Peer-to-Peer-Technologie in der Automatisierung

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Skodzik, Jan, geb. am 25.01.1985 in Neubrandenburg,
aus Rostock

Rostock, 04.02.2015

urn:nbn:de:gbv:28-diss2015-0090-4

Gutachter:

- Prof. Dr.-Ing. Dirk Timmermann (Universität Rostock, Fakultät für Informatik und Elektrotechnik, Institut MD)
- Prof. Dr.-Ing. Norbert Stoll (Universität Rostock, Fakultät für Informatik und Elektrotechnik, Institut Automatisierungstechnik)
- Prof. Dr. rer. nat. Stefan Fischer (Universität zu Lübeck, Sektion Informatik / Technik, Institut für Telematik)

Tag der Einreichung: 04.02.2015
Tag der Verteidigung: 29.05.2015

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

AN	Access Node
ARP	Address Resolution Protocol
CIP	Common Industrial Protocol
CoAP	Constrained Application Protocol
CoHaRT	CoAP HaRTKad
CRC	Cyclic Redundancy Check
(D)DOS	(Distributed) Denial of Service
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DPWS	Device Profile for Web Service
DR	Datenreplikation
DSLAM	Digital Subscriber Line Access Multiplexer
DST	Dynamische Suchtoleranz
ERC	Erasure Resilient Code
ESC	EtherCAT Slave Controller
EXI	Efficient XML Interchange
FCS	Frame Check Sequence
FT-Knoten	First Triggered-Knoten
GB	Gigabyte
GUI	Grafische Benutzerschnittstelle (engl.:Graphical User Interface)
HaRTKad	Hard Real-Time Kademlia
HT-Knoten	Helfende Knoten
IDST	Inverse dynamische Suchtoleranz
IE	Industrial Ethernet
IHL	Internet Header Length
IIoT	Industrial Internet der Dinge
IoT	Internet der Dinge
IP	Internet Protocol
IPv4	IP Version 4
IPv6	IP Version 6
IRT	Isochronous Echtzeit
ISO	International Organization for Standardization
ISP	Internet Service Provider
JMS	Java Message Service
KaDis	Kademlia Discovery
KaDisSy	Kademlia Discovery and Synchronisation
KB	Kilobyte
LWL	Lichwellenleiter
MB	Megabyte
MD4	Message-Digest-Algorithmus 4
MD5	Message-Digest-Algorithmus 5
NDP	Neighbor Discovery Protocol
NTP	Network Time Protocol

OSI	Open Systems Interconnection Model
P2P	Peer-to-Peer
PDP	Prozessdatenprotokoll
PSP	P2P-based Storage Platform
PSP-Auto	PSP-Automation
PTP	Precision Time Protocol
RAM	Random Access Memory
RS-Code	Reed-Solomon-Code
REST	Representational state transfer
RTT	Round Trip Time
SD	Sessiondaten
SDN	Software Defined Network
SFD	Start of Frame Delimiter
SOA	Service orientierten Architektur
SOAP	Simple Object Access Protocol
SPoF	Single Point of Failure
TB	Terabyte
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
ToS	Type of Service
TSN	Time Sensitive Networking
TTE	Time Trigger Event
TTL	Time-to-Live
TLV	Type-Length-Value
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W7-X	Wendelstein 7-X
WS	Web Service
WSDL	Web Services Description Language
XML	Extensible Markup Language

Danksagung

An dieser Stelle möchte ich die Gelegenheit ergreifen, mich bei meinen Eltern und der gesamten Familie zu bedanken, die mich stets aufopferungsvoll bei meinem nun bereits 24-jährigen Bildungsweg unterstützt hat. Ein besonderer Dank gilt auch meiner Freundin Melanie Voß, die mich ebenfalls immer unterstützt hat und das schon seit dem ersten Semester als Student der Informationstechnik. Ohne diese Unterstützung wäre die Verfassung dieser Forschungsarbeit nicht möglich gewesen. Mein großer Dank gilt weiterhin meinem Doktorvater Professor Timmermann, der durch meine Anstellung am Institut für Angewandte Mikroelektronik und Datentechnik die Grundlage für die Erstellung dieser Forschungsarbeit schuf. Er half mir Tiefs und Schwierigkeiten zu meistern, motivierte mich stets und trug damit maßgeblich zu dem Gelingen dieser Forschungsarbeit bei. Darüber hinaus danke ich Peter Danielis, Vlado Altmann, Sebatian Unger, Tim Wegner und Benjamin Beichler, die diese Arbeit fleißig Korrektur gelesen haben, sowie allen Kollegen und Studenten, die an dieser Arbeit beteiligt sind.

Dabei möchte ich besonders Peter Danielis, meinen Zimmerkollegen Vlado Altmann sowie Jörg Schacht vom MPI nennen. Schlussendlich richte ich ein großes Dankeschön an alle Mitarbeiter des Instituts für Angewandte Mikroelektronik und Datentechnik, die mir ein Arbeiten in freundlicher und netter Atmosphäre ermöglichten.

Kapitel 1.

Einleitung

Das Internet ist nicht mehr aus dem heutigen Leben wegzudenken. Anfänglich zum Austausch von Nachrichten zwischen wenigen Nutzern gedacht, ist es heute ein global agierendes Netzwerk. Das Besondere ist dabei, dass nicht nur einzelne Rechner und Server diesem Netzwerk angehören, sondern auch mobile Geräte wie Smartphones. So ist es möglich, dass wir durch unser digitales Ich immer präsent sein können. Wir haben zu jeder Zeit Zugriff auf das Internet und liefern auch Informationen in Form von Wissen und anderen Daten. Was ist, wenn diese Informationen nicht nur vom Menschen kommen, sondern auch von Maschinen, wie z.B. dem eigenen Kühlschrank? Wenn dies der Fall ist, spricht man vom Internet der Dinge (IoT): dies sind Geräte, die über das Internet mittels Internet Protocol (IP)-Adresse ansprechbar sind und ebenfalls Informationen austauschen. In der Zukunft wird es immer mehr Geräte geben, die dem Internet angebunden sind und so ihre Funktionalität erst nutzbar machen. Ein Gebiet, in dem IoT bereits starken Einzug gehalten hat, ist Smart Home, in dem der Kühlschrank bereits mit dem Internet verbunden ist und auch dem Nutzer Dienste bereitstellt [Ves14]. Das Einsatzgebiet ist mannigfaltig und kaum einschränkbar, insbesondere da die Geräte selbstständig miteinander kommunizieren können. Ein weiteres Gebiet ist das Automatisierungsumfeld und zwar im industriellen Umfeld. Zum einen gibt es hier das Synonym Industrial Internet of Things (IIoT), das eben genau den Trend beschreibt, Geräte (insbesondere eingebettete Systeme aus der industriellen Automatisierung) mittels IP zu vernetzen und somit in das bestehende Internet integrierbar zu machen. Ein weiteres auf die Automatisierungstechnik zugeschnittenes Synonym ist Industrie 4.0. Industrie 4.0 als Zukunftsprojekt sieht sich als vierte industrielle Revolution nach der Dampfmaschine, der Massenfertigung durch Fließbänder und der digitalen Revolution zur Intensivierung

der Automatisierung [FWW13]. Mittels Industrie 4.0 soll es möglich sein, auch kleine Produktionsmengen flexibel und gewinnbringend produzieren zu können. Dies soll mit hoher Selbstkonfiguration, -optimierung und -analyse gelingen. IoT ist dabei eine Technologie, um Industrie 4.0 zu realisieren, da eine Kommunikation von einem Gerät bzw. Produkt zu jedem anderen ermöglicht werden soll. Die Anzahl der Geräte soll in Zukunft enorm ansteigen. Erste Untersuchungen prognostizieren bis zu 50 Milliarden verbundene Geräte im Jahre 2020 [Eva11], wohingegen auch die Anzahl der Geräte im industriellen Umfeld stark zunimmt [EA12]. Doch nicht nur die Anzahl der Geräte ist von entscheidender Bedeutung, auch der Fakt, dass die Geräte immer intelligenter werden. Die Frage, die sich jedoch stellt, ist, inwiefern bereits bestehende Lösungen, insbesondere im Automatisierungsumfeld, die zukünftig nötigen Anforderungen erfüllen.

1.1. Die Zielsetzung dieser Arbeit

Diese Arbeit stellt sich genau den Fragen, inwiefern bestehende Lösungen für zukünftige Anforderungen gewappnet sind und welche Alternativen sich anbieten. Im Besonderen fällt dabei auf, dass die meisten bestehenden Lösungen in der Automatisierung auf dem alten Master/Slave oder Client-Server-Modell basieren. Diese weisen jedoch bekannte Schwachstellen in Bezug auf Skalierbarkeit, Flexibilität, Selbstorganisation und Ausfallsicherheit auf. Alle diese Aspekte sind Anforderungen von Industrial Internet of Things und Industrie 4.0 bzw. den darunterliegenden Technologien. Ein Netzwerkparadigma, welches diese Eigenschaften hingegen besitzt, ist Peer-to-Peer (P2P). P2P als bekanntes und etabliertes Netzwerkparadigma wird bereits in anderen Bereichen, wie dem Streaming von Videodaten, angewendet. Deshalb wurde untersucht, inwiefern und mit welcher Leistungsfähigkeit sich P2P-Netzwerke auch im Umfeld der Automatisierung und insbesondere der industriellen Automatisierung einsetzen lassen. Ziel ist es, ein konsistentes System zu erstellen, welches für den Nutzer auch transparent ist. Das erfordert, verschiedenste Aspekte eines solchen Systems zu untersuchen und zu bearbeiten. Der Hauptaspekt ist die Echtzeit, die als Anforderung an Systeme in diesem Umfeld gestellt wird. Das betrifft die Echtzeitfähigkeit eines einzelnen Systems (Echtzeitsystem) und die echtzeitfähige und deterministische Kommunikation (Echtzeitkommunikation) zwischen den Systemen. Da P2P-Netzwerke per se nicht echtzeitfähig sind, muss ein geeignetes P2P-Netzwerk ausgewählt und untersucht werden, ob und wie es für Echtzeitanforderun-

gen erweitert werden kann. Je nach Wahl der Konzepte sind andere Herausforderungen zu lösen, wie z.B. die Synchronisation eines P2P-Netzwerkes. Diese Arbeit soll zeigen, ob und mit welcher Performance ein auf P2P-Technologie-basierendes System in der Automatisierung eingesetzt werden kann, um die genannten zukünftigen Herausforderungen besser als mit dem aktuellen Stand der Technik bewältigen zu können.

1.2. Aufbau der Arbeit

Diese Arbeit besteht aus sechs Kapiteln. Kapitel 1 umfasst eine Einleitung und die Zielsetzung dieser Arbeit. Es folgen die Grundlagen in Kapitel 2, welches die netzwerktechnischen, Echtzeit- und Peer-to-Peer-Grundlagen umfasst. Kapitel 3 stellt Simulationsergebnisse eines Systems für das verteilte Speichern von Daten dar, das ebenfalls für das Automatisierungsumfeld angepasst wurde. Das vierte Kapitel beschreibt zuerst den Stand der Technik bzgl. aktueller vernetzter Systeme im industriellen Umfeld und stellt dann das entwickelte HaRTKad-Verfahren mit Synchronisierung vor. Im fünften Kapitel werden zwei Applikationen präsentiert, welche in Verbindung mit HaRTKad als Middleware im Umfeld der Automatisierung eingesetzt werden können. Abschließend folgt in Kapitel 6 eine Zusammenfassung und ein Ausblick.

In Abbildung 1.1 ist eine Übersicht der Dissertation aufgeführt. Die eigenen Hauptbeiträge sind fett hervorgehoben.

Abbildung 1.1.: Struktur der Kapitel und deren Inhalt.

Kapitel 2.

Grundlagen

In diesem Kapitel werden die netzwerktechnischen Grundlagen erläutert, die für diese Arbeit benötigt werden. Dies umfasst insbesondere kabelgebundene Netzwerke, weil als Grundlage dieser Arbeit Ethernet verwendet wurde. Zusätzlich werden Prinzipien und Protokolle oberhalb von Ethernet erörtert, welche in dieser Arbeit Verwendung finden. Da der Einsatz der Lösungen in der Automatisierung angestrebt wird, werden auch die Grundlagen von Echtzeitsystemen dargelegt. Abschließend werden P2P-Netzwerke erläutert, die die Grundlage aller Realisierungen und dessen Erweiterung um Echtzeitfähigkeit den Innovationskern darstellen.

2.1. Netzwerktechnische Grundlagen

Die netzwerktechnischen Grundlagen lassen sich anhand des International Organization for Standardization (ISO)/Open Systems Interconnection Model (OSI)-Referenzmodells erörtern (siehe Abbildung 2.1). Zusätzlich wurde das in der Praxis verbreitete TCP/IP-Modell dargestellt, welches aus vier Schichten besteht [TW14]. Ergänzend wurden repräsentative Protokollbeispiele für die jeweilige Schicht aufgeführt. Insgesamt existieren sieben Schichten, die kurz erläutert werden:

1. Schicht: Auf der **Bitübertragungsschicht** (engl. physical layer) wird eine Definition für die Übertragung von logischen Bits auf einem Übertragungskanal beschrieben. Zwei Kommunikationspartner müssen die Signale auf dem physikalischen Medium richtig interpretieren, damit ein korrekter Informationsaustausch ermöglicht wird. Schnittstellen zwischen den verschiedenen Übertragungsmedien und Netzwerkgeräten werden darge-

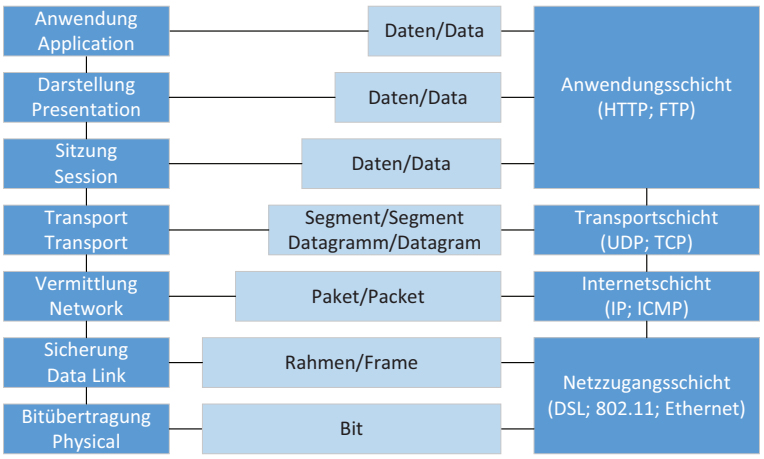

Abbildung 2.1.: Darstellung des ISO/OSI-Schichtenmodells.

legt. Die physikalische Beschreibung des Übertragungsmediums ist ebenfalls Teil dieser Schicht.

2. Schicht: Die **Sicherungsschicht** (engl. data link layer) sichert die Kommunikation mittels Rahmen (engl. frames) ab. Zur Erkennung und damit Absicherung von Fehlern werden z.B. Bitstuffing und Prüfsummen verwendet.

3. Schicht: In der **Vermittlungsschicht** (engl. network layer) werden die Daten innerhalb von Paketen übertragen. Die Pakete lassen sich über Adressen durch ein gegebenes Netzwerk leiten, was als Routing bezeichnet wird. Die in dieser Schicht enthaltenen Routingprotokolle ermöglichen das Routen in großen Netzwerken. Ein Einfluss auf die Dienstgüte ist mittels Fluss- und Überlastkontrollen in dieser Schicht bereits möglich.

4. Schicht: Die zuverlässige Übertragung erfolgt in der **Transportschicht** (engl. transport layer). Es kann eine Ende-zu-Ende-Auslieferung von Segmenten mittels verbindungsorientierter Verbindung realisiert werden. Die zuverlässige Übertragung wird durch Flusskontroll- und Fehlerbehebungsmaßnahmen sichergestellt. Zusätzlich wird auch eine nicht zuverlässige und verbindungslose Verbindung definiert, welche Datagramme anstatt Segmente übermittelt.

5. Schicht: Übertragungssitzungen zwischen mehreren Endsystemen ermöglicht die **Sitzungsschicht** (engl. session layer). Sicherheitsmerkmale, wie verwendete Passworttechniken sowie Synchronisation zwischen Rechner, werden hier vermerkt. Ziel ist es, bei

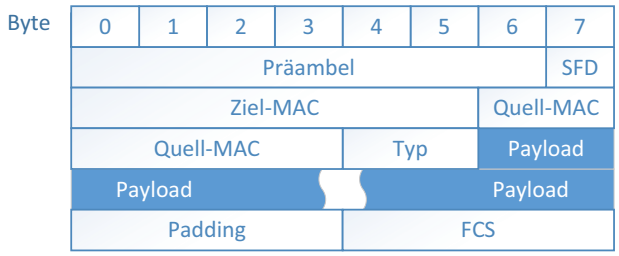

Abbildung 2.2.: Aufbau eines Ethernet-Frames.

einem Ausfall eines Teilnehmers die Sitzung wieder fortführen zu können.

6. Schicht: Da unterschiedliche Rechner Daten intern unterschiedlich darstellen können, wird die **Darstellungsschicht** (engl. presentation layer) benötigt. Mittels der Darstellungsschicht werden die Daten eines Rechners passend für einen anderen Rechner übersetzt, sodass eine verständliche Darstellung und Interpretation der Daten ermöglicht wird.

7. Schicht: Die **Anwendungsschicht** besteht aus einer großen Vielzahl an Protokollen, welche bestimmen, wie eine Anwendung das Netz nutzt. Sie interagiert direkt mit einer Anwendung, welche Netzwerkressourcen benötigt.

Bis auf die unterste Schicht kommunizieren die Teilnehmer virtuell miteinander. In der untersten Schicht hingegen findet eine reale physikalische Übertragung auf dem Übertragungsmedium statt [TW14], [Com98], [Sch09]. Im Folgenden werden weitere spezielle Protokolle und Prinzipien erörtert, welche für diese Arbeit verwendet wurden.

2.1.1. Ethernet

Ethernet für die drahtgebundene Übertragung ist eines der verbreitesten Netzwerkprotokolle in der Netzzugangsschicht des TCP/IP-Modelles zur Übertragung von Daten. Der Erfolg dieser sich immer weiter entwickelnden Technologie wird dadurch deutlich, dass auch konventionelle Einsatzgebiete wie die Automatisierung immer mehr auf Ethernet zurückgreifen [Net12].

In Abbildung 2.2 ist der exemplarische Aufbau eines Ethernet-Frames dargestellt. Ethernet umfasst sowohl eine Beschreibung der Hardware als auch Software zur Übertragung von Daten. Im Folgenden wird der Fokus auf den Softwareanteil und dessen Spezifikation

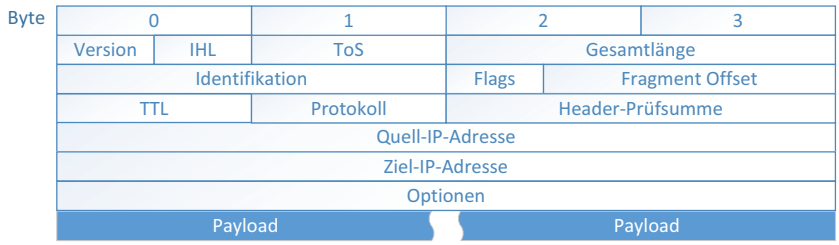

Abbildung 2.3.: Aufbau eines IPv4-Paketes.

gelegt. Ethernet wurde in dem IEEE Standard 802.3 spezifiziert [IEE]. Der Frame beginnt mit einer 7-Byte-Präambel, welche zur Synchronisierung des Empfängers genutzt wird. Dies wird durch das Verwenden des Bitmusters 10101010 erreicht. Es folgt eine Startkennung (Start of Frame Delimiter (SFD)) für den Frame, woraufhin der Header folgt. Zunächst werden die Ziel- und Quell-MAC-Adresse angegeben, welche jeweils 6 Byte umfassen. Das 2 Byte lang folgende Typ-Feld gibt an, welches Protokoll sich anschließt (z.B. 0x0800 für das IPv4 Protokoll). Als nächstes folgt die Payload, in der die Daten des Frames übertragen werden. Die Payload kann eine Größe von 0-1500 Byte haben, wobei ein Ethernet-Frame mindestens 64 und maximal 1518 Byte lang sein darf. Wird ein zusätzliches VLAN-Tag zur Erstellung von virtuellen Netzen verwendet, darf die maximale Größe 1522 Byte betragen. Die Größenvorgabe umfasst die Präambel und den SFD nicht. Sollte ein Frame inklusive Payload zu klein sein, wird er mit Padding-Bytes aufgefüllt, um die Minimalgröße zu erreichen, was als Padding bezeichnet wird. Den Abschluss bildet die Frame Check Sequence (FCS), welche eine Cyclic Redundancy Check (CRC)-Summe umfasst. Dieser 32-Bit-Wert wird genutzt, um Fehler im Ethernet-Frame beim Empfänger erkennen zu können.

2.1.2. IP

Das Internet Protokoll (IP) ermöglicht das Routing durch Netzwerke. Am meisten verbreitet ist nach wie vor IP in der Version 4 (IPv4), was die Zugriffe auf Google belegen. 2014 greifen vorerst nur ca. 3 % weltweit auf Google mittels IP in der Version 6 (IPv6) zu. In Deutschland beträgt der Wert ca. 8 % [Goo14]. In dieser Arbeit wurde in der praktischen Umsetzung nur IPv4 genutzt, sodass in den Grundlagen nur Bezug auf IPv4 genommen wird. Generell ist aber eine Verwendung von IPv6 ebenso möglich.

