern vielmehr nutzen die Monitoring-Mechanismen die Informationen aus dem CMS um die im Prozessschritt *Datenerhebung* gewonnen Monitoringdaten weiterzuverarbeiten. Beispielsweise wenn bei *VMware* auf das CMS zugegriffen wird, um zu ermitteln, zu welchem Kunden eine VM gehört.

Abhängig: Bezieht sich ein Monitoring-Mechanismus auf die Überprüfung von Informationen die ausschließlich im CMS enthalten sind, so dass von dort Monitoringdaten erheben werden, so wird dies als *Abhängig* bezeichnet.

In Abbildung 6.11 wird die entwickelte Taxonomie für Monitoring-Unabhängigkeit abgebildet, wobei in der Klasse *Unabhängig* die höchste Monitoring-Unabhängigkeit erreicht werden kann. Oberste Ebene der Taxonomie bilden dabei Monitoring-Mechanismen.

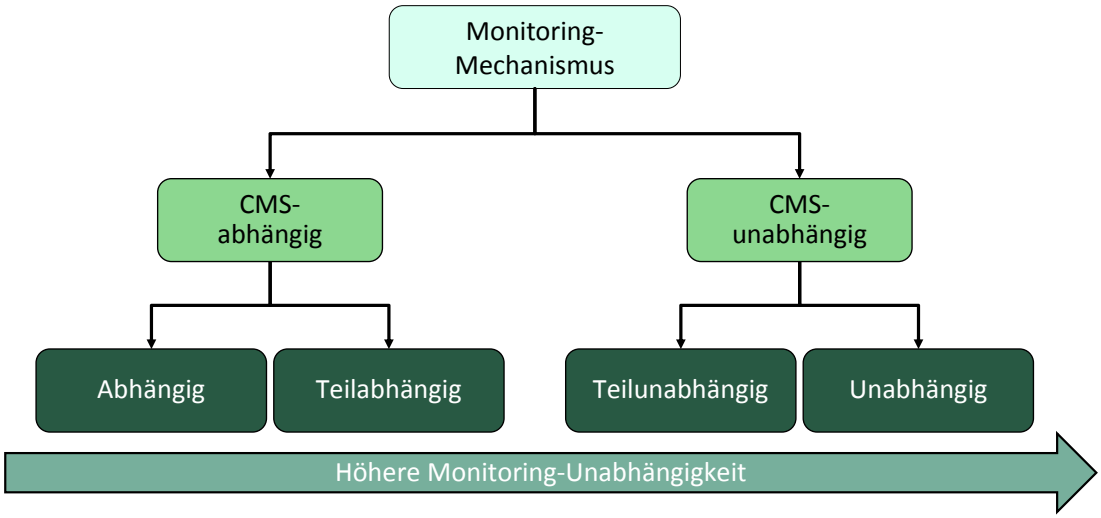

Abbildung 6.11: *Darstellung der Taxonomie für Monitoring-Unabhängigkeit.*

Diese werden in die Subklassen *CMS-abhängig* und *CMS-unabhängig* unterteilt, wobei sich diese in die oben beschriebenen Klassen aufteilen.

Ausgehend von den vorgestellten Klassen und dem Entscheidungskriterium ergibt sich der in Abbildung 6.12 dargestellte Entscheidungsbaum. Je Monitoringprozess wird dabei

die erreichbare Klasse von Monitoring-Unabhängigkeit ermittelt, indem die einzelnen Prozessschritte mit dem Entscheidungsbaum abgeglichen werden. Kommen in einem Prozesschritt mehrere Techniken zum Einsatz, so wird im Rahmen der Taxonomie der mit der geringsten Monitoring-Unabhängigkeit gewertet. Der Prozessschritt *Auslöser* spielt für Monitoring-Unabhängigkeit wie in Abschnitt 6.1 erläutert keine Rolle.

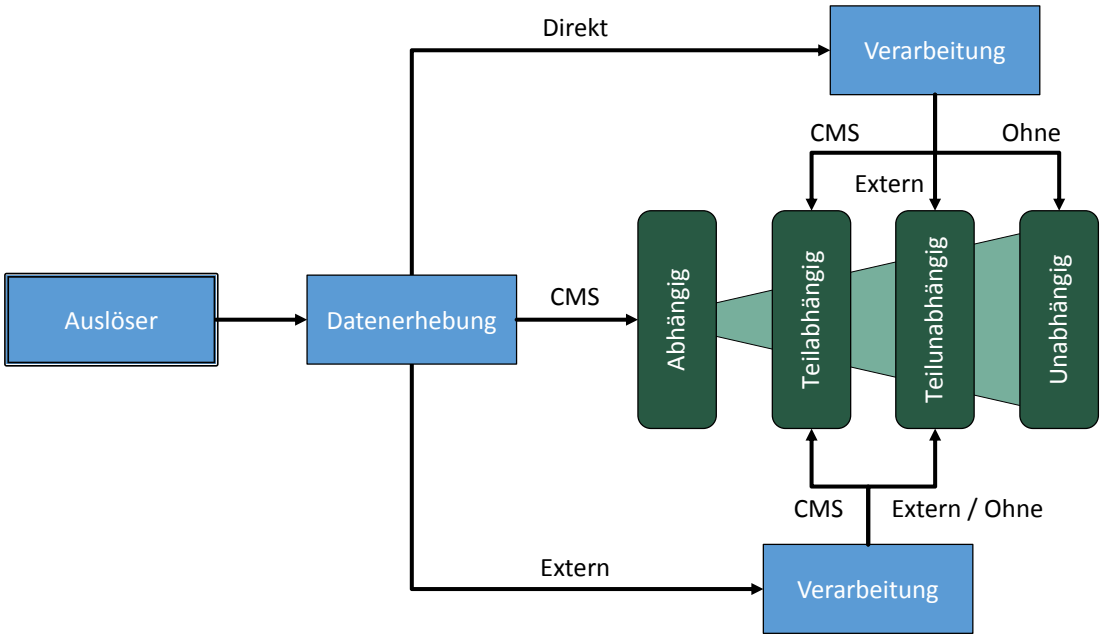

Abbildung 6.12: *Entscheidungsbaum zur Beurteilung von Monitoring-Unabhängigkeit.*

Für den Prozessschritt *Datenerhebung* wird nun unterschieden, ob *Direkt* auf dem physischen Cloud-Knoten befindliche Datenquellen zur Erhebung von Monitoringdaten verwandt werden, oder ob *Externe* Komponenten genutzt werden, beziehungsweise ob direkt auf das *CMS* zugegriffen wird. Wird die Datenerhebung *Direkt* durchgeführt, so wird für den Prozessschritt *Verarbeitung* des fraglichen Monitoring-Mechanismus unterschieden, ob weitere Informationen benötigt werden. Ist dies nicht der Fall (*Ohne*), so ist die höchste Monitoring-Unabhängigkeit erreicht (Klasse *Unabhängig*). Werden Informationen von *Externen* Komponenten benötigt so ist der Monitoring-Mechanismus in die Klasse *Teilunabhängig* einzustufen. Ist allerdings der Zugriff auf das *CMS* im Prozessschritt *Verarbeitung* erforderlich, so liegt die erreichte Monitoring-Unabhängigkeit in der Klasse *Teilabhängig*.

Werden allerdings im Prozessschritt *Datenerhebung* Monitoringdaten von *Externen* Komponenten erhoben, so wird für den Prozessschritt *Verarbeitung* unterschieden, ob weitere Informationen des CMS benötigt werden. Ist dies der Fall, so ist für die Monitoring-Unabhängigkeit die Klasse *Teilabhängig* erreicht. Ist dies nicht der Fall und es werden

weitere Informationen beispielsweise von *Externen* Komponenten benötigt, so wird die Klasse *Teilunabhängig* erreicht. Gleiches trifft zu, falls im Prozessschritt *Verarbeitung* keine weiteren Informationen benötigt werden.

Wird im Prozessschritt *Datenerhebung* direkt auf das CMS zugegriffen um Monitoringdaten zu erheben, so ist direkt die Klasse *Abhängig* erreicht ohne weitere Fallunterscheidung. Für diese Art von Monitoring-Mechanismen ist keine Monitoring-Unabhängigkeit gegeben.

Da die jeweils erreichbare Klasse für Monitoring-Unabhängigkeit von der konkreten Umsetzung der Monitoringprozesse der eingesetzten Monitoring-Mechanismen abhängt, lassen sich keine generellen Aussagen zu Monitoring-Unabhängigkeit ableiten. Beispielsweise lässt sich die Platzierung von VMs von Monitoring-Mechanismen in der Regel sowohl auf den physischen Cloud-Knoten direkt ermitteln (Klasse *Unabhängig*), als auch durch geeignete Monitoring-Mechanismen aus dem CMS auslesen (Klasse *Abhängig*).

Allerdings existieren auch Anwendungsfälle für Cloud-Monitoring bei denen sich nicht durch Austausch von Monitoring-Mechanismen eine höhere Monitoring-Unabhängigkeit erreichen lässt. Beispielsweise wenn überprüft werden soll, wie sich das CMS im Fehlerfall verhält oder ob das CMS auf dem aktuellen Stand ist. Hierzu sind unweigerlich Monitoringdaten aus dem CMS selbst erforderlich und es wird die Klasse *Abhängig* erreicht.

6.6 Analyse von Seiteneffekten

Nachdem nun geklärt wurde unter welchen Bedingungen Monitoring-Unabhängigkeit erreicht werden kann, werden im Folgenden ausgewählte Seiteneffekten betrachtet. Zum einen die Auswirkung von der bei Cloud-Computing typischen Elastizität auf Monitoring-Unabhängigkeit. Zum anderen den durch Monitoring-Unabhängigkeit entstehenden Overhead.

6.6.1 Auswirkung von Elastizität

In diesem Abschnitt werden die Auswirkungen der in Cloud-Umgebungen typischen Elastizität auf Monitoring-Unabhängigkeit untersucht. Da es sich bei einer Cloud-Umgebung dem Wesen nach um ein verteiltes System handelt, ist die konsistente Erfassung des verteilten Systemzustands besonders herausfordernd. Die Forderung nach Unabhängigkeit vom CMS stellt hier eine zusätzliche Hürde dar: würden benötigte Informationen

direkt vom CMS erhoben (Klasse *Abhängig*), stellt konsistente Erfassung des Systemzustands kein besonderes Problem dar, denn die Information liegen im CMS bereits in einem konsistenten Zustand vor. Da Monitoringdaten im Rahmen dieser Dissertation aber möglichst unabhängig erfasst werden sollen, kann in der Regel nicht sichergestellt werden, dass während eines Monitoringprozesses Änderungen an der Cloud-Infrastruktur vom CMS vorgenommenen werden, was zu Inkonsistenzen in den Monitoringdaten führt. Allerdings ist es für Cloud-Monitoring auch nicht immer erforderlich, den Systemzustand der Cloud-Umgebung konsistent zu erfassen, beispielsweise wenn lediglich die Betriebssystemversion der physischen Cloud-Knoten überprüft werden soll. Gleiches gilt, wenn ermittelt werden soll ob genügend Reservekapazität vorgehalten wird oder wenn die CPU-Last eines einzelnen physischen Cloud-Knotens überprüft werden soll.

Ein Beispiel bei dem Konsistenz der Erfassung eine Rolle spielt, ist die Platzierung von VMs auf physischen Cloud-Knoten. Nachstehend wird daher am Beispiel des Monitorings von VM-Platzierungen exemplarisch die Auswirkung von Elastizität auf Monitoring-Unabhängigkeit untersucht.

Zunächst wird dazu die Häufigkeit von *Live-Migrationen* analysiert. Aufbauen darauf soll die Frage geklärt werden, wie groß die Wahrscheinlichkeit ist, dass es während eines Monitoringprozesses nicht zu *Live-Migrationen* kommt.

6.6.1.1 Häufigkeit von Live-Migrationen

Bei Cloud-Computing sprich man von sogenannten *Live-Migrationen*, wenn während der Laufzeit einer VM der physische Cloud-Knoten gewechselt wird. Die während einer *Live-Migration* erhobenen Monitoringdaten werden in diesem Anwendungsfall als inkonsistent angesehen, da die VM sich in einem Transformationszustand befindet. Der genaue Ablauf einer solchen *Live-Migration* wird unter anderem von [196, 64] und Flittner und Bauer [TT-1] (siehe Abbildung 2) analysiert. Dabei dauert im Worst-Case-Szenario eine *Live-Migration* bis zu 50 Sekunden [91, 183] wobei die VM selbst im Mittel $1.15s$ unerreichbar ist [64]. Die Durchführung von *Live-Migrationen* wirkt sich also in der Regel auf die durch die VM bereitgestellten Anwendungen aus. Unter anderem aus diesem Grund setzt beispielsweise *Netflix* darauf, Cloud-Anwendungen so auszulegen, dass diese auch bei zufälligem Komplettausfall einzelner VMs zuverlässig ihre Dienstleistung erbringen können (vergleiche [175]). Da eine einzelne VM nicht erfolgskritisch für die Dienstleistung von *Netflix* ist, kann weitestgehend auf *Live-Migrationen* verzichtet werden.

6.6.1.1.1 Daten von IBM

Nach bestem Wissen ist *IBM* einer der wenigen Cloud-Anbieter, die Details zu VM-Migrationen in der Praxis veröffentlichen [19, 18, 17]. Die von Birke et al. [19] präsentierten Ergebnisse legen dabei nahe, dass im Schnitt 10 VMs auf einem physischen Cloud-Knoten ausgeführt werden. Weiterhin zeigen die in [19] analysierten Daten, dass 90% aller physischen Cloud-Knoten zumindest zwei VMs ausführen, wobei in 95% aller Fälle die VMs auf einem physischen Cloud-Knoten zum selben Kunden gehören. Die von Birke et al. [19] durchgeführte Untersuchung kommt schließlich zu dem Schluss, dass *Live-Migrationen* von VMs ziemlich unwahrscheinlich sind. 78% der analysierten VMs haben im Untersuchungszeitraum ihren physischen Cloud-Knoten nicht gewechselt. Nur einige Ausreißer VMs haben bis zu drei *Live-Migrationen* pro Tag durchgeführt. Außerdem zeigen Birke et al. [19], dass die Zeit zwischen zwei *Live-Migrationen* relativ lang ist – in nur drei Prozent aller Fälle liegt zwischen zwei *Live-Migrationen* eine Zeitspanne kleiner gleich zwei Tage. Außerdem zeigt die Analyse auf, dass VMs meisten nachts migriert werden (in 70% aller Fälle) und dass in den meisten Fällen VMs lediglich zwischen zwei physischen Cloud-Knoten hin und her migriert werden (in rund 68% aller Fälle).

Allerdings liefern die Analysen von Birke et al. [19] keine Informationen darüber, wie häufig die VMs eines einzelnen Kunden von *Live-Migrationen* betroffen sind. Diese Information wäre aber zur Beurteilung der Auswirkung von Monitoring-Unabhängigkeit auf Konsistenz entscheidend – bei *CloudInspector* werden nur die physischen Cloud-Knoten befragt, die virtuelle Ressourcen für den anfragenden Kunden ausführen.

6.6.1.1.2 Aktualisierungshäufigkeit von physischen Cloud-Knoten

Um die Häufigkeit von *Live-Migrationen* für VMs eines einzelnen Kunden besser abschätzen zu können, wurde zusätzlich im Rahmen dieser Dissertation eine Analyse der Aktualisierungshäufigkeit von physischen Cloud-Knoten durchgeführt. In diesem Fall müssen alle VMs auf einen anderen physischen Cloud-Knoten migriert werden. Zur Abschätzung wird im Folgenden die Veröffentlichung von *Xen* Versionen näher betrachtet [312]. Außerdem die Häufigkeit von sicherheitskritischen Aktualisierungen [311]. Dabei wird davon ausgegangen, dass bei einer neuen Version innerhalb eines kurzen Zeitfensters alle physischen Cloud-Knoten der Cloud-Umgebung aktualisiert werden müssen und es daher auch zu *Live-Migrationen* aller VMs kommt. Auf diese Art und Weise kann die untere Schranke für die Häufigkeit von *Live-Migrationen* eines einzelnen Kunden abgeschätzt werden.

In Abbildung 6.13 ist dargestellt, wie sich die Xen-Versionen innerhalb der letzten sechs Jahre in Abhängigkeit zur Unix-Zeit entwickelt haben – im Zeitraum zwischen dem

03.12.2012 und 27.02.2018. In der Abbildung ist außerdem die folgende Ergebnisfunk-

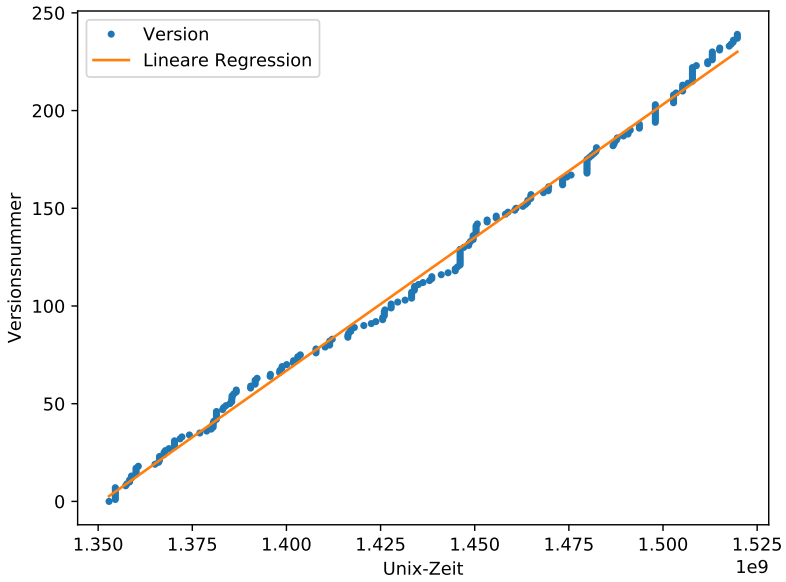

Abbildung 6.13: *Entwicklung der Versionen von Xen im Verhältnis zur Unix-Zeit.*

tion der mittels der *Vandermonde-Matrix* [15] durchgeführten linearen Regression über alle Aktualisierungen dargestellt.

$$f(x) = 1.363 \cdot 10^{-6} \cdot x - 1841$$

Hieraus lässt sich durch bilden der ersten Ableitung das Wachstum von $1.363 \cdot 10^{-6}$ je Sekunde berechnen. Im Schnitt ist also alle 8.5 Tage die Aktualisierung eines physischen Cloud-Knotens fällig (rund 0.12 im Tagesdurchschnitt) und somit auch die *Live-Migration* von VMs eines beliebigen Kunden.

6.6.1.1.3 Ergebnis

Im Ergebnis lässt sich festhalten, dass laut der Analyse von Birke et al. [19] *Live-Migrationen* von VMs eher selten sind, die Zeitspannen zwischen zwei *Live-Migrationen* sehr lang sind, in der Regel zwischen zwei physischen Cloud-Knoten hin und her migriert wird und dass für die deutliche Mehrheit aller physischen Cloud-Knoten die darauf ausgeführten VMs zum selben Kunden gehören. Im Worst-Case-Szenario dauert eine Live-Migration dabei bis zu 50*s*. Weiter ist im Schnitt als untere Abschätzung alle 8.5

Tage die *Live-Migration* von VMs eines Kunden aufgrund von Aktualisierungen von physischen Cloud-Knoten erforderlich. Im Vergleich mit der Analyse von Birke et al. [19] zeigt sich jedoch, dass in der Praxis in rund 50% der untersuchten Fälle mehr als 16 Tage zwischen der *Live-Migration* einer VM lagen. In nur 2% aller untersuchten Fälle liegt die Zeitspanne zwischen zwei *Live-Migration* bei weniger als zwei Tage. Die häufigste Anzahl an *Live-Migrationen* war hier mit drei pro Tag angegeben.

6.6.1.2 Modell für die Analyse

Die folgende analytische Betrachtung hängt von mehreren Parametern ab.

Ereignisrate `CR`: Gibt an, wie viele *Live-Migrationen* die VMs eines Kunden K im Tagesdurchschnitt (innerhalb von 1.440 Minuten) erfahren.

Anfragerate `AR`: Gibt an, wie viele Anfragen ein Kunde K im Tagesdurchschnitt (innerhalb von 1.440 Minuten) an den *CloudInspector* zur Überprüfung von vertraglichen Vereinbarungen stellt.

Überprüfungsdauer $A_{\texttt{DUR}}$: Gibt die maximal erwartete Dauer für eine Überprüfung durch *CloudInspector* in Minuten an.

Änderungsdauer $C_{\texttt{DUR}}$: Gibt die maximal erwartete Dauer einer *Live-Migration* in Minuten an – die maximale Zeit bis eine Änderung aktiv ist.

Die Belegung der aufgeführten Variablen hängt stark von der Elastizität der eingesetzten Cloud-Umgebung ab. Darüber hinaus spielt der konkrete Anwendungsfall eine große Rolle.

Weiter legt die Analyse von Birke et al. [19] nahe, dass die Dauer zwischen zwei *Live-Migration* exponentialverteilt ist. Im Folgenden wird daher ein Poisson-Prozess zur Analyse der Häufigkeit von *Live-Migration* in einem fest vorgegebenen Intervall verwendet. Poisson-Prozesse eigenen sich insbesondere zur Untersuchung von Ereignissen, die im zeitlichen Verlauf zufällig und unabhängig voneinander immer wieder eintreten [139] (insbesondere Seite 209ff.).

Ein Poisson-Prozess ist dabei wie folgt definiert:

$$P(X_t = k) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t}$$

Mit Hilfe des Poisson-Prozesses kann berechnet werden, wie hoch die Wahrscheinlichkeit ist, dass k Ereignisse in einem Beobachtungszeitraum t bei Intensität λ auftreten.

6.6.1.3 Untersuchungen

Mit Hilfe des Poisson-Prozesses wird nun untersucht, wie groß die Wahrscheinlichkeit ist, dass ein Monitoringprozess abgeschlossen werden kann, ohne dass es dabei zu *Live-Migrationen* kommt. Im Folgenden wird diese Wahrscheinlichkeit auch als Erfolgswahrscheinlichkeit bezeichnet. Zu überprüfen ist also, wie groß die Erfolgswahrscheinlichkeit ist, dass in einem vorgegebenen Beobachtungszeitraum t, null ($k = 0$) *Live-Migrationen* auftreten, bei zu parameterisierender Häufigkeit von *Live-Migrationen* pro Tag λ.

Der Beobachtungszeitraum t des Poisson-Prozesses ist dabei so groß zu wählen, dass selbst wenn kurz vor Beginn des Beobachtungszeitraums noch eine *Live-Migration* angestoßen wurde, diese beendet werden und anschließend ein Monitoringprozess ablaufen kann. Dabei wird zunächst davon ausgegangen, dass ein Monitoringprozess mehrere Millisekunden läuft – dies ist natürlich stark anwendungsfallspezifisch und die Annahme muss gegebenenfalls an einen konkreten Monitoringmechanismus angepasst werden. Der Parameter t ist daher auf den Wert $A_{\mathtt{DUR}} + C_{\mathtt{DUR}}$ zu setzen. Die Häufigkeit λ repräsentiert hier die Anzahl der *Live-Migrationen* pro Tag und ist daher gleichzusetzen mit der Ereignisrate $\mathtt{CR}$. Da hier die Wahrscheinlichkeit berechnet werden soll, dass in dem gewählten Beobachtungszeitraum t keine *Live-Migration* auftritt und somit der Monitoringprozess erfolgreich auf konsistenter Basis abgeschlossen werden kann, wird k auf 0 gesetzt.

Nun wird berechnet, wie hoch die Erfolgswahrscheinlichkeit ist, dass in dem Beobachtungszeitraum $A_{\mathtt{DUR}} + C_{\mathtt{DUR}}$ genau 0 *Live-Migrationen* bei Ereignisrate $\mathtt{CR}$ auftreten. Es ergibt sich die folgende Formel zur Berechnung:

$$
\begin{aligned}
P(X_{(A_{\mathtt{DUR}})+C_{\mathtt{DUR}}} = 0) &= \frac{(\frac{\mathtt{CR}}{1.440} \cdot (A_{\mathtt{DUR}} + C_{\mathtt{DUR}}))^0}{0!} \cdot e^{-\frac{\mathtt{CR}}{1.440} \cdot (A_{\mathtt{DUR}} + C_{\mathtt{DUR}})} \\
&= e^{-\frac{\mathtt{CR}}{1.440} \cdot (A_{\mathtt{DUR}} + C_{\mathtt{DUR}})}
\end{aligned}
$$

Im linken Teil der Abbildung 6.14 wird untersucht, wie sich die Dauer des Beobachtungszeitraums t bei unterschiedlicher Häufigkeit an *Live-Migrationen* λ auf die Erfolgswahrscheinlichkeit auswirkt. Als Untersuchungsgegenstand wurden hier Sekunden gewählt, da im Worst-Case eine *Live-Migration* bis zu *50* Sekunden dauern kann [91, 183].

Untersucht werden hier die Ereignisraten für *Live-Migrationen* alle 16 Tage ($\mathtt{CR} = 0.06$), alle 8.5 Tage ($\mathtt{CR} = 0.12$), alle zwei Tage ($\mathtt{CR} = 0.5$) und dreimal pro Tag ($\mathtt{CR} = 3$). Die Untersuchung zeigt, dass mit zunehmender Häufigkeit $\mathtt{CR}$ die Erfolgswahrscheinlichkeit, dass ein Monitoringprozess einen Zeitraum ohne *Live-Migration* erwischt, abnimmt. Allerdings zeigt sich auch, dass für die untersuchten Häufigkeiten von *Live-Migrationen* die Erfolgswahrscheinlichkeit bei über 99.3% liegt. Bei 1.000 Monitoringprozessen pro Tag sind im Schnitt nur 7 durch Inkonsistenzen betroffen.

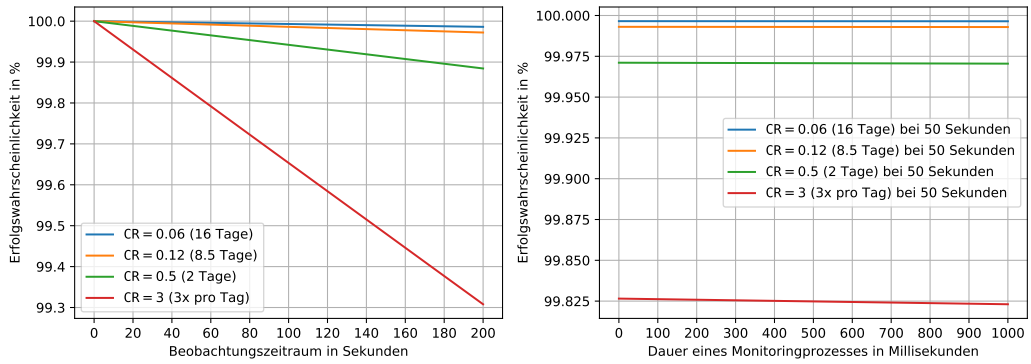

Abbildung 6.14: *Parameterstudie zur Auswirkung von Live-Migrationen auf Monitoringprozesse mit Konsistenz.*

Der rechte Teil der Abbildung 6.14 zeigt, wie sich die Dauer eines Monitoringprozesses $A_{\mathtt{DUR}}$ bei unterschiedlicher Häufigkeit von *Live-Migrationen* auf die Erfolgswahrscheinlichkeit auswirkt. Der Beobachtungszeitraum t wird dabei fest auf den Wert des Worst-Case-Szenarios von 50 Sekunden gesetzt und die Dauer eines Monitoringprozesses $A_{\mathtt{DUR}}$ von 0 bis 1.000 Millisekunden variiert. Für Monitoringprozesse im Millisekundenbereich zeigt sich sehr deutlich, dass dies nur sehr geringe Auswirkungen auf die Erfolgswahrscheinlichkeit eins Monitoringprozesses hat. Die in einer Cloud-Umgebung vorherrschende Elastizität wirkt sich also nur kaum auf Monitoring-Unabhängigkeit aus. Allerdings hängt dies auch sehr vom konkreten Anwendungsfall ab, sollten die Monitoringprozesse deutlich länger dauern oder *Live-Migrationen* häufiger stattfinden, so hat dies erheblichen Einfluss auf erreichbare Monitoring-Unabhängigkeit. Wenn konsistente Erfassung trotzdem relevant ist, muss in solchen Fällen der Monitoringprozess so angepasst werden, dass weniger unabhängige Klassen der Taxonomie erreicht werden – beispielsweise durch Abruf von konsistenten Informationen aus dem CMS.

In Abbildung 6.15 werden die Auswirkungen von deutlich häufigeren *Live-Migrationen* pro Tag (bis zu einer Milliarde) auf die Konsistenz von Cloud-Monitoring untersucht. An dieser Stelle sei noch einmal deutlich hervorgehoben, dass die untersuchte Anzahl an *Live-Migrationen* dabei nicht der Gesamtzahl an *Live-Migrationen* innerhalb der ganzen Cloud-Umgebung entspricht, sondern die Häufigkeit bei der ein einzelner Kunde von *Live-Migrationen* betroffen ist. Hier stellen eine Milliarde *Live-Migration* pro Tag eine sehr deutliche obere Schranke da.

In der linken Hälfte der Abbildung 6.15 wird die Auswirkung verschiedener Häufigkeit von *Live-Migrationen* und Dauer des Beobachtungszeitraums t auf die sogenannte Fehlerwahrscheinlichkeit dargestellt. Unter Fehlerwahrscheinlichkeit wird hier die Gegenwahrscheinlichkeit zur Erfolgswahrscheinlichkeit verstanden. Also die Wahrscheinlichkeit

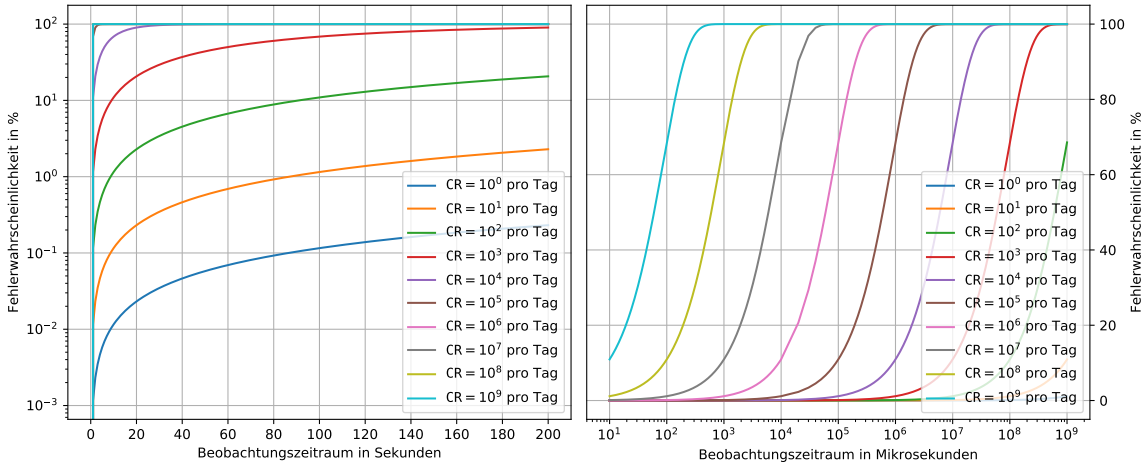

Abbildung 6.15: *Auswirkung von häufigen Live-Migrationen pro Tag auf die Wahrscheinlichkeit, dass ein Monitoringprozess aufgrund von Inkonsistenzen nicht erfolgreich abgeschlossen werden kann. Hier als Fehlerwahrscheinlichkeit bezeichnet.*

dafür, dass ein Monitorinprozess nicht erfolgreich durchgeführt werden kann. Es zeigt sich deutlich, dass mit zunehmender Häufigkeit von *Live-Migrationen* die Fehlerwahrscheinlichkeit drastisch steigt. Bereits ab 10.000 *Live-Migrationen* pro Tag ist diese nahe 100%.

Im rechten Teil der Abbildung 6.15 ist eine detaillierte Betrachtung der Fehlerwahrscheinlichkeit für hohe Häufigkeiten dargestellt. Die Dauer des Beobachtungszeitraums t wird dabei nun in Mikrosekunden variiert. Es zeigt sich noch einmal recht deutlich, dass Monitoring-Mechanismen die konsistente Monitoringdaten benötigen bei hoher Anzahl von *Live-Migrationen* pro Tag kaum anwendbar sind. Selbst bei einer Dauer von wenigen Mikrosekunden schlägt unabhängiges Cloud-Monitoring mit hoher Wahrscheinlichkeit fehl.

6.6.1.4 Ergebnis

Als Ergebnis bleibt festzuhalten, dass der Einfluss von Elastizität auf Monitoring-Unabhängigkeit von mehreren Faktoren abhängt. Zunächst von der konkreten Ausgestaltung eines Monitoring-Mechanismus und dessen Monitoringprozess. Einige Fragen die in diesem Kontext beantwortet werden müssen sind: Welche Klasse der Taxonomie wird erreicht? Wie lange ist die Ausführungsdauer des Prozess? Ist konsistente Datenerhebung überhaupt erforderlich?

Zweiter wesentlicher Faktor ist die Häufigkeit mit der Änderungen an der Cloud-Umgebung vorgenommen werden die die Konsistenz betreffen. Die exemplarisch anhand von *Live-Migrationen* durchgeführte Analyse hat dabei gezeigt, das Monitoring-Unabhängigkeit und hochdynamische Cloud-Umgebungen nur schwer zu vereinen sind. Allerdings zeigt die Betrachtung auch auf, dass im Fall *Live-Migrationen* bei einer Häufigkeit im Bereich von alle *8.5* Tage eine *Live-Migration* bis maximal drei mal pro Tag, unabhängiges Cloud-Monitoring durchaus realisierbar ist.

In weiterführenden Arbeiten können dann konkrete Monitoring-Mechanismen konstruiert werden und der Einfluss von Elastizität hierauf untersucht werden. Allerdings bleibt fraglich, welche Verallgemeinerungen sich aus der Untersuchung einzelner Monitoring-Mechanismen ziehen lassen und wie Daten zu Internas von Cloud-Umgebungen erhoben werden sollen.

6.6.2 Überblick zum entstehenden Overhead

Bei *Unabhängigen* Monitoring-Mechanismen ist insbesondere der auf physischen Cloud-Knoten entstehende Overhead von Interesse. Bei dieser Klasse von Monitoring-Unabhängigkeit werden Monitoringdaten *Direkt* auf den physischen Cloud-Knoten erhoben. Dies beeinflusst direkt die Leistungsfähigkeit der Cloud-Umgebung.

Der auf physischen Cloud-Knoten entstehende Overhead hängt stark von der Art des eingesetzten Monitoring-Mechanismus ab. Hinzu kommt, dass der Overhead proportional zur Anzahl der Abfragen ist – je mehr Anfragen pro Sekunde desto höher der Overhead.

An dieser Stelle soll eine erste Abschätzung des entstehenden Overheads präsentiert werden. Hierzu werden die Ergebnisse aus verschiedenen studentischen Abschlussarbeiten herangezogen. Dabei dienen die ermittelten Werte als wichtiger erster Indikator für die Praktikabilität von Monitoring-Unabhängigkeit.

Unter anderem zeigt die Arbeit von Bauer [TT-12], dass der durch Monitoring-Unabhängigkeit entstehende Overhead selbst bei 250 Monitoringdatenerhebungen pro Sekunde kaum ins Gewicht fällt ($< 5\%$ der totalen CPU Kapazität). Allerdings zeigt die Arbeit auch deutlich, dass hierfür genügend Leistungsreserven in Abhängigkeit der zu überwachenden VMs auf den physischen Cloud-Knoten vorgehalten werden müssen, da ansonsten Auswirkungen auf VMs (in der Regel Paketverlust) messbar sind.

Weiter zeigt die Arbeit von Hoor [TT-6], dass zumindest im Bereich von Blockspeicherdiensten, *LVM* und *CephFast* mit ähnlicher Performance abgefragt werden können. Der Blockspeicherdienst *Ceph* hingegen eignet sich aufgrund des dort entstehenden Overheads nicht für unabhängiges Cloud-Monitoring (über 30% Overhead).

Die Abschlussarbeit von Roth [TT-4] legt weiter nahe, dass der bei Nutzung des Betriebssystems als Datenquelle entstehende Overhead gering ist (zwischen 3% und 6%). Weiter wird gezeigt, dass der Overhead zur Erfassung der CPU Last eines physischen Cloud-Knotens sogar alleine von der Anfragehäufigkeit abhängt und bei Variation der ausgeführten VM konstant bleibt.

6.7 Zusammenfassung

In diesem Kapitel wurde anhand von explorierten Datenquellen und einer Analyse der Implikationen in realen Cloud-Umgebungen eine Taxonomie zur Klassifizierung von Monitoring-Unabhängigkeit präsentiert. Nach bestem Wissen ist diese Taxonomie zum Stand der Forschung derzeit einzigartig.

Die Untersuchung von Seiteneffekten indiziert, dass für konsistente Monitoring-Mechanismen die Häufigkeit und Dauer von Änderungen des CMS an der Cloud-Umgebung eine entscheidende Rolle für die Umsetzung von Monitoring-Unabhängigkeit spielt. Für hochdynamische Cloud-Umgebungen lässt sich dabei keine Monitoring-Unabhängigkeit erreichen – eine konsistente Erhebung von Monitoringdaten ist nicht möglich. Allerdings zeigt die Untersuchung auch, dass zumindest in dem Fall von *Live-Migrationen* in der von IBM beschrieben Cloud-Umgebung [19] CMS-unabhängiges Cloud-Monitoring durchgeführt werden könnte. Weiter fördert die Analyse von Seiteneffekten zu Tage, dass der bei Monitoring-Unabhängigkeit entstehende Overhead je nach Anwendungsfall im vertretbaren Rahmen liegt. Jedoch sind hier je konkretem Anwendungsfall eigene Berechnungen erforderlich, bevor ein Monitoring-Mechanismus ohne weiteres eingesetzt werden könnte.

Kapitel 7

Zusammenspiel der Monitoring-Bausteine

In den vorangegangenen Kapiteln wurden verschiedenen Beiträge für CMS-unabhängiges Monitoring von Cloud-Umgebungen vorgestellt. Unter anderem wurden die Monitoring-Bausteine *Adaptives Transportsystem für Request-Response in Gruppen*, *Verdeckender Ungleichheitsnachweis für Monitoringdaten* und *Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen* diskutiert.

In diesem Kapitel werden die entwickelten Monitoring-Bausteine wieder aufgegriffen und in das *CloudInspector Rahmenwerk für unabhängiges Cloud-Monitoring* aus Kapitel 3 integriert. Dabei wurde das Konzept des *CloudInspectors* für diese Dissertation prototypmäßig realisiert. Außerdem wurde im Rahmen des EU-Projekts *SECCRIT* [300] die Umsetzung verschiedener Monitoring-Mechanismen erprobt. Eine vorläufige Version von *CloudInspector* findet sich hier [301] unter der sogenannten *Reference Use License*.

Nachstehend wird zunächst eine Übersicht der für *CloudInspector* konstruierten Monitoring-Mechanismen und Anwendung der in dieser Dissertation präsentierten Monitoring-Bausteine gegeben. Anschließend werden die so ermittelten Ergebnisse zusammengefasst.

7.1 Konstruktion von Monitoring-Mechanismen

Im Rahmen des SECCRIT-Projekts wurden verschiedene Fallstudien für Cloud-Monitoring untersucht. Siehe hierzu insbesondere Anhang A. Als Monitoring-Mechanismen wurden dabei umgesetzt: *Affinität*, *Anti-Affinität*, *Geo-Lokation*, *Hot-Spare-Setup*, *Isolation* und *Pflichtverletzung*. Eine Erläuterung folgt weiter unten im Text.

Des Weiteren wurden von Flittner et al. [TT-2] Monitoring-Mechanismen zur Überprüfung von SFC vorgeschlagen. Dieser wird hier als *Service-Function-Chaining* bezeichnet. Darüber hinaus erörtern Flittner et. al in [TT-1] Details eines Monitoring-Mechanismus zur kundengesteuerten Analyse des Datenverkehrs von Netzen. Für die weitere Betrachtung wird hierauf mit dem Begriff *Datenverkehr-Extraktion* verwiesen. Außerdem wird in der studentischen Abschlussarbeit von Hoor [TT-6] das CMS-unabhängige Monitoring von Blockspeicherdiensten besprochen. Dieser Monitoring-Mechanismus wird hier als *Blockspeicher* bezeichnet.

In Tabelle 7.1 werden die im Rahmen dieser Dissertation konstruierten Monitoring-Mechanismen zusammengetragen. An dieser Stelle sei noch einmal explizit darauf hingewiesen, dass im Rahmen dieser Dissertation nicht die Konstruktion von Monitoring-Mechanismen im Vordergrund steht, sondern die Betrachtung der aufgeführten Monitoring-Bausteine. So dienen die hier dargestellten Monitoring-Mechanismen nur als Beispiel. So ist die dargestellte Klassifikation auch nicht allgemeingültig, sondern gibt nur einen Überblick über die hier gewählte Umsetzung.

Der im Rahmen von *SECCRIT* untersuchte Monitoring-Mechanismus *Affinität* dient zur Überprüfung ob zwei VMs auf dem gleichen physischen Cloud-Knoten realisiert sind. Als Cloud-Umgebung wurde hier *OpenStack* verwendet. Der Monitoring-Mechanismus wurde über die Oberfläche von *CloudInspector* Kunden zugänglich gemacht. Der Prozessschritt *Datenerhebung* erfolgt hierfür bei OpenStack direkt und es findet in der Verarbeitung keine weitere Interaktion statt – OpenStack liefert bereits die Zugehörigkeit zu einem Kunden automatisch mit (vergleiche Abschnitt 6.4). Daher wird die höchste Monitoring-Unabhängigkeit erreicht. Weiter kommt das in dieser Dissertation entwickelte adaptive *gAUDIT*-Transportsystem zum Einsatz. Der Einsatz des Monitoring-Bausteins *CoCoPAS* (*Verdeckender Ungleichheitsnachweis für Monitoringdaten*) wird in der Fallstudie zwar nicht betrachtet, ließe sich aber ohne weitere integrieren.

Der Monitoring-Mechanismus *Anti-Affinität* ermöglicht es Kunden mit Hilfe von *CloudInspector* zu überprüfen, ob zwei VMs auf unterschiedlichen physischen Cloud-Knoten realisiert wurden. Im Rahmen der Fallstudie wurde der Monitoring-Mechanismus für die auf *VMware*-aufbauende Cloud-Umgebung umgesetzt. Daher erfolgt neben der *direkten* Erfassung von Monitoringdaten im Prozessschritt Verarbeitung ein Rückgriff auf das *CMS* für die Zuordnung von virtuellen Ressourcen zu Kunden. Daher wird hier nur die Klasse *Teilabhängig* erreicht. Außerdem kommt wie bei Affinität *gAUDIT* zur Sammlung von Monitoringdaten zum Einsatz und *CoCoPAS* ließe sich ohne weiteres an der Kundenschnittstelle einsetzen.

Der Monitoring-Mechanismus *Blockspeicher* ermittelt, ob der Blockspeicherdienst die vom Kunden geforderte Redundanz aufweist. Hierfür wurde OpenStack als Cloud-Umgebung verwandt. Dabei werden hier zwei Fälle gleichzeitig betrachtet, verdeutlicht durch die

	Cloud-Umgebung		Unabhängigkeit für Monitoring-Mechanismen												*gAUDIT*	*CoCoPAS*
			Auslöser		Datenerhebung			Verarbeitung			Klassifizierung					
Monitoring-Mechanismus	OS	VW	K	R	D	E	C	O	E	C	AB	TA	TU	UN		
Affinität	●	–	●	–	●	–	–	●	–	–	–	–	–	●	●	(●)
Anti-Affinität	–	●	●	–	●	–	–	–	–	●	–	●	–	–	●	(●)
Blockspeicher	●	–	●	–	●	⁕	–	–	–	–	–	–	⁕	●	●	–
Geo-Lokation	●	–	●	–	●	–	–	●	–	–	–	–	–	●	●	–
Hot-Spare-Setup	●	–	●	–	–	–	●	●	–	–	●	–	–	–	–	–
Isolation	–	●	●	–	●	–	–	–	–	●	–	●	–	–	●	–
Datenverkehr-Extraktion	●	–	●	–	●	–	–	–	●	–	–	●	–	–	–	–
Pflichtverletzung	–	●	–	●	●	–	–	●	–	–	–	–	–	●	–	–
Service-Function-Chaining	●	–	●	–	●	–	–	–	–	●	–	●	–	–	(●)	–

Tabelle 7.1: *Tabellarische Übersicht der im Rahmen dieser Dissertation konstruierten Monitoring-Mechanismen. Achtung: diese Tabelle dient nicht zur generellen Klassifizierung von Monitoring-Mechanismen, sondern zeigt lediglich die im Rahmen dieser Dissertation betrachteten Varianten. Monitoring-Mechanismen lassen sich auf vielfältigste Weise realisieren. Fällt ein konstruierter Monitoring-Mechanismus in die durch Spalten dargestellten Kategorien, so wird das Symbol ● verwandt. Das Symbol ⁕ wird für eine untersuchte Designalternative eingesetzt. Das Symbol – wird verwandt, wenn der untersuchte Monitoring-Mechanismus sich hier nicht einkategorisieren lässt. Das in Klammern gesetzte Symbol (●) zeigt an, dass der Aspekt nicht direkt in der Fallstudie berücksichtigt wurde, aber im Rahmen dieser Dissertation aufgegriffen wird. Die Abkürzungen sind dabei wie folgt: OS steht für OpenStack, VW für VMware, K für Kunde, R für Rahmenwerk, D für Direkt, E für Extern, C für CMS, O für Ohne, AB für Abhängig, TA für Teilabhängig, TU für Teilunabhängig und UN für Unabhängig.*

Verwendung der unterschiedlichen Symbole • und ⁎. Wird ein auf dem physischen Cloud-Knoten lokaler ausgeführter Blockspeicherdienst *Direkt* (Symbol •) abgefragt, so ist vollständige Monitoring-Unabhängigkeit gegeben. Wird allerdings auf einen *Externen* Blockspeicherdienst zurückgegriffen, so lässt sich nur noch die Klasse *Teilunabhängig* erreichen (Symbol ⁎). *gAUDIT* kommt zur Sammlung der Monitoringdaten zum Einsatz.

Der zur Geolokation von VMs eingesetzte Monitoring-Mechanismus *Geo-Lokation* verhält sich dabei identisch wie der Monitoring-Mechanismus *Blockspeicher* bei *Direkter* Erhebung. Nur wird in diesem Fall anstatt der aktuellen Konfiguration des Blockspeichers die Lokation des physischen Cloud-Knotens ausgelesen. Ohne Beschränkung der Allgemeinheit wird in dieser Fallstudie davon ausgegangen, dass die physischen Cloud-Knoten zuvor entsprechend konfiguriert wurden. Gegebenenfalls ist auch Nutzung externer Dienste zur Geolokation von physischen Cloud-Knoten denkbar, allerdings ist hierfür noch weitere Forschung notwendig. In diesem Fall würde der Monitoring-Mechanismus allerdings in die Klasse *Teilunabhängig* fallen.

Als nächstes wird der Monitoring-Mechanismus *Hot-Spare-Setup* zur Überprüfung, ob das CMS zu einer gegebenen VM eine Standby-VM konfiguriert hat, betrachtet. Im Rahmen der Fallstudie wurde hier OpenStack verwendet. Im Prozessschritt der Datenerhebung muss der Monitoring-Mechanismus hier auf das CMS zugreifen, da nur dort Informationen über geplantes Verhalten des CMS vorliegen. Der im vorangegangenen Kapitel entwickelten Taxonomie folgend, fällt der Monitoring-Mechanismus damit automatisch in die Klasse in der keine Monitoring-Unabhängigkeit erreicht werden kann (*Abhängig*). Auch muss nicht *gAUDIT* zur Datensammlung verwandt werden, da die Informationen von einer zentralen Stelle erhoben werden können.

Der Monitoring-Mechanismus *Isolation* wurde genauso wie der Monitoring-Mechanismus *Anti-Affinität* entwickelt, nur dass hier geprüft wird, dass keine VMs anderer Kunden auf dem physischen Cloud-Knoten ausgeführt werden. Es wird die Klasse *Teilabhängig* erreicht. *CoCoPAS* lässt sich hier nicht einsetzen, da keine Überprüfung auf Ungleicheit von Monitoringdaten stattfindet.

Soll Datenverkehr des Kunden automatisch aus der Cloud-Umgebung extrahiert und in einer vom Kunden betriebenen VM analysiert werden, so kann der entwickelte Monitoring-Mechanismus *Datenverkehr-Extraktion* verwandt werden. Dieser wurde für OpenStack umgesetzt und erhebt Monitoringdaten *Direkt* aus dem virtuellen Switch eines physischen Cloud-Knotens. Allerdings muss der extrahierte Datenverkehr in dem Prozessschritt *Verarbeitung* separat gekapselt und an die Analyse-VM des Kunden versandt werden, weswegen hier *Extern* eingetragen ist. Im Ergebnis fällt der Monitoring-Mechanismus daher in die Klasse *Teilabhängig*. Der Einsatz von *gAUDIT* ist nicht notwendig, da die Monitoringdaten nicht von *CloudInspector* zentral gesammelt werden.

Der Monitoring-Mechanismus *Pflichtverletzung* ist der einzig betrachtete Anwendungsfall, bei dem nicht der *Kunde* der Auslöser des Monitoringprozesses ist, sondern das *Rahmenwerk* selbst. Der Mechanismus soll nämlich kontinuierlich Metadaten über das Cloud-Verhalten aufzeichnen um gegebenenfalls eine Pflichtverletzung des Cloud-Anbieters vor Gericht nachweisen zu können. Aus diesem Grund kommt auch nicht *gAUDIT* zum Einsatz, vielmehr werden die Daten an den in Kapitel 3 vorgestellten Log-Server gesandt und von dort sicher bei einer vertrauenswürdigen dritten Partei verwahrt. Realisiert wurde der Monitoring-Mechanismus dabei für die Cloud-Umgebung *VMware*. Die Datenerhebung erfolgt *Direkt* und im Prozessschritt Verarbeitung werden keine weiteren Informationen benötigt (*Ohne*), weswegen der Monitoring-Mechanismus volle Monitoring-Unabhängigkeit erreicht. Bemerkenswert ist, dass hier nicht wie sonst bei VMware üblich das CMS für die Zuordnung zu Kunden befragt werden muss. Vielmehr wird in dieser Fallstudie die Zuordnung von aufgezeichneten Monitoringdaten zu Kunden dem Sachverständigen der die Fehler-Ursachen-Analyse durchführt überlassen.

Durch den letzten hier betrachteten Monitoring-Mechanismus *Service-Function-Chaining* soll die vom Kunden spezifizierte Konfiguration von SFC überprüft werden. Hierzu wurde eine Umsetzung auf Basis von OpenStack realisiert. Zwar erfolgt die Datenerhebung noch *Direkt*, allerdings ist für die Zuordnung von erfassten Monitoringdaten zur vom Kunden spezifizierten SFC-Konfiguration der Zugriff auf das CMS erforderlich, weswegen der Monitoring-Mechanismus in die Klasse *Teilabhängig* fällt. In dem untersuchten Fall wird zunächst nicht *gAUDIT* eingesetzt, jedoch ist dies für zukünftige Versionen geplant.

7.2 Wichtige Ergebnisse

Durch die Konstruktion von neun unterschiedlichen Monitoring-Mechanismen für insgesamt zwei verschiedene Cloud-Umgebungen wird die Leistungsfähigkeit des *CloudInspector*-Rahmenwerks für unabhängiges Cloud-Monitoring demonstriert. Dabei benutzt eine Mehrheit der konstruierten Monitoring-Mechanismen den Monitoring-Baustein des *Adaptiven* gAUDIT-*Transportsystems für Request-Response in Gruppen* (hier sechs von neun). Zwar wurde für die Realisierung meist eine naive Implementierung gewählt, dennoch konnte in rund 44% aller Fälle die höchste Monitoring-Unabhängigkeit erreicht werden. Weitere 44% entfallen auf die Klasse *Teilabhängig* da unter anderem die Zuordnung von virtuellen Ressourcen zu Kunden vom CMS abgerufen werden muss. Hier ist für die Zukunft die Entwicklung neuartiger Monitoring-Mechanismen wünschenswert, so dass der Zugriff auf das CMS entfallen kann – beispielsweise wenn anstatt eines passiven (diese Dissertation) ein aktiver Monitoringansatz gewählt wird. Für den Monitoring-Baustein des *Verdeckenden Ungleichheitsnachweis für Monitoringdaten* lässt

sich festhalten, dass dieser seine Leistungsfähigkeit derzeit nur in speziellen Anwendungsfällen unter Beweis stellen kann (Monitoringdaten-Überprüfung auf Gleichheit oder Ungleicheit). Für die hier dargestellte Analyse wurde jedoch eine breit gefächerte Untersuchung durchgeführt.