Abbildung 2.4.: Aufbau eines UDP-Datagrammes.

In Abbildung 2.3 ist der exemplarische Aufbau eines IPv4-Paketes dargestellt, welches der Internetschicht zuzuordnen ist. Definiert ist der Aufbau im RFC 791 [rfc81]. Im Versions-Feld ist die verwendete Version von IP angegeben. Für IPv4 ist hier der Wert 4 vermerkt. Im Internet Header Length (IHL)-Feld ist die Länge des Headers in Vielfachen von 32-Bit-Wörtern vermerkt. Das Type of Service (ToS)-Feld kann für die Quality of Service-Eigenschaften genutzt werden, da hier Flusskontrollen und Priorisierungen von Paketen möglich sind. IP-Pakete können auch fragmentiert werden, falls diese zu groß für ein Übertragungsmedium sind. Zur Einstellung und Verwaltung der Fragmente werden die Felder Identifikation, Flags und Fragment Offset genutzt. Der betriebssystemabhängige Time-to-Live (TTL)-Wert bestimmt, wie viele Stationen ein Paket durchlaufen darf, wodurch zirkulierende Pakete im Internet vermieden werden. Das Protokoll-Feld gibt an, welches Protokoll sich in der IP-Payload befindet. Als Beispiel sei hier User Datagram Protocol (UDP) genannt, das im Anschluss beschrieben wird. Die Header-Prüfsumme ist im anschließenden Feld angegeben und dient der Fehlererkennung auf Empfängerseite. Die Quell- und Ziel-IP-Adresse mit einer Größe von 32 Bit sind im nächsten Feld angegeben. Es folgt ein optionaler Teil mit Optionen, die vom Sender eingetragen werden können. Sollten diese in ihrer Gesamtheit kein Vielfaches von 32 Bit sein, muss mit Padding aufgestockt werden. Abschließend folgt die Payload mit den Daten des nächsten Protokolls.

2.1.3. UDP/TCP

UDP ist ein leichtgewichtiges Protokoll und besitzt die Eigenschaften, verbindungslos und ungesichert zu sein. In Abbildung 2.4 ist der exemplarische Aufbau eines UDP-Datagrammes dargestellt, welches der Transportschicht zuzuordnen ist. Die ersten 4 Bytes sind der Quellport, der angibt, welcher Applikation beim Sender das Datagramm zuzuordnen ist. Der folgende Zielport gibt die Zielapplikation beim Empfänger an und besitzt eine Größe von 4 Byte. Es folgt ein 4 Byte langes Längenfeld zur Angabe der

Länge der Daten. Optional ist die Verwendung einer 4 Byte langen Prüfsumme, die bei Nichtverwendung mit Nullen angegeben werden muss. Abschließend folgt die Payload, in der sich die Daten der folgenden Protokolle befinden.

Im Gegensatz zu UDP ist das Transmission Control Protocol (TCP) ein verbindungsorientiertes Protokoll, welches eine Sicherung der Datenübertragung ermöglicht. TCP selbst eignet sich ohne Änderungen nicht für den Einsatz im Echtzeitbereich, wo es auf rechtzeitige, zuverlässige und effektive Datenzustellungen ankommt. Das Neusenden von Daten sollte hingegen vermieden werden. Sollten Daten einmal verloren gehen, besitzen die erneut gesendeten Daten oft keinen gültigen Informationsgehalt mehr und erzeugen hierdurch unnötiges Datenvolumen im Netzwerk.

Ein weiterer konträrer Aspekt ist die Überlastkontrolle von TCP. Dabei werden Daten beim Sender u.U. vorgehalten und später gesendet, wenn der Sender der Meinung sein sollte, dass dies den Kanal überlastet. Ebenfalls anzumerken ist die Tatsache, dass TCP komplexer und umfangreicher ist als UDP, was zu einem größeren Overhead bzgl. der Performance führt, was in Echtzeitapplikationen mit hoher Anforderung an die Reaktionsgeschwindigkeit von Nachteil ist [Bid03]. Aus diesem Grund wird UDP in den nachfolgend vorgestellten Applikationen mit Echtzeitanforderungen bevorzugt.

2.1.4. Web Services

Im Verlauf der Arbeit kommen auch Web Services zum Einsatz. Web Services (WS) sind ein einheitlicher offener Standard zur Interaktion zwischen Entitäten in Netzwerken mittels Services. WS stellen eine mögliche Implementierung von serviceorientierten Architekturen (SOA) dar. SOAs als Architekturmuster beziehen sich nicht direkt auf einzelne Geräte, sondern der Service steht im Vordergrund. Services werden von den eigentlichen Entitäten im Netzwerk entkoppelt und bereitgestellt. Eine Entität besitzt intern eine Logik, z.B. in Form von Quellcode. Services bieten die Möglichkeit, diesen Quellcode nicht nach außen preisgeben zu müssen, da Services diese Logik nach außen hin kapseln. Eigenschaften von serviceorientierten Systemen sind nachfolgend aufgelistet:

- Lose Kopplung: Durch Services werden die Abhängigkeiten zwischen Instanzen auf ein Minimum reduziert.
- Service Contract: Services vereinbaren die Kommunikation mittels Servicebeschrei-

bungen und entsprechender Dokumente.

- Autonomie: Services haben die Kontrolle über die Logik, die sie einkapseln.
- Abstraktion: Die Logik wird nicht nach außen preisgegeben, nur die angebotenen Services.
- Komposition: Mehrere Services können einen neuen Service bilden.
- Wiederverwendbarkeit: Teile interner Logik können und sollten durch mehrere Services wiederverwendbar sein.
- Zustandslos: Mittels Services ist es möglich, weniger Informationen über die aktuelle Aktivität ermitteln zu müssen.
- Discoverability: Services müssen so entwickelt und beschrieben werden, dass sie mittels Discovery-Mechanismen gefunden und auf sie zugegriffen werden kann [Erl09].

WS basieren ebenfalls auf offenen Standards und Protokollen, wie dem Simple Object Access Protocol (SOAP) [W3C07], Extensible Markup Language (XML) [W3C08], Web Services Description Language (WSDL) [W3C01] und Universal Description, Discovery and Integration (UDDI) [OAS02]. SOAP wird verwendet, um Nachrichten und Dateien/Daten zwischen verschiedenen Teilnehmern zu senden. Es verwendet meist HTTP und TCP als darunterliegende Protokolle, kann aber auch in Kombination mit UDP verwendet werden [OAS09b]. Die Daten sind in der Regel mittels XML kodiert, um diese menschenlesbar zu machen. Das WSDL-Dokument beschreibt die Schnittstellen eines Services. Die Beschreibung in WSDL ist ebenfalls mittels XML kodiert.

In einigen Fällen existieren Servicebroker, welche das Wissen über die angebotenen Services im Netzwerk besitzen. UDDI ist ein Dienst, der als Servicebroker realisiert und in den Ursprüngen der Web Services verwendet wurde [Erl09]. Aus heutiger Sicht bieten sich Service-Discovery-Mechanismen an, die auf einen Servicebroker verzichten. Der Standard WS-Discovery bietet die Möglichkeit, auch ohne zentrale Einheit ein Service Discovery im Netzwerk durchzuführen. Hierdurch ist es möglich, in Ad-hoc-Netzwerken nach verfügbaren Services zu suchen, wodurch eine zentralisierte Struktur vermieden wird.

In der Abbildung 2.5 ist der Aufbau einer SOA mit Web Services und UDDI-Dienst dargestellt. Die komplette Kommunikation findet mittels des SOAP-Protokolls statt. Besitzt

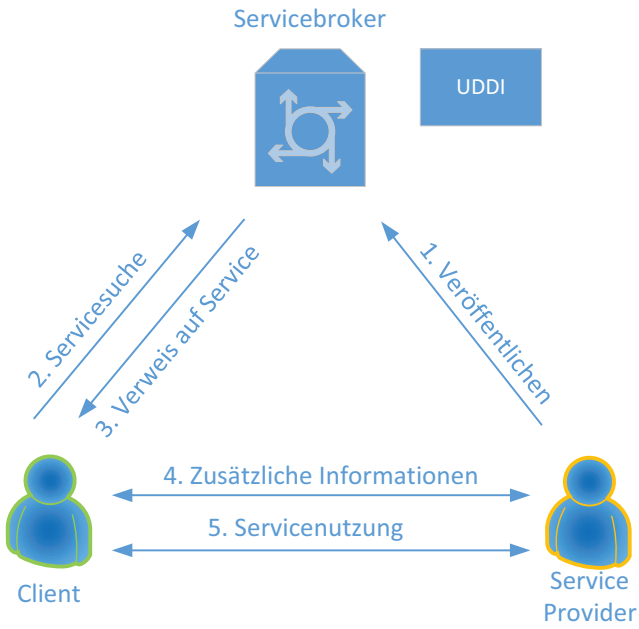

Abbildung 2.5.: UDDI-basierte Web Service SOA-Architektur.

eine Entität/ein Service Provider einen Service und möchte diesen im Netzwerk anbieten, kontaktiert er den Servicebroker, der einen UDDI-Dienst anbietet. An diesen meldet er seine angebotenen Services mittels Meta-Informationen oder ggf. WSDL-Dokument. Sucht ein Client nach einem Service, fragt er direkt den ihm bekannten Servicebroker an. Dies kann mit einem Verweis auf den Service über Meta-Informationen oder ggf. WSDL-Dokument erfolgen. Anschließend kontaktiert der Client den Service Provider und erfragt weitere Informationen wie z.B. das WSDL-Dokument, falls er dieses noch benötigt und nicht erhalten hat. Abschließend kann die eigentliche Nutzung des Services erfolgen. In einem Ad-hoc-Netzwerk ohne Servicebroker würden die Schritte 1 bis 3 abgewandelt werden. Die Informationen werden dabei direkt von den Service Providern eingeholt. Ein Auffinden der Services erfolgt meist mit Multicast-Nachrichten, um alle potentiellen Service Provider zu erreichen [OAS09c]. Im Umfeld von WS gibt es noch weitere Standards, die je nach Applikation verwendet werden. Ein weiteres Beispiel wäre WS-Eventing, welches ein Eventing-System mittels WS ermöglicht.

2.1.5. CoAP

Der CoAP-Standard steht für Constrained Application Protocol und ist ein Protokoll, welches für Zielplattformen mit limitierten Ressourcen, wie z.B. Sensoren, entwickelt wurde [SHB14]. Als Protokoll ist es der Applikationsschicht zuzuordnen. CoAP ist nach dem Representational state transfer (REST)-Softwarearchitekturstil konzipiert, was einem ressourcenbasierten Ansatz entspricht. REST-Architekturen weisen mehrere Merkmale auf, die im Folgenden kurz erläutert werden [Fie00].

Client/Server: Bei REST wird das Client/Server-Modell verwendet. Ein Server hält die Daten bzw. Ressourcen vor. Der Client greift auf den Server zu, um mit dessen Ressourcen zu interagieren. CoAP nutzt diese Benennung konsistent weiter und verwendet CoAP-Clients und CoAP-Server. Zusätzlich wird noch ein Proxy verwendet, der auf zwei verschiedene Arten definiert sein kann. Der Proxy verhindert eine direkte Kommunikation zwischen Client und Server und ermöglicht so eine höhere Sicherheit und Entlastung der Server. Zum einen existiert ein Forward-Proxy, bei dem ein Client den dahinterliegenden Server kennen und benennen muss, da der Proxy selbst den Server nicht kennt. Anders ist dies bei einem Reverse-Proxy, welcher für den Client transparent ist. Der Client nimmt somit an, dass er direkt mit dem Server kommuniziert.

Zustandslosigkeit: Der Server speichert keine Sitzungsdaten zwischen zwei Client-Interaktionen. Es besteht keine Abhängigkeit zwischen den Anfragen. Hierdurch wird die Zuverlässigkeit und Skalierbarkeit verbessert. Die Nachvollziehbarkeit von Aktionen durch den Server ist ebenfalls stark verbessert.

Einheitliche Schnittstelle: Die Schnittstelle bei REST ist stark verallgemeinert und vereinfacht, was diese Systeme weiträumig einsetzbar macht. Durch die einfache Schnittstelle ist dessen Verwendung ebenfalls leicht verständlich.

CoAP basiert auf vier Methoden, um mit einer Ressource zu interagieren:

- GET: Abrufen einer Repräsentation einer Ressource.
- POST: Senden einer Repräsentation. Die Verarbeitung der Repräsentation bestimmt das Ziel selbstständig und abhängig von der Zielressource.
- PUT: Erstellen oder Aktualisieren einer adressierten Ressource. Die Repräsentation der Ressource wird mit angegeben.
- DELETE: Eine Ressource soll gelöscht werden.

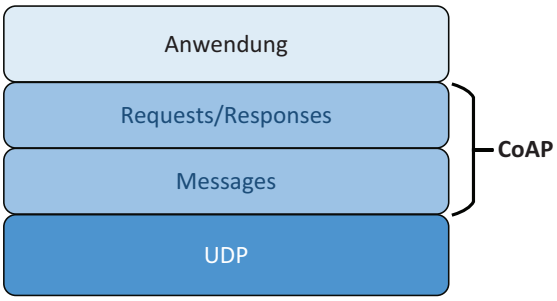

Abbildung 2.6.: Abstrahiertes CoAP-Schichtenmodell [SHB14].

Cache: Ein Client verfügt über einen Cache, wodurch Anfragen direkt vom Client-Cache beantwortet werden können. Dies ermöglicht eine Einsparung von Kommunikationsvolumen. Zu beachten ist aber der Kompromiss bezüglich Einsparung von Kommunikationsvolumen und der Aktualität der gecachten Daten.

In der Abbildung 2.6 ist das abstrahierte Schichtenmodell von CoAP dargestellt. CoAP basiert meist auf UDP, kann jedoch auch andere Protokolle verwenden. Alle folgenden Betrachtungen gehen aber von der Nutzung von UDP aus. Da CoAP direkt auf UDP aufsetzt, ergibt dies insgesamt einen sehr kompakten und schlanken Protokollstack.

Die Kommunikation in CoAP oberhalb von UDP basiert auf zwei Ebenen. In der Nachrichtenebene (engl. Message Layer) werden Nachrichten der Typen Confirmable (CON), Non-Confirmable (NON), Acknowledge (ACK) und Reset (RST) übertragen. Durch die Verwendung von CON-Nachrichten kann eine zuverlässige Kommunikation ähnlich der Verwendung von TCP erreicht werden. Nach dem Senden einer CON-Nachricht wird eine ACK-Nachricht erwartet, damit die Übersendung als erfolgreich quittiert werden kann. Die Zuordnung der ACK-Nachrichten zu den CON-Nachrichten erfolgt über eine Nachrichten-ID. Ist ein Kommunikationspartner nicht in der Lage, eine CON-Nachricht zu interpretieren, sendet dieser eine RST-Nachricht an den Sender der CON-Nachricht. Erhält der Sender einer CON-Nachricht nach einem definierten Timeout keine Antwort vom Empfänger, sendet er die Nachricht erneut.

Um ein wiederholtes Senden von Informationen zu vermeiden, kann die Kommunikation auf NON-Nachrichten basieren. Diese erwarten keine Antwort in Form von ACK-Nachrichten. Optional können RST-Nachrichten versendet werden, wenn der Empfänger einer NON-Nachricht nicht in der Lage ist, diese zu interpretieren.

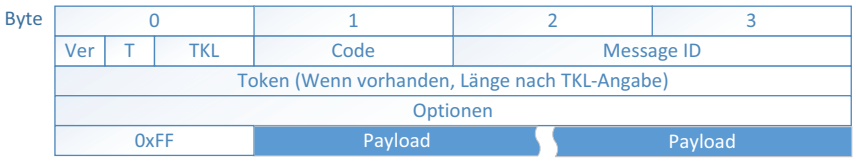

Abbildung 2.7.: CoAP-Paketaufbau [SHB14].

Über der Nachrichtenebene befindet sich die Request/Response-Ebene. In dieser werden die Grundmethoden durchgeführt, wie z.B. die GET-Methode. Zwischen den Teilnehmern wird das Request/Response-Prinzip als Kommunikationsmodell verwendet. Requests können als CON- oder NON-Nachrichten verschickt werden. Antworten in Form von Responses können als CON- oder NON-Nachrichten sowie „piggy-backed“ innerhalb einer ACK-Nachricht übermittelt werden.

Durch die Verwendung von UDP/IP werden auch Multicasts im Netzwerk unterstützt. Durch die Einfachheit und den geringen Overhead eignet sich CoAP besonders für eingebettete Plattformen.

2.1.5.1. Genereller CoAP-Paketaufbau

In der Abbildung 2.7 ist ein CoAP-Paket dargestellt. Das erste 2-Bit Feld (Ver) gibt die CoAP-Version an. Das nächste 2-Bit Feld (T) gibt den Typ der Nachricht an, also ob es sich um eine Nachricht des Typs Confirmable, Non-Confirmable, Acknowledgement oder Reset handelt. Das 4-Bit Feld (TKL) gibt die Länge des späteren Tokens in Byte an. Das folgende 8-Bit Code-Feld enthält die Information, ob es sich bei der Nachricht um ein Request, Response oder um eine Fehlermeldung handelt. Die 16-Bit Message-ID wird verwendet, um Duplikate von Nachrichten erkennen zu können und um Nachrichten vom Typ Acknowledgement/Reset mit den Typen Confirmable/Non-Confirmable abzugleichen. Das anschließende Token der im TKL-Feld angegebenen Länge ermöglicht die Zuordnung von Request/Response-Paaren. Nach dem Token folgen optionale Optionen. Die Optionen werden im Type-Length-Value (TLV)-Format angegeben. In den Optionen wird z.B. die Zieladresse einer Ressource in Form des Uniform Resource Identifier (URI) angegeben. Nach einer Option können sich das Ende der Nachricht, weitere Optionen oder ein Payload-Marker und die entsprechende Payload befinden. Sollte eine Payload folgen, muss ein Payload-Marker im Vorfeld mit dem Wert 0xFF gesetzt werden. Die

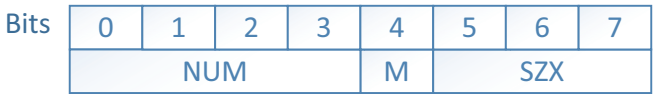

Abbildung 2.8.: Aufbau der CoAP-Blockoption [BS13].

Länge der Payload wird über Länge des Datagrammes bestimmt.

2.1.5.2. Die Blockoption in CoAP

CoAP bietet mit der blockweisen Übertragung die Möglichkeit, große Datenmengen in einem eingebetteten Umfeld zu realisieren. Diese zusätzliche Möglichkeit existiert bereits als Draft in [BS13] und soll zukünftig zum Standard werden.

In der Abbildung 2.8 ist der Aufbau der Blockoption dargestellt.

SZX-Feld: Der Wert im SZX-Feld ist eine exponentielle Größenangabe. Durch die Formel 2.1 wird die Größe der übertragenden Bytes einer Repräsentation einer Ressource im Paket angegeben.

$$N_{Bytes} = 2^{SZX+4} \tag{2.1}$$

M-Feld: Das im M-Feld definierte Bit bestimmt, ob die Übertragung der Repräsentation abgeschlossen ist. Bei einem nicht gesetzten Bit ist die Übertragung abgeschlossen. Ist das Bit gesetzt, sind weitere Blöcke zu übertragen.

NUM-Feld: Die Blocknummer ist im NUM-Feld angegeben und wird verwendet, um die einzelnen Blöcke zuordnen zu können. In Verbindung mit der im SZX-Feld angegebenen Größenangabe kann jederzeit die Startbytenummer eines Blockes ermittelt werden (siehe Formel 2.2).

$$Startbyte_{Block} = Block_{NUM} * 2^{SZX+4} \tag{2.2}$$

Als Beispiel sei die Übertragung von 1024 Bytes pro Paket vorgesehen. Dazu wird der Wert SZX auf 6 gesetzt. 3000 Bytes sollen übertragen werden. Daraus ergeben sich drei Blöcke, Blocknummer 0 (Byte 0-1023), Blocknummer 1 (Byte 1024 - 2047) und Blocknummer 3 (Byte 2049 - 2099). Der letzte Block muss nicht komplett gefüllt sein.

Block1-Options: Bei der Angabe der Blockoption im CoAP-Header wird zusätzlich zwischen der Block1- und Block2-Option unterschieden. Block1-Option wird verwendet, wenn ein CoAP-Client Daten an einen CoAP-Server senden will, was durch PUT-Anfragen realisiert wird. Im PUT-Request vom Client bestimmt SXZ die übertragenen Byte pro Paket. Im M-Feld wird bestimmt, ob noch Blöcke folgen, und die Blockfeldnummer im NUM-Feld bestimmt, um welches Block es sich gerade handelt. Die Block1-Option im Response vom Server wird folgendermaßen definiert. Das NUM-Feld gibt an, der wievielte Block erhalten wurde. Im M-Feld wird definiert, ob weitere Pakete erwartet werden. Sollte der Server z.B. überlastet sein, kann dieser im SZX-Feld auch eine gewünschte Nutzlast für die folgenden PUT-Requests definieren. Ein Beispiel der Verwendung von Block1-Optionen ist in Anhang C.1 zu sehen.