7.3 Zusammenfassung

In diesem Kapitel wird das Zusammenspiel der im Rahmen dieser Dissertation untersuchten Monitoring-Bausteine in dem *CloudInspector* Rahmenwerk für unabhängiges Cloud-Monitoring beleuchtet. Dabei werden verschiedene Monitoring-Mechanismen für reale Cloud-Umgebungen konstruiert und die Anwendbarkeit der Monitoring-Bausteine dieser Dissertation verdeutlicht. Aufgrund der schier unbegrenzten Möglichkeiten zur Realisierung von Monitoring-Mechanismen dienen die hier vorgestellten Ergebnisse als wichtiger Wegweiser für in der Praxis erreichbare Monitoring-Unabhängigkeit. Der Entwurf von hochentwickelten Monitoring-Mechanismen, die vollständig ohne Beteiligung des CMS auskommen, bleibt Gegenstand zukünftiger Arbeiten. Im Rahmen dieser Dissertation konnte zumindest demonstriert werden, dass die Vision nicht unerreichbar scheint.

Kapitel 8

Zusammenfassung und Ausblick

In der heutigen IT-Landschaft spielt Cloud-Computing bereits eine prominente Rolle und ist Dank treibender Kräfte wie *Smart Cities*, *5G*, *Internet-of-Things* oder *Machine Learning* auch in Zukunft nicht mehr wegzudenken. Dabei wird der Siegeszug von Cloud-Computing durch die rasante technologische Entwicklung noch befeuert, etwa durch die Fortschritte im Bereich von Virtualisierungstechnologien durch *Software-Defined-Networking*. Allerdings wird es bei dieser rasanten Entwicklung immer schwieriger, sicherzustellen, dass die vom Kunden geforderten Randbedingungen tatsächlich in Cloud-Umgebungen umgesetzt werden. Daher werden zusätzliche Monitoringmöglichkeiten für Cloud-Umgebungen benötigt, um das Vertrauen von Kunden in CMS wiederherzustellen.

Die in dieser Arbeit verfolgte Vision ist es, den bei Cloud-Computing vorherrschenden Mangel an Transparenz durch die Ausgestaltung von CMS-unabhängigem Monitoring zumindest in Teilen zu beseitigen. Dies erfordert unter anderem die Entwicklung von geeigneten Monitoring-Bausteinen. Das Ziel dieser Dissertation war daher Entwicklung drei ausgewählter Monitoring-Bausteine: 1) Adaptives Request-Response-Transportsystem für Gruppen, 2) Verdeckender Ungleichheitsnachweis für Monitoringdaten und 3) Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen. Die entwickelten Monitoring-Bausteine wurden in ein neuartiges Rahmenwerk für unabhängiges Cloud-Monitoring integriert und für zwei reale Cloud-Umgebungen prototypmäßig umgesetzt beziehungsweise erprobt.

Relevante Herausforderungen in diesem Kontext waren: 1) die Konzeption von effizienter und adaptiver Request-Response-Kommunikation im Kontext von Gruppen, 2) die Überwältigung des Spannungsverhältnisses zwischen Überprüfbarkeit und Schutz von

Betriebsgeheimnissen bei der Überprüfung von Monitoringdaten auf Ungleichheit und 3) die Eingrenzung von Monitoring-Unabhängigkeit anhand theoretischer und praktischer Erkenntnisse.

Nachstehend werden die wichtigsten Ergebnisse dieser Doktorarbeit zusammengetragen und anschließend wird ein Ausblick auf weiterführende Arbeiten geben.

8.1 Ergebnisse dieser Arbeit

Im Kontrast zu verwandten Arbeiten werden in dieser Doktorarbeit die folgenden Ergebnisse vorgestellt.

Adaptives Transportsystem für Request-Response in Gruppen: Die durchgeführte simulative Parameterstudie zeigt verschiedene Aspekte auf. Im ersten Teil wird recht deutlich, dass aufgrund des in Cloud-Umgebungen zu erwartenden geringen Nachrichtenverlusts, bereits vereinzelte Sendewiederholungen per Unicast ausreichen, um die anvisierte Response-Vollständigkeit zu erreichen. Im Gegensatz dazu betrachten verwandte Arbeiten Nachrichtenverlust entweder überhaupt nicht oder berücksichtigen Response-Vollständigkeit nur am Rande. Nach bestem Wissen ist das hier vorgestellte Transportsystem derzeit das einzige, was die Durchführung von Zufallsstichproben aus einer Gruppe von Empfängern erlaubt. Aus diesem Grund wird auch eine deutlich höhere bidirektionale-Zuverlässigkeit erreicht als in vergleichbaren Fällen. Im zweiten Teil wird der Effekt der Auswahl zwischen Multicast und Unicast für den Versand von Request-Nachrichten auf die im Netz erreichbare Effizienz untersucht. Dabei wird gezeigt, dass wenn die Anzahl der erwarteten Response-Nachrichten 50% der Gesamtgruppengröße übersteigt, die Verwendung von Multicast im Allgemeinen den höchsten Einsparungsfaktor bietet. Allerdings zeigt die Parameterstudie auch auf, dass bei Durchführung innerhalb der Klasse *Statistisch-zuverlässig* die erreichbare Effizienz auch von der Stichprobengröße abhängt. Insgesamt hat es sich dabei als sinnvoll herausgestellt, die Auswahl zwischen Multicast und Unicast sowohl initial, als auch für Sendewiederholungen durchzuführen, wobei im Fall von Cloud-Umgebungen in der Regel, wenn überhaupt, nur eine einzelne Response-Nachricht fehlt und daher nur eine Sendewiederholung per Unicast in Frage kommt. Im dritten Teil wird das Incast-Verhalten des entwickelten Transportsystems näher betrachtet. Für das untersuchte *Basis Incast Szenario* konnte gezeigt werden, dass die Wahl einer Gleichverteilungsfunktion unter Berücksichtigung der erwarteten Response-Nachrichten auf Seiten von Respondents am Besten zur Desynchronisation von Response-Nachrichten geeignet ist. Dabei hat sich herausgestellt, dass die Erhöhung des Intervalls der Gleichverteilungsfunktion um 77.6ns je erwarteter Response-Nachricht am Besten geeignet

ist. Im Vergleich zur Exponentialfunktion bringt diese Wahl für die Ausführungszeit von Request-Response einen rund 4.4-fachen Vorteil mit sich, bei gleichzeitiger Desynchronisation von Response-Nachrichten.

Verdeckender Ungleichheitsnachweis für Monitoringdaten: Der Ungleichheitsnachweis wurde im Rahmen dieser Dissertation hauptsächlich analytisch untersucht. Allerdings wurde auch ein kurzer Ausblick auf den in der Praxis entstehenden Overhead präsentiert. Dabei konnte im theoretischen Teil gezeigt werden, dass der neuartige Ungleichheitsnachweis für Hinterlegungsverfahren so ausgelegt werden kann, dass dieser die Sicherheitseigenschaften *Geheimhaltung*, *Eindeutigkeit*, *Zurechenbarkeit* und *Nichtabstreitbarkeit* einhalten kann. Im Stand der Forschung findet sich bisher nur ein Nachweis für Gleichheit, die Betrachtung von Ungleichheit ist neu. Darüber hinaus wurde in der Analyse für den Nachweis der Zero-Knowledge-Eigenschaft ein Simulator skizziert. Der wesentliche Beitrag dieser Dissertation liegt dabei jedoch nicht in der Ausgestaltung eines effizienten kryptographischen Überprüfungsmechanismus, sondern vielmehr im ersten Einsatz einer solchen Technologie für Cloud-Monitoring. So wurde bei der Studie zum Overhead aufgedeckt, dass dieser sich im Wesentlichen parallel zum beim Einsatz von OpenSSL entstehenden Overhead entwickelt. Bei der weiteren Betrachtung der entstehenden Datenmenge wurden die Signaturen als wesentlicher Verursacher identifiziert.

Unabhängigkeit für Monitoring-Mechanismen in Cloud-Umgebungen: Für diesen Monitoring-Baustein wurde ein zweistufiger Ansatz aus der Kombination von Theorie und Praxis zur Entwicklung einer Taxonomie für Monitoring-Unabhängigkeit gewählt. Die zentrale Forderung nach Unabhängigkeit vom CMS ist nach Bestem Wissen derzeit im Stand der Forschung nicht abgedeckt – hier kann diese Dissertation einen bedeutenden Beitrag leisten. Anhand einer theoretischen Analyse wurde aufgezeigt, dass sich prinzipiell der *VMM*, *Network-Hypervisor*, *Storage-Hypervisor*, das *Betriebssystem* und der Kontrollkanal des CMS als Datenquelle für CMS-unabhängiges Cloud-Monitoring nutzen lassen. Allerdings hat die anschließende Untersuchung der Implikationen in realen Cloud-Umgebungen verdeutlicht, dass in der Praxis ein Monitoringprozess nur selten vollständig unabhängig ausgelegt werden kann. In der Regel erfordern Monitoring-Mechanismen die Zuordnung zwischen physischen und virtuellen Ressourcen, was häufig nur durch das CMS zu lösen ist. Daher unterscheidet die entwickelte Taxonomie und der zugehörige Entscheidungsbaum zwischen den Klassen *abhängig*, *teilabhängig*, *teilunabhängig* und *unabhängig*. Außerdem wurden Seiteneffekte *von* und *auf* Monitoring-Unabhängigkeit untersucht. Dabei wurde festgestellt, dass die in Cloud-Umgebungen typische Elastizität keinen großen Einfluss auf die Konsistenz von Monitoring-Unabhängigkeit hat, wenn man die Anzahl der einen Kunden betreffenden *Live-Migrationen* pro Tag als gering voraussetzt (unter drei *Live-Migrationen* pro Tag). Dass dies eine realistische Annahme ist, belegt

die im Kapitel durchgeführte Analyse zur Aktualisierungshäufigkeit von physischen Cloud-Knoten. Allerdings wurde auch ermittelt, dass in hoch-dynamischen Cloud-Umgebungen Konsistenz für Monitoring-Unabhängigkeit nicht erreichbar ist (weit über 1.000 *Live-Migrationen* für einen Kunden pro Tag). Die kurze Zusammenfassung der studentischen Untersuchungen zum Monitoring-Overhead auf physischen Cloud-Knoten zeigt einerseits, dass dieser in der Regel im vertretbaren Rahmen liegt. Andererseits hängt hier der tatsächlich entstehende Overhead maßgeblich von der Ausgestaltung eines konkreten Monitoring-Mechanismus und der Abfragehäufigkeit durch Kunden ab.

Rahmenwerk für unabhängiges Cloud-Monitoring: Anhand von verschiedenen Fallstudien wurde die Integration der im Rahmen dieser Dissertation entwickelten Monitoring-Bausteine in ein Rahmenwerk für unabhängiges Cloud-Monitoring demonstriert. Hierfür wurden exemplarisch neun Monitoring-Mechanismen konstruiert. Die Ergebnisse legen nahe, dass die im Rahmen dieser Dissertation vorgestellten Monitoring-Bausteine für den Einsatz in der Praxis relevant sind. Da im Rahmen dieser Dissertation die beschrieben Monitoring-Bausteine im Vordergrund standen, bleibt für zukünftige Arbeiten eine umfassendere Untersuchung zur Konstruktion von hochentwickelten CMS-unabhängigen Monitoring-Mechanismen.

8.2 Weiterführende Arbeiten

Nachdem nun die relevanten Ergebnisse dieser Dissertation zusammengetragen wurden, werden in diesem Abschnitt einige wichtige Ansatzpunkte für weiterführende Arbeiten herausgestellt.

Eine Herausforderung für zukünftige Cloud-Umgebungen ist der Wandel hin zu föderierten Ansätzen, beispielsweise im Kontext von *Edge-* und *Fog-Computing* oder *Microclouds*. In diesem Zusammenhang dürfte es spannend sein, die im Rahmen dieser Arbeit entwickelten Monitoring-Bausteine für diese neuen Anwendungsfälle zu adaptieren. Sicherlich wäre es auch vielversprechend, die in dieser Dissertation gewonnen Erkenntnisse auf höhere Abstraktionsebenen von Cloud-Computing wie PaaS oder SaaS zu übertragen, oder gar Cloud-Umgebungen mit mehreren CMSs zu betrachten (sogenanntes Slicing).

Die Ergebnisse dieser Arbeit zum adaptiven Transportsystem für Request-Response in Gruppen bieten eine gute Ausgangslage für weiterführende Arbeiten zur Untersuchung der Abfrage größerer Mengen an Monitoringdaten mittels Request-Response – derzeit wird je Kommuikationsrichtung nur eine Nachricht übertragen. Hierzu müssen Aspekte von Stau- und Flusskontrolle im Zusammenhang mit dem im Rahmen dieser Arbeit

entwickelten Ansatz für Request-Response in Gruppen in Einklang gebracht werden. Hierbei ergeben sich ganz neue Herausforderung. Beispielsweise wie das Zusammenspiel zwischen Staukontrolle und adaptiver Request-Response-Kommunikation, die zwischen Multicast und Unicast wechselt, ausgestaltet werden kann oder wie sich Transportsystem je nach Hintergrundverkehr an die jeweilige Lastsituation im Netz anpassen kann.

Für den *Verdeckenden Ungleichheitsnachweis* wäre es sicher herausfordernd, die aktuellen Entwicklungen im Bereich von gitterbasierten Hinterlegungsverfahren aufzugreifen und passend um einen Ungleichheitsnachweis zu ergänzen. Allerdings sind nach bestem Wissen derzeit noch keine effizienten Verfahren hierfür absehbar. Des Weiteren könnte in weiterführenden Arbeiten auch die Effizienz des bisherigen Verfahrens untersucht oder der Nachweis stärkerer Sicherheitseigenschaften angestrebt werden.

Im Rahmen dieser Arbeit konnte die Konstruktion von konkreten CMS-unabhängigen Monitoring-Mechanismen nur am Rande behandelt werden. Zukünftige Arbeiten können sich daher dem Thema der Ausgestaltung von Monitoring-Mechanismen mit Monitoring-Unabhängigkeit intensiver widmen. Allerdings sind hier nur auf einzelne Anwendungsfälle bezogene Fortschritte zu erwarten. Eine Generalisierung von CMS-unabhängigen Monitoring-Mechanismen auf alle denkbaren Anwendungsfälle scheint derzeit nicht möglich.

Als Resümee lässt sich am Ende zusammenfassend festhalten, dass die in dieser Dissertation vorgestellten Monitoring-Bausteine einen wichtigen ersten Beitrag für CMS-unabhängiges Monitoring von Cloud-Umgebungen und Transparenz aus Kundensicht liefern.

Anhang A

SECCRIT Fallstudien

Im Rahmen dieser Dissertation wurden im EU-Projekt *SECCRIT* [300] verschiedene Fallsstudien und Demonstrationen von Cloud-Monitoring in realistischen Cloud-Umgebungen durchgeführt. Diese werden im Folgenden kurz beschrieben.

A.1 Getrennte physische Cloud-Knoten & Isolation von VMs

Die hier vorgestellten Ergebnisse werden in [TT-10] detailiert beschrieben. Der Anwendungsfall gestaltet sich dabei wie folgt.

Monitoringgegenstand

Ein Kunde richtet mit Hilfe des CMS die VMs `Master`, `HotStandbyMaster` und `Database` in der Cloud-Umgebung eines Cloud-Anbieters ein. Zwischen Kunde und Cloud-Anbieter wird ferner vereinbart, dass die VM `Master` und `HotStandbyMaster` nicht auf dem selben physischen Cloud-Knoten ausgeführt werden dürfen (*anti-affinity*). Ferner wird vereinbart, dass für die VM `Database` sichergestellt werden muss, dass der physische Cloud-Knoten, der für die Ausführung verantwortlich ist, keine VMs anderer Kunden ausführt (*isolation*). Mit Hilfe von *CloudInspector* soll gezeigt werden, dass diese beiden Anforderungen zur Laufzeit überprüfbar sind.

In der Kundenoberfläche des CMS kann der Kunde zwar seine VMs verwalten, allerdings kann er hier nicht einsehen, auf welchen physischen Cloud-Knoten diese ausgeführt

werden oder ob die physischen Cloud-Knoten, welche für die Ausführung verantwortlich sind, weitere VMs anderer Kunden ausführen. Vergleiche hierzu Abbildung A.1.

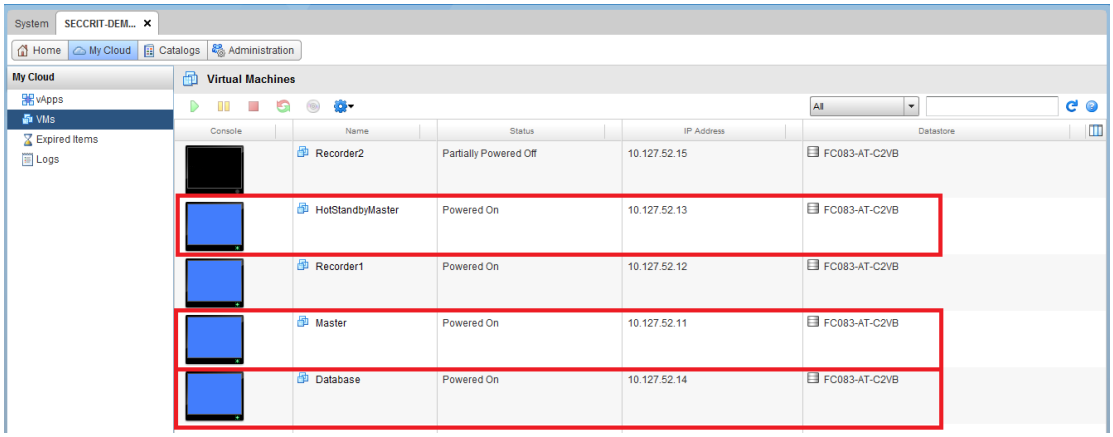

Abbildung A.1: *Überblick des CMS für Kunden auf die VMs innerhalb der Cloud-Umgebung. Quelle [TT-10].*

Allerdings liefert das CMS dem Cloud-Anbieter eine ausführlichere Sicht auf die Cloud-Umgebung als den Kunden. So kann der Cloud-Anbieter einsehen, auf welchen physischen Cloud-Knoten sich die VMs `Master`, `HotStandbyMaster` und `Database` derzeit befinden und ob VMs anderer Kunden auf den selben physischen Cloud-Knoten ausgeführt werden.

So liefert das CMS dem Cloud-Anbieter die Information, dass die VM `Master` derzeit auf dem physischen Cloud-Knoten `atviectesx090` ausgeführt wird – siehe Abbildung A.2.

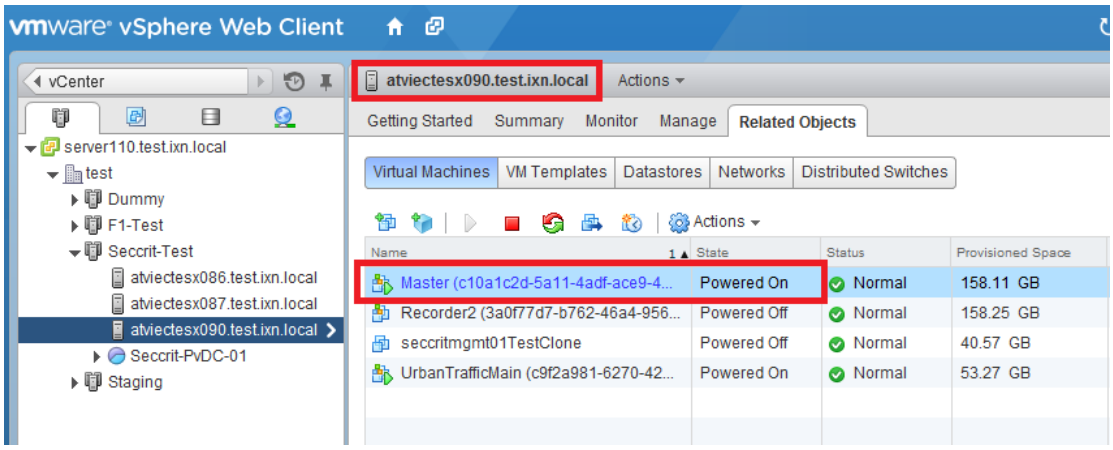

Abbildung A.2: *Sicht der Cloud-Anbieters beziehungsweise des Cloud-Management-Systems auf die Cloud-Infrastruktur. Hier für den physischen Cloud-Knoten `atviectesx090`. Quelle [TT-10].*

Außerdem lässt sich durch den Cloud-Anbieter einsehen, dass die VM `HotStandbyMaster` derzeit auf dem physischen Cloud-Knoten `atviectesx086` ausgeführt wird – siehe Abbildung A.3.

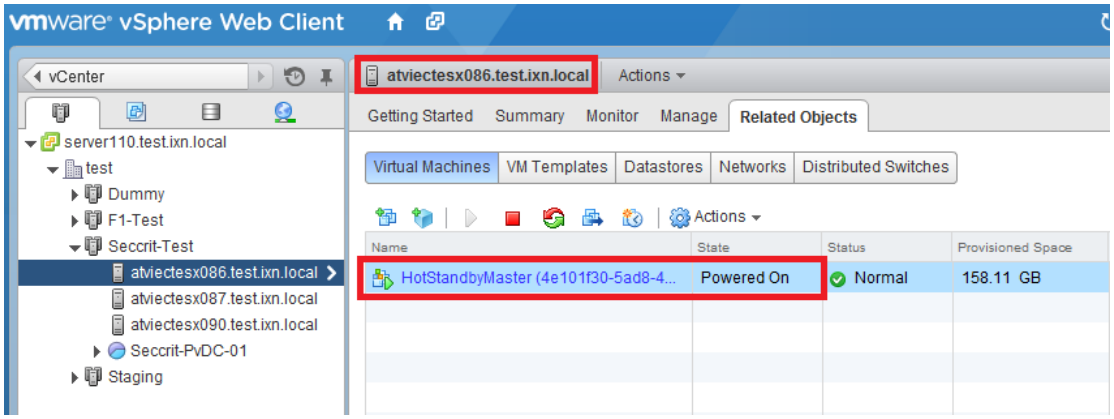

Abbildung A.3: *Detailansicht des Cloud-Management-Systems für den physischen Cloud-Knoten `atviectesx086`. Quelle [TT-10].*

Darüber hinaus wird die VM `Database` derzeit auf dem physischen Cloud-Knoten `atviectesx087` ausgeführt auf dem sich keine VMs anderer Kunden befinden.

Cloud-Monitoring

Da der Kunde keinen Zugriff auf die Sicht des Cloud-Anbieters hat, ist der Einsatz von *CloudInspector* notwendig um die Überprüfung von vertraglichen Vereinbarungen durch Kunden zu ermöglichen. Hinzu kommt, dass *CloudInspector* die Monitoringdaten unabhängig vom CMS erhebt und somit nicht durch Fehlfunktion des selbigen betroffen ist. In Abbildung A.4 wird die grafische Benutzeroberfläche von *CloudInspector* dargestellt.

Hier kann der Kunde auswählen, für welche VMs welche vertragliche Vereinbarung überprüft werden soll. Beispielhaft wird hier *affinity*, *anti-affinity* und *isolation* angeboten.

Der Kunde fordert nun eine Überprüfung auf *anti-affinity* für die VMs `Master` und `HotStandbyMaster` an – siehe Abbildung A.5.

Wie aus der Abbildung A.5 ersichtlich wird, liefert *CloudInspector* dem Kunden als Ergebnis zurück, dass die Cloud-Umgebung die von ihm geprüfte vertragliche Eigenschaft derzeit einhält und das somit die VMs auf unterschiedlichen physischen Cloud-Knoten ausgeführt werden. Bemerkenswert ist allerdings, dass der Kunde von *CloudInspector* keine weiteren Details wie beispielsweise die Hostnamen der physischen Cloud-Knoten

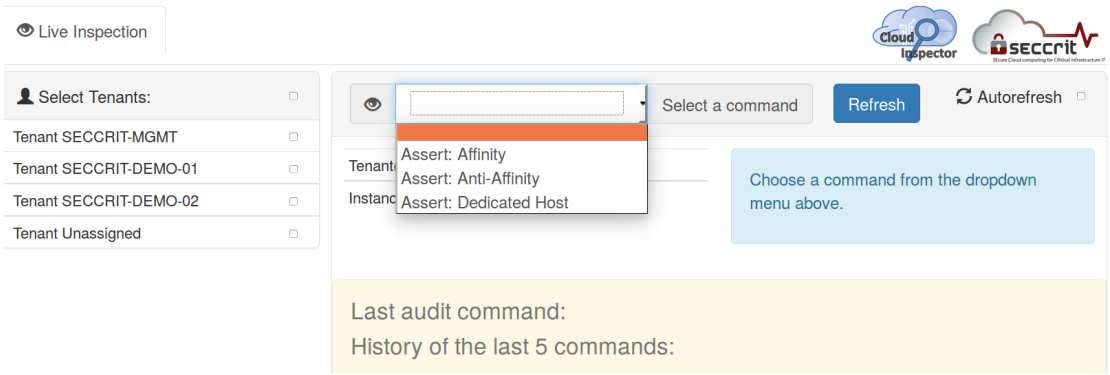

Abbildung A.4: *Grafische Oberfläche des CloudInspectors für Kunden. Die Kunden können zu überprüfende VMs und Art der Überprüfung auswählen. Quelle [TT-10].*

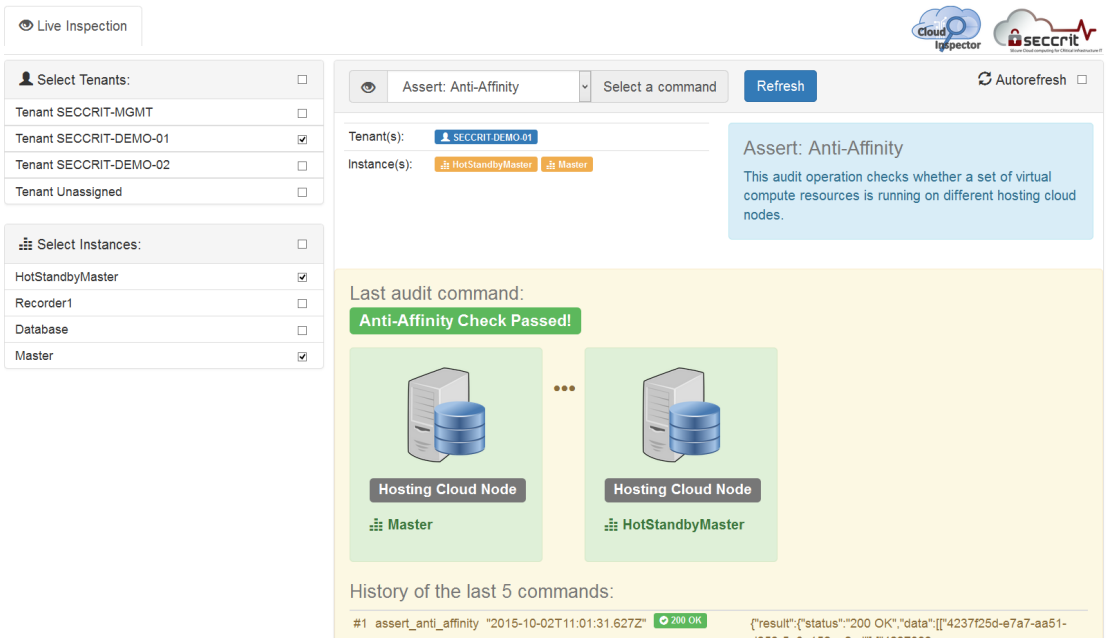

Abbildung A.5: *Ergebnis der Überprüfung der vertraglichen Vereinbarung anti-affinity für die VM* `Master` *und* `HotStandbyMaster` *durch CloudInspector. Die Überprüfung ist erfolgreich. Quelle [TT-10].*

ausgeliefert bekommt – dies dient vor allem dem Schutz von Betriebsgeheimnissen des Cloud-Anbieters.

Als zweite Überprüfung fragt der Kunde über die grafische Oberfläche von *CloudInspector* an, ob die VM `Database` derzeit auf einem physischen Cloud-Knoten ausgeführt wird, auf dem keine VMs anderer Kunden vom CMS zur Ausführung gebracht wurden. Hierzu wählt der Kunde die Überprüfung auf *isolation* in der Kundenoberfläche von *CloudInspector* aus – siehe Abbildung A.6.

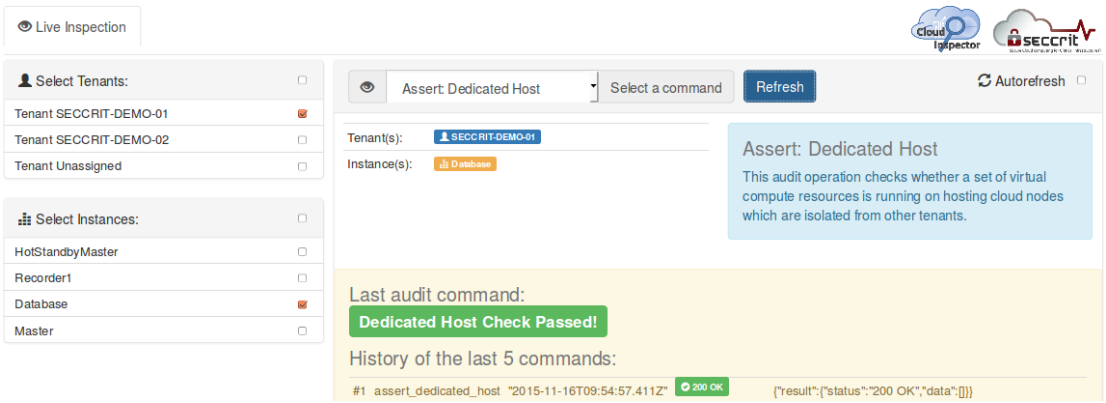

Abbildung A.6: *Ergebnis der Überprüfung der vertraglichen Vereinbarung isolation für die VM* `Database` *durch CloudInspector. Die Überprüfung ist erfolgreich. Quelle [TT-10].*

Auch diese Überprüfung läuft erfolgreich und *CloudInspector* informiert den Kunden über die Einhaltung der vertraglichen Vereinbarung. Wie beim ersten Beispiel werden auch hier keine Betriebsgeheimnisse an den Kunden ausgeliefert.

Zusammenfassend lässt sich festhalten, dass durch *CloudInspector* neue Möglichkeiten zum Monitoring von Cloud-Umgebungen für Kunden geschaffen werden. Während das CMS des Cloud-Anbieters Internas der Cloud-Umgebung vor Kunden versteckt, ermöglicht *CloudInspector* die CMS-unabhängige Überprüfung von vertraglichen Vereinbarungen.

A.2 Hot-Spare VM

Die hier vorgestellten Ergebnisse werden in [TT-10] detailiert beschrieben. Als Anwendungsfall wird die Absicherung einer VM mittels einer Hot-Spare VM untersucht.

Monitoringgegenstand

In diesem Fall hat ein Kunde mit dem Cloud-Anbieter vereinbart, dass im Falle eines Ausfalls der VM `MASTER_SERVER_ABRP-active_instance` die Netzonfiguration auf die VM `MASTER_SERVER_ABRP-backup_instance` umgezogen wird (vergleichbar mit einer Hot-Standby-Lösung). Die Synchronisation des internen, durch `MASTER_SERVER_ABRP-active_instance` bereitgestellten, Dienstes mit `MASTER_SERVER_ABRP-backup_instance` wird dabei nicht von der Cloud-Umgebung übernommen, sondern der Kunde setzt hier ein manuelles Setup auf. Weiterhin hat der

Kunde mit dem Cloud-Anbieter vereinbart, dass die beiden VMs nicht auf dem selben physischen Cloud-Knoten ausgeführt werden sollen. Auf diese Art und Weise soll sichergestellt werden, dass, falls eine Störung des physischen Cloud-Knotens die Ursache des Ausfalls von `MASTER_SERVER_ABRP-active_instance` ist, nicht gleichermaßen die Hot-Spare VM `MASTER_SERVER_ABRP-backup_instance` ausfällt.

Cloud-Monitoring

Da diese Hot-Spare Anforderung lediglich über die Kundenoberfläche des CMS konfiguriert wurde, lassen sich hierzu auf den physischen Cloud-Knotens keine Monitoringdaten erheben. Der geplante Umzug der Netzonfiguration im Fall eines Ausfalls der VM `MASTER_SERVER_ABRP-active_instance` wird von der Orchestrierungskomponente des CMS verwaltet – bei OpenStack ist dies beispielsweise die Komponente *Heat* [267]. Aus diesem Grund ist zur Überprüfung dieser vertraglichen Vereinbarung nicht allein der Abruf von Monitoringdaten eines physischen Cloud-Knoten ausreichend, sondern es werden weitere Informationen aus der Orchestrierungskomponente benötigt, um zu überprüfen, ob die vertragliche Vereinbarung eingehalten wird.

Dies verletzt zwar einerseits die Anforderung nach Unabhängigkeit vom CMS. Andererseits ist alleine die Orchestrierungskomponente des CMS dafür verantwortlich, im Fehlerfall die Netzkonfiguration anzupassen. Daher wird in diesem Fall auf Informationen der Orchestrierungskomponente (hier *Heat*) zugegriffen und überprüft, wie diese sich im Falle eines Ausfalls der VM `MASTER_SERVER_ABRP-active_instance` verhalten würde. In Abbildung A.7 wird gezeigt, wie *CloudInspector* den Kunden über die Einhaltung der überprüften Hot-Spare-Anforderung informiert.

In diesem Anwendungsfall konnte gezeigt werden, dass mit Hilfe von *CloudInspector* selbst komplexe Überprüfungen wie *Hot-Spare* durchgeführt werden können. Zwar war hier eine Aufweichung der Anforderung nach Unabhängigkeit vom CMS notwendig, dennoch hält sich die Anpassung noch in einem vertretbaren Rahmen. So wird der Zugriff auf die Orchestrierungskomponente des CMS auf das Notwendigste beschränkt – die Informationen über die aktuelle Platzierung der VMs wird immer noch auf den physischen Cloud-Knoten erhoben.

A.3 Pflichtverletzung & Fehler-Ursachen-Analyse

In diesem Abschnitt wird das Beispiel zur Aufzeichnung von Protokollierungsdaten aus [TT-10] Abschnitt 4.7 “Test Case TC-009 - Legal evidence provision” und [TT-11] Abschnitt

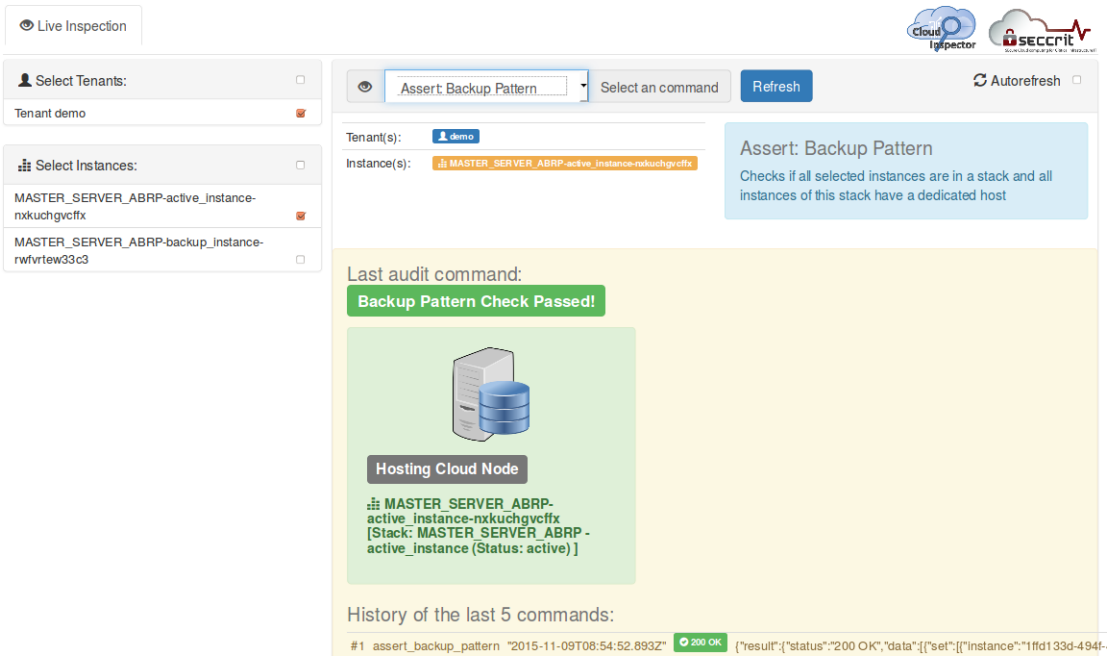

Abbildung A.7: *Ergebnis der Überprüfung durch CloudInspector auf Einhaltung der Hot-Spare-Anforderung. Quelle [TT-10].*

4.1.3.6 “TC-009 Audit Trail provision for Root Cause Analysis in court” zusammengefasst.

Monitoringgegenstand

Als Beispiel soll gezeigt werden, dass sich Verletzungen der Sorgfaltspflicht des Cloud-Anbieters durch den Einsatz von *CloudInspector* aus Kundensicht vor Gericht belegen lassen Dies betrifft in diesem Beispiel vor allem die Sorgfaltspflicht des Cloud-Anbieters, die VMMs auf den physischen Cloud-Knoten auf die jeweils aktuellste Version zu bringen. Beispielsweise weisen die Versionen von *VMware ESXi* vor dem Stand *ESXi 5.5 Update 3a* (Versionsnummer 2403361) die Schwachstelle auf, das beim Löschen von Speicherabbildern einer VM, diese unbeabsichtigt beendet wird (siehe KB2133118 [275]).

Angenommen der Cloud-Anbieter betreibt nun physische Cloud-Knoten auf einer älteren und somit anfälligen Versionsnummer, beispielsweise bei *ESXi 5.5 Update 3* die Versionsnummer 3029944, dann kann es zu unbeabsichtigten Störungen bei der Ausführung von VMs kommen – nämlich dann, wenn diese auf dem anfälligen physischen Cloud-Knoten ausgeführt werden und der Kunde ein Speicherabbild vom CMS löschen lässt. Ohne *CloudInspector* ist es für den Kunden im Falle einer rechtlichen Auseinandersetzung

vor Gericht nahezu unmöglich die Pflichtverletzung des Cloud-Anbieters nachzuweisen. Wurde vom Kunden allerdings die Aufzeichnung von Monitoringdaten konfiguriert und entsprechende Protokollierungsdaten bei einer vertrauenswürdigen dritten Partei hinterlegt, so sollte aus den Protokollierungsdaten klar werden, zu welchem Zeitpunkt welche VM auf welchem physischen Cloud-Knoten ausgeführt wurde und mit welcher Versionsnummer die VMMs auf den jeweiligen physischen Cloud-Knoten zum Zeitpunkt der Ausführung betrieben wurden.

Im Folgenden wird gezeigt, wie genau dieser Nachweis durch die Aufzeichnung von Monitoringdaten durch *CloudInspector* für eine Fehler-Ursachen-Analyse nützlich ist.

Konfiguration zur Aufzeichnung durch den Kunden

In diesem Beispiel konfiguriert der Kunde die Aufzeichnung der Versionsnummer des VMMs der physischen Cloud-Knoten, auf welchem physischen Cloud-Knoten welche VM platziert wird und ob Speicherabbilder (sogenannte *Snapshots*) erstellt oder gelöscht werden. Auf diese Art und Weise werden kontinuierlich Metadaten über das Cloud-Verhalten bei der vertrauenswürdigen dritten Partei hinterlegt und stehen für Fehler-Ursachen-Analysen vor Gericht zur Verfügung.

Beispielablauf

Der Kunde betreibt eine VM mit dem Namen *CrashVM* in der Cloud-Umgebung des Cloud-Anbieters. Der Cloud-Anbieter hat einen Teil der physischen Cloud-Knoten auf die aktuellste Version des VMMs aktualisiert (Versionsnummer 2403361). Ein Teil der physischen Cloud-Knoten ist allerdings anfällig für die Schwachstelle KB2133118 (Versionsnummer 3029944) – hiermit verletzt der Cloud-Anbieter seine Sorgfaltspflicht.

Interaktion des Kunden

Während der Kunde in der Oberfläche des CMS die VM verwalten kann, hat er keine Information darüber, ob die VM derzeit auf einem aktualisierten oder anfälligen physischen Cloud-Knoten ausgeführt wird. Abbildung A.8 zeigt die Sicht des Kunden auf die VM *CrashVM*.

Allerdings macht Abbildung A.9 deutlich, dass der Cloud-Anbieter einsehen kann, auf welchem physischen Cloud-Knoten die VM derzeit ausgeführt wird und mit welcher

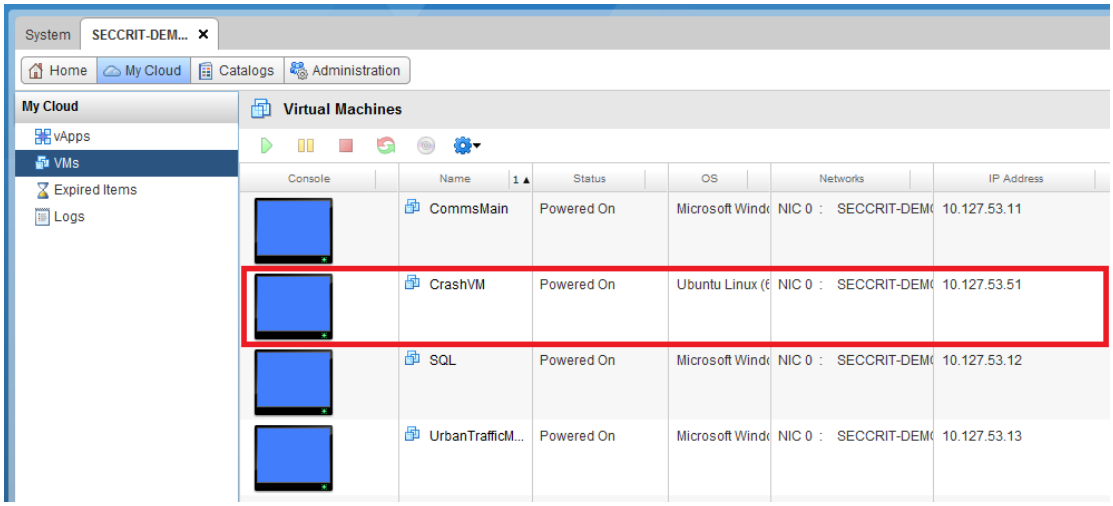

Abbildung A.8: *Beispielhafte Kundenoberfläche des CMS mit einer Übersicht aller derzeit ausgeführten VMs. Ob diese auf einem anfälligen physischen Cloud-Knoten ausgeführt werden ist nicht ersichtlich. Quelle [TT-10].*

Versionsnummer der VMM betrieben wird. In diesem Beispiel mit der Versionsnummer 2403361, also der nicht anfälligen Variante.

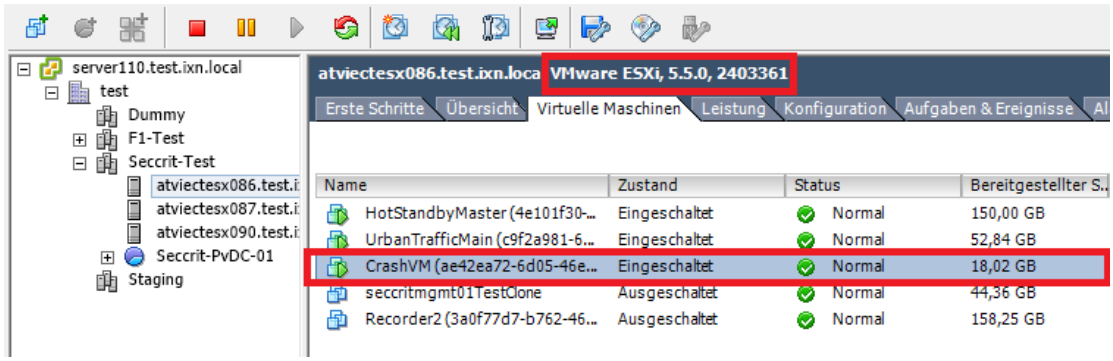

Abbildung A.9: *Oberfläche des CMS für den Cloud-Anbieter. Hier sind mehr Details sichtbar, unter anderem auf welchem physischen Cloud-Knoten die VMs ausgeführt werden und mit welcher Versionsnummer der Virtual-Machine-Monitor betrieben wird. Quelle [TT-10].*

Erstellt nun der Kunde ein Speicherabbild der VM *CrashVM* (siehe Abbildung A.10) und löscht dieses Speicherabbild anschließend wieder (siehe Abbildung A.11), so wird der Betrieb der VM dadurch wie erwartet nicht beeinträchtigt.

Wird nun allerdings die VM *CrashVM* auf einen anfälligen physischen Cloud-Knoten migriert, beispielsweise weil der Cloud-Anbieter Wartungsarbeiten vornehmen möchte, so kann es bei der Erstellung von Speicherabbildern zu Fehlern kommen. Abbildung A.12

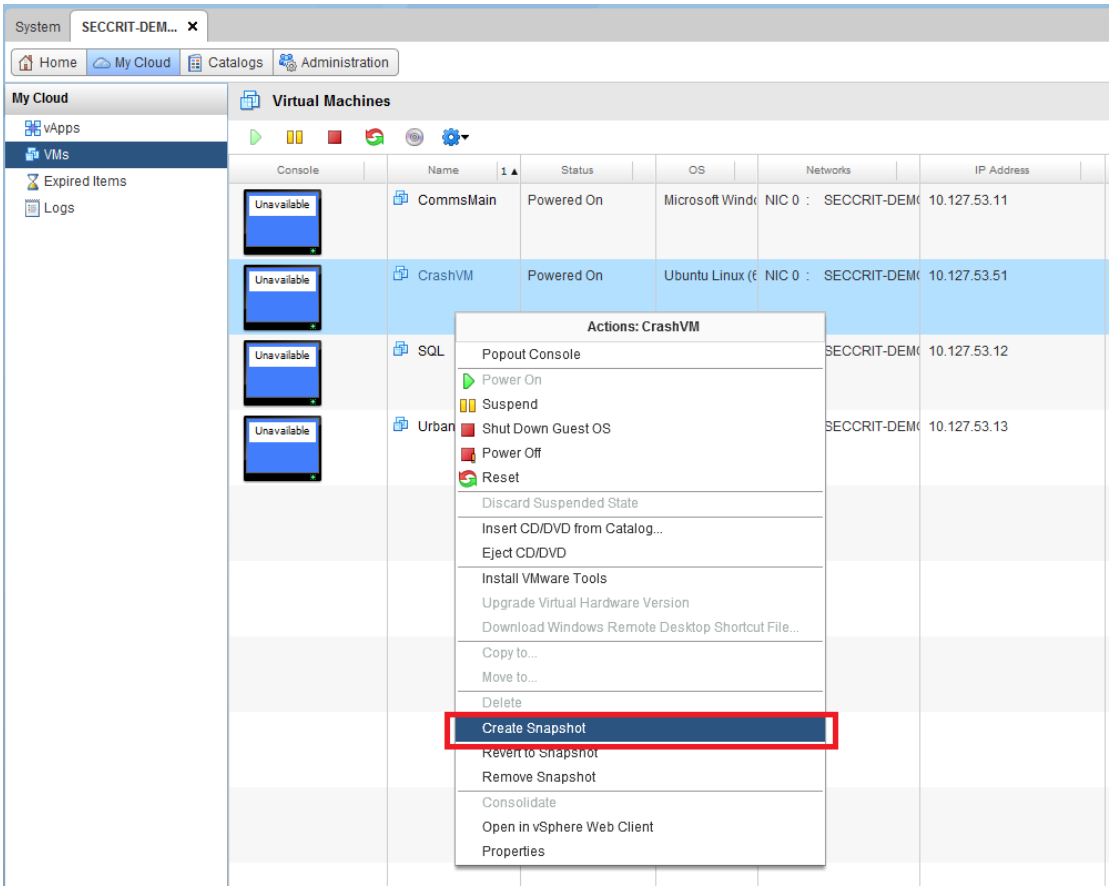

Abbildung A.10: *Kundenoberfläche des CMS beim Erstellen von Speicherabbildern (Snapshots). Hier am Beispiel der VM CrashVM. Quelle [TT-10].*

zeigt, dass die VM *CrashVM* auf den physischen Cloud-Knoten `atviectesx090` migriert wurde, dessen Version 3029944 des VMMs anfällig für die Schwachstelle KB2133118 ist. Der Kunde wiederholt nun das Erstellen eines Speicherabbilds für die VM CrashVM und das anschließende Löschen des erstellen Speicherabbilds. Anders als beim ersten Versuch stürzt dabei die VM wider Erwarten ab und es kommt zu Beeinträchtigungen bei der vom Cloud-Anbieter für den Kunden versprochenen Dienstleistung.

Fehler-Ursachen-Analyse

Initiiert der Kunde wegen der nicht erbrachten Dienstleistung einen Rechtsstreit mit dem Cloud-Anbieter, so können die von *CloudInspector* bei einer vertrauenswürdigen dritten Partei hinterlegten Protokollierungsdaten zur Fehler-Ursachen-Analyse vor Gericht verwendet werden. Dabei garantiert die vertrauenswürdige dritte Partei, dass die

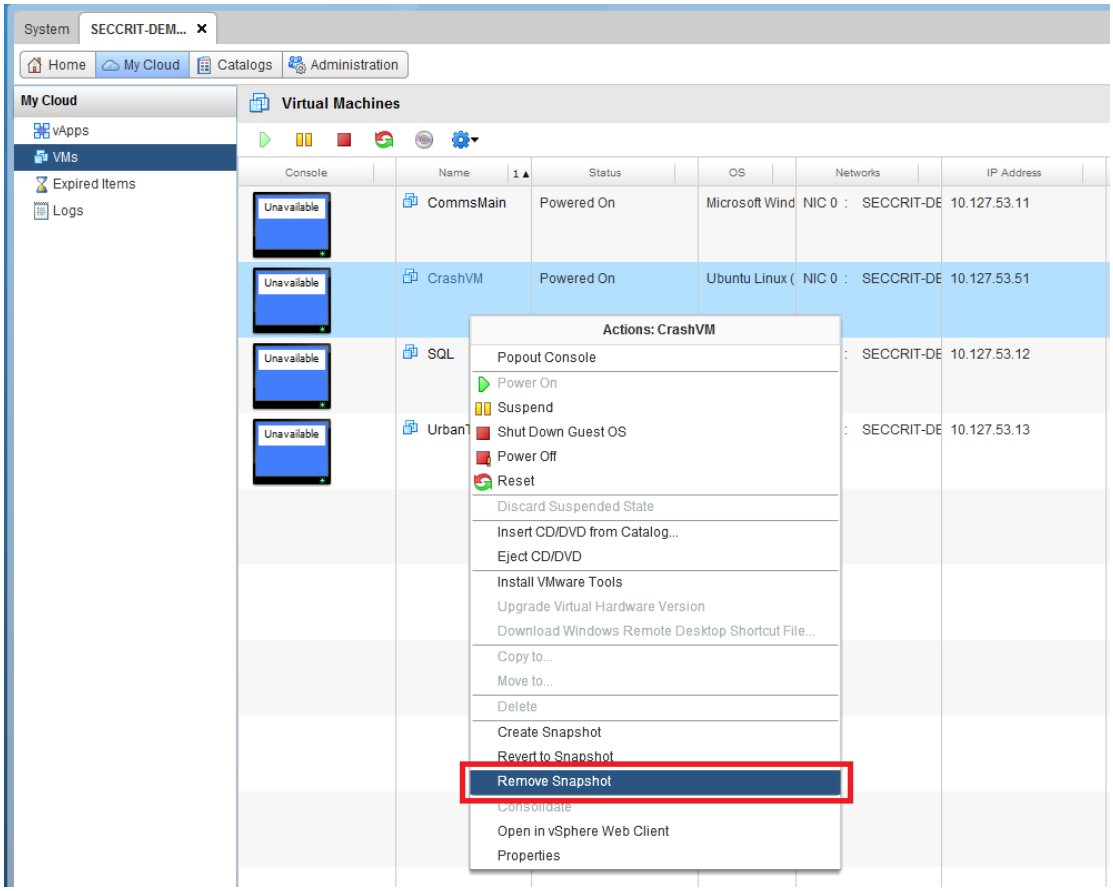

Abbildung A.11: *Kundenoberfläche des CMS beim Löschen von Speicherabbildern (Snapshots). Hier am Beispiel der VM CrashVM. Quelle [TT-10].*

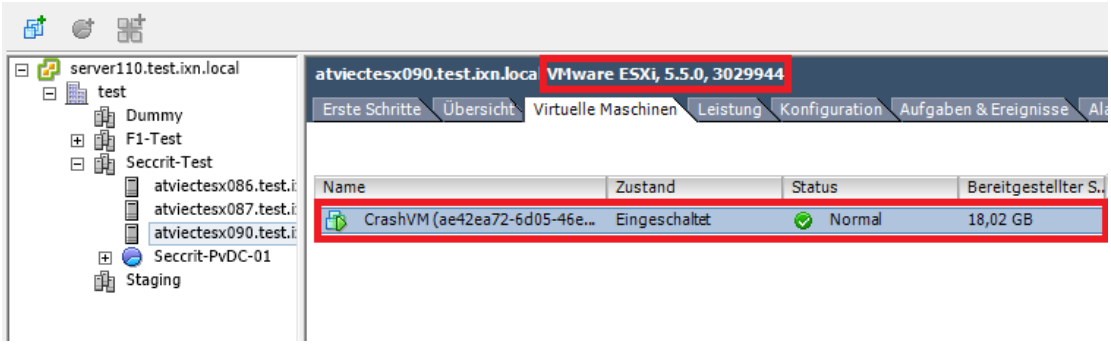

Abbildung A.12: *Cloud-Anbieter-Oberfläche des CMS nach Migration der VM CrashVM. Für den Cloud-Anbieter würde ersichtlich, dass die VM CrashVM derzeit auf einem anfälligen (Versionsnummer 3029944) physischen Cloud-Knoten ausgeführt wird. Für einen Kunden ist diese Information nicht einsehbar. Quelle [TT-10].*

protokollierten Daten verfügbar und unverändert sind. In der Regel wird die Fehler-Ursachen-Analyse vor Gericht von einem Sachverständigen durchgeführt, dem eine nicht aktualisierte Version des VMMs auffällt, beziehungsweise der die Schwachstelle KB2133118 kennt.