Block2-Options: Die Block2-Optionen werden i.d.R. verwendet, wenn Daten durch einen Client vom Server angefragt werden, was der GET-Funktion entspricht. In den Responses gibt der Server die Anzahl der versendeten Bytes im SZX-Feld an. Zusätzlich übermittelt er die Blocknummer im NUM-Feld und ob noch weitere Pakete folgen, durch das Setzen des Bits im M-Feld. Der Client kann in der Block2-Option durch Setzen des Wertes im SZX-Feld im Request angeben, wie viele Bytes er in der Payload haben möchte. Durch den NUM-Eintrag kann er zusätzlich mitteilen, welchen Block er benötigt. Ein Beispiel der Verwendung von Block2-Optionen ist in Anhang C.2 zu sehen.

2.2. Echtzeit

In diesem Abschnitt wird der Begriff Echtzeit bzw. Echtzeitsystem erörtert. Zusätzlich wird der Begriff Echtzeitkommunikation erläutert. Diese Definitionen sind nötig, da im Umfeld der Automatisierung besondere Bedingungen hinsichtlich der Echtzeitfähigkeit gelten. Es gibt viele Beschreibungen des Echtzeitbetriebs bzw. wie entsprechende Rechensysteme arbeiten müssen, um als Echtzeitsystem zu gelten. In der Echtzeitverarbeitung ist die Korrektheit des Systems nicht nur abhängig von den errechneten Werten, sondern auch von der Zeit, in der ein Wert ermittelt wurde [Sta88].

Eine standardisierte Beschreibung findet sich in der DIN 44300 Norm [ND85]:

> (Ist der) Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verar-

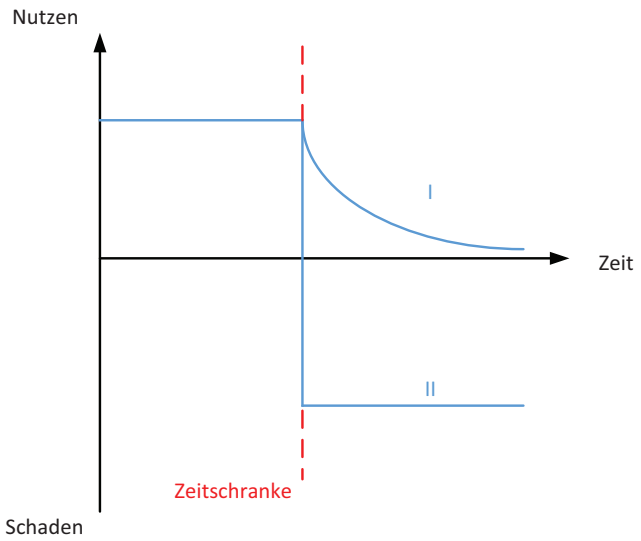

Abbildung 2.9.: Darstellung weicher (I) und harter (II) Echtzeit.

beitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlichen zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.

Abstrakt betrachtet, besteht ein Echtzeitsystem aus einem externen und internen System. Das externe System gibt dem internen System Zeitvorgaben für die Abarbeitung von Aufgaben vor. Das interne System hat diese Zeitvorgaben zu beachten. Jedoch unterscheidet man, je nachdem, wie streng die Zeitvorgaben sind, zwischen weicher und harter Echtzeit. In Abbildung 2.9 ist der Nutzen und der Schaden bzgl. weicher und harter Echtzeit dargestellt.

Bei **weicher Echtzeit (I)** entsteht kein Schaden, wenn die Zeitbedingung für eine Aufgabe nicht eingehalten wird. Es kann sogar vorkommen, dass der Nutzen nach der vorgegebenen Zeitschranke abnimmt und für eine Zeit danach immer noch ein Nutzen besteht.

Bei **harter Echtzeit (II)** hingegen muss die vorgegebene Zeitschranke eingehalten werden, da der Nutzen gleich null ist und es zu Schaden kommt. Dieser Schaden kann finanzieller Natur in Automatisierungsanlagen sein, sowie Schaden an Menschen, die sich z.B. im selben Umfeld wie die Echtzeitsysteme befinden. Daher sind Vorkehrungen zu treffen, um die Zeitschranken einzuhalten und die Wahrscheinlichkeit der Nichteinhaltung

zu minimieren [Zöb08]. Die Abarbeitung einer Aufgabe, welche eine Zeit Δe beansprucht und nach dem Zeitpunkt r starten kann, muss zum Zeitpunkt d erledigt sein. Diese Bedingung wird auch als Rechtzeitigkeit bezeichnet. Die entsprechende Zeitbedingung ist in 2.3 beschrieben.

$$A \equiv r + \Delta e \leq d \tag{2.3}$$

Bei harter Echtzeit und Echtzeitsystemen gilt, dass die Zeitbedingung A unter den günstigen Randbedingungen B immer zutreffen muss (Siehe Formel 2.4).

$$P(A \mid B) = 1 \tag{2.4}$$

Verteilte Echtzeitsysteme bestehen aus mehreren Echtzeitsystemen, welche miteinander kommunizieren müssen, da Aufgaben verteilt vorliegen. Die Besonderheit besteht darin, dass jedes Echtzeitsystem für sich die harten Echtzeitbedingungen erfüllen muss und diese Bedingungen auch für die Kommunikation gelten. Die erste Bedingung kann z.B. durch ein Echtzeitbetriebssystem gelöst werden, wenn es sich bei der Zielplattform beispielsweise um ein eingebettetes System mit entsprechender Hardware handelt. Je nach Rechensystem unterscheidet sich die verwendete Software.

2.2.1. Verfahren für harte Echtzeitkommunikation

Die harte Echtzeitkommunikation hängt vom physikalischen Aufbau des Netzwerkes und von der Art des Zugriffes auf das physikalische Medium ab. Man unterscheidet dabei zwischen der drahtgebundenen und drahtlosen Kommunikation. Im Anschluss erfolgt eine Beschränkung auf die drahtgebundene Kommunikation, da diese in dieser Arbeit verwendet wird.

Zugriffsprotokolle, mit denen sich der Zugriff kontrollieren lässt, sind in mehrere Typen unterteilbar [Zöb08]:

- Zentralisierte Verfahren
- Arbitrationsverfahren
- Token-Verfahren

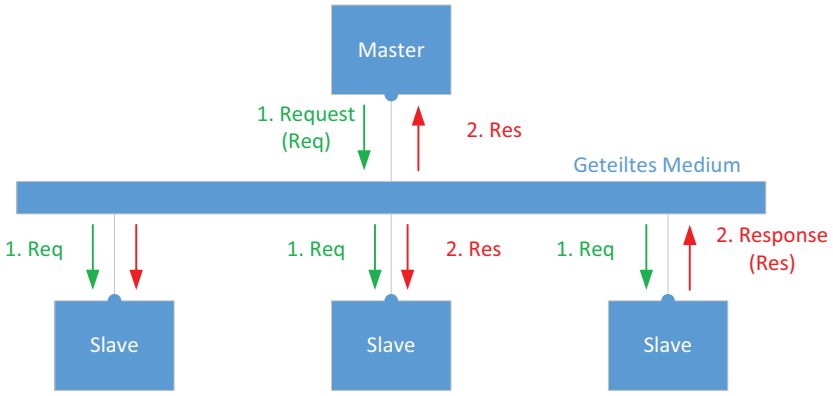

Abbildung 2.10.: Darstellung des Master-Slave-Zugriffsprotokolls.

- Zeitmultiplexverfahren

Zentralisierte Verfahren beruhen auf dem Master-Slave-Prinzip. Der Master ist direkt verantwortlich für den Medienzugriff der einzelnen Teilnehmer (Slaves). Das Einsammeln von Statusinformationen und das Versenden von Befehlen an Slaves wird i.d.R. mittels eines Broadcasts durch den Master gelöst. Jeder Slave lauscht dabei auf den Frame und erkennt anhand einer Zieladresse, ob das Paket für den Slave bestimmt ist, und verwertet die Nutzdaten des Frames. Zusätzlich zu den Nutzdaten sind meist auch Synchronisierungs- und Diagnosebits enthalten. Antworten von den Slaves werden auch als Broadcast versendet, sodass alle anderen Knoten ebenfalls die Nutzdaten erhalten. Die Funktionalität des Gesamtsystems hängt komplett vom Master ab. Die Leistungsfähigkeit des Masters bestimmt die Größe und teils auch Performance des Netzwerkes. Der Master ist zum einen ein Engpass (engl. bottleneck), da die gesamte Kommunikation über ihn läuft, und zum anderen ein sensibler Angriffspunkt, weil es reicht, den Master auszuschalten, um den größtmöglichen Schaden zu erreichen. Eine Darstellung des Zugriffsmechanimus ist in Abbildung 2.10 zu sehen. Es wird deutlich, dass ausschließlich der Master eine Anfrage (Request) stellt und die Slaves immer eine Antwort (Response) generieren. Die Fairness der Teilnehmerkommunikation hängt in diesem Fall ausschließlich von der Implementierung des Masters ab. Zudem ist und bleibt der große Nachteil der steuernde Master selbst, welcher als Single Point of Failure (SPoF) und Flaschenhals im zentralisierten System präsent ist.

Beim **Arbitrierungsverfahren** werden ebenfalls Daten mittels Broadcast übertragen

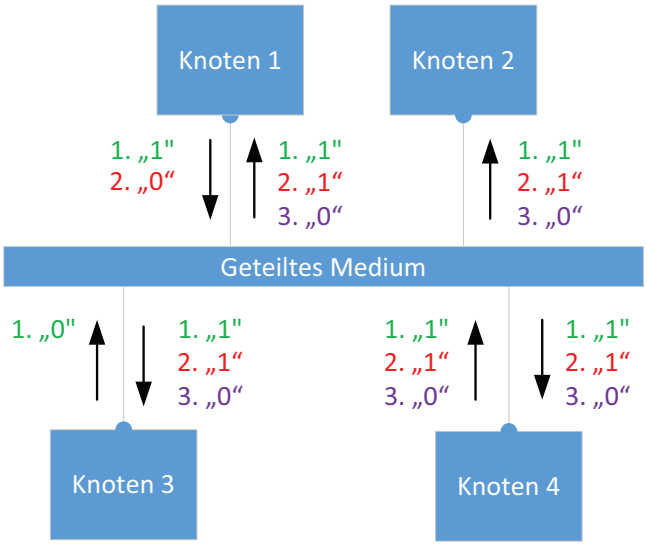

Abbildung 2.11.: Darstellung des Arbitration-Zugriffsprotokolls.

und man führt dominante und rezessive Bits ein. Ist das gemeinsame Medium frei und wollen mehrere Teilnehmer senden, starten sie mit dem Senden ihrer Priorität. Bei der Priorität handelt es sich üblicherweise um einen Bitstring. Angenommen, es existieren zwei Teilnehmer, die auf das gemeinsame Medium zugreifen wollen, starten beide Teilnehmer mit dem Senden ihres ersten Prioritätsbits. Wenn sie beide das gleiche Bit besitzen, wird sich keiner durchsetzen, und es wird mit dem Senden des zweiten Bits gestartet. Sollte jedoch ein Teilnehmer das dominante Bit besitzen, bemerkt der andere Teilnehmer dies, da sein Bit überlagert wird, und er beendet das Senden.

In Abbildung 2.11 sind vier Knoten dargestellt, die mittels des Arbitrierungsverfahrens auf ein gemeinsames Medium zugreifen wollen. Die logische Eins gilt dabei als dominant gegenüber der rezessiven logischen Null. Es ist ersichtlich, dass sich Knoten 4 nach drei übertragenen Bits durchsetzt. Knoten 3 nimmt seinen Anspruch auf das Medium zuerst zurück, da er beim Lauschen der Leitung feststellt, dass seine Null überlagert wurde. Anschließend nimmt Knoten 1 seinen Anspruch zurück. Im dritten Schritt sendet Knoten 4 auf Grund seiner gegebenen Priorität eine Null und lauscht auf dem Medium. Die Null wird nicht überlagert, wodurch Knoten 4 erkennt, dass er nun mit dem Senden seiner Daten starten darf. Bei Switched Ethernet kann dieses Verfahren nicht eingesetzt werden, weil da die Teilnehmer nicht mehr einen gemeinsamen Bus verwenden und somit eine

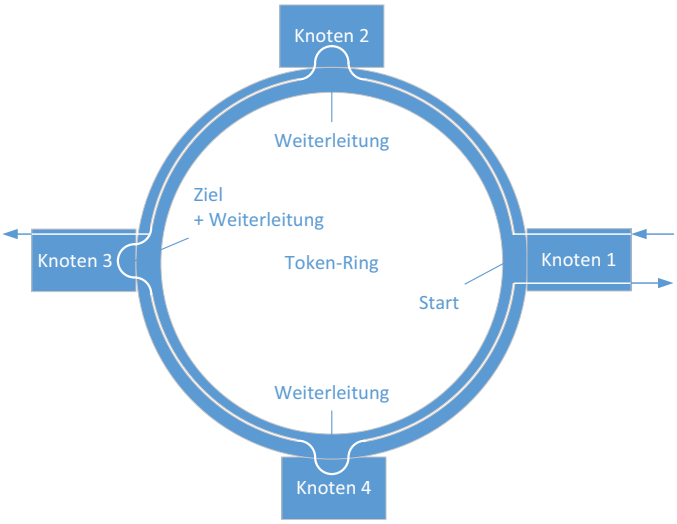

Abbildung 2.12.: Darstellung des Tokenprinzips.

physikalische Realisierung unmöglich ist. Zudem ist dieses Verfahren nicht fair, da im Worst Case nicht jeder gleich oft Zugriff auf das Medium hat. Ein Knoten mit hoher Priorität ist in der Lage, das Medium komplett zu blockieren.

Die dritte Möglichkeit ist das **Tokenprinzip**. Ein Teilnehmer darf auf einem gemeinsamen Medium nur senden, wenn er ein sogenanntes Token besitzt. Dieses Token existiert nur einmal im Netzwerk, wodurch ein geregelter Zugriff auf das Medium möglich ist. Die Teilnehmer sind meist physikalisch als Ring aufgebaut. Alternative Vernetzungen sind möglich, aber ein virtueller Ring ist in jedem Fall präsent. Durch diese Ringstruktur ist der Nachfolger immer bekannt. Ist ein Teilnehmer mit dem Senden fertig, gibt er das Token an den Nachfolgeknoten weiter, welcher nun das Medium nutzen darf.

Tokenverfahren garantieren ein deterministisches Verhalten und arbeiten fair. Jeder Teilnehmer besitzt bei gleichem Sendebedarf die gleiche Medienzugriffszeit. Allerdings ist eine Überwachung des Tokens selbst bzw. auch der Ausfall einzelner Teilnehmer notwendig. Fällt ein Teilnehmer aus, muss die vorherige Station i.d.R. die Nachrichten wieder in umgekehrter Richtung weitersenden, da ansonsten ein kompletter Ausfall der Kommunikation stattfindet. Dadurch, dass eine Nachricht über die Knoten im Ring weitergeleitet wird, fungieren diese als Repeater. Allerdings kann dies auch eine hohe Latenz bei einer sehr großen Anzahl an Teilnehmern bedeuten. Logisch ließe sich dieses Prinzip auch

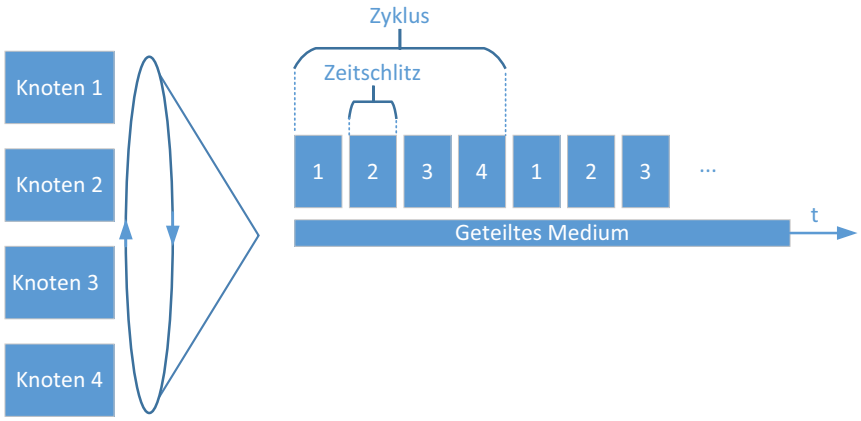

Abbildung 2.13.: Darstellung des Zeitmultiplexing.

auf Ethernet anwenden. Ein Beispiel ist in Abbildung 2.12 zu sehen. Knoten 1 möchte an Knoten 3 eine Nachricht schicken, besitzt das Token und kann daher direkt senden. Der direkte Nachfolger ist Knoten 2, welcher die Nachricht selbst weiterleitet. Knoten 3 erhält nun die Nachricht, macht sich eine Kopie der Nachricht und leitet sie weiter. Abschließend sendet Knoten 4 die Nachricht an den Ursprungsknoten 1 zurück, welcher dadurch erfährt, dass die Nachrichtenübermittlung mittels des Token-Rings erfolgreich durchgeführt wurde.

Die vierte Möglichkeit ist ein **Zeitmultiplex-Verfahren**. Hierbei bekommt jeder Teilnehmer einen eigenen Zeitschlitz. Innerhalb dieses Zeitschlitzes kann der Teilnehmer seine gewünschte Kommunikation durchführen. Nachdem jeder Teilnehmer Zugriff auf das Medium hatte, wird wieder beim ersten Teilnehmer begonnen, was dem Round-Robin-Prinzip entspricht. Ein Durchlauf, in dem jeder Teilnehmer Zugriff hatte, wird auch als Zyklus bezeichnet. Der große Vorteil ist der hohe Grad an Determiniertheit und Fairness bei der Zuteilung des Medienzugriffes. Der große Nachteil ist die aufwendige Planung der Zeitschlitze und Zyklen. Die einfachste Möglichkeit bietet eine zentralisierte Lösung, welche die Zeitschlitze den einzelnen Teilnehmer zuteilt. Alternativ kann auch eine Vorkonfiguration durch einen Nutzer während der Anbindung des Gerätes an das Netzwerk erfolgen. In Abbildung 2.13 ist das Zeitmultiplexing für vier Knoten dargestellt, welche die Teilnehmer darstellen. Nach dem Round-Robin-Prinzip darf zuerst Knoten 1 senden und erhält einen Zeitschlitz, gefolgt von den anderen Knoten. Nachdem Knoten 4 gesendet hat, ist Knoten 1 wieder an der Reihe und ein Zyklus ist beendet. In Kapitel 4 wird

ein System vorgestellt, welches ebenfalls Zeitschlitze nutzt, jedoch selbstorganisierend die Zeitschlitze ohne eine zentrale Instanz zuteilt.

2.2.2. Anwendbarkeit der harten Echtzeit bei Ethernet

Ethernet nutzt ebenfalls ein Zeitmultiplexverfahren, den sogenannten CMSA/CD-Algorithmus, für den Medienzugriff (siehe Abschnitt 2.1.1). Allerdings ist ein Determinismus nicht gegeben. Im Worst Case erhält ein Teilnehmer bei hoher Auslastung des Netzwerkes keinen Medienzugriff, da es zu permanenten Kollisionen kommen kann. Prinzipiell lassen sich die vorgestellten Verfahren teilweise auch oberhalb des Ethernetprotokolles anwenden. Jedoch besitzt Ethernet aus heutiger Sicht keine Busstruktur mehr, sondern es existiert ein geswitchtes Ethernet, das eher der Sternstruktur entspricht. Zwar kann es zu keinen Kollisionen mehr kommen, jedoch zu Paketverwürfen in der gepufferten Struktur. Daher sind in Switched Ethernet-Netzwerken in Echtzeitumgebungen Überlastsituationen zu vermeiden. Da oftmals der Kommunikationsbedarf im Gesamtnetzwerk nicht vorhersagbar ist, muss der Medienzugriff gesteuert werden, um Ethernet für den Einsatz in Echtzeitumgebungen zu ermöglichen. Der Zugriffsmechanismus ermöglicht dabei den Determinismus und die Rechtzeitigkeit der Nachrichtenzustellung, welche die Grundvoraussetzungen für die Echtzeitkommunikation darstellen. Es muss in diesem Falle auch unterschieden werden, ob die Netzwerkstruktur bekannt ist. Sollte dies nicht der Fall sein, ist der Worst Case derart definiert, dass theoretisch alle Daten an eine Senke im Netzwerk gehen bzw. ein Knotenpunkt im Netzwerk vorliegt, den der gesamte Datenverkehr passiert.

2.3. Peer-to-Peer-Netzwerke

Peer-to-Peer stellt ein Netzwerkparadigma dar, welches einen Gegensatz zu dem traditionellen Client-Server- oder Master-Slave-Modell darstellt. Die Idee dabei ist es, dass Teilnehmer in einem P2P-System alle dieselben Rechte und Pflichten besitzen. Man spricht in diesem Umfeld von einem Peer auch als Servent, da er Funktionen vom Server und Client vereint.