Zunächst analysiert der Sachverständige die von *CloudInspector* bei der vertrauenswürdigen dritten Partei hinterlegten Protokollierungsdaten für den physischen Cloud-Knoten `atviectesx086`. Abbildung A.13 zeigt einen Ausschnitt dieser Protokollierungsdaten. Hier ist keine Pflichtverletzung des Cloud-Anbieters und keine Fehler-Ursache für den Absturz der VM CrashVM erkennbar. Der physische Cloud-Knoten wird mit der aktuellen Version 2403361 des VMMs betrieben.

```
[...]

2015-11-23, 11:04:42.157556, TEM atviectesx086, TENANT system, MSG [CHECK_BUILD_NO]
    build number 2403361 did not change

2015-11-23, 11:10:25.994848, TEM atviectesx086, TENANT system, MSG [VM_PLACEMENT]
    virtual machine CrashVM of tenant SECCRIT-DEMO-02 instantiated

2015-11-23, 11:30:23.129523, TEM atviectesx086, TENANT system, MSG [SNAPSHOT]
    New Snapshot (Nr. 1) for virtual machine CrashVM

2015-11-23, 11:33:41.327488, TEM atviectesx086, TENANT system, MSG [SNAPSHOT]
    Removed Snapshot (Nr. 1) for VM CrashVM

2015-11-23, 12:04:42.157556, TEM atviectesx086, TENANT system, MSG [CHECK_BUILD_NO]
    build number 2403361 did not change

[...]
```

Abbildung A.13: *Ausschnitt der Protokollierungsdaten für den physischen Cloud-Knoten `atviectesx086`. Keine Pflichtverletzung erkennbar. Quelle [TT-10].*

Als nächstes analysiert der Sachverständige die Protokollierungsdaten für den physischen Cloud-Knoten `atviectesx090`. Hier wird ersichtlich, dass die VM CrashVM auf den physischen Cloud-Knoten migriert wurde und dass anschließend ein Speicherabbild erstellt und gelöscht wurde. Außerdem wird ersichtlich, dass der VMM des physischen Cloud-Knotens nicht mit der aktuellen Version betrieben wird, sondern mit der Versionsnummer 3029944. Abbildung A.14 zeigt einen Ausschnitt dieser Protokollierungsdaten.

Hieraus kann der Sachverständige schließen, dass der physische Cloud-Knoten zum Zeitpunkt der Speicherabbilderstellung (und Löschung) anfällig für die Schwachstelle KB2133118 war. Daher liegt der Verdacht nahe, dass dies die Ursache für den Absturz der VM CrashVM war. Darüber hinaus belegen die Protokollierungsdaten die Pflichtverletzung des Cloud-Anbieters, da der VMM des physischen Cloud-Knotens nicht mit der aktuellen Version betrieben wird.

```
[...]

2015-11-23, 19:36:30.154961, TEM atviectesx090, TENANT system, MSG [CHECK_BUILD_NO]
    build number 3029944 did not change

2015-11-23, 19:40:11.650185, TEM atviectesx090, TENANT system, MSG [VM_PLACEMENT]
    virtual machine CrashVM of tenant SECCRIT-DEMO-02 migrated

2015-11-23, 19:45:31.211011, TEM atviectesx090, TENANT system, MSG [SNAPSHOT]
    New Snapshot (Nr. 2) for virtual machine CrashVM

2015-11-23, 19:48:24.049438, TEM atviectesx090, TENANT system, MSG [SNAPSHOT]
    Removed (Nr. 2) for VM CrashVM

2015-11-23, 20:36:30.154961, TEM atviectesx090, TENANT system, MSG [CHECK_BUILD_NO]
    build number 3029944 did not change

[...]
```

Abbildung A.14: *Ausschnitt der Protokollierungsdaten für den physischen Cloud-Knoten* `atviectesx090`*. Beleg für die Pflichtverletzung des Cloud-Anbieters, der physische Cloud-Knoten wird nicht auf der aktuellsten Version 2403361 betrieben. Außerdem wird die Fehler-Ursache für den Absturz der VM CrashVM ersichtlich – die Schwachstelle KB2133118 wurde durch Erstellen und Löschen eines Speicherabbilds ausgelöst. Quelle [TT-10].*

A.4 Geo-Lokation von VMs

In diesem Abschnitt wird der Fall “TC-008 - Geolocation of sensitive data” aus [TT-10] Abschnitt 3.3 und [TT-11] Abschnitt 4.1.1.3 “TC-008 Asserting Right of Access (Data Protection Law) - Geolocation of personal data’ zusammengefasst.

Monitoringgegenstand

Als Beispiel soll gezeigt werden, wie mit Hilfe von *CloudInspector* das im BDSG vorgesehene Auskunftsrecht von Kunden gestärkt werden kann. Hierfür wird angenommen, ein Kunde hat eine Datenbank mit personenbezogenen Daten in die Cloud-Umgebung ausgelagert. Mit dem Cloud-Anbieter hat der Kunde dabei vereinbart, dass die VM *Database*, in der die Daten der Datenbank abgelegt wurden, nur auf physischen Cloud-Knoten in Europa ausgeführt werden darf, so dass europäisches Datenschutzrecht gilt und im Zweifelsfall der Ort der Jurisdiktion geklärt ist. Außerdem soll so sichergestellt werden, dass nicht-europäische Nachrichtendiensten keinen Zugriff auf die Daten der VM *Database* erhalten. Der Cloud-Anbieter verfügt allerdings über mehrere Datenzentren in und außerhalb der Europäischen Union.

Ohne *CloudInspector* hat der Kunde in diesem Szenario keinen Zugriff auf die Lokation des physischen Cloud-Knotens der die VM *Database* zur Verfügung stellt. In der Kundenoberfläche des CMS wird lediglich angezeigt, ob die VM aktiv ist. Vergleiche dazu Abbildung A.15

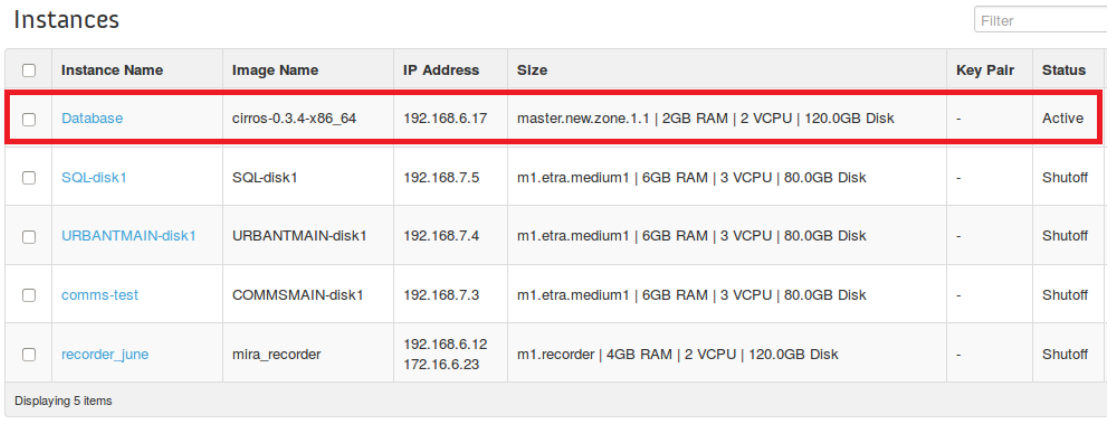

Abbildung A.15: *Kundenoberfläche des Cloud-Management-Systems auf die VM Database. Der Kunde kann nicht überprüfen in welchem Land sich der für die Ausführung der VM verantwortliche physische Cloud-Knoten befindet. Quelle [TT-10].*

In der Kundenoberfläche wird weder ersichtlich auf welchem physischen Cloud-Knoten die VM *Database* ausgeführt wird, noch in welchem Land sich der physische Cloud-Knoten befindet. Der Kunde kann so nicht überprüfen ob seine vertragliche Vereinbarung eingehalten wird.

Der Cloud-Anbieter kann hingegen in seiner Oberfläche des CMS zumindest einsehen auf welchem physischen Cloud-Knoten die VM *Database* ausgeführt wird. Vergleiche dazu Abbildung A.16.

Allerdings kann der Anbieter auch nicht direkt entnehmen in welchem Datenzentrum sich der jeweilige physische Cloud-Knoten befindet. Hierzu kann er aber im Gegensatz zum Kunden weitere Nachforschungen anstellen.

Cloud-Monitoring

Wird nun durch *CloudInspector* ein Monitoring-Mechanismus zur Überprüfung der geografischen Lokation einer VM zur Verfügung gestellt (hier innerhalb der Europäischen Union), kann der Kunde eigenständig überprüfen ob seine vertragliche Vereinbarungen

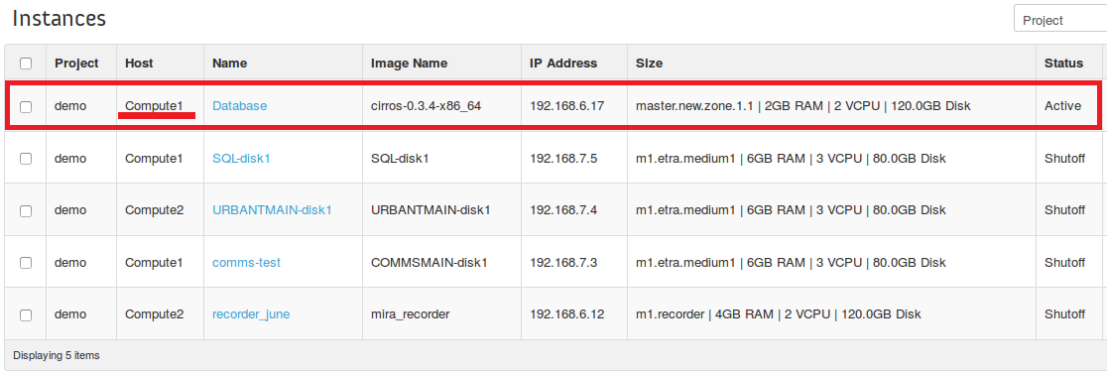

Abbildung A.16: *Oberfläche des Cloud-Management-Systems für den Cloud-Anbieter. Der zur VM gehörende physische Cloud-Knoten ist sichtbar, der Ort allerdings nicht direkt. Quelle [TT-10].*

mit dem Cloud-Anbieter erfüllt ist. Außerdem wird so das Auskunftsrecht des Kunden gestärkt, da dieser nun selbst steuern kann wann eine Überprüfung stattfindet.

Zur Realisierung des Anwendungsfalls wird hier zunächst davon ausgegangen, dass auf jedem physischen Cloud-Knoten die entsprechende Lokation hinterlegt ist. Dies ist zwar zunächst eine ziemlich stringente Forderung, allerdings kann der Cloud-Anbieter bei der Installation von *CloudInspector* gleich mit dafür sorgen, dass diese Forderung erfüllt ist. In aller Regel sind die physischen Cloud-Knoten in einem Datenzentrum auch nicht mobil und wechseln die Örtlichkeit, so dass diese Konfiguration nur einmal vorgenommen werden muss. Soll nicht auf jeden physischen Cloud-Knoten die entsprechende Lokation hinterlegt werden, so muss im Datenzentrum des Cloud-Anbieters ein Lokationsdienst zur Verfügung gestellt werden, den *CloudInspector* dann bezüglich der Lokation einzelner physischen Cloud-Knoten anfragen kann.

In Abbildung A.17 ist die Oberfläche von *CloudInspector* zur Überprüfung der Geolokation dargestellt. Bemerkenswert ist dabei, dass *CloudInspector* über die Oberfläche zwar das Ergebnis der Überprüfung liefert, aber keine genauen Daten über die Lokation an den Kunden übermittelt. Dies dient in erster Linie dem Schutz des Cloud-Anbieters, denkbar wäre etwa, dass ein Konkurent die vorgestellte Schnittstelle zum ausspionieren der geografischen Verbreitung seines Mitbewerbers nutzen würde.

Mit Hilfe dieses Monitoring-Mechanismus von *CloudInspector* können Kunden nun in nahezu Echtzeit überprüfen ob die vertraglich vereinbarte Geolokation von VMs eingehalten wird.

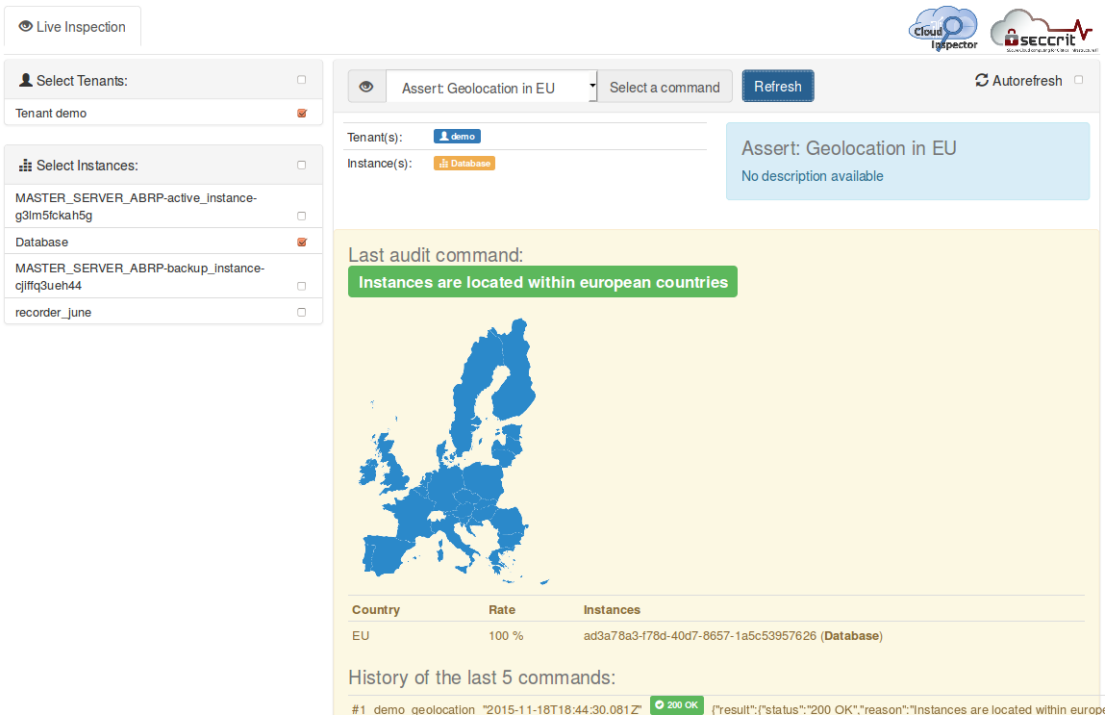

Abbildung A.17: *Die von CloudInspector zur Verfügung gestellte Oberfläche zur Überprüfung ob eine VM auf physischen Cloud-Knoten eines Datenzentrum innerhalb der Europäischen Union ausgeführt wird. Quelle [TT-10].*

Anhang B

Umfrage zu unabhängigem Cloud-Monitoring

Um eine erste Abschätzung für die Nachfrage un Akzeptanz des *CloudInspector*-Ansatzes treffen zu können, wurde in Zusammenarbeit mit dem Institut für Informationswirtschaft und Marketing [271] eine Umfrage unter potentiellen Nachfragern (Cloud-Anbietern) durchgeführt (siehe Bachelorarbeit von Rieth [TT-7]). Die Erkenntnisse der Umfrage wurden dann für eine stichprobenartige quantitative Schätzung des Absatzpotenzials von *CloudInspector* verwendet. Des Weiteren wurden die hier beschriebenen Erkenntnisse im Rahmen des EU-Projekts SECCRIT [300] verwendet.

Methodik

Da es sich bei *CloudInspector* um einen Forschungsprototypen handelt, wurde zur Analyse des Marktpotenzials zunächst ermittelt, welche Faktoren bei der Erstellung eines marktfähigen Produkts zu berücksichtigen sind. Als Anlehnung wurden hier die für Cloud-Computing typischen Faktoren betrachtet: Sicherheitsbedenken, Vorteil, Kompatibilität, Komplexität, Erprobbarkeit, Unternehmensgröße, Top-Management Unterstützung, Innovationsfähigkeit, Technologiereife, Wettbewerbsdruck und staatliche Regulierung.

Des Weiteren wurde sich bei der Marktpotenzialanalyse auf den deutschen IaaS Markt konzentriert. Hierzu wurde in der Bachelorarbeit die Marktübersicht von [255] herangezogen. Für die Marktpotenzialanalyse sollte anhand der öffentlich einsehbaren Bilanzsumme der IaaS Cloud-Anbieter (z.B. hier im Bundesanzeiger [251]) ermittelt werden, wie groß der Markt für *CloudInspector* wäre. Da der *CloudInspector* auf jedem

physischen Cloud-Knoten installiert werden muss, musste hierzu neben der öffentlich einsehbaren Bilanzsumme die für IaaS eingesetzte Serveranzahl der Cloud-Anbieter ermittelt werden. Des Weiteren spielt die generelle Bereitschaft der Cloud-Anbieter für den Einsatz von *CloudInspector* und das potentielle Interesse von Kunden eine wichtige Rolle für eine erste Abschätzung. Hinzukommt, dass die Zahlungsbereitschaft für *CloudInspector* ermittelt werden musste.

Da die benötigten Informationen nicht allesamt öffentlich einsehbar sind, wurden von den 54 für den deutschen IaaS Markt relevanten Cloud-Anbieter 41 zu telefonischen Interviews eingeladen. Von den eingeladenen Cloud-Anbietern haben sich schließen 11 an der Umfrage im Rahmen der Bachelorarbeit beteiligt, wobei von den gemachten Angaben 10 verwendbar waren. Tabelle B.1 gibt eine Übersicht der befragten Cloud-Anbieter hinsichtlich des Umsatzes, der Bilanzsumme und der Anzahl der Server, die als physischer Cloud-Knoten für IaaS eingesetzt werden.

Unternehmen	Umsatz	Bilanzsumme	Anzahl IaaS-Server
U1	1.000.000 €	146.889 €	80
U2	4.500.000 €	1.188.087 €	20
U3	3.000.000 €	4.007.558 €	75
U4	43.600.000 €	5.027.198 €	8
U5	60.000.000 €	6.410.000 €	500
U6	52.000.000 €	7.893.597 €	150
U7	30.000.000 €	8.519.499 €	60
U8	730.000 €	22.224.636 €	7.500
U9	25.000.000 €	25.064.752 €	500
U10	3.065.000.000 €	1.323.430.000 €	1.100

Tabelle B.1: *Übersicht der Unternehmen die zur Analyse des CloudInspector-Marktpotenzials befragt wurden. Entnommen aus der zu Grunde liegenden Bachelorarbeit [TT-7].*

Anhand der telefonisch ermittelten Informationen und der öffentlich einsehbaren Daten aus dem [251] wurde dann das Marktpotenzial für *CloudInspector* abgeschätzt. Für ausführlichere Details zur Methodik, wie zum Beispiel den konkreten Ablauf der Interviews, siehe Arbeit von Rieth [TT-7].

Ergebnisse

Die Ergebnisse der Analyse lassen sich in zwei Kategorien unterteilen. Zum einen in konkrete Ergebnisse aus den Telefoninterviews und zum anderen in eine Hochrechnung beziehungsweise erste Annäherung des Absatzpotenzials von *CloudInspector*.

Von den befragten 10 Cloud-Anbietern gaben fünf an, dass der *CloudInspector* für sie, beziehungsweise ihre Kunden, interessant sei und einen Mehrwert darstellen könnte. Genauso viele Cloud-Anbieter (fünf) gaben allerdings an, dass die Verwendung des *CloudInspectors* für sie vermutlich keinen Mehrwert bieten würde. Die interessierten Cloud-Anbieter schätzten dabei den Mehrwert von *CloudInspector* auf ungefähr 10% der Kosten, die üblicherweise je physischem Cloud-Knoten für ein CMS anfallen. Weiter gaben die befragten Cloud-Anbieter an (vergleiche [TT-7], S. 21ff), dass zur Nutzung des *CloudInspectors* unbedingt eine externe Zertifizierung beziehungsweise Unbedenklichkeitsbescheinigung für den Einsatz notwendig ist, da ansonsten eventuell neue Sicherheitsprobleme entstehen können. Außerdem sollen die von *CloudInspector* bei einer vertrauenswürdigen dritten Partei gelagerten Informationen über das Cloud-Verhalten durch höchste Sicherheitsstandards geschützt werden, beispielsweise durch Lagerung in einem Hochsicherheitsdatenzentrum.

Zur Abschätzung des Absatzpotenzials wurden zwei verschiedene Methoden gewählt. Zum einen wurde in der Bachelorarbeit anhand der Bilanzsumme und des Interesses innerhalb der befragten Stichprobe eine segmentbasierte Hochrechnung auf den gesamten Markt durchgeführt, zum anderen wurde eine konservative durchschnittsbasierte Schätzung vorgenommen – auch wenn die Stichprobengröße mit *10* befragten Cloud-Anbietern recht klein und auf den deutschen Markt fokussiert ist. Die segmentbasierte Hochrechnung ergibt einen theoretischen maximalen Absatz von 7.800.000 €, wohingegen die konservative durchschnittsbasierte Schätzung ein Absatzpotenzial von 2.600.000 € für *CloudInspector* sieht (vergleiche [TT-7], Anhang 9). Des Weiteren lässt sich aus den ermittelten Daten abschätzen (vergleiche [TT-7], S. 23ff), dass Kleinstunternehmen nicht als geeignete Zielgruppen für die Vermarktung für *CloudInspector* in Frage kommen, da hier die Kundenbindung traditionell sehr hoch ist, was zusätzliche Transparenzmechanismen vermutlich überflüssig macht. Dahingegen zeigt sich, dass kleine und große Cloud-Anbieter als Zielgruppe für *CloudInspector* in Frage kommen, während mittelgroße dem Einsatz kritisch gegenüberstehen.

Abschließend bleibt festzuhalten, dass die durchgeführte Stichprobe gezeigt hat, dass es gewisses Marktpotenzial und Interesse bei Cloud-Anbietern für den Einsatz des *CloudInspectors* gibt – auch wenn die Stichprobe recht klein und auf den deutschen Markt fokussiert ist. Wie groß dieses Absatzpotenzial tatsächlich ist, lässt sich allerdings nicht

abschließend klären, da dazu weitere betriebswissenschaftlich fundierte Untersuchungen angestellt werden müssten. Beispielsweise durch intensivere und weitere Interviews, durch Produktentwicklung aus dem bestehenden Prototypen und Betrachtung des europäischen Marktes. Gerade bei Unternehmen, die nicht nur allein Standorte in Deutschland haben, kommt eine Überprüfungsmöglichkeit durch *CloudInspector* bezüglich des Standorts des aktuell verwendeten Datenzentrums gänzlich zum Tragen – bei Unternehmen die ausschließlich Datenzentren in Deutschland betreiben, macht diese Überprüfungsmöglichkeit keinen Sinn. Diese und andere tiefergehende betriebswissenschaftliche Aspekte sind allerdings nicht Bestandteil dieser Dissertation. In erster Annäherung sollte hier nur gezeigt werden, dass der praktische Nutzen von *CloudInspector* durchaus von der Industrie gesehen wird.