Ein typischer Client-Server-Aufbau ist in Abbildung 2.14 dargestellt. Die Sicherung der Funktionalität hängt direkt von einem zentralen Server ab. Fällt der Server aus, ist das

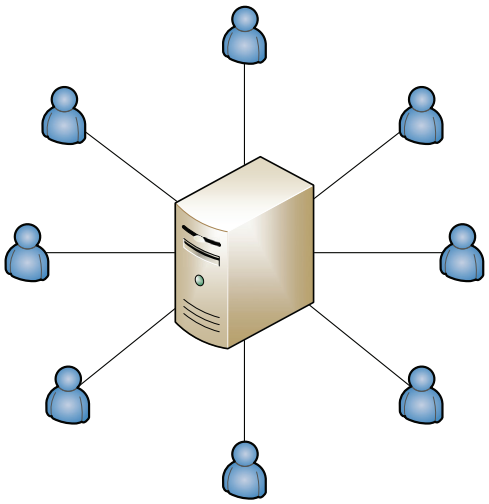

Abbildung 2.14.: Konventionelles Client-Server-Netzwerkmodell.

Netzwerk nicht mehr einsatzfähig: Der Server wird daher auch als SPoF bezeichnet. Eine weitere Besonderheit ist der Umgang mit dem Churn. Hierbei handelt es sich um das dynamische Verhalten des Nutzers, die in das Netzwerk ein- und austreten. Hier sind P2P-Netzwerke auf Grund ihres dezentralen Ansatzes sehr flexibel und passen sich den resultierenden Netzwerkstrukturen an.

Ein weiterer Aspekt ist die schlechte Skalierbarkeit. Ein Server kann nicht unbegrenzt viele Clients verwalten, weil es zu einer Überlastsituation kommen kann. Demnach stellt der Server zusätzlich den Flaschenhals im Netzwerk dar. Server sind, verglichen mit den Clients, daher oftmals leistungsstarke Rechner bzw. Rechnerfarmen, um diese Nachteile auszugleichen. Diese Struktur nutzen Angreifer gezielt durch sogenannte (Distributed) Denial of Service ((D)DoS)-Attacken aus. Der Server wird dabei durch eine Überflut an Anfragen überlastet, sodass ein normaler Nutzer nur sehr schwer oder gar keinen Kontakt zum Server und seinen Diensten erhält. Beliebt ist dabei der Einsatz von Bot-Netzen, um gewöhnliche aber infizierte Rechner zur Erzeugung der Überlast zu nutzen [PKB10]. Die Beseitigung der genannten Defizite besteht darin, auf einen Server möglichst zu verzichten, was durch P2P-Netzwerke erreicht werden kann. Ein vollständig dezentrales P2P-Netzwerk ist in Abbildung 2.15 dargestellt. In der Praxis wird P2P meist direkt in der Anwendungsschicht realisiert, wodurch es die Vor- und Nachteile aller unteren Schichten erbt.

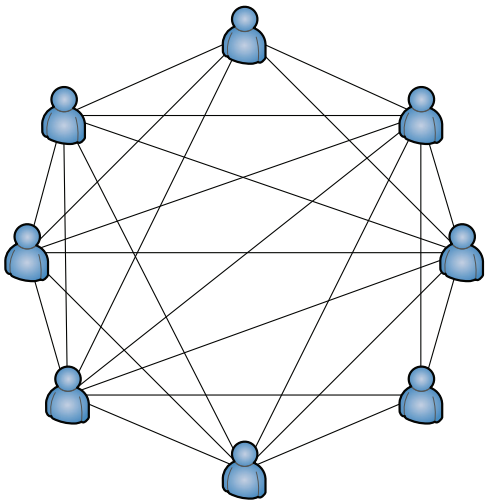

Abbildung 2.15.: Darstellung eines komplett dezentralisierten P2P-Netzwerkmodells.

2.3.1. Klassifizierung von P2P-Systemen

P2P-Netzwerke lassen sich in verschiedene Klassen unterteilen. In Abbildung 2.16 ist die Klassifizierung dargestellt. Man unterscheidet zunächst zwischen unstrukturierten und strukturierten P2P-Netzwerken. Unstrukturierte Netzwerke werden in zentralisierte, dezentralisierte und hybride Netzwerke eingeteilt. Bei strukturierten Netzwerken gibt es die DHT-basierten Netzwerke [SW05].

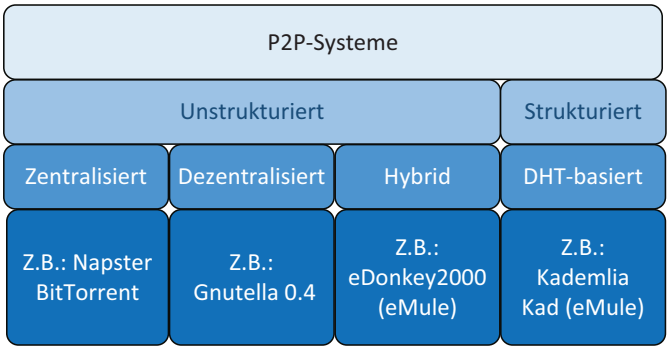

Abbildung 2.16.: Klassifizierung von P2P-Netzwerken.

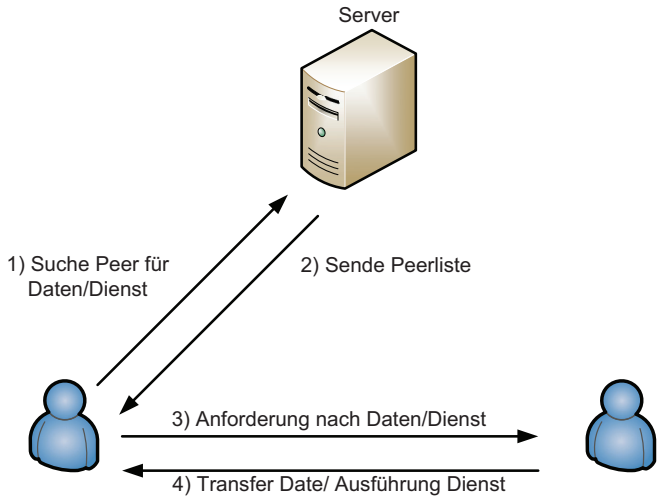

Abbildung 2.17.: Darstellung eines unstrukturierten, zentralen P2P-Netzwerkes.

2.3.2. Unstrukturierte, zentralisierte P2P-Systeme

Eines der bekanntesten unstrukturierten, zentralisierten P2P-Systeme ist Napster. Ein unstrukturiertes, zentralisiertes P2P-System besitzt immer noch einen zentralen Server, welcher als Index dient. Sucht ein Peer Daten oder Dienste, fragt dieser erst den Server an, welcher entsprechende Peers als Antwort liefert, die diese Daten oder Dienste zur Verfügung stellen.

In Abbildung 2.17 ist ein unstrukturiertes, zentralisiertes System dargestellt. Ein weiterer bekannter Vertreter ist BitTorrent. Generell existiert immer noch ein SPoF und Flaschenhals in Form des Servers. Dieser wird jedoch nur für Suchanfragen nach Daten und Diensten genutzt, da er selbst keine direkten Daten oder Dienste bereitstellt.

2.3.3. Unstrukturierte, dezentralisierte P2P-Systeme

Unstrukturierte, dezentralisierte P2P-Systeme besitzen hingegen keinen zentralen Server. Jeder Peer besitzt eine Nachbarschaftsliste mit benachbarten Peers. Routinginformationen dagegen besitzt der Peer nicht. Dies bedeutet, dass er bei der Suche nach Daten oder Diensten das komplette Netzwerk fluten muss. Um zyklische durch das Netzwerk laufende Pakete zu vermeiden, wurde eine Lebenszeit für die Pakete eingeführt. Nach einer

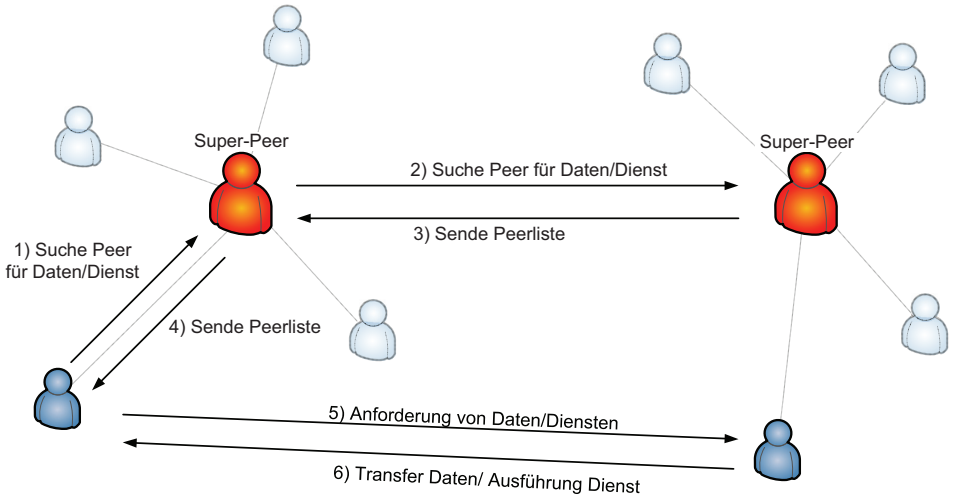

Abbildung 2.18.: Darstellung eines unstrukturierten, hybriden P2P-Netzwerkes.

definierten Anzahl an Hops werden die Pakete nicht weiter gereicht. Nach einer erfolgreichen Suche findet der Austausch von Daten oder Diensten statt. Das Gnutella-Protokoll der Version 0.4 ist ein Beispiel für dieses System. Die nicht vorhandene Strukturierung der Daten im Netzwerk hat den Nachteil, dass die Suche im Netzwerk durchaus nicht erfolgreich sein kann. Somit sind Systeme dieser Klasse ungeeignet, wenn der Determinismus zum Finden von zuständigen Knoten im Vordergrund steht.

2.3.4. Unstrukturierte, hybride P2P-Systeme

Unstrukturierte, hybride P2P-Systeme führen sogenannte Super-Peers ein. Statt einer zentralen Instanz wird ein zweites Netzwerk aus Super-Peers aufgebaut. An jeden Super-Peer sind einige Peers angeschlossen. Stellt ein Peer eine Anfrage, richtet sich diese zunächst an den Super-Peer, der möglicherweise einen entsprechenden Peer kennt und diesen zurückliefert. Andernfalls fragt er andere Super-Peers, ob diese einen entsprechenden Peer kennen. Der Austausch von Daten und Diensten erfolgt bei erfolgreicher Suche direkt zwischen den Peers. eDonkey 2000, welches als Protokoll in eMule integriert ist, ist einer der populärsten Vertreter. Eine Darstellung eines unstrukturierten, hybriden Systems ist in Abbildung 2.18 ersichtlich.

2.3.5. Strukturierte DHT-basierte P2P-Systeme

Strukturierte, dezentralisierte P2P-Systeme basieren auf einer verteilten Hashtabelle (engl. Distributed Hash Table (DHT)) und dem Schaffen eines Zusammenhanges zwischen Daten/Diensten und Peers. Mittels der entstehenden Relation können Daten/ Dienste gezielt Peers im Netzwerk zugeordnet werden, ohne auf eine zentrale Instanz angewiesen zu sein. Hierdurch schafft man eine Struktur, welche dem Nutzer auch ein deterministisches Verhalten zusichern kann, da nicht „irgendein“ Knoten für „irgendeinen“ Dienst zuständig ist. Erreicht wird dies, indem Peers und Daten Hashwerte in einem gemeinsamen Hashadressraum zugeordnet bekommen. Der Hashwert einer Bitbreite n kann z.B. aus der IP-Adresse oder MAC-Adresse eines Knotens erzeugt werden. Bei den Daten bietet sich der Dateiname bzw. bieten sich die Daten selbst an. Zur Erzeugung des Hashwertes wird eine Hashfunktion wie der Message-Digest Algorithm 4 (MD4) oder Message-Digest Algorithm 5 (MD5) verwendet [Riv92]. Die Breite n und die verwendete Hashfunktion hängen dabei vom verwendeten Protokoll und der entsprechenden Implementierung ab. Nach der Bitbreite n richtet sich auch die Größe des Hashraumes $0...2^n - 1$. Ein Peer ist nicht nur zuständig für ein Datum, wenn der Hashwert identisch ist, sondern auch wenn er ihm ähnlich genug ist. Ein gleicher Hashwert stellt somit einen Sonderfall in der Ähnlichkeit dar. Der Hashraum wird in der Literatur oft als Ring dargestellt. In Abbildung 2.19 sind einige Peers und Daten auf dem Hashring dargestellt.

Symbolische Umrandungen stellen die Zuständigkeiten der Peers dar. Die Zuständigkeit wird dabei durch die Ähnlichkeit, wie bei dem DHT-basierten System Kad durch die Suchtoleranz ST definiert, geregelt. Die Suchtoleranz ist der Grad der Abweichung, die maximal erlaubt ist, damit ein Peer für ein Datum zuständig ist. Die Suchtoleranz wird als numerischer Wert definiert und ist im Normalfall zur Kompilierzeit der P2P-Systeme festgelegt. Auch ist erkennbar, dass je nach Suchtoleranz auch mehrere Knoten für ein Datum zuständig sein können, was oftmals zur Erhöhung der Zuverlässigkeit durch Redundanz gewünscht ist. Jeder Peer besitzt eine Routingtabelle mit einer Untermenge aller Peers. Sucht ein Peer nach einem zuständigen Peer für einen gegebenen Hashwert, kontaktiert er den Knoten in der Routingtabelle, der dem Hashwert am ähnlichsten ist. Dabei kann es sein, dass er das Ziel erst über mehrere Hops erreicht.

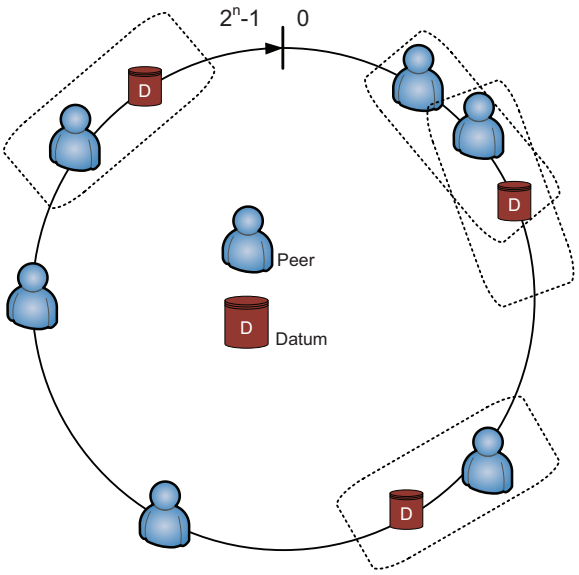

Abbildung 2.19.: Darstellung eines DHT-Rings mit Peers und Daten. Die Zuständigkeitsbereiche dienen nur der Veranschaulichung.

2.3.6. Vergleich der P2P-Systeme

In diesem Abschnitt wird die Verwendung des P2P-Protokolls Kademlia begründet, da Kad (eine Implementierung von Kademlia) als Grundlage für die späteren Forschungsergebnisse gewählt wurde. Die Wahl fiel auf strukturierte DHT-basierte P2P-Systeme, weil diese keine SPoFs/Flaschenhälse, wie die unstrukturierten, zentralisierten Systeme, aufweisen und im Gegensatz zu unstrukturierten, dezentralisierten Systemen eine deterministische Suche gewährleisten. Außerdem bieten die strukturierten DHT-basierten P2P-Systeme den besten Kompromiss bzgl. Such- und Speicherkomplexität. In Abbildung 2.20 ist die Such- und Speicherkomplexität in Abhängigkeit von der Anzahl der Peer N im Netzwerk dargestellt.

Durch den Vergleich der verschiedenen P2P-Systeme wird klar, dass unstrukturierte, zentralisierte Systeme einen großen Nachteil bzgl. der Verwendung eines zentralen Servers besitzen. Der Server ermöglicht zwar eine relativ schnelle Suche nach Knoten, da nur der Server selbst kontaktiert werden muss, um einen Knoten zu finden (Suchkomplexität $O(1)$). Jedoch ist die Zusicherung der Funktionalität des Netzwerks nur über den Server möglich. Der Server stellt somit auch einen sicherheitskritischen Teil des Netzwerkes

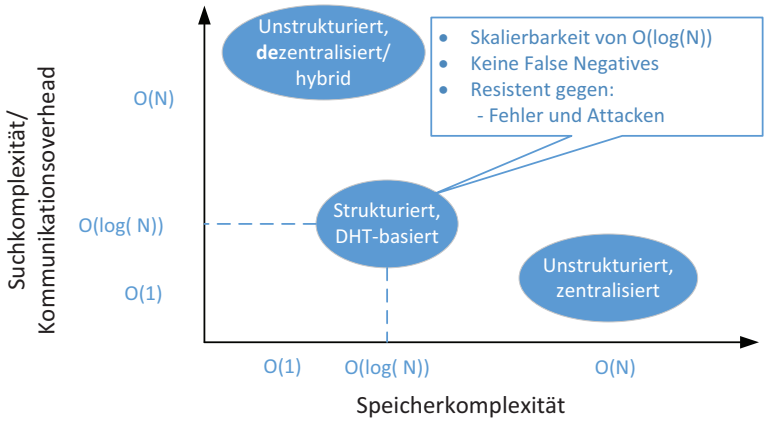

Abbildung 2.20.: Vergleich der Komplexitäten der drei Systeme.

dar. Zudem muss der Server alle Knoten und die dazugehörigen Informationen speichern und konsistent halten (Speicherkomplexität $O(N)$). Da der Server aber nicht per se mit der Anzahl der Peers skaliert, stellt dieser einen Flaschenhals bzgl. seiner eigenen Ressourcenlimitierung der CPU, des Speichers und der eigenen Netzwerkanbindung dar.

Unstrukturierte, dezentralisierte Systeme vermeiden jede Form einer zentralen Instanz und weisen damit eine hohe Skalierbarkeit auf. Jedoch stehen die Daten nicht in Relation zu den Peers, was eine Suche mittels Fluten bedingt (Suchkomplexität $O(N^2)$). Da die Suche nach Passieren einer bestimmten Anzahl an Peers abgebrochen wird, kann es vorkommen, dass Daten nicht gefunden werden, obwohl sie im Netzwerk existieren (False Negatives). Ein Vorteil ist dagegen die geringe Speicherkomplexität von $O(1)$, weil nur eine bestimmte Anzahl an Nachbar-Peers gespeichert wird.

Hybride Netzwerke werden in diesem Zusammenhang den unstrukturierten, dezentralisierten Systemen zugeordnet, da die Superpeers meist maximal 50-100 Peerkontakte abspeichern. Dies ergibt eine Speicherkomplexität von $O(1)$. Mit steigender Anzahl an Peers im Netzwerk steigt damit auch die Anzahl an Superpeers. Eine Suche selbst ähnelt dem Fluten, da über die Superpeers alle Teilnehmer gefragt werden müssen.

Den Kompromiss stellen strukturierte DHT-basierte Netzwerke dar. Die Such- und Speicherkomplexität beträgt $O(log(N))$. Sie bieten eine deterministische Suche (Vermeidung von False Negatives), unter der Voraussetzung, dass die Suchtoleranz entsprechend eingestellt ist. Da sie keine zentralen Instanzen besitzen, sind sie zudem robust gegenüber

Netzwerkfehlern oder Attacken.

Vergleich von strukturierten DHT-basierten P2P-Systemen: Die Wahl der Zielplattform fällt auf Grund der zuvor genannten Argumente auf strukturierte DHT-basierte P2P-Netze. Nun stellt sich abschließend die Frage nach dem konkreten Protokoll. Es existieren diverse DHT-basierte Realisierungen wie Chord, Pastry, Kademlia, Symphony und Viceroy. Im Folgenden werden Chord, Pastry und Kademlia untersucht, da diese eine deterministisch bestimmbare Suchkomplexität aufweisen. Dieses Kriterium ist ausschlaggebend, weil der spätere Einsatz im Bereich der (harten) Echtzeit liegt und der Determinismus eine Grundvoraussetzung darstellt.

Chord weist eine starre Routingtabelle auf, was zu höheren Beitritts- und Austrittskosten für neue Knoten führt. Pastry und Kademlia hingegen besitzen eine flexible Routingtabelle, was in einem geringen Wartungsaufwand resultiert.

Kademlia besitzt zusätzlich vorteilhafte Analyseeigenschaften, was die Bestimmung des Worst Case-Verhaltens ermöglicht. Bei Pastry entspricht die Routing-Metrik nicht unbedingt der tatsächlichen numerischen Nähe der Knoten-IDs. Die Lösung ist die Verwendung von zwei Routing-Phasen, was die Lookup-Performance verschlechtert. Außerdem resultiert daraus eine komplizierte formale Worst Case-Analyse des Systems. Die Suchoperation (Lookup) als wichtigste Operation, da sie am häufigsten ausgeführt wird, soll als abschließendes Entscheidungskriterium gelten. Chord ist in seiner Suchperformance nicht beeinflussbar. Pastry und Kademlia hingegen können über den Parameter b beeinflusst werden. b ist die Anzahl der Bits, die pro Suchschritt übersprungen werden können. Je höher der Wert für b ist, desto weniger Lookupschritte werden benötigt, sodass die Suche früher abgeschlossen werden kann. Für Pastry beträgt der Wert 4 und für Kademlia 5 [Stu06, SW05, MM02]. Kad als Implementation besitzt im Mittel sogar den Wert 6,98. In Tabelle 2.1 ist die Suchperformance von Chord, Pastry und Kademlia aufgeführt.