Anhang C

Simulationsparameter für OMNeT++

Parameter	**Wert**
Simulationsaufbau	
OMNeT++ Version	4.6
INET Version	3.1.1
Simulationsrechner	AMD Opteron 6282SE CPU (32 Cores je 2.6 GHz), 514 Gigabyte Arbeitsspeicher, Ubuntu 14.04 LTS
Zufallszahlengenerator	*Mersenne Twister* [131]
Durchläufe Insgesamt	900
Allgemeine Parameterwerte	
Maximale Simulationszeit	1500s
Wiederholungen	10
Einschwingphase	40s
MLDv2 Startzeit	4s
MLDv2 Intervall	125s
Szenario Startzeit	7s
Datenrate je Link	10Gbps
Verzögerung je Link	0.3μs
Startzeit der Links	30s
Linkfehlerraten	{0, 0.0001, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.007, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5}
gAUDIT Parameter	
Initiator Startzeit	3.5s
Respondents Startzeit	3.5s
Request-Response Verfahren	{gAUDIT, Survey}
Request-Response Startzeit	40s
Request-Response Intervall	1s
Request-Response Anzahl	100
Anzahl Sendewiederholungen	{0, 1, 4, 9, 14}

Tabelle C.2: *Simulationsparameter für Parameterstudie in Abschnitt 4.9.1.*

Parameter	**Wert**
Simulationsaufbau	
OMNeT++ Version	5.4.1
INET Version	4.0.0
Simulationsrechner	AMD Opteron 6282SE CPUs (32 Cores je 2.6 GHz), 514 Gigabyte Arbeitsspeicher, Ubuntu 18.04.1 LTS
	AMD Opteron 6140 CPUs (32 Cores je 2.6 GHz), 514 Gigabyte Arbeitsspeicher, Ubuntu 18.04.1 LTS
	Intel Xeon L5420 CPU (8 Cores je 2.5 GHz), 12 Gigabyte Arbeitsspeicher, Ubuntu 18.04.1 LTS
	3x Intel Xeon X3430 CPU (4 Cores je 2.4 GHz), 12 Gigabyte Arbeitsspeicher, Ubuntu 18.04.1 LTS
Zufallszahlengenerator	*Mersenne Twister* [131] mit wählbarem Seed
Standardparameter	
Maximale Simulationszeit	100s
Wiederholungen	5
Einschwingphase	20s
Szenario Startzeit	15s
Datenrate je Link	10Gbps
Linkfehlerrate	0
Verzögerung je Link	$0.3\mu s$
Warteschlangen	Nop (Keine)
ARP	GlobalArp
PIM Mode	Dense
Respondents Startzeit	15s
Respondents Anzahl	{2..1002 step 25}
Respondent Verlust	{0, intuniform(0,2)}
Initiator Startzeit	20s
Response-Vollständigkeiten	{0.25, 0.5, 0.75, 1.0}
Request-Response Verfahren	{Unicast, Multicast, gAUDIT}
Parameterstudie zur Bestimmung von s	
W	{0..1 step 0.1}
s	{0..100 step 10}
Detailierte Parameterstudie für s im Intervall $[30; 60]$	
Wiederholungen	3
Request-Response Verfahren	gAUDIT
W	{0..1 step 0.1}
s	{30..60 step 1}

Tabelle C.4: *Simulationsparameter für Parameterstudien in Abschnitt 4.9.2.*

Parameter	**Wert**
Simulationsaufbau wie in Tabelle C.4	
Zusätzliche Standardparameter zu denen in Tabelle C.4	
Warteschlangen	DropTail
Warteschlangen Kapazität	100 Rahmen
Response-Vollständigkeit	1.0
Respondent Verlust	0
W	0 (Best-Case-Szenario)
Verteilungsfunktionen	{Gleichverteilung, Exponentialverteilung, *gAUDIT*-Gleichverteilung, *gAUDIT*-Exponentialverteilung}
Parameterstudie zum Incast-Verhalten	
s	{0..100 step 10}
Parameterwerte	{0, 10^{-13}, 10^{-12}, $2 \cdot 10^{-12}$, $3 \cdot 10^{-12}$, $4 \cdot 10^{-12}$, $5 \cdot 10^{-12}$, 10^{-11}.10^{-10}..10^{-1}}
Detailierte Parameterstudie zum Incast-Verhalten im Intervall $[10^{-8}; 10^{-6}]$	
s	50
Parameterwerte	{10^{-8}..10^{-6} step 10^{-8}}

Tabelle C.6: *Simulationsparameter für Parameterstudien in Abschnitt 4.9.3.*

Anhang D

Variablenübersicht für CoCoPAS

Variable	Verwendung / Bedeutung
Q_x	Anfrage (im englischen *Question*) Q von einem Kunden mit x als Index zur Unterscheidung verschiedener Anfragen.
V_x	Virtuelle Maschine (VM) V mit x als Index zur Unterscheidung verschiedener VMs.
N_x	Physischer Cloud-Knoten (im englischen *Node*) N mit x als Index zur Unterscheidung verschiedener Cloud-Knoten.
S_x	Monitoringdatum S mit x als Index zur Unterscheidung verschiedener Monitoringdaten.
A_x	Auditwert A mit x als Index zur Unterscheidung verschiedener Auditwerte. Dabei wird x passend zu dem im Auditwert mittels ElGamal hinterlegten Monitoringdatum S_x gewählt.
G	Diffie-Helllman-Gruppe.
g	Generator der Diffie-Helllman-Gruppe G.
h	Es gilt $h := g^x$ mit Geheimnis x.
z_1 und z_2 oder z_3 und z_4	Ergebnis der Hinterlegung mittels ElGamal. Es gehört immer ein Paar aus geradem und ungeradem Index zusammen.
r_x	Zufallswert (im englischen *random*) r der für die Hinterlegung mittels ElGamal verwendet wurde. Der Index x wird passend zum Auditwert A_x und Monitoringdatum S_x gewählt. Wird $*$ als Index verwendet, so handelt es sich um einen Zufallswert der für einen Rechenschritt benötigt wird. Bei Verwendung von `val1` im Index handelt es sich um einen speziellen Zufallswert der für den Ungleichheitsnachweis benötigt wird.
`Csign`	Signatur des Cloud-Anbieters.
`Tsign`	Signatur einer vertrauenswürdigen dritten Partei.
`Ttime`	Zeitstempel einer vertrauenswürdigen dritten Partei.
D	Berechnung des Differenz Auditwerts von per ElGamal hinterlegten Werten durch Bilden des Quotienten. Beispielsweise $D := \frac{g^{S_1} \cdot h^{r_1}}{g^{S_2} \cdot h^{r_2}} = g^{S_1 - S_2} \cdot h^{r_1 - r_2}$
i	Multiplikativ inverses Element. Hier verwendet als inverses Element zur Differenz zweier per ElGamal hinterlegten Werte. Beispielsweise i multiplikativ invers zu $S_1 - S_2$.
T, E, B und P	Hilfswerte für Zero-Knowledge-Beweise.
C_x	Zufallswert (im englischen *Challenge*) C des Kunden der an den Cloud-Anbieter für die Durchführung eines Zero-Knowledge-Beweises versandt wird. Index x zur Unterscheidung verschiedener Werte.

Tabelle D.2: *Übersicht der Variablen des Verdeckenden Ungleichheitsnachweis für Monitoringdaten.*

Arbeiten des Autors

Publikationen, Beteiligung an Projektarbeiten und betreute studentische Arbeiten.

[TT-1] M. Flittner und R. Bauer. „TREX: Tenant-driven network traffic extraction for SDN-based cloud environments“. In: *2017 Fourth International Conference on Software Defined Systems (SDS)*. Mai 2017, S. 48–53. DOI: 10.1109/SDS.2017.7939140.

[TT-2] M. Flittner, J. Scheuermann und R. Bauer. „ChainGuard: Controller-independent Verification of Service Function Chaining in Cloud Computing“. In: *IEEE Conference on Network Function Virtualization and Software Defined Networks*. Nov. 2017.

[TT-3] M. Flittner, A. Weigel und M. Zitterbart. „gAUDIT: A group communication-capable request-response middleware for auditing clouds“. In: *2017 International Conference on Networked Systems (NetSys)*. März 2017.

[TT-4] M. Roth. *Cloudinspector – Analysis and Evaluation*. Bachelorarbeit. Institut für Telematik, Karlsruhe Institut für Technologie (KIT). Betreuender Mitarbeiter: Matthias Flittner. 2017.

[TT-5] M. Flittner, S. Balaban und R. Bless. „CloudInspector: A Transparency-as-a-Service Solution for Legal Issues in Cloud Computing“. In: *IEEE International Conference on Cloud Engineering Workshop* (Apr. 2016).

[TT-6] J. Hoor. *Transparency-as-a-Service für Blockspeicherlösungen in Cloud-Umgebungen*. Masterarbeit. Institut für Telematik, Karlsruhe Institut für Technologie (KIT). Betreuender Mitarbeiter: Matthias Flittner. 2016.

[TT-7] T. Rieth. *Marktpotentialanalyse einer IT-Innovation am Beispiel des Cloud Computing-Produkts Cloud Inspector*. Bachelorarbeit. Institut für Informationswirtschaft und Marketing, Institut für Telematik, Karlsruhe Institut für Technologie (KIT). Betreuender Mitarbeiter: Maximilian Lüders, Matthias Flittner. 2016.

[TT-8] A. Weigel. *Analyse und simulationsbasierte Evaluierung von Request-Response-Verfahren in elastischer Cloud-Umgebung*. Masterarbeit. Institut für Telematik, Karlsruhe Institut für Technologie (KIT). Betreuender Mitarbeiter: Matthias Flittner. 2016.

[TT-9] R. Bless, M. Flittner, J. Horneber, A. Hudic, D. Hutchison, C. Jung, F. Pallas, M. Schöller, S. N. ul Hassan Shirazi, S. Simpson und P. Smith. *Whitepaper AF 2.0 - SECCRIT Architectural Framework*. 2015.

[TT-10] M. Matinlassi, S. Cáceres, T. Puschacher, M. Flittner, E. Sfakianakis, C. Jung, A. Zambrano, I. Chochliouros, K. Chelidonis, N. Shirazi, M. Rudolph, C. Jung und S. Oechsner. *SECCRIT Deliverable D6.2 – Demonstrators validation*. 2015.

[TT-11] A. Zambrano, S. Cáceres, P. Smith, M. Matinlassi, M. Flittner, A. Hudic, N. Shirazi, S. Balaban, C. Wagner, M. Rudolph und S. Oechsner. *SECCRIT Deliverable D6.3 – Report on validation results*. 2015.

[TT-12] R. Bauer. *Infrastruktur-Auditing als vertrauensbildende Massnahme für Cloud-Computing: Entwicklung eines unabhängigen Transparency-as-a-Service Rahmenwerks*. Diplomarbeit. Institut für Telematik, Karlsruhe Institut für Technologie (KIT). Betreuender Mitarbeiter: Matthias Flittner, Roland Bless. 2014.

[TT-13] R. Bless und M. Flittner. „Towards corporate confidentiality preserving auditing mechanisms for Clouds“. In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. Okt. 2014, S. 381–387. DOI: 10.1109/CloudNet.2014.6969025.

[TT-14] R. Bless, M. Flittner, J. Horneber, D. Hutchison, C. Jung, F. Pallas, M. Schöller, S. N. ul Hassan Shirazi, S. Simpson und P. Smith. *Whitepaper AF 1.0 SECCRIT Architectural Framework*. 2014.

Veröffentlichungen und Literatur

[1] I. M. Abbadi. „Clouds' Infrastructure Taxonomy, Properties, and Management Services“. In: *Advances in Computing and Communications*. Hrsg. von A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki und S. M. Thampi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 406–420. ISBN: 978-3-642-22726-4.

[2] M. Abe, R. Cramer und S. Fehr. „Non-interactive Distributed-Verifier Proofs and Proving Relations among Commitments“. In: *Advances in Cryptology — ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings*. Hrsg. von Y. Zheng. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, S. 206–224. ISBN: 978-3-540-36178-7. DOI: 10.1007/3-540-36178-2_13. URL: http://dx.doi.org/10.1007/3-540-36178-2_13.

[3] G. Aceto, A. Botta, W. de Donato und A. Pescapè. „Cloud monitoring: A survey“. In: *Computer Networks* 57.9 (2013), S. 2093–2115. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2013.04.001. URL: http://www.sciencedirect.com/science/article/pii/S1389128613001084.

[4] *Algorithms, Key Size and Protocols*. Report. Abgerufen am 06.12.2018. H2020-ICT-2014 – Project 645421 – ECRYPT – Coordination & Support Action, 2018. URL: http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[5] M. Alhamad, T. Dillon und E. Chang. „A survey on SLA and performance measurement in cloud computing“. In: *OTM Confederated International Conferences 'On the Move to Meaningful Internet Systems'*. Springer. 2011, S. 469–477.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta und M. Sridharan. „Data Center TCP (DCTCP)“. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM '10. New Delhi, India: ACM, 2010, S. 63–74. ISBN: 978-1-4503-0201-2. DOI: 10.1145/1851182.1851192. URL: http://doi.acm.org/10.1145/1851182.1851192.

[7] C. A. Ardagna, R. Asal, E. Damiani und Q. H. Vu. „On the Management of Cloud Non-Functional Properties: The Cloud Transparency Toolkit“. In: *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*. März 2014, S. 1–4. DOI: 10.1109/NTMS.2014.6814039.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica und M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Techn. Ber. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, Feb. 2009. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica und M. Zaharia. „A View of Cloud Computing“. In: *Commun. ACM* 53.4 (Apr. 2010), S. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: http://doi.acm.org/10.1145/1721654.1721672.

[10] S. Baker. *CORBA distributed objects: using Orbix*. ACM Press/Addison-Wesley Publishing Co., 1997.

[11] R. Baldoni und A. Virgillito. „Distributed event routing in publish/subscribe communication systems: a survey“. In: *DIS, Universita di Roma La Sapienza, Tech. Rep* 5 (2005).

[12] J. Baliga, R. W. A. Ayre, K. Hinton und R. S. Tucker. „Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport“. In: *Proceedings of the IEEE* 99.1 (Jan. 2011), S. 149–167. ISSN: 0018-9219. DOI: 10.1109/JPROC.2010.2060451.

[13] B. Ban. „JavaGroups–Group Communication Patterns in Java“. In: *Department of Computer Science Cornell University* (1998), S. 1–18.

[14] A. Beloglazov, J. Abawajy und R. Buyya. „Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing“. In: *Future Generation Computer Systems* 28.5 (2012). Special Section: Energy efficiency in large-scale distributed systems, S. 755–768. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2011.04.017. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11000689.

[15] A. Beutelspacher. *Lineare Algebra*. Springer Spektrum, 2014. ISBN: 978-3-658-02413-0.

[16] S. Bhardwaj, L. Jain und S. Jain. „Cloud computing: A study of infrastructure as a service (IAAS)“. In: *International Journal of engineering and information Technology* 2.1 (2010), S. 60–63.

[17] R. Birke, L. Y. Chen und E. Smirni. „Data Centers in the Cloud: A Large Scale Performance Study“. In: *2012 IEEE Fifth International Conference on Cloud Computing*. Juni 2012, S. 336–343. DOI: 10.1109/CLOUD.2012.87.

[18] R. Birke, A. Podzimek, L. Y. Chen und E. Smirni. „State-of-the-practice in data center virtualization: Toward a better understanding of VM usage“. In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Juni 2013, S. 1–12. DOI: 10.1109/DSN.2013.6575350.

[19] R. Birke, A. Podzimek, L. Y. Chen und E. Smirni. „Virtualization in the Private Cloud: State of the Practice“. In: *IEEE Transactions on Network and Service Management* 13.3 (Sep. 2016), S. 608–621. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2601646.

[20] K. P. Birman, F. Mattern und A. Schiper. *Theory and Practice in Distributed Systems: International Workshop, Dagstuhl Castle, Germany, September 5 - 9, 1994. Selected Papers (Lecture Notes in Computer Science)*. Springer, 1995. ISBN: 978-3-540-60042-8.

[21] A. D. Birrell und B. J. Nelson. „Implementing Remote Procedure Calls“. In: *ACM Trans. Comput. Syst.* 2.1 (Feb. 1984), S. 39–59. ISSN: 0734-2071. DOI: 10.1145/2080.357392. URL: http://doi.acm.org/10.1145/2080.357392.

[22] R. B. Bohn, J. Messina, F. Liu, J. Tong und J. Mao. „NIST Cloud Computing Reference Architecture“. In: *2011 IEEE World Congress on Services*. Juli 2011, S. 594–596. DOI: 10.1109/SERVICES.2011.105.

[23] A. Botta, W. de Donato, V. Persico und A. Pescapé. „Integration of Cloud computing and Internet of Things: A survey“. In: *Future Generation Computer Systems* 56 (2016), S. 684–700. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2015.09.021. URL: http://www.sciencedirect.com/science/article/pii/S0167739X15003015.

[24] *BSI TR-02102-1: Kryptographische Verfahren: Empfehlungen und Schlusselläng*en. Technischen Richtlinie. Bundesamt für Sicherheit in der Informationstechnik, 2017.

[25] *BSI-Gesetz vom 14. August 2009 (BGBl. I S. 2821), das zuletzt durch Artikel 1 des Gesetzes vom 23. Juni 2017 (BGBl. I S. 1885) geändert worden ist*. https://www.gesetze-im-internet.de/bsig_2009/index.html. 2017.

[26] *Bundesdatenschutzgesetz in der Fassung der Bekanntmachung vom 14. Januar 2003 (BGBl. I S. 66), das zuletzt durch Artikel 10 Absatz 2 des Gesetzes vom 31. Oktober 2017 (BGBl. I S. 3618) geändert worden ist*. 2017.

[27] *Bundesdatenschutzgesetz vom 30. Juni 2017 (BGBl. I S. 2097)*. 2017.

[28] R. Buyya, A. Beloglazov und J. Abawajy. „Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges“. In: *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*. 2010, S. 6–12.

[29] J. M. A. Calero und J. G. Aguado. „Comparative analysis of architectures for monitoring cloud computing infrastructures“. In: *Future Generation Computer Systems* 47 (2015). Special Section: Advanced Architectures for the Future Generation of Software-Intensive Systems, S. 16–30. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2014.12.008. URL: http://www.sciencedirect.com/science/article/pii/S0167739X14002684.

[30] J. Camenisch und V. Shoup. *Practical Verifiable Encryption and Decryption of Discrete Logarithms*. Cryptology ePrint Archive, Report 2002/161. http://eprint.iacr.org/2002/161. 2002.

[31] R. C. Chalmers und K. C. Almeroth. „Modeling the branching characteristics and efficiency gains in global multicast trees“. In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Bd. 1. Apr. 2001, 449–458 vol.1. DOI: 10.1109/INFCOM.2001.916734.

[32] R. C. Chalmers und K. C. Almeroth. „On the topology of multicast trees“. In: *IEEE/ACM Transactions on Networking* 11.1 (Feb. 2003), S. 153–165. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.804835.

[33] Y. Chen, R. Griffith, J. Liu, R. H. Katz und A. D. Joseph. „Understanding TCP incast throughput collapse in datacenter networks“. In: *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM. 2009, S. 73–82.

[34] Y. Cheng, X.-Y. Li und M.-Q. Ling. „A Trusted Cloud Service Platform Architecture“. In: *Information Science and Applications (ICISA), 2012 International Conference on*. Mai 2012, S. 1–6. DOI: 10.1109/ICISA.2012.6220973.

[35] Y. Cherdantseva. „Secure*BPMN - a graphical extension for BPMN 2.0 based on a Reference Model of Information Assurance & Security“. Diss. Cardiff University – School of Computer Science & Informatics, 2014. URL: http://orca.cf.ac.uk/id/eprint/74432.

[36] D. M. Chiu, S. Hurst, M. Kadansky und J. Wesley. *TRAM: A Tree-based Reliable Multicast Protocol*. Techn. Ber. Mountain View, CA, USA, 1998.

[37] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka und J. Molina. „Controlling Data in the Cloud: Outsourcing Computation Without Outsourcing Control“. In: *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. CCSW '09. Chicago, Illinois, USA: ACM, 2009, S. 85–90. ISBN: 978-1-60558-784-4. DOI: 10.1145/1655008.1655020.

[38] J. C.-I. Chuang und M. A. Sirbu. „Pricing Multicast Communication: A Cost-Based Approach“. In: *Telecommunication Systems* 17.3 (Juli 2001), S. 281–297. ISSN: 1572-9451. DOI: 10.1023/A:1016695006342. URL: https://doi.org/10.1023/A:1016695006342.

[39] L. Clarke, I. Glendinning und R. Hempel. „The MPI message passing interface standard“. In: *Programming environments for massively parallel distributed systems*. Springer, 1994, S. 213–218.

[40] C. Clos. „A study of non-blocking switching networks“. In: *The Bell System Technical Journal* 32.2 (März 1953), S. 406–424. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[41] M. Cucciniello, G. A. Porumbescu und S. Grimmelikhuijsen. „25 Years of Transparency Research: Evidence and Future Directions“. In: *Public Administration Review* 77.1 (), S. 32–44. DOI: 10.1111/puar.12685. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/puar.12685.

[42] G. Cugola, E. Di Nitto und A. Fuggetta. „The JEDI event-based infrastructure and its application to the development of the OPSS WFMS“. In: *IEEE transactions on Software Engineering* 27.9 (2001), S. 827–850.

[43] G. Cugola, M. Migliavacca und A. Monguzzi. „On Adding Replies to Publish-subscribe“. In: *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*. DEBS '07. Toronto, Ontario, Canada: ACM, 2007, S. 128–138. ISBN: 978-1-59593-665-3.

[44] G. Da Cunha Rodrigues, R. N. Calheiros, V. T. Guimaraes, G. L. d. Santos, M. B. de Carvalho, L. Z. Granville, L. M. R. Tarouco und R. Buyya. „Monitoring of Cloud Computing Environments: Concepts, Solutions, Trends, and Future Directions“. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. SAC '16. Pisa, Italy: ACM, 2016, S. 378–383. ISBN: 978-1-4503-3739-7. DOI: 10.1145/2851613.2851619. URL: http://doi.acm.org/10.1145/2851613.2851619.

[45] M. Denzel, M. Ryan und E. Ritter. „A Malware-Tolerant, Self-Healing Industrial Control System Framework“. In: *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference, SEC 2017, Rome, Italy, May 29-31, 2017, Proceedings*. Hrsg. von S. De Capitani di Vimercati und F. Martinelli. Cham: Springer International Publishing, 2017, S. 46–60. ISBN: 978-3-319-58469-0. DOI: 10.1007/978-3-319-58469-0_4. URL: http://dx.doi.org/10.1007/978-3-319-58469-0_4.

[46] W. Diffie und M. E. Hellman. „New directions in cryptography“. In: *Information Theory, IEEE Transactions on* 22.6 (Nov. 1976), S. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638.

[47] C. Diot, W. Dabbous und J. Crowcroft. „Multipoint communication: a survey of protocols, functions, and mechanisms“. In: *IEEE Journal on Selected Areas in Communications* 15.3 (Apr. 1997), S. 277–290. ISSN: 0733-8716. DOI: 10.1109/49.564128.

[48] F. Doelitzscher, C. Fischer, D. Moskal, C. Reich, M. Knahl und N. Clarke. „Validating Cloud Infrastructure Changes by Cloud Audits“. In: *Services (SERVICES), 2012 IEEE Eighth World Congress on*. Juni 2012, S. 377–384. DOI: 10.1109/SERVICES.2012.12.

[49] D. Dolev und A. C. Yao. „On the security of public key protocols“. In: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*. Okt. 1981, S. 350–357. DOI: 10.1109/SFCS.1981.32.

[50] J.-P. Ebert, A. Willig et al. „A Gilbert-Elliot bit error model and the efficient use in packet level simulation“. In: *TKN Technical Reports Series*. TKN-99-002 (1999). URL: http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/tkn_report02.pdf.

[51] T. Elgamal. „A public key cryptosystem and a signature scheme based on discrete logarithms“. In: *Information Theory, IEEE Transactions on* 31.4 (Juli 1985), S. 469–472. ISSN: 0018-9448. DOI: 10.1109/TIT.1985.1057074.

[52] *ENISA TP-05-14-084-EN-N: Algorithms, key size and parameters report*. Report. European Union Agency for Network und Information Security, Nov. 2014.

[53] P. T. Eugster, P. A. Felber, R. Guerraoui und A.-M. Kermarrec. „The Many Faces of Publish/Subscribe“. In: *ACM Comput. Surv.* 35.2 (Juni 2003), S. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: http://doi.acm.org/10.1145/857076.857078.

[54] F. Faniyi und R. Bahsoon. „A Systematic Review of Service Level Management in the Cloud“. In: *ACM Comput. Surv.* 48.3 (Dez. 2015), 43:1–43:27. ISSN: 0360-0300. DOI: 10.1145/2843890. URL: http://doi.acm.org/10.1145/2843890.

[55] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison und T. Lynn. „A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives“. In: *Journal of Parallel and Distributed Computing* 74.10 (2014), S. 2918–2933. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2014.06.007. URL: http://www.sciencedirect.com/science/article/pii/S0743731514001099.

[56] P. Felber. „The CORBA object group service: A service approach to object groups in CORBA“. Diss. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998.

[57] P. Felber, R. Guerraoui und A. Schiper. „The Implementation of a CORBA Object Group Service“. In: *Theor. Pract. Object Syst.* 4.2 (Apr. 1998), S. 93–105. ISSN: 1074-3227. DOI: 10.1002/(SICI)1096-9942(1998)4:2<93::AID-TAPO4>3.3.CO;2-E. URL: http://dx.doi.org/10.1002/(SICI)1096-9942(1998)4:2%3C93::AID-TAPO4%3E3.3.CO;2-E.

[58] C. Fiandrino, D. Kliazovich, P. Bouvry und A. Y. Zomaya. „Performance and Energy Efficiency Metrics for Communication Systems of Cloud Computing Data Centers“. In: *IEEE Transactions on Cloud Computing* 5.4 (Okt. 2017), S. 738–750. ISSN: 2168-7161. DOI: 10.1109/TCC.2015.2424892.

[59] M. Fischlin. „Trapdoor commitment schemes and their applications“. Diss. Johann Wolfgang Goethe-Universität, Frankfurt am Main, 2001.

[60] R. W. Floyd. „Algorithm 97: Shortest Path“. In: *Commun. ACM* 5.6 (Juni 1962), S. 345–. ISSN: 0001-0782. DOI: 10.1145/367766.368168. URL: http://doi.acm.org/10.1145/367766.368168.

[61] S. Floyd und V. Jacobson. „The Synchronization of Periodic Routing Messages“. In: *IEEE/ACM Trans. Netw.* 2.2 (Apr. 1994), S. 122–136. ISSN: 1063-6692. DOI: 10.1109/90.298431. URL: http://dx.doi.org/10.1109/90.298431.

[62] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu und L. Zhang. „A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing“. In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '95. Cambridge, Massachusetts, USA: ACM, 1995, S. 342–356. ISBN: 978-0-89791-711-7. DOI: 10.1145/217382.217470. URL: http://doi.acm.org/10.1145/217382.217470.

[63] J. Fox. „The uncertain relationship between transparency and accountability“. In: *Development in practice* 17.4-5 (2007), S. 663–671.

[64] T. Fukai, T. Shinagawa und K. Kato. „Live Migration in Bare-metal Clouds“. In: *IEEE Transactions on Cloud Computing* (2018), S. 1–1. ISSN: 2168-7161. DOI: 10.1109/TCC.2018.2848981.

[65] P. Gallagher, D. D. Foreword und C. F. Director. *FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS)*. 2009.

[66] N. Garg. *Apache Kafka*. Packt Publishing Ltd, 2013.

[67] J. Gemmell, T. Montgomery, T. Speakman und J. Crowcroft. „The PGM reliable multicast protocol“. In: *IEEE Network* 17.1 (Jan. 2003), S. 16–22. ISSN: 0890-8044. DOI: 10.1109/MNET.2003.1174173.

[68] *Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz)*. http://www.bgbl.de/xaver/bgbl/start.xav?startbk=Bundesanzeiger_BGBl&jumpTo=bgbl115s1324.pdf. 2015.

[69] S. Ghazouani und Y. Slimani. „A survey on cloud service description“. In: *Journal of Network and Computer Applications* 91 (2017), S. 61–74. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.04.013. URL: http://www.sciencedirect.com/science/article/pii/S1084804517301613.

[70] S. Gilboa, S. Gueron und B. Morris. „How Many Queries are Needed to Distinguish a Truncated Random Permutation from a Random Function?“ In: *Journal of Cryptology* 31.1 (Jan. 2018), S. 162–171. ISSN: 1432-1378. DOI: 10.1007/s00145-017-9253-0. URL: https://doi.org/10.1007/s00145-017-9253-0.

[71] D. Gonzales, J. M. Kaplan, E. Saltzman, Z. Winkelman und D. Woods. „Cloud-Trust—a Security Assessment Model for Infrastructure as a Service (IaaS) Clouds“. In: *IEEE Transactions on Cloud Computing* 5.3 (Juli 2017), S. 523–536. ISSN: 2168-7161. DOI: 10.1109/TCC.2015.2415794.