Aufgrund der geringen Suchkomplexität, der einfacheren Durchführung der Worst Case-Analyse, des deterministischen Suchverhaltens und der Einstellbarkeit der Such- und Speicherkomplexität mittels des Parameters b wurde Kademlia ausgewählt, um ein vernetztes System in der Automatisierungsumgebung zu realisieren. Zusätzlich weist Kademlia eine höhere Routing-Performance als Chord und Pastry auf [KGS11]. Als Grundlage für die P2P-basierten Entwicklungen in dieser Arbeit wurde Kad gewählt, was eine

Protokoll	Suchkomplexität
Chord	$O(\frac{1}{2}log_2(N))$
Pastry	$O(log_{2^b}(N))$
Kademlia	$O(log_{2^b}(N))$

Tabelle 2.1.: Angabe der Suchkomplexität von DHT-basierten P2P-Netzwerken mit deterministischen Suchverhalten in Abhängigkeit von der Knotenanzahl N und dem Parameter b [SW05].

Implementierung von Kademlia darstellt. Kad wurde ursprünglich für den eMule-Client implementiert [emu].

2.3.7. Kad(emlia) Grundlagen

Die Ähnlichkeit zwischen einer Information/Datei und einem Knoten wird durch die XOR-Operation bestimmt. Im Folgenden wird der Fokus auf Dateien gelegt, welche aber auch Informationen oder Dienste darstellen können. Die XOR-Operation wird mittels Hashwerten als Parameter ausgeführt, um die XOR-Distanz zu ermitteln. Die Hashwerte werden z.B. durch die MD5-Funktion generiert. Bei Dateien kann der Inhalt der Datei oder der Dateiname verwendet werden, um den Hashwert zu erzeugen. Um den Peers einen Hashwert zuzuweisen, bietet sich z.B. die IP-Adresse oder ein selbstgewählter String an, welcher als Argument für den MD5-Algorithmus gewählt werden kann. Die Zuständigkeit ist dann gegeben, wenn die ermittelte Distanz kleiner als eine gegebene Suchtoleranz ST ist (siehe Formel 2.5).

$$MD5(Peer) \oplus MD5(Datei) < ST \tag{2.5}$$

Damit ein zuständiger Peer im Netzwerk gefunden wird, besitzt jeder Knoten eine Routingtabelle. Die Routingtabelle besteht bei Kad mit einer Hashbitbreite von 128 Bits aus 128 k-Buckets. Buckets stellen Listenspeicherelemente dar, die Knotenkontaktinformationen speichern. Die Anzahl an Buckets ist gleich der Bitbreite der Hashwerte. Jeder Bucket hält dabei maximal k Peer-Kontakte. k ist eine Systemvariable, die im Vorfeld gesetzt ist (für Kad beträgt diese 10).

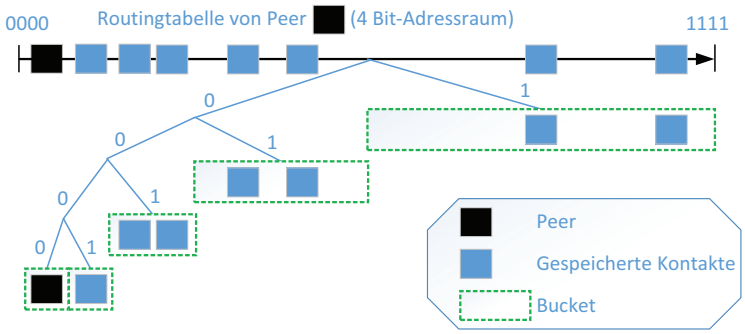

Abbildung 2.21.: Darstellung einer Routingtabelle eines Peers in einem 4-Bit-Adressraum mit Beispielkontakten.

Eine Routingtabelle für einen 4-Bit-Adressraum ist in Abbildung 2.21 dargestellt. In diesem Beispiel sei zur Vereinfachung angenommen, dass die Buckets maximal zwei Kontakte halten dürfen ($k = 2$). Für einen Hashwert hält ein Knoten nur einen Kontakt in seiner Routingtabelle vor. Dies erklärt, warum in dem Bucket, der sich nur um ein Bit an der niederwertigsten Stelle unterscheidet, ein Knotenkontakt befindet, da es nur einen möglichen Eintrag pro Hashwert für den Bucket geben kann.

Die Peer-Kontakte sind nach der XOR-Distanz zu dem Peer selbst sortiert. Als Resultat dieser Struktur geht hervor, dass ein Peer viele nahe Knoten und weniger entfernte Knoten kennt, was zu einem logarithmischen Suchverhalten führt. Will ein Peer dem Kad-Netzwerk beitreten, muss er ein sogenanntes Bootstrapping durchführen. Um ein Bootstrap durchzuführen, benötigt man mindestens einen Knotenkontakt (Bootstrap-Peer), den man kontaktieren muss, um in das Netzwerk zu gelangen. Der Bootstrap-Peer nimmt den neuen Peer in seiner Routingtabelle auf und sendet in einer Beitrittsbestätigung neue Kontakte. Jeder Peer-Eintrag in der Routingtabelle verfügt zusätzlich über eine Lebenszeit. Ist die Lebenszeit abgelaufen, wird dieser Knoten erneut angefragt. Sollte er nicht mehr antworten, wird dieser aus der Routingtabelle gelöscht.

Zusätzlich werden zwei weitere Wartungsoperationen verwendet, um die Routingtabelle aktuell zu halten. Es handelt sich dabei um Random- und Self-Lookups. Bei einem Random-Lookup wird ein zufälliger Hashwert generiert und nach diesem im Kad-Netzwerk gesucht. Somit werden auch entfernte Peers für die Routingtabelle gesucht. Bei einem Self-Lookup wird mittels des eigenen Hashwertes im Kad-Netzwerk gesucht,

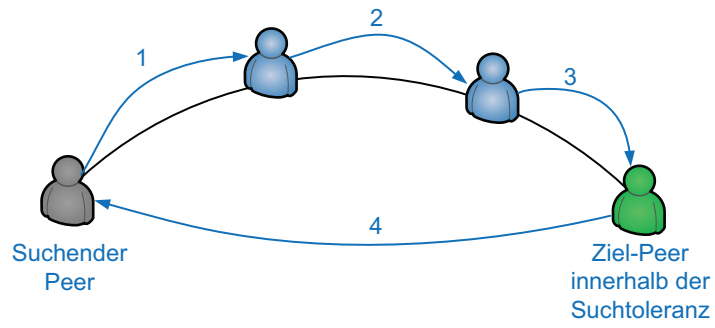

Abbildung 2.22.: Beispiel einer rekursiven Suche im Kad-Netzwerk.

um so möglichst nahe Peers zu finden. Diese Operationen dienen der Stabilisierung und Aktualisierung der Routingtabelle [SW05].

Bei der Suche (Lookup) nach einem Peer mit Hilfe der Routingtabelle gibt es zwei Verfahren: die rekursive und iterative Suche.

Rekursive Suche: In der Abbildung 2.22 ist die rekursive Suche exemplarisch dargestellt. Der suchende Peer befragt den Peer aus seiner Routingtabelle, der am dichtesten am gesuchten Hashwert liegt. Dieser kann, sollte er nicht selbst für den Hashwert zuständig sein, weitere dichtere Peers befragen. Ist ein zuständiger Peer gefunden, antwortet dieser dem ursprünglich suchenden Peer. Vorteil dieser Suchart ist die hohe Lookup-Performance. Jedoch besteht das Risiko, dass eine Suche nie zum Ziel führt, wenn ein Peer auf dem Suchpfad ausfällt.

Iterative Suche: Ein suchender Knoten fragt bei einem Peer aus seiner Routingtabelle, welcher dem gesuchten Hashwert am nächsten ist, nach. Dieser liefert neue Kontakte, insofern dieser nicht selbst zuständig ist. Die neuen Kontakte werden ebenfalls vom suchenden Peer gefragt. Auf diese Weise nähert sich der Peer dem Ziel, bis er genug zuständige Peers gefunden hat.

In Abbildung 2.23 ist dieser Ablauf schematisch dargestellt. Der suchende Peer besitzt keinen direkten Kontakt von einem Peer im Ziel-Hashbereich und kontaktiert Peers aus seiner Routingtabelle, die dicht am Ziel-Hashwert liegen. Die kontaktierten Knoten liefern direkt neue Kontakte, die dichter am Ziel-Hashwert liegen, aber auch noch außerhalb der Suchtoleranz liegen können. Der suchende Knoten kontaktiert selbst die neuen Kontakte, bis er einen zuständigen Knoten gefunden hat. Die iterative Suche wird in Kad

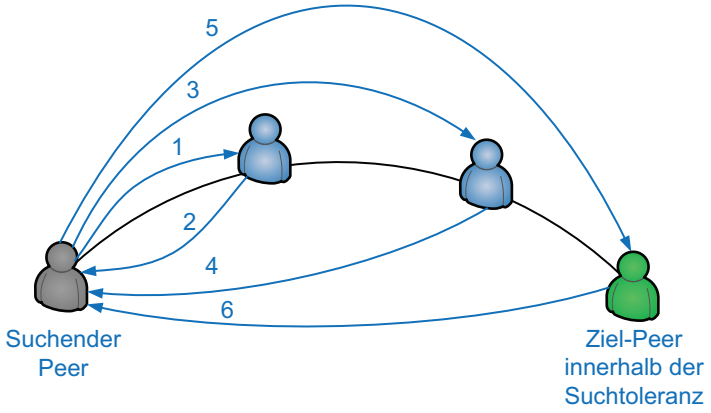

Abbildung 2.23.: Beispiel einer iterativen Suche im Kad-Netzwerk.

genutzt, während Kademlia eine rekursive Suche vorschlägt. Der Nachteil der Mehrkommunikation durch die iterative Suche wird durch den Fakt der einfacheren Implementierung, Wartbarkeit, einfacheren Fehlersuche und besseren Kompensation von Ausfällen entlang des Suchpfades ausgeglichen. Bei der rekursiven Suche muss der Timeout deutlich höher gesetzt werden, da der letzte Knoten ausgefallen sein könnte. Aus diesen Gründen wird die iterative Suche für die in dieser Arbeit beschriebenen Entwicklungen ausgewählt.

Im Folgenden wird der Lookup mittels der iterativen Suche beschrieben, wie er in Kad stattfindet. Die Hashwerte besitzen eine Breite von 128 Bit. Die Anzahl der kontaktierten Peers pro Schritt hängt vom Parameter α ab (bei Kad $\alpha = 3$). In der Abbildung 2.24 ist der Lookup dargestellt.

Der schwarz gekennzeichnete Peer sucht nach einem Hashwert im rot gekennzeichneten Bereich (I). Er besitzt jedoch zunächst selbst keinen Kontakt in diesem Bereich und fragt daher Peers aus seiner Routingtabelle, die dem Hashwert am nächsten sind. Die Anzahl der Peers, die mittels eines Requests (REQ) gefragt werden, beträgt in diesem Beispiel 3 ($\alpha = 3$). Die drei Peers antworten durch Responses (RES) und geben neue Kontakte zurück, die in (II) als orange markierte Peers zu sehen sind. Die neuen Peer-Kontakte befinden sich im Hashbereich, der durch die Suchtoleranz bestimmt ist. Diese werden ebenfalls mittels REQ kontaktiert, um zu überprüfen, ob diese sich noch im Netz-

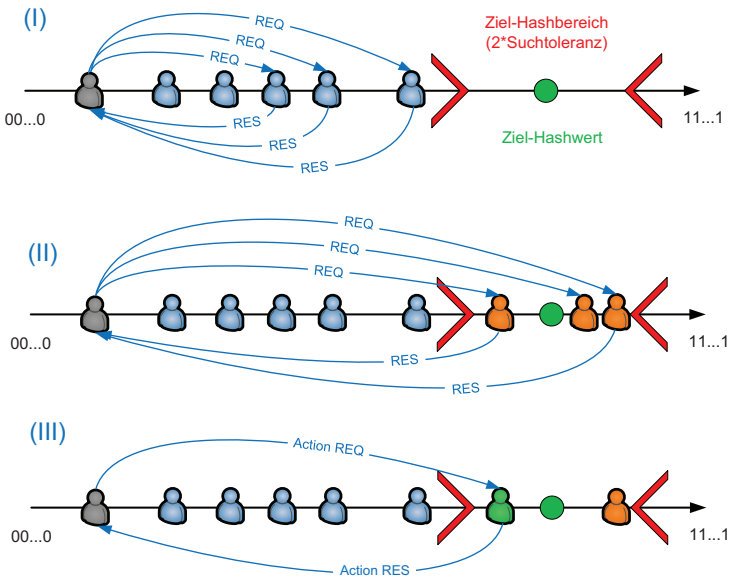

Abbildung 2.24.: Iterativer Lookup-Prozess im Kad-Netzwerk mit 128-Bit breitem Hashraum. Der suchende Peer besitzt zunächst keinen Kontakt im Ziel-Hashbereich.

werk befinden. Die aktiven Peers antworten mittels RES-Nachrichten und liefern u.U. weitere Kontakte. Im nächsten Schritt (III) wird ein zuständiger Peer (i.d.R. der Peer mit der größten Ähnlichkeit zum Ziel-Hashwert) mittels eines Action-Requests (Action REQ) kontaktiert. Der angefragte Peer antwortet mit einem Action-Response (Action RES). Action REQ/RES-Nachrichten stellen die Aktion, wie den Datenaustausch, dar. Eine durchgeführte Suche wird immer durch ein lokales Suchobjekt überwacht, welches als Instanz für jede Suche beim suchenden Peer existiert. Das Suchobjekt überwacht eine Suche und überprüft, ob eine der zwei Abbruchkriterien erfüllt ist. Das erste Abbruchkriterium besteht darin, ob genügend Peers für eine Suche gefunden wurden. Das zweite Abbruchkriterium stellt einen Timeout dar. Dieser findet statt, wenn nicht genügend Peers für eine Suche gefunden werden, die dem gesuchten Hashwert genügend ähneln. Die Anzahl der zu antwortenden Peers und der Timeout können vom Nutzer zur Kompilierzeit variiert werden und sind essentiell für die Zielapplikation. Als Szenario ist zu unterscheiden, ob ein Knoten einer möglichst großen Anzahl an Teilnehmern eine Nachricht zukommen lassen will oder ob es reicht, mindestens einen anderen Teilnehmer

zu kontaktieren. Dies kann bereits mittels der variierbaren Anzahl an zu antwortenden Knoten realisiert werden.

Kapitel 3.

Peer-to-Peer-Technologie für verteiltes Speichern

Im ersten Unterkapitel 3.1 wird ein bereits bestehendes P2P-System zum verteilten Speichern vorgestellt. Bei dem Einsatzgebiet handelt es sich um Zugangsnetze, welche später erörtert werden. Dieses System konnte bereits prototypisch umgesetzt werden und zeigt, dass sich P2P-Technologie in verschiedensten Einsatzbereichen sinnvoll einsetzen lässt. Es ließen sich jedoch keine Aussagen über das Verhalten des Systems bei mehreren Tausend Knoten treffen. Hierzu wird in dieser Arbeit ein Simulator entwickelt, welcher es ermöglicht, die Funktionalität in Bezug auf die Zuverlässigkeit der Daten, die verteilt im Netzwerk gespeichert werden, und des entstehenden Datenvolumens nachweisen zu können.

Im zweiten Unterkapitel 3.2 wird die Idee des verteilten Speichers mittels P2P-Technologie aufgegriffen und in der Automatisierung eingesetzt. Hierzu werden die Parameter, abhängig vom neuen Einsatzgebiet, neu ermittelt. Der Simulator wird für das neue Einsatzgebiet weiterentwickelt und wird in seiner Zeitauflösung verbessert. Zusätzlich fließen prototypische Ergebnisse eines für die Automatisierung entwickelten Prototypens mit ein, um realistische Eigenschaften des Systems im eingebetteten Umfeld ermitteln zu können. PSP erlangt durch die gewählte Plattform verbesserte Eigenschaften, insbesondere die weiche Echtzeit. Zudem werden die Grenzen des Systems aufgezeigt, die das System unter Extrembedingungen besitzt.

3.1. PSP - Ein System zur sicheren verteilten Speicherung von Daten in Zugangsnetzen

3.1.1. Einleitung

Das ursprüngliche entwickelte P2P-System läuft auf sogenannten Zugangsknoten (engl. Access Nodes(ANs)). Durch den Verbund verschiedener ANs wurde ein verteilter Speicher entwickelt [DGT+10]. Internetnutzer mit einem Heimanschluss sind an diese ANs angeschlossen, welche sich in der Nähe des Anschlusses befinden. Die ANs selbst sind Bestandteil der Zugangsnetze. Eine Aufgabe der ANs ist das Speichern der sogenannten Session-Daten (SD) zur Verwaltung der angeschlossenen Nutzer.

SD werden direkt auf den ANs erstellt. Die SD stellen Informationen bzgl. IP-Adressen, physikalischen Ports, MAC-Adressen und Vergabezeiten von IP-Adressen bereit. Internet Service Provider (ISP) müssen die SD speichern, da sie für operative, administrative und kontrolltechnische Aufgaben benötigt werden. SD ändern sich permanent aufgrund des dynamischen Verhaltens der Nutzer, die mit den ANs verbunden sind. Daher wird regelmäßig auf den Speicher zugegriffen, um die Daten zu aktualisieren.

Ein Nutzer, der sich mit dem Internet verbindet, fordert automatisch eine IP-Adresse mittels Dynamic Host Configuration Protocol (DHCP) an. Session-Filter blockieren den Datenverkehr von nicht konfigurierten Teilnehmern, wie Nutzer, die noch keine IP-Adresse angefordert haben. Im Falle eines Neustarts oder Absturzes eines AN müssen die SD neu geladen werden, um somit den Session-Filter zu rekonfigurieren. Erst nach der Wiederherstellung hat der Nutzer wieder Zugriff auf das Internet. Ein Neustart eines AN geschieht im Worst Case 2- bis 3-mal in der Woche, im Mittel einmal alle 4 Wochen und im Best Case einmal im Jahr [Nin11]. Der Flash-Speicher eines AN wird genutzt, um eine persistente Speicherung der SD auch im Worst Case zu garantieren. Allerdings ist der Flash-Speicher in der Verfügbarkeit und Anzahl der Schreibzyklen beschränkt. Zusätzlich ist er bereits für andere Konfigurationsparameter vorgesehen. Flash-Speicher haben typischerweise eine Haltbarkeit von 10.000 Schreibzyklen. Ein Update der SD geschieht alle 15 Minuten bis zu einer Stunde, was einen häufigen Schreibzugriff auf den Speicher zur Folge hat. Unter diesen Umständen könnte der Flash-Speicher bereits nach 104 Tagen ausfallen [Nin11].

Anstatt einen Flash-Speicher zu nutzen, wird vorgeschlagen, den freien verfügbaren flüch-

tigen Random Access Memory (RAM) und die Rechenleistung zu nutzen. Im Gegensatz zum Flash-Speicher kann der RAM nahezu uneingeschränkt oft beschrieben werden. Ein auf verteilten Hashtabellen (DHT)-basierendes Kad-Netzwerk erlaubt es dem System, den RAM aller Knoten logisch zu verbinden und zu teilen. Dabei arbeitet das Kad-Netzwerk als ein semi-permanenter verteilter Speicher für das strukturierte Abspeichern der SD. Jeder AN teilt seine Speicher- und Rechenkapazität mit den Teilnehmern des Kad-Netzwerkes. Nach dem Neustart eines AN wird die Wiederherstellung der SD durch das Einsammeln der nötigen Daten aus dem Kad-Netzwerk erreicht. SD müssen mit einer sehr hohen Zuverlässigkeit verfügbar sein, damit dem Nutzer am AN keine negativen Einschränkungen widerfahren. Die Zuverlässigkeit ist mit einem typischen Wert von 99,999% gegeben. Wenn die Daten allerdings auf eine verteilte Weise im RAM der ANs gespeichert werden, muss die Verfügbarkeit mit geeigneten Mitteln gesichert werden, da ANs ausfallen können. Die Redundanz wird dabei trotz Zusicherung einer hohen Datenverfügbarkeit durch Nutzung von Erasure Resilient Codes (ERCs) niedrig gehalten [LCL04]. Hierdurch wird die Persistenz der Daten gesichert. Das System, genannt „P2P based Storage Platform“ (PSP), wurde bereits entwickelt, um dieses Konzept umzusetzen. PSP wurde dabei erfolgreich als Softwareprototyp implementiert und ist lauffähig auf ANs, weil diese genügend Ressourcen zur Verfügung stellen [DGT+10]. Hierbei konnte die Funktionalität des Systems und die Entlastung einzelner Knoten nachgewiesen werden.

Allerdings ist es sehr aufwendig, ein Test-Setup mit mehreren hunderten bzw. tausenden Knoten zu erstellen, sodass es bisher noch nicht untersucht wurde [DGT+10]. Um trotzdem Aussagen über das Systemverhalten bei einem hochskalierten Netzwerk machen zu können, wurde für das bestehende System eine Simulation entwickelt. Es wurde ein Simulationsmodell implementiert, um die Verfügbarkeit der Daten, die Skalierbarkeit des P2P-Netzwerkes und die Netzwerkauslastung zu untersuchen. Mit Hilfe eines PSP-Prototyps wurden die Parameter ermittelt, die für die Simulation nötig sind. Die prototypischen Werte gehen als Eingabeparameter in die Simulation ein und ermöglichen es, das Verhalten des Systems realitätsnah zu simulieren. Mittels verschiedener Ausfallszenarien werden die generierten Datenmengen aufgrund der ausgetauschten Daten im P2P-Netzwerk ermittelt.

Nachfolgend werden die Hauptbeiträge kurz aufgeführt:

- Ableitung realistischer Simulationsparameter für die Simulation.

- Evaluierung der Simulationsergebnisse im Bezug auf Datenverfügbarkeit, Skalierbarkeit und zusätzliches Datenaufkommen.

Die Hauptbeiträge dieses Unterkapitels sind in [SDAT13a] publiziert. Unterkapitel 3.1.2 stellt den Stand der Technik dar. Abschnitt 3.1.3 gibt einen Überblick über das verwendete PSP-System. Die benötigten Simulationsparameter werden im Unterkapitel 3.1.4 abgeleitet. Unterkapitel 3.1.7 evaluiert die Simulationsergebnisse, bevor Unterkapitel 3.1.8 mit einem Fazit endet.

3.1.2. Stand der Technik

Der Stand der Technik wurde detailliert in [DGT+10] beschrieben, sodass hier nur die wesentlichen Arbeiten vorgestellt werden. Zusätzlich werden zwei aktuelle Arbeiten beschrieben, um die Aktualität des Standes der Technik zu gewährleisten.

[KBC+00, RA10, MKS08, DR01, YYXT08] schlagen eine DHT-basierte Lösung für die verteilte Speicherung der Daten vor. In [KBC+00] sollen global verteilte nicht vertrauenswürdige Server genutzt werden. Um eine Vertrauenswürdigkeit zu erreichen, müssen zusätzliche Vorkehrungen getroffen werden. Im Gegenzug dazu können bei PSP zusätzliche Vorkehrungen vermieden werden, da ANs vertrauenswürdige Instanzen sind.

In [RA10] präsentieren Ribeiro et al. eine DHT-basierte Plattform zur Datenspeicherung mit einer geringen Verfügbarkeit von Peers aufgrund von hohem Churn. Die Lookup-Komplexität dieses Ansatzes beträgt $O(N/log_2(N))$ Hops für jeden Lookup. PSP nutzt den Vorteil von Kad-Netzwerken, die eine geringe Lookup-Komplexität von $O(log_{2^b}(N))$ aufweisen [MM02].

Die Ansätze in [MKS08] und [DR01] stellen eine Pastry-basierende skalierbare Speicherplattform dar. [DR01] nutzt zudem Datenreplikation, um eine hohe Datenverfügbarkeit zu garantieren. Das in PSP verwendete Kademlia weist gegenüber Pastry eine bessere Lookup-Performance auf und lässt sich bzgl. des Worst Case-Verhaltens besser analysieren [SW05]. Zudem nutzt PSP die Reed-Solomon (RS)-Codes, um eine hohe Datenverfügbarkeit bei geringerer Datenredundanz als Datenreplikation zu gewährleisten.

Ye et al. [YYXT08] bieten eine verteilte Speicherplattform mit relativ vertrauenswürdigen Peers mit dem Fokus auf Aktualität und Sicherheit der Daten an. Ye et al. benötigten zusätzliche Server, um das Konzept des One-Hop-Routings zu gewährleisten. PSP ver-

zichtet auf die Verwendung von zentralen Instanzen, wie Server als Single Point of Failure (SPoF), um ein voll skalierbares und fehlerresistentes Netzwerk zu bilden.

Aktuelle Arbeiten sind in [XSL+12] und [PLR13] zu finden.

In [XSL+12] wird ein hybrider Ansatz vorgestellt, der aus normalen Knoten und Knoten mit hoher Zuverlässigkeit und Leistungsfähigkeit basiert. Von Vorteil ist dieses System, wenn ein stark heterogenes Netzwerk gegeben ist. Knoten mit hoher Zuverlässigkeit bilden einen Ring mittels Chord, an denen normale mobile Knoten mit hohem Churn angeschlossen sind. PSP hingegen besteht selbst nur aus zuverlässigen Knoten mit geringen Churn, was dem Chord-basierten Kernnetz aus [XSL+12] entspricht. Chord ist allerdings Kademlia bezüglich der Suchkomplexität unterlegen (siehe Tabelle 2.1).

In [PLR13] wird ebenfalls ein hybrider Ansatz vorgestellt, welcher im Wesentlichen ein Chord-basiertes Routing zur konsistenten Datenhaltung bei hohem Churn realisiert. Der hybride Ansatz verringert zwar das Problem der gleichmäßigen Verteilung der Daten durch Bildung von ausgeglichenen Clustern, die für bestimmte Datensätze zuständig sind, jedoch wird der Verwaltungsaufwand deutlich komplexer. PSP hingegen besitzt eine bessere Performance als Chord und setzt, wie bereits erwähnt, nicht auf simple Datenreplikation, sondern auf die ERCs. Durch die ERCs wird nicht nur weniger Speicher auf den einzelnen Instanzen benötigt, sondern es wird auch Bandbreite gespart, da weniger Daten übertragen werden müssen.

3.1.3. PSP-Grundlagen

In diesem Abschnitt werden die wesentlichen Aspekte von PSP erläutert, um so die nötigen Parameter (*kursiv* hervorgehoben) für eine Simulation zu bestimmen. Detaillierte Erörterungen sind in der bestehenden Vorarbeit [DGT+10] zu finden.

Abbildung 3.1 stellt ein typisches Zugangsnetzwerk dar. Zugangsnetzwerke umfassen die ANs, wie den IP Digital Subscriber Line Access Multiplexer (DSLAM). PSP wird genau an diesem Punkt auf den ANs integriert. Dabei werden die frei verfügbaren Ressourcen (RAM und CPU) genutzt, um das Speichern von SD zu realisieren. Alle ANs sind Peers mit Client- und Server-Funktionalität und bilden ein neues logisches Overlay über der existierenden Topologie. Das P2P-Netzwerk selbst wird zu einer gemeinsam genutzten Speicher-Ressource, bei der jeder AN seine verfügbaren RAM-Ressourcen bereitstellt. Jeder AN speichert nur einen Teil der gesamten SD des Netzwerkes. Zusätzlich

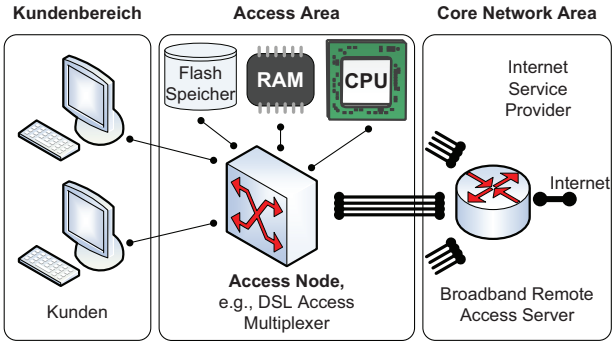

Abbildung 3.1.: Zugangsknoten mit einem PSP-Knoten.

teilen die ANs ihre Rechenleistung und das Netzwerk bildet damit ebenfalls eine verteilte Rechenressource. Für weitere Informationen zur Nutzung des Systems als verteilte Rechenressource sei auf [SDA+11a] und [SDA+11b] verwiesen. PSP nutzt Kad als eine Implementierungsvariante von Kademlia. Durch die Nutzung von Kad besitzt PSP keinen SPoF, sodass eine hohe Resistenz gegen DDoS-Attacken und Netzwerkausfällen [SW05] gegeben ist. Redundante Datenspeicherung wird per se in Kad unterstützt, sodass bereits eine höhere Zuverlässigkeit der SD erreicht wird. Sollten ein AN oder mehrere ANs ausfallen, sind die Daten immer noch mit einer hohen Wahrscheinlichkeit verfügbar. Um allerdings eine Verfügbarkeit der SD von fast 100 % im flüchtigen RAM zu erreichen, sind weitere Maßnahmen nötig. Ziel ist es eine Datenverfügbarkeit von 99,999 % zu erreichen. Dies wird durch zusätzliche Redundanz der Daten erreicht. Anstatt einfacher Datenreplikation (DR) durch z.B. redundanten Speicherns auf verschiedenen PSP-Instanzen, wurden ERCs gewählt, um den redundanten Anteil an Daten zu minimieren.

Bei ERCs werden die zu speichernden Daten in m Stücke (sogenannten Chunks) aufgeteilt. Anschließend werden k kodierte Chunks generiert, die Informationen von allen m-Chunks durch einen Kodierungsgprozess enthalten, welcher die Zeit $T_{EncChunks}$ benötigt. Insgesamt werden $n = m + k$ Chunks generiert, die die Daten repräsentieren. Mit lediglich m Chunks können effiziente ERCs die Originaldaten wieder herstellen, auch wenn einige Chunks verloren gegangen sind [LCL04]. Die Verfügbarkeit der Daten $P_{d,ERC}$ ist in Formel 3.1 beschrieben. P_n ist die konstante AN-Verfügbarkeit und es wurde ein Wert

von 90 % angenommen, was bereits einen sehr instabilen Knoten darstellt.

$$P_{d,ERC} = \sum_{i=m}^{n} \binom{n}{i} P_n^i (1 - P_n)^{n-i} \tag{3.1}$$

Durch die Verwendung von ERCs (mit $n = 32$ und $m = 16$) kann der redundante Anteil gegenüber der einfachen DR um den Faktor 3 reduziert werden. Aus diesem Grund wurde in [DGT+10] entschieden, in PSP ERC-Codes zu nutzten, um eine hohe Datenverfügbarkeit mit hoher Effizienz in Bezug auf zusätzlichen Overhead zu realisieren. Der ERC wurde mit den Reed Solomon(RS)-Codes realisiert. RS-Codes sind durch eine hohe Komplexität bei der Generierung der kodierten Chunks und deren Dekodierung beim Erstellen der Originaldaten gekennzeichnet [RL05, Huf03].

Allerdings haben die ANs genügend verfügbare Rechenkapazitäten, um diese Berechnungen durchzuführen. Ein weiterer Vorteil ergibt sich daraus, dass RS-Codes symmetrische Blockcodes darstellen. Dies bedeutet, dass Daten und kodierte Chunks geordnet sind. Auf die Daten-Chunks folgen die kodierten Chunks. Sollten alle m unkodierten Daten-Chunks verfügbar sein, kann die ganze Datei durch simples Zusammenfügen der Chunks ohne jeglichen Bedarf einer Dekodierung wiederhergestellt werden. Dadurch sind RS-Codes effizient und sparsam und vermeiden somit die Rechenzeit $T_{DecodeChunks}$ und Energieaufnahme bei der Dekodierung.

3.1.3.1. Kad-basierte Realisierung von PSP

PSP wird durch die Verbindung der ANs eines ISP durch ein Kad-Netzwerk realisiert. Das Kad-Protokoll ist erweitert worden, um den Austausch und das Löschen von Daten-Chunks zu ermöglichen. Die ANs erreichen ein strukturiertes Speichern der SD und kodierten Daten-Chunks anderer PSP-Knoten. Zusätzlich wird jeder AN ein Peer bzw. PSP-Knoten und bekommt einen Hashwert zugewiesen. Der Hashwert kann auf verschiedene Arten generiert werden, z.B. durch die eigene IP-Adresse. Die ANs werden im Kad-Netzwerk entsprechend ihres Hashwertes in der DHT platziert. Jeder AN ist für die Speicherung seiner eigenen SD im RAM zuständig, um alle verbundenen Teilnehmer mit einer IP zu versehen, was der DHCP-Funktionalität der ANs entspricht. Zusätzlich muss der AN die Chunks der SD anderer ANs speichern. Der Knoten muss die Chunks speichern, wenn der Hashwert der Datei (Name des Chunks) kombiniert mit der ID des

ANs dem Knotenhashwert ähnelt. Eine detaillierte Beschreibung der Knotenarchitektur, welche die DHCP- und die Kad-Funktionalität ermöglicht, ist in [DGT+10] wiedergegeben.

3.1.3.2. PSP-Knoteninteraktionen

PSP-Knotenaktivitäten, die eine Änderung der SD zur Folge haben, können durch interne und externe Events ausgelöst werden. Deren Auslöser sind zu bestimmen, da diese für das Simulationsmodell nötig sind, um möglichst realitätsnahe Ergebnisse zu erzeugen.

Interne Events: Diese geschehen, wenn ein AN eine Lese-, Lösch- oder Speicheroperation der eigenen SD im Kad-Netzwerk durchführt. Ein AN muss die SD aus dem Kad-Netzwerk lesen, wenn er zuvor ausgefallen ist. Einen anderen AN zu finden wird mit der Komplexität $O(log_{2^b}(N)) * T_{Hop}$ durchgeführt, wobei T_{Hop} die Round Trip Time (RTT) zwischen zwei beliebigen ANs darstellt. $O(log_{2^b}(N))$ gibt die nötigen Sprünge während des Lookup-Prozesses zum Ziel an. Das Löschen als Basisoperation ist nötig, um ungültige SD aus dem Kad-Netzwerk zu löschen. Sollte ein AN Änderungen an den eigenen SD feststellen, werden die alten SD gelöscht. Änderungen können durch DHCP-Operationen oder administrative Operationen ausgelöst werden. Wenn der AN DHCP Acknowledgements (DHCP ACKs) oder ein Client DHCP Releases sendet, sind die SD geändert worden und müssen aktualisiert werden. Zusätzlich können administrative Änderungen an den SD zu einem Aktualisieren der SD führen. Nachdem die alten SD gelöscht wurden, speichert der Knoten die neuen Daten im Kad-Netzwerk, was die dritte mögliche Speicheroperation darstellt. Die Speicheroperation benötigt die Zeit $T_{TransData}$, um die Daten zu übertragen.

Externe Events: Diese finden statt, wenn eine administrative Instanz eines ISP Kommandos sendet, um die SD- oder PSP-Parameter zu ändern. PSP bietet dabei die Möglichkeit, direkt Operationen auf dem AN unabhängig von der DHCP-Funktionalität auszuführen. Folgende Operationen werden unterstützt:

- Speicherung von aktuellen SD.
- Löschen eigener gespeicherter SD eines ANs.
- Löschen der verteilt gespeicherten SD eines ANs.
- Senden von durch administrative Instanz angeforderten Datenteilen.

- Modifikation von RS-Parametern.

Die genannten Operationen erlauben es der administrativen Instanz, die volle Kontrolle über das Verhalten des PSP-Systems in Bezug auf Datenverfügbarkeit und gespeicherten Daten zu erhalten. Wenn Daten gelöscht, gespeichert oder gesucht werden, nutzt PSP den Lookup-Prozess, um in Kontakt mit den zuständigen Knoten zu treten. Um die Suche und das Übertragen von Daten zu ermöglichen, integriert PSP Suchobjekte in das Kad-Protokoll. Für die Kommunikation werden UDP-Pakete verwendet. Die Datenübertragung zwischen zwei Knoten wird mittels TCP durchgeführt, was einen zuverlässigen Datentransfer ermöglicht.

3.1.4. Simulationssetup

Mittels Simulation sollen die Performance, Skalierbarkeit und Auslastung der Verbindungen von PSP untersucht werden. Hierdurch soll die hohe Verfügbarkeit und Zuverlässigkeit der SD im Kad-Netzwerk nachgewiesen werden. Um das Verhalten von tausenden Knoten aufzuzeigen, wurde ein Simulationsmodell basierend auf der Funktionalität eines lauffähigen PSP-Prototyps entwickelt [DGT+10]. Ein diskreter C++-Simulator mit einer Zeitauflösung von einer Sekunde wurde entwickelt, um eine performante Simulation über eine Simulationszeit von einem Jahr für mehrere tausend Knoten zu ermöglichen. Um die Simulation zu beschleunigen und um die Simulation in einer adäquaten Zeit zu ermöglichen, ist die Simulation rein funktional und unabhängig von einer darunterliegenden Netzwerktopologie. Der Simulator ist in Anhang A näher erläutert. Durch die Simulation ist es möglich, das generierte Datenvolumen und die ausgetauschten Datenchunks pro Sekunde zu messen sowie die Funktionalität des PSP-Systems zu validieren. Als Netzwerkgröße für die Simulation wurde das Backbone-Netzwerk der Deutschen Telekom gewählt, welches aus ca. 8.000 Hauptverteilern (beinhalten die DSL-Knoten) besteht, d.h. den ANs [Sch10]. Alle ANs sind über eine 10-GBit/s Bandbreite miteinander verbunden.

Es gibt zwei Hauptoperationen, die ein AN ausführen muss und auf denen der Fokus liegt. Die erste Hauptoperation ist das Speichern von Daten im Kad-Netzwerk, die zweite Operation das Zurückholen eigener Daten eines ANs aus dem Netzwerk.

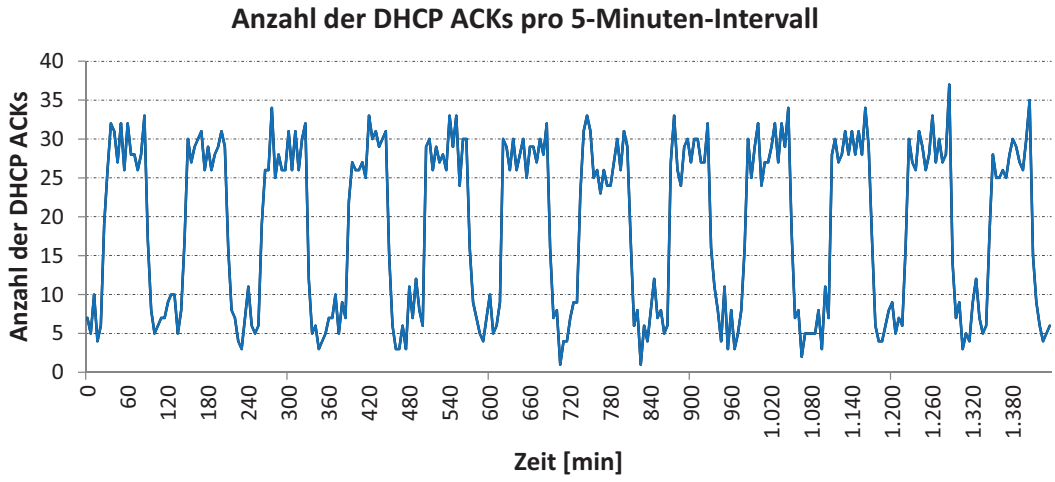

Abbildung 3.2.: DHCP-Datenverkehr in Form der DHCP-ACK-Anfragen.

.

3.1.5. Speichern der SD im Kad-Netzwerk

For $n = 0, 1...\infty$:

$$\text{rect}(t) = \sqcap(t) = \begin{cases} 8 & \text{if } n * 120 \text{ min} - 25 \text{ min} < |t| \leq n * 120 \text{ min} + 25 \text{ min} \\ 30 & \text{if } n * 120 \text{ min} + 25 \text{ min} < |t| \leq (n+1) * 120 \text{ min} - 25 \text{ min} \end{cases} \tag{3.2}$$

Die ANs müssen ihre Daten verteilt im Netzwerk speichern. Dies geschieht dann, wenn sich die SD auf einem AN geändert haben. Änderungen geschehen durch ein DHCP ACK vom DHCP-Server zu einem Client oder durch ein DHCP Release in umgekehrter Richtung. Um ein realistisches Verhalten des DHCP-Verkehrs zu emulieren, wurde der Verkehr des institutseigenen DHCP-Servers für eine Woche mit dem Programm Wireshark aufgenommen und nach DHCP ACK-Paketen gefiltert [Wir14]. Die Darstellung des DHCP-Verkehrs ist Abbildung 3.2 dargestellt. Die DHCP ACKs sind einem Wireshark-Trace entnommen (5-Minuten-Intervalle) dargestellt. Es genügt dabei die DHCP ACKs zu verwenden, da die Anzahl der DHCP Release-Anfragen auf Grund ihrer geringen Anzahl zu vernachlässigen sind. Hieraus konnte die Rechteckfunktion (siehe Formel 3.2) abgeleitet werden, um das DHCP-Verhalten der Teilnehmer nachzubilden. Es sind 200 Anwender an den DHCP-Server angebunden. Die DHCP Lease Time beträgt zwei Stun-

den.

Es gibt eine Phase mit weniger DHCP-Aktivität, welche als Low-Phase bezeichnet wird und der Dauer T_{Low} von 50 min entspricht. Nach der Low-Phase folgt eine High-Phase, die durch eine höhere durchschnittliche Anzahl an DHCP ACKs charakterisiert ist. Die High-Phase entspricht der Dauer T_{High} von 70 min. Die Low-Phase und die High-Phase werden über die Zeit hinweg periodisch wiederholt. In der Tabelle 3.1 werden zusammenfassend die Parameter aufgelistet, die das Verhalten des DHCP-Servers charakterisieren. Dieses Verhalten wurde direkt in der Simulation umgesetzt.

Parameter		Wert
Low-Phase		50 min
High-Phase		70 min
#DHCP ACKs pro 5 min	Low	8
	High	30

Tabelle 3.1.: Parameter der periodischen Rechteckfunktion zur Nachbildung des DHCP-Serververhaltens

Anhand des Verhaltens des DHCP-Servers können die Zeiten bestimmt werden, an denen die SD im Kad-Netzwerk gespeichert werden. Jeder AN in der Simulation startet zu einem zufälligen Zeitpunkt innerhalb eines 120-min-Intervalles, was der Low- und High-Phase entspricht. Innerhalb der Low-Phase verteilt ein AN seine SD alle 30 min und während der High-Phase alle 10 min, da sich die SD hier häufiger ändern. Diese Zeitintervalle zum Speichern der SD im Kad-Netzwerk entsprechen den Zeitintervallen von 15 min bis zu einer Stunde zum Speichern im Flash-Speicher und übertreffen sie sogar, was eine höhere Aktualität der SD garantiert. Einige Parameter müssen bestimmt werden, um die Zeit T_{Store} zum Enkodieren und Speichern der SD im Kad-Netzwerk berechnen zu können (siehe Formel 3.3).

$$T_{Store} = T_{EncChunks} + T_{Hop} * \lceil Hops_{Avg} \rceil + T_{TransData} \tag{3.3}$$

Die durchschnittliche Zeit $T_{EncChunks}$ zum Enkodieren der SD hängt von deren Größe ab.