[72] R. L. Graham, D. E. Knuth und O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science, Second Edition*. Addison Wesley Pub Co Inc, 1994. ISBN: 978-0-201-55802-9.

[73] S. Grimmelikhuijsen. „Transparency and trust. An experimental study of online disclosure and trust in government“. Diss. Utrecht University, 2012.

[74] W. D. Gropp, W. Gropp, E. Lusk und A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Bd. 1. MIT press, 1999.

[75] W. Gropp, E. Lusk, N. Doss und A. Skjellum. „A high-performance, portable implementation of the MPI message passing interface standard“. In: *Parallel computing* 22.6 (1996), S. 789–828.

[76] I. Gul, A. ur Rehman und M. H. Islam. „Cloud computing security auditing“. In: *Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on*. Juni 2011, S. 143–148.

[77] A. Haeberlen. „A Case for the Accountable Cloud“. In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), S. 52–57. ISSN: 0163-5980.

[78] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi und M. Wójcik. „Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance“. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Los Angeles, CA, USA: ACM, 2017, S. 29–42. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098825. URL: http://doi.acm.org/10.1145/3098822.3098825.

[79] R. Hans, U. Lampe und R. Steinmetz. „QoS-Aware, Cost-Efficient Selection of Cloud Data Centers“. In: *2013 IEEE Sixth International Conference on Cloud Computing*. Juni 2013, S. 946–947. DOI: 10.1109/CLOUD.2013.113.

[80] M. Hapner, R. Burridge, R. Sharma, J. Fialli und K. Stout. „Java message service“. In: *Sun Microsystems Inc., Santa Clara, CA* (2002), S. 9.

[81] G. Haßlinger und O. Hohlfeld. „The Gilbert-Elliott model for packet loss in real time services on the Internet“. In: *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2008 14th GI/ITG Conference-*. VDE. 2008, S. 1–15.

[82] L. Henckel und H. Stüttgen. „Transportdienste in Breitbandnetzen“. In: *Kommunikation in verteilten Systemen: Grundlagen, Anwendungen, Betrieb GI/ITG-Fachtagung, Mannheim, 20.–22. Februar 1991, Proceedings*. Hrsg. von W. Effelsberg, H. W. Meuer und G. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, S. 96–111. ISBN: 978-3-642-76462-2. DOI: 10.1007/978-3-642-76462-2_8. URL: https://doi.org/10.1007/978-3-642-76462-2_8.

[83] M. Henning. „The Rise and Fall of CORBA“. In: *Queue* 4.5 (Juni 2006), S. 28–34. ISSN: 1542-7730. DOI: 10.1145/1142031.1142044. URL: http://doi.acm.org/10.1145/1142031.1142044.

[84] J. C. Hill, J. C. Knight, A. M. Crickenberger und R. Honhart. *Publish and subscribe with reply*. Techn. Ber. University of Virginia: Department of Computer Science, 2002.

[85] G. J. Hofstede. „Transparency in netchains“. In: *Information Technology for a better Agri-Food Sector, Environment and Rural Living. Debrecen University, Debrecen, Hungary* (2003), S. 17–29.

[86] G. Hohpe und B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004. ISBN: 978-0-321-20068-6.

[87] H. W. Holbrook, S. K. Singhal und D. R. Cheriton. „Log-based Receiver-reliable Multicast for Distributed Interactive Simulation“. In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '95. Cambridge, Massachusetts, USA: ACM, 1995, S. 328–341. ISBN: 978-0-89791-711-7. DOI: 10.1145/217382.217468. URL: http://doi.acm.org/10.1145/217382.217468.

[88] M. Holloway. *Service Level Management in Cloud Computing: Pareto-Efficient Negotiations, Reliable Monitoring, and Robust Monitor Placement*. Springer, 2017. ISBN: 978-3-658-18773-6. DOI: 10.1007/978-3-658-18773-6.

[89] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong und A. Vahdat. „B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-defined WAN“. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Budapest,

Hungary: ACM, 2018, S. 74–87. ISBN: 978-1-4503-5567-4. DOI: 10.1145/3230543.3230545. URL: http://doi.acm.org/10.1145/3230543.3230545.

[90] M. Hosseini, D. T. Ahmed, S. Shirmohammadi und N. D. Georganas. „A survey of application-layer multicast protocols“. In: *IEEE Communications Surveys Tutorials* 9.3 (Third 2007), S. 58–74. ISSN: 1553-877X. DOI: 10.1109/COMST.2007.4317616.

[91] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull und J. N. Matthews. „A Quantitative Study of Virtual Machine Live Migration“. In: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. CAC '13. Miami, Florida, USA: ACM, 2013, 11:1–11:10. ISBN: 978-1-4503-2172-3. DOI: 10.1145/2494621.2494622. URL: http://doi.acm.org/10.1145/2494621.2494622.

[92] F. K. Hwang, D. S. Richards und P. Winter. *The Steiner tree problem*. Bd. 53. Elsevier, 1992.

[93] *IEEE Std 802.3-2015: IEEE Standard for Ethernet*. Standard. Institute of Electrical und Electronics Engineers, 2015.

[94] P. Z. Ingerman. „Algorithm 141: Path Matrix“. In: *Commun. ACM* 5.11 (Nov. 1962), S. 556–. ISSN: 0001-0782. DOI: 10.1145/368996.369016. URL: http://doi.acm.org/10.1145/368996.369016.

[95] *ISO/IEC 11889:2015: Information technology – Trusted platform module library*. Standard. International Organization for Standardization, Aug. 2015.

[96] *ISO/IEC 13239:2002: Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*. Standard. International Organization for Standardization, 2002.

[97] *ISO/IEC 19464:2014: Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification*. Standard. International Organization for Standardization, Mai 2014.

[98] *ISO/IEC 27001:2013: Information technology – Security techniques – Information security management systems – Requirements*. Standard. International Organization for Standardization, 2013.

[99] *ISO/IEC 27017:2015 / ITU-T X.1631 – Information technology – Security techniques – Code of practice for information security controls based on ISO/IEC 27002 for cloud services*. Standard. International Organization for Standardization, 2015.

[100] *ISO/IEC 27018:2014 – Information technology – Security techniques – Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors*. Standard. International Organization for Standardization, 2014.

[101] *ISO/IEC 7498-1:1994: Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Standard. International Organization for Standardization, Nov. 1994.

[102] *ITU-T X.210: Information technology – Open Systems Interconnection – Basic Reference Model: Conventions for the definition of OSI services*. Recommendation. International Telecommunications Union, Nov. 1993.

[103] *ITU-T X.690: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. Recommendation. International Telecommunications Union, Aug. 2015.

[104] B. Jennings und R. Stadler. „Resource management in clouds: Survey and research challenges“. In: *Journal of Network and Systems Management* 23.3 (2015), S. 567–619.

[105] W. Jia und W. Zhou. *Distributed network systems: From concepts to implementations*. Springer, 2005. ISBN: 978-0-387-23840-1.

[106] C. Jung, A. Eitel und R. Schwarz. „Enhancing Cloud Security with Context-aware Usage Control Policies.“ In: *GI-Jahrestagung*. 2014, S. 211–222.

[107] M. F. Kaashoek, A. S. Tanenbaum und K. Verstoep. „Group communication in Amoeba and its applications“. In: *Distributed Systems Engineering* 1.1 (1993), S. 48.

[108] M. F. Kaashoek und A. S. Tanenbaum. „An evaluation of the Amoeba group communication system“. In: *Proceedings of 16th International Conference on Distributed Computing Systems*. Mai 1996, S. 436–447. DOI: 10.1109/ICDCS.1996.507992.

[109] S. K. Kasera, J. Kurose und D. Towsley. „Scalable Reliable Multicast Using Multiple Multicast Groups“. In: *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '97. Seattle, Washington, USA: ACM, 1997, S. 64–74. ISBN: 978-0-89791-909-8. DOI: 10.1145/258612.258676. URL: http://doi.acm.org/10.1145/258612.258676.

[110] J. Katz und Y. Lindell. *Introduction to Modern Cryptography, Second Edition (Chapman & Hall/CRC Cryptography and Network Security Series)*. Chapman und Hall/CRC, 2014. ISBN: 978-1-4665-7026-9.

[111] A. Keller und H. Ludwig. „The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services“. In: *Journal of Network and Systems Management* 11.1 (März 2003), S. 57–81. ISSN: 1573-7705. DOI: 10.1023/A:1022445108617. URL: https://doi.org/10.1023/A:1022445108617.

[112] R. Knode und D. Egan. *A Precis for the CloudTrust Protocol (V2.0)*. 2010.

[113] R. K. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang und B. S. Lee. „TrustCloud: A Framework for Accountability and Trust in Cloud Computing“. In: *Services (SERVICES), 2011 IEEE World Congress on*. Juli 2011, S. 584–588. DOI: 10.1109/SERVICES.2011.91.

[114] R. L. Ko, B. Lee und S. Pearson. „Towards Achieving Accountability, Auditability and Trust in Cloud Computing“. In: *Advances in Computing and Communications*. Hrsg. von A. Abraham, J. Mauri, J. Buford, J. Suzuki und S. Thampi. Bd. 193. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, S. 432–444. ISBN: 978-3-642-22725-7. DOI: 10.1007/978-3-642-22726-4_45.

[115] H. König. *Protocol Engineering: Prinzip, Beschreibung und Entwicklung von Kommunikationsprotokollen*. Springer-Verlag, 2013. ISBN: 978-3-322-80066-4. DOI: 10.1007/978-3-322-80066-4. URL: https://doi.org/10.1007/978-3-322-80066-4.

[116] J. Kurose und R. Keith. *Computer Networking: A Top-Down Approach, Global Edition*. Bd. Seventh Edition. Pearson Higher Education, 2016. ISBN: 978-1-292-15359-9.

[117] N. Kusolitsch. „Messbare Funktionen – Zufallsvariable“. In: *Maß- und Wahrscheinlichkeitstheorie: Eine Einführung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 83–108. ISBN: 978-3-642-45387-8. DOI: 10.1007/978-3-642-45387-8_7. URL: https://doi.org/10.1007/978-3-642-45387-8_7.

[118] G. Leurent, M. Nandi und F. Sibleyras. „Generic Attacks Against Beyond-Birthday-Bound MACs“. In: *Advances in Cryptology – CRYPTO 2018*. Hrsg. von H. Shacham und A. Boldyreva. Cham: Springer International Publishing, 2018, S. 306–336. ISBN: 978-3-319-96884-1.

[119] B. N. Levine und J. J. Garcia-Luna-Aceves. „A comparison of known classes of reliable multicast protocols“. In: *Proceedings of 1996 International Conference on Network Protocols (ICNP-96)*. Okt. 1996, S. 112–121. DOI: 10.1109/ICNP.1996.564920.

[120] D. Li und D. R. Cheriton. „OTERS (on-tree efficient recovery using subcasting): a reliable multicast protocol“. In: *Proceedings Sixth International Conference on Network Protocols (Cat. No.98TB100256)*. Okt. 1998, S. 237–245. DOI: 10.1109/ICNP.1998.723744.

[121] H. Li, G. Zhu, C. Cui, H. Tang, Y. Dou und C. He. „Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing“. In: *Computing* 98.3 (März 2016), S. 303–317. ISSN: 1436-5057. DOI: 10.1007/s00607-015-0467-4. URL: https://doi.org/10.1007/s00607-015-0467-4.

[122] S. Lin, D. J. Costello und M. J. Miller. „Automatic-repeat-request error-control schemes“. In: *IEEE Communications Magazine* 22.12 (Dez. 1984), S. 5–17. ISSN: 0163-6804. DOI: 10.1109/MCOM.1984.1091865.

[123] S. Lins, P. Grochol, S. Schneider und A. Sunyaev. „Dynamic Certification of Cloud Services: Trust, but Verify!“ In: *IEEE Security Privacy* 14.2 (März 2016), S. 66–71. ISSN: 1540-7993. DOI: 10.1109/MSP.2016.26.

[124] S. Lins, S. Schneider und A. Sunyaev. „Trust is Good, Control is Better: Creating Secure Clouds by Continuous Auditing“. In: *IEEE Transactions on Cloud Computing* 6.3 (Juli 2018), S. 890–903. ISSN: 2168-7161. DOI: 10.1109/TCC.2016.2522411.

[125] B. Littlewood und L. Strigini. „Redundancy and Diversity in Security“. In: *Computer Security – ESORICS 2004: 9th European Symposium on Research in Computer Security, Sophia Antipolis, France, September 13 - 15, 2004. Proceedings*. Hrsg. von P. Samarati, P. Ryan, D. Gollmann und R. Molva. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 423–438. ISBN: 978-3-540-30108-0. DOI: 10.1007/978-3-540-30108-0_26. URL: http://dx.doi.org/10.1007/978-3-540-30108-0_26.

[126] Y. Liu, B. Plale et al. „Survey of publish subscribe event systems“. In: *Computer Science Dept, Indian University* 16 (2003).

[127] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola und J. M. Hellerstein. „Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud“. In: *Proc. VLDB Endow.* 5.8 (Apr. 2012), S. 716–727. ISSN: 2150-8097. DOI: 10.14778/2212351.2212354. URL: https://doi.org/10.14778/2212351.2212354.

[128] A. Maarouf, A. Marzouk und A. Haqiq. „A review of SLA specification languages in the cloud computing“. In: *Intelligent Systems: Theories and Applications (SITA), 2015 10th International Conference on*. IEEE. 2015, S. 1–6.

[129] S. Maffeis. „The Object Group Design Pattern“. In: *Proceedings of the 2Nd Conference on USENIX Conference on Object-Oriented Technologies (COOTS) - Volume 2*. COOTS'96. Toronto, Ontario, Canada: USENIX Association, 1996, S. 12–12. URL: http://dl.acm.org/citation.cfm?id=1268049.1268061.

[130] A. W. Malik und S. U. Khan. „Data center modeling and simulation using OMNET++“. In: *Handbook on Data Centers*. Springer, 2015, S. 839–855.

[131] M. Matsumoto und T. Nishimura. „Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator“. In: *ACM Trans. Model. Comput. Simul.* 8.1 (Jan. 1998), S. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995. URL: http://doi.acm.org/10.1145/272991.272995.

[132] D. McGrew. „Galois Counter Mode“. In: *Encyclopedia of Cryptography and Security*. Hrsg. von H. C. A. van Tilborg und S. Jajodia. Boston, MA: Springer US, 2011, S. 506–508. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_451. URL: https://doi.org/10.1007/978-1-4419-5906-5_451.

[133] D. McGrew und J. Viega. „The Galois/counter mode of operation (GCM)“. In: *submission to NIST Modes of Operation Process* 20 (2004).

[134] A. Meera und S. Swamynathan. „Agent based Resource Monitoring System in IaaS Cloud Environment“. In: *Procedia Technology* 10 (2013). First International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013, S. 200–207. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2013.12.353. URL: http://www.sciencedirect.com/science/article/pii/S2212017313005070.

[135] R. M. Metcalfe und D. R. Boggs. „Ethernet: Distributed Packet Switching for Local Computer Networks“. In: *Commun. ACM* 19.7 (Juli 1976), S. 395–404. ISSN: 0001-0782. DOI: 10.1145/360248.360253. URL: http://doi.acm.org/10.1145/360248.360253.

[136] P. V. Mieghem und M. Janic. „Stability of a multicast tree“. In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Bd. 2. Juni 2002, 1099–1108 vol.2. DOI: 10.1109/INFCOM.2002.1019358.

[137] C. K. Miller, T. Andresen, T. Gardner, C. Michelson, K. Cates, M. White und K. Robertson. *System and method for sending packets over a computer network*. US Patent 6,873,627. März 2005.

[138] K. Miller, K. Robertson, A. Tweedly und M. White. *StarBurst Multicast File Transfer Protocol (MFTP) Specification*. Internet-Draft draft-miller-mftp-spec-03. IETF, Apr. 1998. URL: https://tools.ietf.org/html/draft-miller-mftp-spec-03.

[139] M. Mürmann. *Wahrscheinlichkeitstheorie und stochastische Prozesse*. Masterclass. Springer Spektrum, 2014. ISBN: 978-3-642-38160-7.

[140] M. Nanavati, P. Colp, B. Aiello und A. Warfield. „Cloud Security: A Gathering Storm“. In: *Commun. ACM* 57.5 (Mai 2014), S. 70–79. ISSN: 0001-0782. DOI: 10.1145/2593686. URL: http://doi.acm.org/10.1145/2593686.

[141] M. Naor und A. Ziv. *Primary-Secondary-Resolver Membership Proof Systems*. Cryptology ePrint Archive, Report 2014/905. http://eprint.iacr.org/2014/905. 2014.

[142] *NIST SP 500-292: NIST Cloud Computing Reference Architectur*. Recommendation. National Institute of Standards und Technology, Sep. 2011.

[143] *NIST SP 800-145: The NIST Definition of Cloud Computing*. Recommendation. National Institute of Standards und Technology, Sep. 2011.

[144] *NIST SP 800-57: Part 1 Rev. 4: Recommendation for Key Management, Part 1: General*. Recommendation. National Institute of Standards und Technology, Juli 2012.

[145] K. Obraczka. „Multicast Transport Protocols: A Survey and Taxonomy“. In: *IEEE Communications Magazine* 36.1 (Jan. 1998), S. 94–102. ISSN: 0163-6804. DOI: 10.1109/35.649333.

[146] P. Patel, A. H. Ranabahu und A. P. Sheth. *Service level agreement in cloud computing*. 2009.

[147] S. Paul, K. K. Sabnani, J. C. H. Lin und S. Bhattacharyya. „Reliable multicast transport protocol (RMTP)“. In: *IEEE Journal on Selected Areas in Communications* 15.3 (Apr. 1997), S. 407–421. ISSN: 0733-8716. DOI: 10.1109/49.564138.

[148] *PCI DSS v3.2: Data Security Standard – Requirements and Security Assessment Procedures – Version 3.2*. Standard. Payment Card Industry Security Standard Council, Apr. 2016.

[149] S. Pearson. „Toward Accountability in the Cloud“. In: *IEEE Internet Computing* 15.4 (Juli 2011), S. 64–69. ISSN: 1089-7801. DOI: 10.1109/MIC.2011.98.

[150] T. P. Pedersen. „Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing“. In: *Advances in Cryptology — CRYPTO '91: Proceedings*. Hrsg. von J. Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, S. 129–140. ISBN: 978-3-540-46766-3. DOI: 10.1007/3-540-46766-1_9. URL: http://dx.doi.org/10.1007/3-540-46766-1_9.

[151] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar et al. „The Design and Implementation of Open vSwitch.“ In: *NSDI*. 2015, S. 117–130.

[152] G. Phillips, S. Shenker und H. Tangmunarunkit. „Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law“. In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '99. Cambridge, Massachusetts, USA: ACM, 1999, S. 41–51. ISBN: 978-1-58113-135-2. DOI: 10.1145/316188.316205. URL: http://doi.acm.org/10.1145/316188.316205.

[153] D. Powell. „Group Communication“. In: *Commun. ACM* 39.4 (Apr. 1996), S. 50–53. ISSN: 0001-0782. DOI: 10.1145/227210.227225. URL: http://doi.acm.org/10.1145/227210.227225.

[154] C. Price, S. Rivera, A. Peled, M. Wolpin, F. Brockners, P. Chinnakannan, A. Sardella, P. Hou, M. Young, P. Mehta, T. Nguyenphu und D. Neary. „OPNFV: An Open Platform to Accelerate NFV“. In: *White Paper. A Linux Foundation Collaborative Project* (2012).

[155] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. http://eur-lex.europa.eu/eli/reg/2016/679/oj. 2016.

[156] R. van Renesse, K. P. Birman und S. Maffeis. „Horus: A Flexible Group Communication System“. In: *Commun. ACM* 39.4 (Apr. 1996), S. 76–83. ISSN: 0001-0782. DOI: 10.1145/227210.227229. URL: http://doi.acm.org/10.1145/227210.227229.

[157] B. P. Rimal, E. Choi und I. Lumb. „A Taxonomy and Survey of Cloud Computing Systems“. In: *2009 Fifth International Joint Conference on INC, IMS and IDC*. Aug. 2009, S. 44–51. DOI: 10.1109/NCM.2009.218.

[158] T. Ruffing und G. Malavolta. „Switch Commitments: A Safety Switch for Confidential Transactions“. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, S. 170–181.

[159] J. H. Saltzer, D. P. Reed und D. D. Clark. „End-to-end Arguments in System Design“. In: *ACM Trans. Comput. Syst.* 2.4 (Nov. 1984), S. 277–288. ISSN: 0734-2071. DOI: 10.1145/357401.357402. URL: http://doi.acm.org/10.1145/357401.357402.

[160] C. P. Schnorr. „Efficient Identification and Signatures for Smart Cards“. English. In: *Advances in Cryptology — EUROCRYPT '89*. Hrsg. von J.-J. Quisquater und J. Vandewalle. Bd. 434. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1990, S. 688–689. ISBN: 978-3-540-53433-4. DOI: 10.1007/3-540-46885-4_68.

[161] M. Schöller, R. Bless, F. Pallas, J. Horneber und P. Smith. „An Architectural Model for Deploying Critical Infrastructure Services in the Could“. In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Bd. 1. Dez. 2013, S. 458–466. DOI: 10.1109/CloudCom.2013.67.

[162] R. Sharp. *Principles of protocol design*. Springer, 2008. ISBN: 978-3-540-77541-6.

[163] W. Shi, J. Cao, Q. Zhang, Y. Li und L. Xu. „Edge Computing: Vision and Challenges“. In: *IEEE Internet of Things Journal* 3.5 (Okt. 2016), S. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[164] M. Siebenhaar, U. Lampe, D. Schuller und R. Steinmetz. „Robust Cloud Monitor Placement for Availability Verification“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science*. CLOSER 2014. Barcelona, Spain: SCITEPRESS - Science und Technology Publications, Lda, 2014, S. 193–198. ISBN: 978-989-758-019-2. DOI: 10.5220/0004958101930198. URL: http://dx.doi.org/10.5220/0004958101930198.

[165] M. Siebenhaar, D. Schuller, O. Wenge und R. Steinmetz. „Heuristic Approaches for Robust Cloud Monitor Placement“. In: *Service-Oriented Computing*. Hrsg. von X. Franch, A. K. Ghose, G. A. Lewis und S. Bhiri. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 321–335. ISBN: 978-3-662-45391-9.

[166] M. Siebenhaar, O. Wenge, R. Hans, H. Tercan und R. Steinmetz. „Verifying the Availability of Cloud Applications“. In: *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013*. 2013, S. 489–494.

[167] M. Simsek, A. Aijaz, M. Dohler, J. Sachs und G. Fettweis. „5G-Enabled Tactile Internet“. In: *IEEE Journal on Selected Areas in Communications* 34.3 (März 2016), S. 460–473. ISSN: 0733-8716. DOI: 10.1109/JSAC.2016.2525398.

[168] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart und A. Vahdat. „Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network“. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: ACM, 2015, S. 183–197. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787508.

[169] W. Stallings. *Data and computer communications*. Bd. 10th international edition. Pearson Higher Education, Dez. 2013. ISBN: 978-1-292-01438-8.

[170] A. Sunyaev und S. Schneider. „Cloud Services Certification“. In: *Commun. ACM* 56.2 (Feb. 2013), S. 33–36. ISSN: 0001-0782. DOI: 10.1145/2408776.2408789. URL: http://doi.acm.org/10.1145/2408776.2408789.

[171] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan und A. I. A. Ahmed. „Cloud monitoring: A review, taxonomy, and open research issues“. In: *Journal of Network and Computer Applications* 98 (2017), S. 11–26. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.08.021. URL: http://www.sciencedirect.com/science/article/pii/S1084804517302783.

[172] A. Tanenbaum. *Computer Networks*. Bd. 5th Edition. Pearson India, 2010. ISBN: 978-93-325-1874-2.

[173] B. H. Tay und A. L. Ananda. „A Survey of Remote Procedure Calls“. In: *SIGOPS Oper. Syst. Rev.* 24.3 (Juli 1990), S. 68–79. ISSN: 0163-5980. DOI: 10.1145/382244.382832. URL: http://doi.acm.org/10.1145/382244.382832.

[174] *TCG TPM 2.0: Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.38*. Specification. Trusted Computing Group, Sep. 2016.

[175] A. Tseitlin. „The Antifragile Organization“. In: *Commun. ACM* 56.8 (Aug. 2013), S. 40–44. ISSN: 0001-0782. DOI: 10.1145/2492007.2492022. URL: http://doi.acm.org/10.1145/2492007.2492022.

[176] K. W. Ullah, A. S. Ahmed und J. Ylitalo. „Towards Building an Automated Security Compliance Tool for the Cloud“. In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. Juli 2013, S. 1587–1593. DOI: 10.1109/TrustCom.2013.195.

[177] R. B. Uriarte, F. Tiezzi und R. D. Nicola. „Slac: A formal service-level-agreement language for cloud computing“. In: *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society. 2014, S. 419–426.

[178] P. Van Mieghem, G. Hooghiemstra und R. van der Hofstad. „On the Efficiency of Multicast“. In: *IEEE/ACM Trans. Netw.* 9.6 (Dez. 2001), S. 719–732. ISSN: 1063-6692. DOI: 10.1109/90.974526. URL: http://dx.doi.org/10.1109/90.974526.

[179] A. Varga et al. „The OMNeT++ discrete event simulation system“. In: *Proceedings of the European simulation multiconference (ESM'2001)*. Bd. 9. S 185. sn. 2001, S. 65.

[180] A. Varga und R. Hornig. „An Overview of the OMNeT++ Simulation Environment“. In: *Proceedings of the 1st ICST International Workshop on OMNeT++*. ICST. Marseille, France, März 2008. ISBN: 978-963-9799-20-2. URL: http://dl.acm.org/citation.cfm?id=1416222.1416290.

[181] B. Varghese und R. Buyya. „Next generation cloud computing: New trends and research directions“. In: *Future Generation Computer Systems* 79 (2018), S. 849–861. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.020. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17302224.

[182] S. Vinoski. „CORBA: integrating diverse applications within distributed heterogeneous environments“. In: *IEEE Communications magazine* 35.2 (1997), S. 46–55.

[183] W. Voorsluys, J. Broberg, S. Venugopal und R. Buyya. „Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation“. In: *Cloud Computing*. Hrsg. von M. G. Jaatun, G. Zhao und C. Rong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 254–265. ISBN: 978-3-642-10665-1.

[184] K. Wada. „Logical Volume Manager“. In: *Encyclopedia of Database Systems*. Springer, 2009, S. 1659–1660.

[185] B. Wang, Z. Qi, R. Ma, H. Guan und A. V. Vasilakos. „A survey on data center networking for cloud computing“. In: *Computer Networks* 91 (2015), S. 528–547.