Jeder AN mit seiner DHCP-Funktionalität besitzt ein SD-Volumen von 4 MB[1]. Wenn die Daten im Netzwerk gespeichert werden, müssen sie mittels der RS-Codes aufgeteilt werden. Die Parameter m und k sind auf den Wert 16 gesetzt. Dies führt zu 32 Chunks mit einer Größe von je 0,25 MB. Zusammen ergibt sich pro SD eine Datenmenge $Data_{Node}$ von 8 MB. Die Zeit, um die k-Chunks zu enkodieren und die m-Chunks zu erstellen, hängt von der Leistungsfähigkeit der AN-Hardware ab. Ein gewöhnlicher AN erreicht 1500 bis 3000 Dhrystone MIPS Rechenleistung, wenn als typischer Prozessor ein PowerQuicc II Pro oder PowerQuicc III verwendet wird [sem07]. Mittels eines vergleichbaren PC wurde ein durchschnittlicher Wert für $T_{EncChunks}$ von 7,125 s erreicht. Des Weiteren ist es wichtig zu wissen, welche Zeit benötigt wird, um die Chunks im Kad-Netzwerk zu speichern. Die Zeit für einen Lookup-Prozess, um verantwortliche Knoten zu finden, ist durch die durchschnittliche Anzahl an Hops bestimmt. Die Hopanzahl für einen Lookup-Prozess in Kad ist durch die Formel 3.4 gegeben. N ist die Anzahl aller Knoten im Testszenario und umfasst 1000 bis 8000 Knoten, was der Größe des Backbonenetzes der Deutschen Telekom entspricht. Die Anzahl der Knoten wurde variiert, um die Skalierbarkeit von PSP zu beweisen. Der Parameter b bestimmt die Anzahl der Bits, welche mit jedem Lookup-Schritt durchschnittlich übersprungen werden, und wird auf 6 gesetzt. Hieraus ergibt sich eine durchschnittliche Anzahl an Hops für 8000 Knoten von $Hops_{Avg} = 2.16$. Für die Worst Case-Analyse wurde $Hops_{Avg}$ daher auf $\lceil Hops_{Avg} \rceil = 3$ gesetzt. Zusätzlich ist es nötig, den RTT-Wert eines Hops T_{Hop} zu kennen. Als typischer RTT-Wert in Deutschland wird ein Wert von 50 ms angenommen [LW06].

$$Hops_{Avg} = log_{2^b}(N) \tag{3.4}$$

Die Zeit für den Datentransfer ist durch Formel 3.5 beschrieben.

$$T_{TransData} = \frac{Data\ volume}{Bandwidth} \tag{3.5}$$

Die Bandbreite in der Simulation wurde auf 10-GBit/s und die Datenmenge auf 8 MB gesetzt. Daraus ergibt sich eine Transferzeit $T_{Transdata}$ von 6,250 ms. Das ergibt zusammenfassend die Zeit T_{Store} von 7,281 s zum Speichern im Kad-Netzwerk.

[1] Alle Größenangabe sind Dezimalpräfixe, falls nicht anders angegeben. 1000 Byte entsprechen 1 KB.

3.1.6. Wiederherstellen von SD auf dem Netzwerk

Ein AN muss seine SD aus dem Kad-Netzwerk nach einem Reset oder Ausfall wiederherstellen. Die Ausfallzeit T_{Reset} durch einen Reset variiert von 10 bis 15 Minuten und ist auf 15 min für den Worst Case gesetzt worden. Während der Reset-Prozedur ist keine Interaktion mit den Knoten aus dem Kad-Netzwerk möglich. Neben dem Worst Case, Durchschnittsfall und Best Case wurden zusätzliche theoretische Ausfallprofile (gekennzeichnet als Zusatzprofil) definiert, um die Zuverlässigkeit und Performance von PSP aufzuzeigen. Die Anzahl der benötigten Resets nach einem Ausfall von ANs und die entsprechenden P_N-Werte sind in Tabelle 3.2 gegeben.

Ausfallprofil	jährliche Resets	P_N
Worst Case	130	99,629
Durchschnittsfall	13	99,963
Best Case	1	99,997
Zusatz 1	50	99,857
Zusatz 2	100	99,715
Zusatz 3	250	99,267
Zusatz 4	365	98,958
Zusatz 5	500	98,573

Tabelle 3.2.: Ausfallprofile der ANs.

Die Zeit T_{Rest} für das Wiederherstellen der SD auf dem Kad-Netzwerk nach einem Reset ist in Formel 3.6 beschrieben.

$$T_{Rest} = T_{DecodeChunks} + T_{Hop} * \lceil Hops_{Avg} \rceil + T_{TransData} \tag{3.6}$$

$Hops$ und T_{Hop} haben dieselben Werte wie beim Speichern der Daten. $T_{Transdata}$ ist nur halb so groß wie beim Speichern der SD, da nur 16 Chunks benötigt werden, um die SD wiederherzustellen. Die Suche wird dabei mit den unkodierten Chunks gestartet. Daher sendet PSP keine weiteren Requests für Chunks, sobald 16 Chunks verfügbar sind. Die Zeit für das Dekodieren der eingesammelten Chunks variiert je nach Art der verfügbaren Chunks. Die maximale Zeit wurde praktisch für den Worst Case bestimmt. Im Worst

Case wird angenommen, dass PSP 16 kodierte Chunks erhält. Dies führt zu einer Zeit $T_{DecodeChunks}$ von 18,760 s. Zusammenfassend ergibt sich für die Zeit T_{Rest} der Wert 18,913 s für einen AN, welcher bereits wieder in das Kad-Netzwerk eingetreten ist und seine SD wiederherstellt.

Die Zeitauflösung im Simulator beträgt eine Sekunde. Daher werden T_{Store} auf 8 s und T_{Rest} auf 19 s gerundet. In der Tabelle 3.3 sind die Simulationsparameter zusammenfassend aufgelistet.

Parameter	Wert	Beschreibung
N	1000-8000	Anzahl Knoten
$Data_{Node}$	8 MB	Datenmenge pro Knoten
$Days$	365	Anzahl der simulierten Tage
T_{Store}	8 s	Zeit zum Erstellen/Speichern der SD
T_{Rest}	19 s	Zeit zum Wiederherstellen der SD
T_{Low}	30 min	Periodendauer der Low-Phase
T_{High}	10 min	Periodendauer der High-Phase
T_{Reset}	15 min	Resetzeit eines ausgefallenen Knotens

Tabelle 3.3.: Simulationsparameter der Simulationen.

3.1.7. Evaluierung der Simulationsergebnisse

Lineare Skalierbarkeit: Das ausgetauschte SD-Datenvolumen ist in der Abbildung 3.3 zu sehen. Es ist ersichtlich, dass die Menge der transferierten Datenmenge pro Jahr mit der Anzahl der Knoten N im Kad-Netzwerk linear skaliert. Wenn 8000 Knoten, die mit Durchschnittsausfallprofil arbeiten, angenommen werden, erhält man eine durchschnittliche Datenrate für jeden Knoten von 9,1 KB/s, was einen sehr geringen Datenoverhead darstellt. Dies führt zu einer Kanalauslastung von 0,0069 ‰ für eine 10-GBit/s-Verbindung, was keinen Einfluss auf die Funktionalität eines AN hat. Die Anzahl der übertragenen Chunks nimmt mit steigender Anzahl an Ausfällen pro Jahr leicht ab. Dies ist durch die Inaktivität ausgefallener Knoten zu erklären, da diese nicht mehr ihre SD im Kad-Netzwerk speichern, bis sie wieder aktiv sind. Dieses Verhalten ist exemplarisch für die Netzwerkgrößen von 8000, 7000 und 6000 Knoten in Abbildung 3.4 dargestellt.

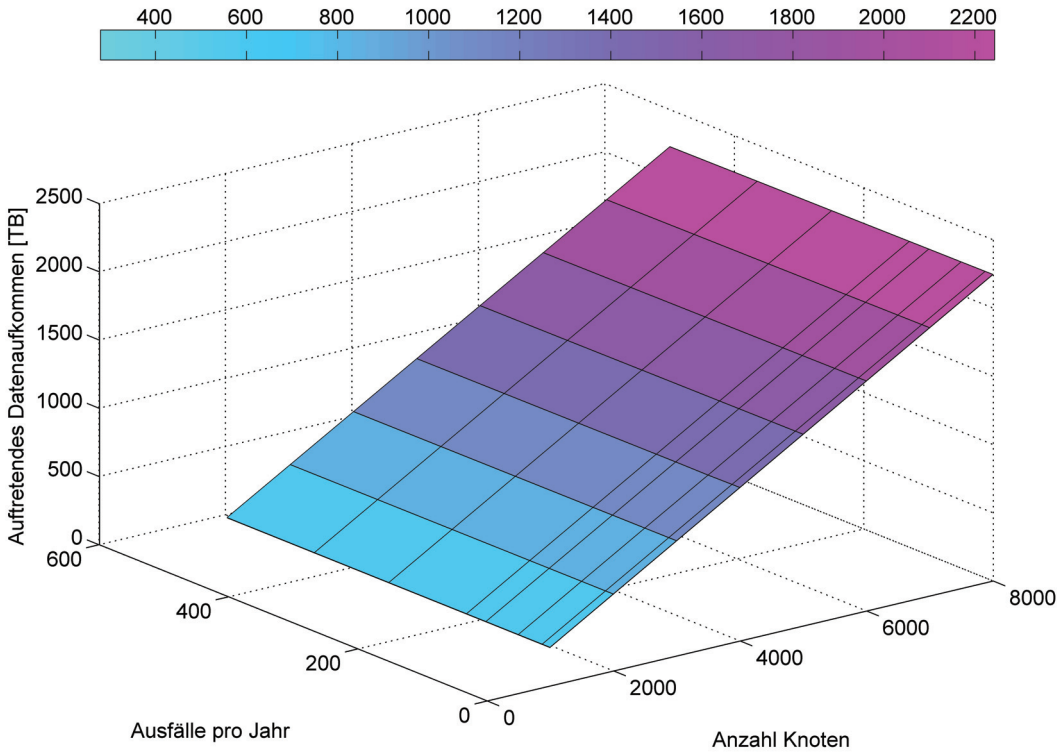

Abbildung 3.3.: Erzeugte Gesamtdatenmenge in TB pro Jahr.

Geringes Verkehrsaufkommen: Ein weiterer wichtiger Wert ist die höchste Spitzendatenlast, die pro Sekunde ausgetauscht wird, als Indikator für die maximale auftretende Netzwerklast im System. Die entsprechenden Ergebnisse sind in Abbildung 3.5 zu sehen. Die maximale Spitzendatenlast pro Sekunde ist direkt abhängig von der Anzahl an Knoten im Kad-Netzwerk. Die größte Lastspitze konnte bei 8000 Knoten festgestellt werden und beträgt beim Durchschnittsausfallprofil 175.25 MB/s pro Sekunde. Wenn die große Anzahl an Knoten berücksichtigt wird, ist dies ein relativ geringes Datenvolumen.

Permanente Datenverfügbarkeit: Während der Simulationen ist kein Fall aufgetreten, in denen PSP nicht in der Lage war, die SD aus dem Kad-Netzwerk wiederherzustellen. Dies kann ebenfalls bei den Zusatzausfallprofilen garantiert werden, welche

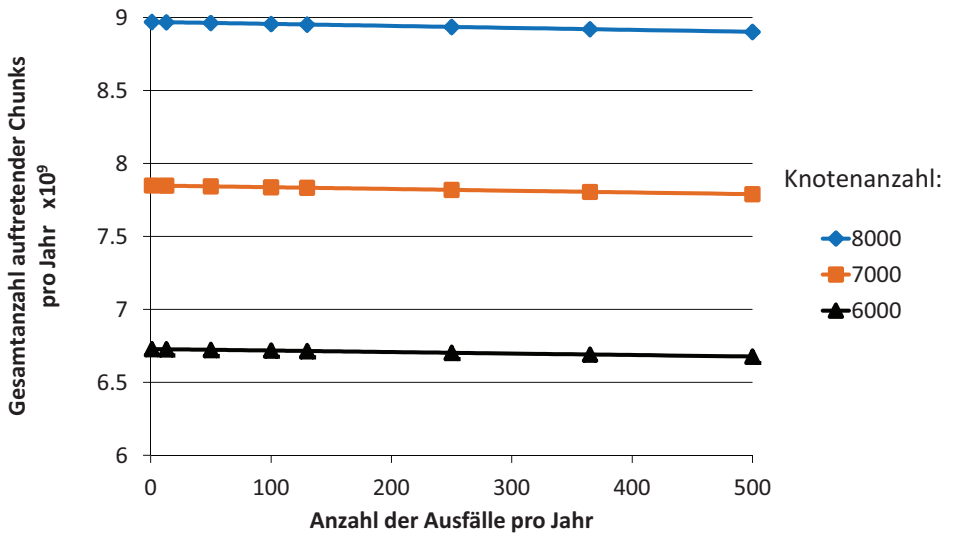

Abbildung 3.4.: Gesamtanzahl Chunks pro Jahr.

eine signifikante Mehranzahl an Ausfällen als der praktische Worst Case aufweisen. Die Ergebnisse der Simulationen unterstreichen die Eignung von PSP für hochskalierte Netzwerke.

3.1.8. Fazit

Die Simulationsergebnisse des verteilten Speichersystems PSP im Umfeld der Zugangsnetze wurden vorgestellt. Mittels Simulation konnten die zuvor getroffenen Annahmen für eine prototypische Umsetzung [DGT+10, Dan12] auch für mehrere tausend Knoten hinsichtlich Skalierbarkeit, Overhead bzgl. des Datenverkehrs und der hohen Verfügbarkeit der SD untersucht werden. Umfangreiche Simulationen mit bis zu 8000 Knoten zeigen die lineare Skalierbarkeit des Systems beim generierten Datenvolumen und der maximalen Spitzendatenlast pro Sekunde. Der generierte Datenverkehr lastet die 10-GBit/s durchschnittlich mit nur 0,0069 ‰ aus und beeinflusst die AN-Funktionalität damit nicht. Außerdem war jeder ausfallende PSP-Knoten in der Lage, seine SD während der simulierten Zeit von einem Jahr wiederherzustellen, bei einem Wiedereintritt in das Netzwerk nach einem Ausfall. Dies konnte sogar für höhere Ausfallraten als im praktischen Worst Case erreicht werden. Nach der Auswertung der quantitativen Ergebnisse durch die Simulation zeigt sich, dass PSP ein DHT-basiertes verteiltes Speichersystem

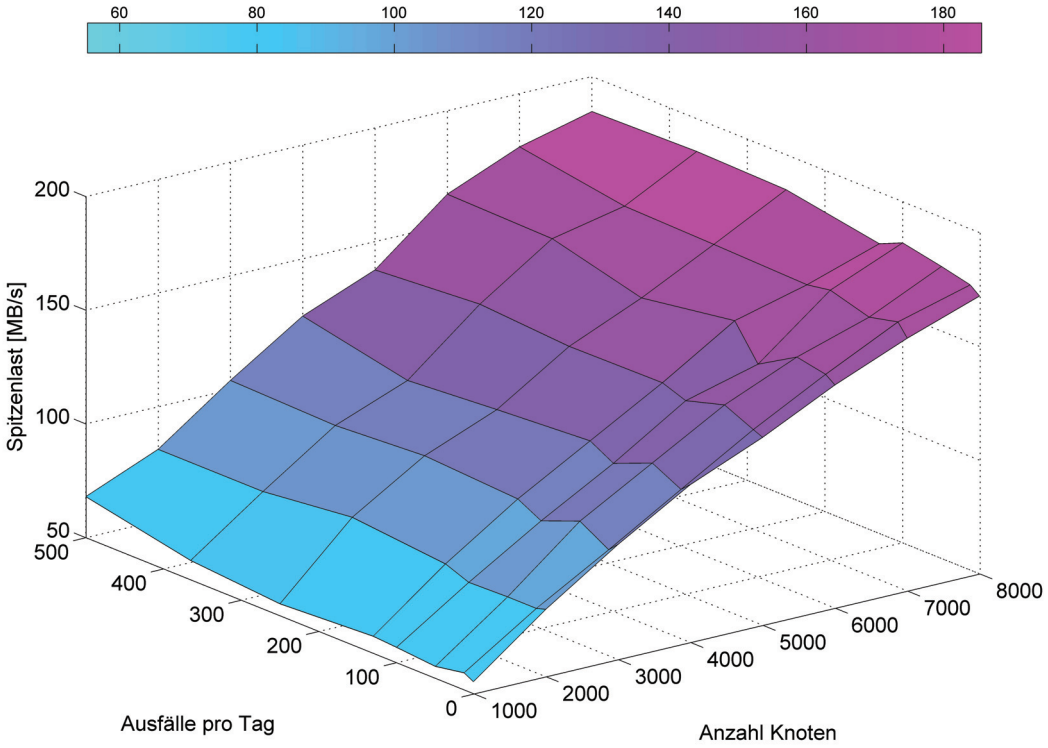

Abbildung 3.5.: Auftretende Spitzendatenlast pro Sekunde in MB.

mit hoher Verfügbarkeit und Zuverlässigkeit der SD in hochskalierten Netzwerken darstellt.

Einordnung in den Gesamtkontext der Arbeit: Am Ende des Fazits wird zur besseren Übersicht eine inhaltliche Einordnung des jeweiligen Beitrages im Unterkapitel vorgenommen. Dies wird an den entsprechenden Stellen anhand der entwickelten Protokollstacks der jeweiligen Anwendung dargelegt. Die Reihenfolge entspricht dabei nicht zwingend der Weiterentwicklung des vorherigen Protokollstacks, bezieht sich aber auf die inhaltliche Reihenfolge. Die Darstellung wird von einer kurzen Erläuterung unterstützt.

Der inhaltlichen Reihenfolge folgend ist PSP der erste Beitrag und der entsprechende Protokollstack ist in Abbildung 3.6 zu sehen. PSP nutzt im Wesentlichen UDP zur Kom-

munikation innerhalb des Kad-Netzwerkes. Jedoch wird TCP verwendet, wenn es um den Austausch der eigentlichen Daten geht. Bei dem Anwendungsszenario im Zugangsnetzwerk ist dies jedoch kein Nachteil, da keine weiche oder gar harte Echtzeit gefordert ist und somit nicht-deterministische Protokolle verwendet werden können. Vorteil ist, dass beliebige zusammenhängende Datemengen übertragen werden können. Das IP-Protokoll wird bei allen entwickelten Anwendungen in dieser Arbeit verwendet, um die Interoperabilität zu gewährleisten. Dies ermöglicht ein konsistentes System, bei dem jeder Teilnehmer per IP-Adresse ansprechbar ist. PSP zeigt an dieser Stelle schon deutlich, dass es Sinn ergibt, P2P-Technologien auch im Umfeld konservativer Industriezweige zu verwenden.

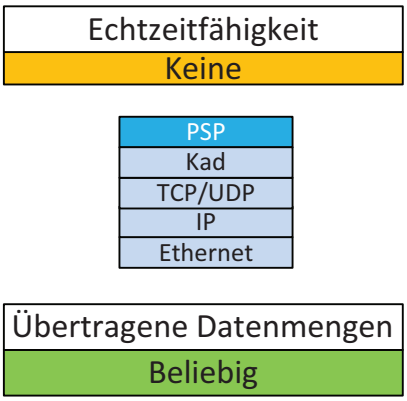

Abbildung 3.6.: Darstellung des PSP-Protokollstacks und Einordnung in den Gesamtkontext.

3.2. PSP-Auto - Verteilter DHT-basierter Speicher in der Automatisierung

3.2.1. Einleitung

Dieser Abschnitt befasst sich mit PSP in Bezug auf ein neues Umfeld - dem der Automatisierung mit Echtzeitanforderungen. Ziel ist es, PSP auch in Automatisierungsanlagen verwenden zu können, um ein hochskalierbares und ausfallsicheres verteiltes System zu ermöglichen. Auch in der Automatisierung (inklusive Heimautomatisierung) ist es sinnvoll, einen verteilten Speicher zu nutzen. An dieser Stelle ist ebenfalls eine Lösung, die auf einem volatilen Speicher basiert, zu bevorzugen, da Kleinstgeräte oftmals einen geringen bzw. keinen Festspeicher besitzen. Gegenüber dem Einsatz von PSP im Umfeld der Zugangsnetze weist das Umfeld der Automatisierung eine andere Charakteristik auf. Das resultierende System zum verteilten Speichern soll über die Eigenschaft der weichen Echtzeit verfügen. Dies kann durch die Verwendung einer geeigneten Plattform gelingen, welche später vorgestellt wird und aus der sich Parameter für die Simulation ableiten lassen. Die Prinzipien sind identisch mit dem in Unterkapitel 3.1 vorgestellten PSP-System. Als Einsatzszenario wird die Laborüberwachung gewählt, wie z.B. eines Chemielabors. Diese Ergebnisse repräsentieren ein eingebettetes System aus dem Bereich der Automatisierung. Ziel ist es, die Einsatzmöglichkeit von PSP und die Leistungsfähigkeit bzgl. der Datensicherheit und Skalierbarkeit im Umfeld der Automatisierung zu analysieren. Das entstandene System wird PSP-Auto genannt. Auch hier wurden mehrere tausend Geräte simuliert, um die Skalierbarkeit des Systems untersuchen zu können. PSP-Auto stellt den Einstieg in die Automatisierung durch die Verwendung einer geeigneten eingebetteten Plattform und entsprechender Software (Echtzeit) dar. Da die Kommunikation der Teilnehmer über Ethernet noch nicht gesteuert ist, erreicht PSP-Auto weiche Echtzeit. Zudem wird durch Simulationen aufgezeigt, wo die Grenzen des Systems liegen und ab wann und in welchem Umfang Ausfälle von Knoten Auswirkungen auf die Funktionalität des Systems haben.