[186] J. S. Ward und A. Barker. „Observing the clouds: a survey and taxonomy of cloud monitoring“. In: *Journal of Cloud Computing* 3.1 (Dez. 2014), S. 24. ISSN: 2192-113X. DOI: 10.1186/s13677-014-0024-2. URL: https://doi.org/10.1186/s13677-014-0024-2.

[187] S. Warshall. „A Theorem on Boolean Matrices“. In: *J. ACM* 9.1 (Jan. 1962), S. 11–12. ISSN: 0004-5411. DOI: 10.1145/321105.321107. URL: http://doi.acm.org/10.1145/321105.321107.

[188] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long und C. Maltzahn. „Ceph: A scalable, high-performance distributed file system“. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, S. 307–320.

[189] P. Wieder, J. M. Butler, W. Theilmann und R. Yahyapour. *Service Level Agreements for Cloud Computing*. Springer Publishing Company, Incorporated, 2014. ISBN: 1489987754, 9781489987754.

[190] R. Wittmann und M. Zitterbart. *Multicast Communication: Protocols, Programming, & Applications*. Elsevier, 2000. ISBN: 978-1-55860-645-6.

[191] D. Wybranietz. *Multicast-Kommunikation in verteilten Systemen*. Springer-Verlag, 1990. ISBN: 978-3-540-52551-6.

[192] W. Xia, P. Zhao, Y. Wen und H. Xie. „A Survey on Data Center Networking (DCN): Infrastructure and Operations“. In: *IEEE Communications Surveys Tutorials* 19.1 (Firstquarter 2017), S. 640–656. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2626784.

[193] X. Xu, Y. Chen und J. M. Alcaraz Calero. „Distributed decentralized collaborative monitoring architecture for cloud infrastructures“. In: *Cluster Computing* 20.3 (Sep. 2017), S. 2451–2463. ISSN: 1573-7543. DOI: 10.1007/s10586-016-0675-5. URL: https://doi.org/10.1007/s10586-016-0675-5.

[194] C. Yang, Q. Huang, Z. Li, K. Liu und F. Hu. „Big Data and cloud computing: innovation opportunities and challenges“. In: *International Journal of Digital Earth* 10.1 (2017), S. 13–53. DOI: 10.1080/17538947.2016.1239771. eprint: https://doi.org/10.1080/17538947.2016.1239771. URL: https://doi.org/10.1080/17538947.2016.1239771.

[195] L. Youseff, M. Butrico und D. D. Silva. „Toward a Unified Ontology of Cloud Computing“. In: *2008 Grid Computing Environments Workshop*. Nov. 2008, S. 1–10. DOI: 10.1109/GCE.2008.4738443.

[196] F. Zhang, G. Liu, X. Fu und R. Yahyapour. „A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues“. In: *IEEE Communications Surveys Tutorials* 20.2 (Secondquarter 2018), S. 1206–1243. ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2794881.

[197] Q. Zhang, L. Cheng und R. Boutaba. „Cloud computing: state-of-the-art and research challenges“. In: *Journal of Internet Services and Applications* 1.1 (Mai 2010), S. 7–18. ISSN: 1869-0238. DOI: 10.1007/s13174-010-0007-6. URL: https://doi.org/10.1007/s13174-010-0007-6.

[198] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy und T. Anderson. „Understanding and Mitigating Packet Corruption in Data Center Networks“. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Los Angeles, CA, USA: ACM, 2017, S. 362–375. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098849. URL: http://doi.acm.org/10.1145/3098822.3098849.

[199] H. Zimmermann. „OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection“. In: *IEEE Transactions on Communications* 28.4 (Apr. 1980), S. 425–432. ISSN: 0090-6778. DOI: 10.1109/TCOM.1980.1094702.

[200] M. Zitterbart und T. Braun. *Hochleistungskommunikation – Band 2: Transportdienste und -protokolle*. Oldenbourg, 1995. ISBN: 978-3-486-23088-8.

Request for Comments (RFCs)

[201] R. T. Braden, D. A. Borman und C. Partridge. *Computing the Internet checksum*. RFC 1071 (Informational). RFC. Updated by RFC 1141. Fremont, CA, USA: RFC Editor, Sep. 1988. DOI: 10.17487/RFC1071. URL: https://www.rfc-editor.org/rfc/rfc1071.txt.

[202] D. Waitzman, C. Partridge und S. E. Deering. *Distance Vector Multicast Routing Protocol*. RFC 1075 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Nov. 1988. DOI: 10.17487/RFC1075. URL: https://www.rfc-editor.org/rfc/rfc1075.txt.

[203] R. Braden (Ed.) *Requirements for Internet Hosts - Communication Layers*. RFC 1122 (Internet Standard). RFC. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864, 8029. Fremont, CA, USA: RFC Editor, Okt. 1989. DOI: 10.17487/RFC1122. URL: https://www.rfc-editor.org/rfc/rfc1122.txt.

[204] S. Armstrong, A. Freier und K. Marzullo. *Multicast Transport Protocol*. RFC 1301 (Informational). RFC. Fremont, CA, USA: RFC Editor, Feb. 1992. DOI: 10.17487/RFC1301. URL: https://www.rfc-editor.org/rfc/rfc1301.txt.

[205] J. Moy. *Multicast Extensions to OSPF*. RFC 1584 (Historic). RFC. Fremont, CA, USA: RFC Editor, März 1994. DOI: 10.17487/RFC1584. URL: https://www.rfc-editor.org/rfc/rfc1584.txt.

[206] H. Harney und C. Muckenhirn. *Group Key Management Protocol (GKMP) Specification*. RFC 2093 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Juli 1997. DOI: 10.17487/RFC2093. URL: https://www.rfc-editor.org/rfc/rfc2093.txt.

[207] A. Ballardie. *Core Based Trees (CBT) Multicast Routing Architecture*. RFC 2201 (Historic). RFC. Fremont, CA, USA: RFC Editor, Sep. 1997. DOI: 10.17487/RFC2201. URL: https://www.rfc-editor.org/rfc/rfc2201.txt.

[208] S. Deering und R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). RFC. Obsoleted by RFC 8200, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Fremont, CA, USA: RFC Editor, Dez. 1998. DOI: 10.17487/RFC2460. URL: https://www.rfc-editor.org/rfc/rfc2460.txt.

[209] M. Crawford. *Transmission of IPv6 Packets over Ethernet Networks*. RFC 2464 (Proposed Standard). RFC. Updated by RFCs 6085, 8064. Fremont, CA, USA: RFC Editor, Dez. 1998. DOI: 10.17487/RFC2464. URL: https://www.rfc-editor.org/rfc/rfc2464.txt.

[210] C. Adams, P. Cain, D. Pinkas und R. Zuccherato. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. RFC 3161 (Proposed Standard). RFC. Updated by RFC 5816. Fremont, CA, USA: RFC Editor, Aug. 2001. DOI: 10.17487/RFC3161. URL: https://www.rfc-editor.org/rfc/rfc3161.txt.

[211] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera und L. Vicisano. *PGM Reliable Transport Protocol Specification*. RFC 3208 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Dez. 2001. DOI: 10.17487/RFC3208. URL: https://www.rfc-editor.org/rfc/rfc3208.txt.

[212] B. Cain, S. Deering, I. Kouvelas, B. Fenner und A. Thyagarajan. *Internet Group Management Protocol, Version 3*. RFC 3376 (Proposed Standard). RFC. Updated by RFC 4604. Fremont, CA, USA: RFC Editor, Okt. 2002. DOI: 10.17487/RFC3376. URL: https://www.rfc-editor.org/rfc/rfc3376.txt.

[213] M. Crispin. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard). RFC. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858, 7817, 8314, 8437, 8474. Fremont, CA, USA: RFC Editor, März 2003. DOI: 10.17487/RFC3501. URL: https://www.rfc-editor.org/rfc/rfc3501.txt.

[214] T. Kivinen und M. Kojo. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. RFC 3526 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Mai 2003. DOI: 10.17487/RFC3526. URL: https://www.rfc-editor.org/rfc/rfc3526.txt.

[215] D. Whiting, R. Housley und N. Ferguson. *Counter with CBC-MAC (CCM)*. RFC 3610 (Informational). RFC. Fremont, CA, USA: RFC Editor, Sep. 2003. DOI: 10.17487/RFC3610. URL: https://www.rfc-editor.org/rfc/rfc3610.txt.

[216] R. Vida (Ed.) und L. Costa (Ed.) *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. RFC 3810 (Proposed Standard). RFC. Updated by RFC 4604. Fremont, CA, USA: RFC Editor, Juni 2004. DOI: 10.17487/RFC3810. URL: https://www.rfc-editor.org/rfc/rfc3810.txt.

[217] P. Karn (Ed.), C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig, J. Mahdavi, G. Montenegro, J. Touch und L. Wood. *Advice for Internet Subnetwork Designers*. RFC 3819 (Best Current Practice). RFC. Fremont, CA, USA: RFC Editor, Juli 2004. DOI: 10.17487/RFC3819. URL: https://www.rfc-editor.org/rfc/rfc3819.txt.

[218] A. Adams, J. Nicholas und W. Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*. RFC 3973 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Jan. 2005. DOI: 10.17487/RFC3973. URL: https://www.rfc-editor.org/rfc/rfc3973.txt.

[219] Y. Rekhter (Ed.), T. Li (Ed.) und S. Hares (Ed.) *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard). RFC. Updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705, 8212. Fremont, CA, USA: RFC Editor, Jan. 2006. DOI: 10.17487/RFC4271. URL: https://www.rfc-editor.org/rfc/rfc4271.txt.

[220] R. Hinden und S. Deering. *IP Version 6 Addressing Architecture*. RFC 4291 (Draft Standard). RFC. Updated by RFCs 5952, 6052, 7136, 7346, 7371, 8064. Fremont, CA, USA: RFC Editor, Feb. 2006. DOI: 10.17487/RFC4291. URL: https://www.rfc-editor.org/rfc/rfc4291.txt.

[221] M. Pullen, F. Zhao und D. Cohen. *Selectively Reliable Multicast Protocol (SRMP)*. RFC 4410 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Feb. 2006. DOI: 10.17487/RFC4410. URL: https://www.rfc-editor.org/rfc/rfc4410.txt.

[222] A. Conta, S. Deering und M. Gupta (Ed.) *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443 (Internet Standard). RFC. Updated by RFC 4884. Fremont, CA, USA: RFC Editor, März 2006. DOI: 10.17487/RFC4443. URL: https://www.rfc-editor.org/rfc/rfc4443.txt.

[223] M. Christensen, K. Kimball und F. Solensky. *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*. RFC 4541 (Informational). RFC. Fremont, CA, USA: RFC Editor, Mai 2006. DOI: 10.17487/RFC4541. URL: https://www.rfc-editor.org/rfc/rfc4541.txt.

[224] H. Holbrook, B. Cain und B. Haberman. *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*. RFC 4604 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2006. DOI: 10.17487/RFC4604. URL: https://www.rfc-editor.org/rfc/rfc4604.txt.

[225] P. Savola. *Overview of the Internet Multicast Routing Architecture*. RFC 5110 (Informational). RFC. Fremont, CA, USA: RFC Editor, Jan. 2008. DOI: 10.17487/RFC5110. URL: https://www.rfc-editor.org/rfc/rfc5110.txt.

[226] R. Thurlow. *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 5531 (Draft Standard). RFC. Fremont, CA, USA: RFC Editor, Mai 2009. DOI: 10.17487/RFC5531. URL: https://www.rfc-editor.org/rfc/rfc5531.txt.

[227] R. Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, Sep. 2009. DOI: 10.17487/RFC5652. URL: https://www.rfc-editor.org/rfc/rfc5652.txt.

[228] B. Adamson, C. Bormann, M. Handley und J. Macker. *NACK-Oriented Reliable Multicast (NORM) Transport Protocol*. RFC 5740 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Nov. 2009. DOI: 10.17487/RFC5740. URL: https://www.rfc-editor.org/rfc/rfc5740.txt.

[229] S. Gundavelli, M. Townsley, O. Troan und W. Dec. *Address Mapping of IPv6 Multicast Packets on Ethernet*. RFC 6085 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Jan. 2011. DOI: 10.17487/RFC6085. URL: https://www.rfc-editor.org/rfc/rfc6085.txt.

[230] V. Paxson, M. Allman, J. Chu und M. Sargent. *Computing TCP's Retransmission Timer*. RFC 6298 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Juni 2011. DOI: 10.17487/RFC6298. URL: https://www.rfc-editor.org/rfc/rfc6298.txt.

[231] M. Watson, A. Begen und V. Roca. *Forward Error Correction (FEC) Framework*. RFC 6363 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Okt. 2011. DOI: 10.17487/RFC6363. URL: https://www.rfc-editor.org/rfc/rfc6363.txt.

[232] J. E. White. *High-level framework for network-based resource sharing*. RFC 707. RFC. Fremont, CA, USA: RFC Editor, Dez. 1975. DOI: 10.17487/RFC0707. URL: https://www.rfc-editor.org/rfc/rfc707.txt.

[233] R. Fielding (Ed.) und J. Reschke (Ed.) *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Juni 2014. DOI: 10.17487/RFC7230. URL: https://www.rfc-editor.org/rfc/rfc7230.txt.

[234] R. Droms. *IPv6 Multicast Address Scopes*. RFC 7346 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2014. DOI: 10.17487/RFC7346. URL: https://www.rfc-editor.org/rfc/rfc7346.txt.

[235] M. Belshe, R. Peon und M. Thomson (Ed.) *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Mai 2015. DOI: 10.17487/RFC7540. URL: https://www.rfc-editor.org/rfc/rfc7540.txt.

[236] J. Halpern (Ed.) und C. Pignataro (Ed.) *Service Function Chaining (SFC) Architecture*. RFC 7665 (Informational). RFC. Fremont, CA, USA: RFC Editor, Okt. 2015. DOI: 10.17487/RFC7665. URL: https://www.rfc-editor.org/rfc/rfc7665.txt.

[237] J. Postel. *User Datagram Protocol*. RFC 768 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/rfc/rfc768.txt.

[238] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang und L. Zheng. *Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)*. RFC 7761 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, März 2016. DOI: 10.17487/RFC7761. URL: https://www.rfc-editor.org/rfc/rfc7761.txt.

[239] J. Postel. *Internet Protocol*. RFC 791 (Internet Standard). RFC. Updated by RFCs 1349, 2474, 6864. Fremont, CA, USA: RFC Editor, Sep. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/rfc/rfc791.txt.

[240] J. Postel. *Transmission Control Protocol*. RFC 793 (Internet Standard). RFC. Updated by RFCs 1122, 3168, 6093, 6528. Fremont, CA, USA: RFC Editor, Sep. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/rfc/rfc793.txt.

[241] L. Eggert, G. Fairhurst und G. Shepherd. *UDP Usage Guidelines*. RFC 8085 (Best Current Practice). RFC. Fremont, CA, USA: RFC Editor, März 2017. DOI: 10.17487/RFC8085. URL: https://www.rfc-editor.org/rfc/rfc8085.txt.

[242] S. Deering und R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, Juli 2017. DOI: 10.17487/RFC8200. URL: https://www.rfc-editor.org/rfc/rfc8200.txt.

[243] D. Plummer. *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. RFC 826 (Internet Standard). RFC. Updated by RFCs 5227, 5494. Fremont, CA, USA: RFC Editor, Nov. 1982. DOI: 10.17487/RFC0826. URL: https://www.rfc-editor.org/rfc/rfc826.txt.

[244] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446.txt.

Weitere Referenzen

[245] *Amazon EC2 – A web service that provides secure, resizable compute capacity in the cloud*. https://aws.amazon.com/ec2/. Abgerufen am 08.02.2018.

[246] *Amazon EC2 Dedicated Hosts – A physical server with EC2 instance capacity fully dedicated to your use*. https://aws.amazon.com/ec2/dedicated-hosts//. Abgerufen am 08.02.2018.

[247] *Apache Kafka*. https://kafka.apache.org/. Abgerufen am 12.06.2018.

[248] *AUDITOR - European Cloud Service Data Protection Certification*. http://www.auditor-cert.de/. Abgerufen am 08.08.2018.

[249] *AWS Compliance*. https://aws.amazon.com/compliance/. Abgerufen am 08.08.2018.

[250] *AWS Elastic Beanstalk – An easy-to-use service for deploying and scaling web applications and services*. https://aws.amazon.com/elasticbeanstalk/. Abgerufen am 08.02.2018.

[251] *Bundesanzeiger*. https://www.bundesanzeiger.de/. Abgerufen am 08.02.2018.

[252] *Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability*. https://ceph.com/. Abgerufen am 08.02.2018.

[253] *Cinder, the OpenStack Block Storage Service*. https://docs.openstack.org/cinder/latest/. Abgerufen am 08.02.2018.

[254] *Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper*. https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019. Abgerufen am 04.12.2018.

[255] *Cloudscape Germany*. http://www.themetisfiles.com/2014/09/cloudscape-germany/. Abgerufen am 08.02.2018.

[256] *Common Criteria for Information Technology Security Evaluation*. https://www.commoncriteriaportal.org/cc/. Abgerufen am 08.08.2018.

[257] *CSA Security, Trust & Assurance Registry (STAR)*. https://cloudsecurityalliance.org/star/. Abgerufen am 08.08.2018.

[258] *CSA/ISACA Cloud Market Maturity – Study Results*. https://downloads.cloudsecurityalliance.org/initiatives/collaborate/isaca/2012-Cloud-Computing-Market-Maturity-Study-Results.pdf. Abgerufen am 28.11.2018.

[259] *DataCenter Knowledge – Microsoft Unveils Its Container-Powered Cloud*. https://www.datacenterknowledge.com/archives/2009/09/30/microsoft-unveils-its-container-powered-cloud. Abgerufen am 04.12.2018.

[260] *Distributed Component Object Model (DCOM) Remote Protocol Specification*. https://msdn.microsoft.com/library/cc201989.aspx. Abgerufen am 08.02.2018.

[261] *Dropbox – The secure file sharing and storage solution that employees and IT admins trust*. https://www.dropbox.com/. Abgerufen am 08.02.2018.

[262] *Electra Object Request Broker*. http://www.maffeis.com/electra.html. Abgerufen am 28.11.2018.

[263] *Forbes – 83% Of Enterprise Workloads Will Be In The Cloud By 2020*. https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/. Abgerufen am 04.12.2018.

[264] *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019*. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html. Abgerufen am 07.12.2018.

[265] *Geek – Just how big is Amazon's AWS business? (hint: it's absolutely massive)*. https://www.geek.com/chips/just-how-big-is-amazons-aws-business-hint-its-absolutely-massive-1610221/. Abgerufen am 04.12.2018.

[266] *Google App Engine – Build scalable web and mobile backends in any language on Google's infrastructure*. https://cloud.google.com/appengine/. Abgerufen am 08.02.2018.

[267] *Heat – OpenStack Orchestration*. https://docs.openstack.org/heat/latest/. Abgerufen am 08.02.2018.

[268] *IBM Cloud Dedicated – An isolated and managed IBM Cloud environment with a curated catalog of cloud services*. https://www.ibm.com/cloud/dedicated. Abgerufen am 09.12.2018.

[269] *IEEE 802.3 Ethernet Working Group*. http://www.ieee802.org/3/. Abgerufen am 04.03.2018.

[270] *INET Framework – An open-source OMNeT++ model suite for wired, wireless and mobile networks*. https://inet.omnetpp.org/. Abgerufen am 08.02.2018.

[271] *Institut für Informationswirtschaft und Marketing (IISM) – Information & Market Engineering – Prof. Dr. Christof Weinhardt*. https://im.iism.kit.edu/. Abgerufen am 08.02.2018.

[272] *Intel Software Guard Extensions (Intel SGX)*. https://software.intel.com/en-us/sgx. Abgerufen am 08.02.2018.

[273] *JBoss Community Documentation – A Pub-Sub Example*. https://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/JMS_Examples-A_Pub_Sub_Example.html. Abgerufen am 12.06.2018.

[274] *JGroups*. http://www.jgroups.org/. Abgerufen am 12.06.2018.

[275] *KB2133118: Virtual machine fails unexpectedly after a snapshot consolidation task (2133118)*. https://kb.vmware.com/s/article/2133118. Abgerufen am 08.02.2018.

[276] KPMG. *Cloud-Monitor 2018: Strategien für eine zukunftsorientierte Cloud Security und Cloud Compliance*. https://klardenker.kpmg.de/cloud-monitor-2018-strategien-fuer-cloud-security-und-cloud-compliance/. Abgerufen am 28.11.2018.

[277] *KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions*. https://www.linux-kvm.org. Abgerufen am 08.02.2018.

[278] *libvirt – Domain XML format*. https://libvirt.org/formatdomain.html. Abgerufen am 08.02.2018.

[279] *libvirt: The virtualization API*. https://libvirt.org/. Abgerufen am 08.02.2018.

[280] *Microsoft Azure – An open, flexible, enterprise-grade cloud computing platform*. https://azure.microsoft.com/. Abgerufen am 08.02.2018.

[281] *Mininet – An Instant Virtual Network on your Laptop (or other PC)*. http://mininet.org/. Abgerufen am 08.02.2018.

[282] *Nanomsg – nn_reqrep(7) Manual Page*. http://nanomsg.org/v1.1.0/nn_reqrep.html/. Abgerufen am 08.02.2018.

[283] *Nanomsg – socket library that provides several common communication patterns*. http://nanomsg.org/. Abgerufen am 08.02.2018.

[284] *Nanomsg – survey(7) Manual Page*. https://nanomsg.org/v1.1.3/nn_survey.html. Abgerufen am 12.06.2018.

[285] *Nimbus API is a .NET API for building messaging applications with Azure Service Bus, Windows Service Bus, and Redis*. http://nimbusapi.com/. Abgerufen am 08.02.2018.

[286] *Nova – Hypervisor Feature Capability Matrix*. https://github.com/openstack/nova/blob/master/doc/source/user/support-matrix.ini. Abgerufen am 08.02.2018.

[287] *Nova, OpenStack Compute Service*. https://docs.openstack.org/nova/latest/. Abgerufen am 08.02.2018.

[288] *Office 365 – Microsoft Office*. https://www.office.com/. Abgerufen am 08.02.2018.

[289] *OMNeT++ Discrete Event Simulator*. http://omnetpp.org/. Abgerufen am 08.02.2018.

[290] *Open Platform for NFV (OPNFV)*. https://www.opnfv.org/. Abgerufen am 08.03.2018.

[291] *OpenStack – AMQP and Nova*. https://docs.openstack.org/nova/latest/reference/rpc.html. Abgerufen am 08.02.2018.

[292] *OpenStack – Open source software for creating private and public clouds*. https://www.openstack.org/. Abgerufen am 08.02.2018.

[293] *OpenStack Block Storage (aka Cinder) Drivers*. https://wiki.openstack.org/wiki/CinderSupportMatrix. Abgerufen am 08.02.2018.

[294] *OpenStack Nova Remote Procedure Call*. https://github.com/openstack/nova/blob/master/nova/rpc.py. Abgerufen am 08.02.2018.

[295] *OvS Open vSwitch: Production Quality, Multilayer Open Virtual Switch*. https://www.openvswitch.org/. Abgerufen am 08.03.2018.

[296] *Python library for working with Open vSwitch*. https://github.com/openvswitch/ovs/tree/master/python. Abgerufen am 08.03.2018.

[297] *pyVmomi is the Python SDK for the VMware vSphere API that allows you to manage ESX, ESXi, and vCenter*. https://github.com/vmware/pyvmomi. Abgerufen am 08.02.2018.

[298] *QEMU is a generic and open source machine emulator and virtualizer*. https://www.qemu.org/. Abgerufen am 08.02.2018.

[299] *RabbitMQ – The most widely deployed automation and operator friendly open source message broker*. https://www.rabbitmq.com/. Abgerufen am 08.02.2018.

[300] *SECCRIT - SEcure Cloud computing for CRitical infrastructure IT*. http://seccrit.eu/. Abgerufen am 08.02.2018.

[301] *SECCRIT – CloudInspector Reference Use License*. https://telematics.tm.kit.edu/3872_seccrit.php. Abgerufen am 09.12.2018.

[302] A. Selzer. *Die Kontrollpflicht nach § 11 Abs. 2 Satz 4 BDSG im Zeitalter des Cloud Computing*. https://www.sit.fraunhofer.de/fileadmin/dokumente/artikel/DuD_Selzer_ADV_Testat.pdf. Abgerufen am 08.02.2018.

[303] *StarAudit*. https://staraudit.org/. Abgerufen am 08.08.2018.

[304] *Transport Message Exchange Pattern: Single-Request-Response*. https://www.w3.org/2000/xp/Group/1/10/11/2001-10-11-SRR-Transport_MEP. Abgerufen am 22.03.2018.

[305] *vCloud Director REST API Reference Documentation*. https://www.vmware.com/support/vcd/doc/rest-api-doc-1.5-html/. Abgerufen am 08.02.2018.

[306] *VMware vCenter Server – Centralized visibility, proactive management and extensibility for VMware vSphere from a single console*. https://www.vmware.com/products/vcenter-server.html. Abgerufen am 08.02.2018.

[307] *VMWare vCloud Director – Deliver Virtual Data Centers*. https://www.vmware.com/products/vcloud-director.html. Abgerufen am 08.02.2018.

[308] *VMware vSphere 6.0 Documentation Center: Data Object - VirtualMachineConfigSummary*. https://pubs.vmware.com/vsphere-6-0/index.jsp?topic=%2Fcom.vmware.wssdk.smssdk.doc%2Fvim.vm.Summary.ConfigSummary.html. Abgerufen am 08.02.2018.

[309] *VMware vSphere Hypervisor (ESXi) – A free bare-metal hypervisor that virtualizes servers so you can consolidate your applications on less hardware*. https://www.vmware.com/products/vsphere-hypervisor.html. Abgerufen am 08.02.2018.

[310] *vSphere and vSphere with Operations Management – The best foundation for your apps, your cloud and your business*. https://www.vmware.com/products/vsphere.html. Abgerufen am 08.02.2018.

[311] *Xen – Advisories, publicly released or pre-released*. https://xenbits.xen.org/xsa/. Abgerufen am 19.02.2018.

[312] *Xen Source Repositories*. https://xenbits.xen.org/. Abgerufen am 19.02.2018.

[313] *ZeroMQ – Distributed Messaging*. http://zeromq.org/. Abgerufen am 08.02.2018.

[314] *ZeroMQ – The Guide*. http://zguide.zeromq.org/page:all. Abgerufen am 08.02.2018.

[315] *ZeroMQ zguide – Chapter 5 - Advanced Pub-Sub Patterns*. http://zguide.zeromq.org/php:chapter5. Abgerufen am 12.06.2018.