Die Hauptbeiträge sind:

- Anpassung des Simulators und der Simulation im Umfeld der Automatisierung.
- Simulative Untersuchungen in Bezug auf Datenverfügbarkeit, Skalierbarkeit und zusätzliches Datenaufkommen.

Die Hauptbeiträge dieses Unterkapitels sind in [SDA+15b] publiziert. Der Rest des Unterkapitels ist wie folgt organisiert: Der Stand der Technik für das gewählte Szenario ist in Abschnitt 3.2.2 beschrieben. In den Abschnitten 3.2.3 und 3.2.4 erfolgt die Anpassung der Simulation und des Simulators an das neue Anwendungsszenario in der Automatisierung. Die neuen Simulationsergebnisse und deren Evaluierung werden in Abschnitt 3.2.5 präsentiert.

3.2.2. Stand der Technik

Beim Stand der Technik werden Systeme vorgestellt, die ebenfalls zur Überwachung und ggf. Steuerung von Laboreinrichtungen geeignet sind. In [ZWLW11] wird ein System zur Überwachung von Laboren vorgestellt. Als zentrales Element dient ein Monitoring Server, der als Gateway zu einem drahtlosen Sensornetzwerk dient. Zusätzlich lässt sich das Labor über externe Terminals überwachen, die mittels Übertragungsmedien Ethernet und GSM direkt am Monitoring Server angebunden sein können. Eine Aussage über die Performance wird allerdings nicht gegeben. In [LZHZ09] wird ein weiteres System zur Laborüberwachung vorgestellt. Dieses bietet ebenfalls die Möglichkeit, extern über GSM, Telefon oder Remote-PC Informationen per TCP/IP über den Laborzustand anzeigen zu lassen. Die Anbindung erfolgt ausschließlich über ein sogenanntes Monitoring Center. Am Monitoring Center befinden sich Controller-Plattformen, an die die Sensoren angeschlossen sind. Die Anbindung an das Monitoring Center erfolgt per RS232 Schnittstelle. Liu et al. präsentieren in [LT10] ein System, welches auf ZigBee basiert. Aufgabe des Systems ist die Lokalisierung von Personen in Räumen mittels per ZigBee angeschlossener Sensoren. Dabei wird ein zentrales ZigBee Extension Gateway vorgestellt, welches als Datenserver und Server zur Positionierung der Personen arbeitet. In [QLZH10] wird ein System namens NCSLab vorgestellt. Dieses erlaubt die komfortable Kontrolle und Überwachung von Experimenten sowie Simulationen. Diese erfolgt unabhängig von Endgeräten mittels Webbrowser. Hierzu wird allerdings ein zentraler Webserver benötigt. Für die einzelnen Experimente benötigt man Experiment-Server, welche Zugriff auf lokale Experimente haben. Zusätzlich existiert ein Matlab-Server, der es den Nutzern ermöglicht, Simulationen durchzuführen. Eine Aussage über die Performance, besonders in Bezug auf den zentralen Webserver, ist nicht angegeben. Robinson et al. nutzen in [RFSC+05] zur Überwachung von Experimenten (auch Chemielabore) das Protokoll MQTT [Loc].

Die gesamte Kommunikation und Überwachung des Experiments läuft über einen Micro-Broker. Nutzer haben über lokale Workstations Zugriff auf das Experiment. Nach außen ist das Netzwerk über eine Bridge mit einem Enterprise Broker verbunden, welcher in der Lage ist, die Nutzer über GPRS über das Experiment zu informieren.

Ein System, das zur Überwachung P2P-Technologie verwendet, wird in [GMR+09] beschrieben. P2P-Technologie wird genutzt, um die Zuverlässigkeit des Systems zu steigern. Es handelt sich um ein Überwachungssystem für Container auf Frachtschiffen. Die Container sind miteinander über ein P2P-Netzwerk verbunden. Die Veröffentlichung lässt keine Aussage über die Performance und somit Einsatzfähigkeit im Echtzeitbereich zu. Auch die Kommunikation ist nicht deterministisch ausgelegt und soll auf kabellosen Übertragungsmedien basieren. Fokus war ein geringes Datenaufkommen und ein geringer Energieverbrauch der Geräte in den Containern.

Bis auf das letzte System basieren alle auf hierarchischen und zentralisierten Verfahren. Diese weisen stets das Problem des Flaschenhalses im Netzwerk auf. Sollten zentrale Elemente der Systeme ausfallen, ist die Funktionalität nicht mehr gegeben. Zudem sind sie für Angriffe von außen anfällig. Nahezu alle Lösungen setzen dagegen Firewalls ein.

Mittels Kad wird untersucht, inwieweit sich z.B. Laborüberwachung auch dezentral gestalten lässt. Dies bezieht sich im Besonderen auf die Datenhaltung. Gerade in Laboren oder Szenarien mit Extrembedingungen ist es von Vorteil, Daten aus dem Netzwerk erhalten zu können, auch wenn ein Teil der Instanzen ausfällt. Das resultierende PSP-Auto-System besitzt bereits potentiell weiche Echtzeiteigenschaften. Durch die Simulation ist es möglich, das Verhalten auch für hochskalierte Netzwerke bestimmen zu können, was bei den vorgestellten Arbeiten nicht der Fall ist. Die Ergebnisse werden durch das Hinzuziehen von prototypischen Werten realistischer und lassen genauere Rückschlüsse auf das Verhalten des Systems zu.

3.2.3. Anpassung an das Automatisierungsszenario

Im Vorfeld müssen neue Anforderungen definiert werden. Die wichtigste Neuerung ist das Erhöhen der Simulationsauflösung von Sekunden auf Millisekunden. Dies ist nötig, da auch die zeitlichen Anforderungen steigen. Des Weiteren werden in Tabelle 3.4 alle angepassten Parameter aufgeführt. Als Anwendungsszenario dient ein Netzwerk, welches einen hohen Churn in Extremsituationen aufweisen kann. Als Beispiel dient die

permanente Überwachung eines Chemielabors mittels Sensordaten. Solche Überwachungen benötigen oftmals einen zeitlichen Bezug zu den Messdaten, weil Schlussfolgerungen über Zustände nur über den Verlauf von Daten gezogen werden können. Als Beispiel wird ein Datensatz einzelner Teilnehmer von jeweils 1 KB angenommen. Die Verteilung (Speicherung der Daten eines Teilnehmers im Netzwerk) bei PSP-Auto ist im gewählten Szenario auf eine Periode T_{Dist} von zwei Sekunden festgelegt, was einer hohen Aktualisierungsrate gegenüber dem ursprünglichen PSP-System entspricht. Somit speichert ein Teilnehmer alle zwei Sekunden seinen Datensatz verteilt im Netzwerk. Zudem ist es bei solch extremen Umgebungen, die auch zu einer erhöhten Ausfallwahrscheinlichkeit der Geräte führen, von großer Bedeutung, die Stabilität des Systems zu untersuchen. Als Ergebnis der Simulationen soll erkennbar sein, ab welchen Bedingungen das System seine Funktionalität nicht mehr garantieren kann. Die simulierte Zeit entspricht einem Tag ($Hours$=24). Die Knotenanzahl N wurde von 25 bis 2000 Knoten variiert. Die restlichen Parameter sind wie im Vorfeld für die Zugangsnetze zu bestimmen, da dieselben Prozesse durchlaufen werden. Dies umfasst die Prozesse der En- und Dekodierung durch die RS-Codes, die Datenübertragung und die Zeit für die Suche im Kad-Netzwerk. Die Zeit zum Enkodieren und Speichern eines Datensatzes im Netzwerk T_{Store} beträgt 675 ms. Die Zeit für das Zurückholen aus dem Netzwerk und dem Dekodieren des Datensatzes T_{Rest} beträgt 863 ms. Im Simulator wird automatisch ein Zeitwert auf T_{Store} und T_{Rest} hinzuaddiert. Dieser Zeitwert ist abhängig von der Anzahl der Knoten N und stellt die Suchzeit im Kad-Netzwerk dar, die sich aus der Anzahl der Suchschritte $log_2(N)$ ergibt. Zusätzlich wird zur Suchzeit die eine zweite Zeit für das Erstellen des Suchobjektes selbst berücksichtigt. Beide Zeiten müssen für jeden der 255 Chunks berücksichtigt werden. Die Ausfallzeit wird erneut mit 15 min angenommen. Diese Annahme unterscheidet sich nicht signifikant von Szenarien, bei denen die Instanzen komplett ohne Wiedereintritt ausfallen. Die umgesetzte Annahme hingegen stellt höheren Ansprüche an das System, da die Daten nach Wiedereintritt in jedem Fall verfügbar sein müssen. Zusätzlich entsteht ein höheres Datenvolumen, welches es zu berücksichtigen gilt.

Die Zeiten T_{Store} und T_{Rest} stammen von Messungen eines Prototyps. Der Prototyp basiert auf einem ZedBoard mit dem Echtzeit-Betriebssystem FreeRTOS [Avn, Rea]. Weitere Details zum Prototypen sind in Abschnitt 4.3.3 aufgeführt. Erweitert wurde der Prototyp um die Implementierung der Reed-Solomon-Codes. Es wurden für PSP-Auto, wie auch bei PSP, mehrere Ausfallprofile simuliert, um verschiedene (Extrem)-

Parameter	Wert	Beschreibung
N	25-2000	Anzahl der Knoten
$Data_{Node}$	15 KB	Datenmenge pro Knoten
$Hours$	24	Simulationszeit
T_{Store}	675 ms	Zeit zur Datenspeicherung
T_{Rest}	863 ms	Zeit zur Datenwiederherstellung
T_{Dist}	2 s	Verteilungsperiode
T_{Reset}	15 min	Ausfallzeit eines Knotens

Tabelle 3.4.: Simulationsparameter des Automatisierungsszenarios.

Situationen zu untersuchen. Die Ausfallprofile sind in Tabelle 3.5 dargestellt. Es ist bereits ersichtlich, dass die Ausfallprofile deutlich pessimistischer ausfallen als für das Umfeld der Zugangsnetze.

Ausfallprofil	tägliche Resets	P_N
1	0	100,000
2	1	98,958
3	2	97,917
4	5	94,792
5	6	93,750
6	8	91,667
7	10	89,583

Tabelle 3.5.: Ausfallprofile der ANs im Automatisierungsszenario.

3.2.4. Anpassung der Reed Solomon-Codes

Für die prototypische Entwicklung der Reed-Solomon-Codes auf der Zielplattform ZedBoard wurden einige Anpassungen gegenüber der ursprünglichen Variante in PSP durchgeführt. Grund hierfür ist die Limitierung der Zielplattform. Aus diesem Grund weichen die RS-Werte für PSP-Auto von denen bei PSP ab. In der prototypischen Umsetzung

wird eine RS(255,127)-Kodierung vorgenommen. Dabei bestehen 255 Bytes aus 127-Byte unkodierten Daten und 128 Bytes stellen die kodierten Bytes dar. Die implementierte Version benötigt noch $m + k/2$ Symbole zum Dekodieren, sodass hier noch weiteres Optimierungspotential vorliegt. Es lassen sich jedoch sehr gut die bereits genannten Zeiten für das En- und Dekodieren ermitteln, um eine realitätsnahe Simulation zu ermöglichen.

3.2.5. Simulationsergebnisse und Auswertung

Durch die Anpassung an das neue Szenario ergeben sich neue Simulationsparameter. Anhand der ermittelten Simulationsparameter für PSP-Auto wurden erneut Simulationen mit dem Ziel durchgeführt, die Datenmengen, Lastspitzen und vor allem die Robustheit von PSP-Auto zu untersuchen.

In Abbildung 3.7 ist das auftretende Datenaufkommen in GB für einen Tag angegeben. Dieses steigt linear mit der Anzahl der Knoten. Auch hier ist erkennbar, dass bei einer hohen Ausfallrate die gesamte Datenmenge abnimmt, da die ausgefallenen Knoten ihre Daten nicht mehr verteilt im Netzwerk speichern. Nimmt man an, dass es keine Ausfälle gibt, entspricht dies dem Fall, bei dem am meisten Daten generiert werden. Daraus ergibt sich eine durchschnittliche Datenrate für einen Knoten von 7,57 Byte/s. Diese geringe Last sollte die eingebetteten Systeme nicht überlasten.

Die Spitzenlast in KB pro Millisekunde ist in Abbildung 3.8 zu sehen. Mit steigender Knotenanzahl erhöht sich ebenfalls die Spitzenlast, weil die Wahrscheinlichkeit steigt, dass mehrere Knoten zum selben Zeitpunkt Daten austauschen. Ein weiterer Aspekt ist, dass mit mehr Ausfällen die Spitzenlast steigt, jedoch nicht das verteilte Datenvolumen. Dies ist dadurch zu erklären, dass ein Knoten nach einem Ausfall sofort wieder Daten verteilt und zuvor zusätzlich seine Daten einsammelt. Somit kann, wenn viele Knoten in einem zusammenhängenden Zeitraum zufällig ausfallen und kurz danach wieder ihre Daten verteilen, die Wahrscheinlichkeit für erhöhte Lastspitzen zunehmen. Abbildung 3.9 stellt dar, ob und wie oft Knoten ihre Daten nicht wiederherstellen konnten. Hier sieht man deutlich, dass wenige Knoten und hohe Ausfallraten zu einer exponentiellen Steigerung der Ausfälle führen. Bei nur 25 Knoten führen 5 Ausfälle pro Tag und pro Knoten bereits zu ersten Szenarien, bei denen Daten nicht wiederhergestellt werden können. Bei 100 Knoten ist dies bei 8 Ausfällen pro Tag und pro Knoten der Fall. Mit steigender Knotenanzahl wird das Netz dabei deutlich stabiler.

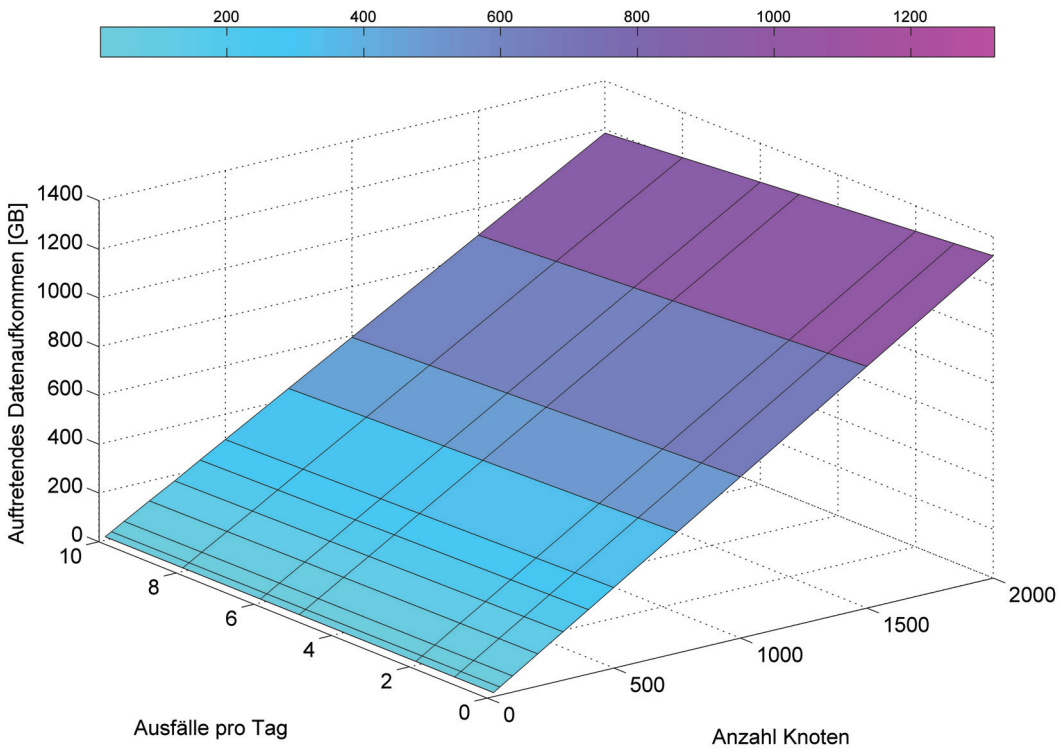

Abbildung 3.7.: Erzeugte Gesamtdatenmenge in GB für einen Tag.

Als weitere Schlussfolgerung ist festzustellen, dass, wenn die Suchtoleranz so eingestellt ist, dass immer jeder Chunk einmal gespeichert werden kann, dies bei einer niedrigen Anzahl an Knoten zu einer hohen Suchtoleranz führt. Damit tragen die einzelnen Knoten eine größere Last. Eine hohe Last bedeutet in diesem Zusammenhang eine hohe Anzahl an Chunks, die auf einer Instanz gespeichert wird. Wie ersichtlich ist, hat eine geringe Anzahl an Knoten den Nachteil, dass sie deutlich anfälliger sind für Ausfälle, da meist mehrere Chunks bei dem Ausfall eines Knotens verloren gehen. Mit steigender Anzahl der Knoten und der damit verbundenen kleineren Suchtoleranz werden die Chunks besser im Netzwerk verteilt und die Knoten müssen nicht mehr so viele Chunks speichern. Dadurch wird das Netzwerk mit zunehmender Knotenanzahl, welche beim verteilten

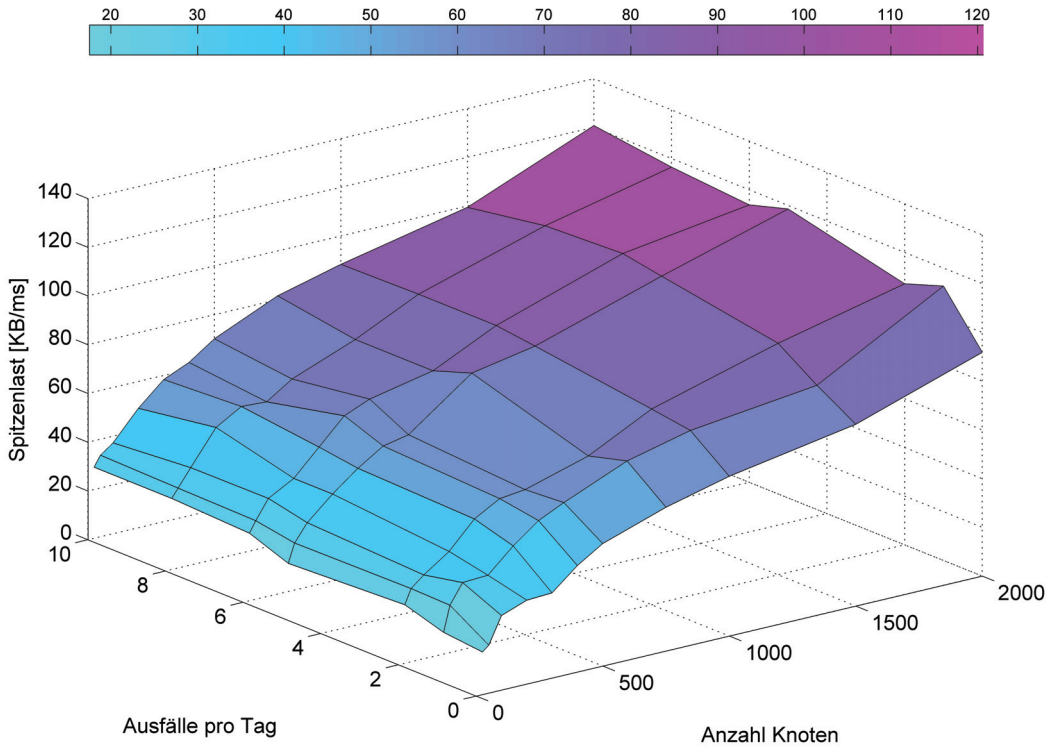

Abbildung 3.8.: Auftretende Spitzendatenlast pro Millisekunde in KB.

Speichern helfen, robuster. Bei 10 Ausfällen pro Tag und Knoten treten jedoch bei jeder simulierten Netzwerkgröße Szenarien auf, bei denen eine Rekonstruktion der Daten nicht möglich ist. Die Werte liegen bei ca. 0,1 bis 0,22 nicht mehr rekonstruierbaren Daten im Durchschnitt pro Tag. Dies liegt daran, dass die Suchtoleranz *ST* immer an die Netzwerkgröße angepasst wird und somit unnötige Redundanz vermieden wird. Da auch der Anteil des Netzes, welcher ausfällt, prozentual konstant ist, ist dieses Verhalten zu erwarten. Abhilfe würde hier nur die Erhöhung der *ST* schaffen, wenn ein derart pessimistisches Szenario als Ziel vorgegeben ist.

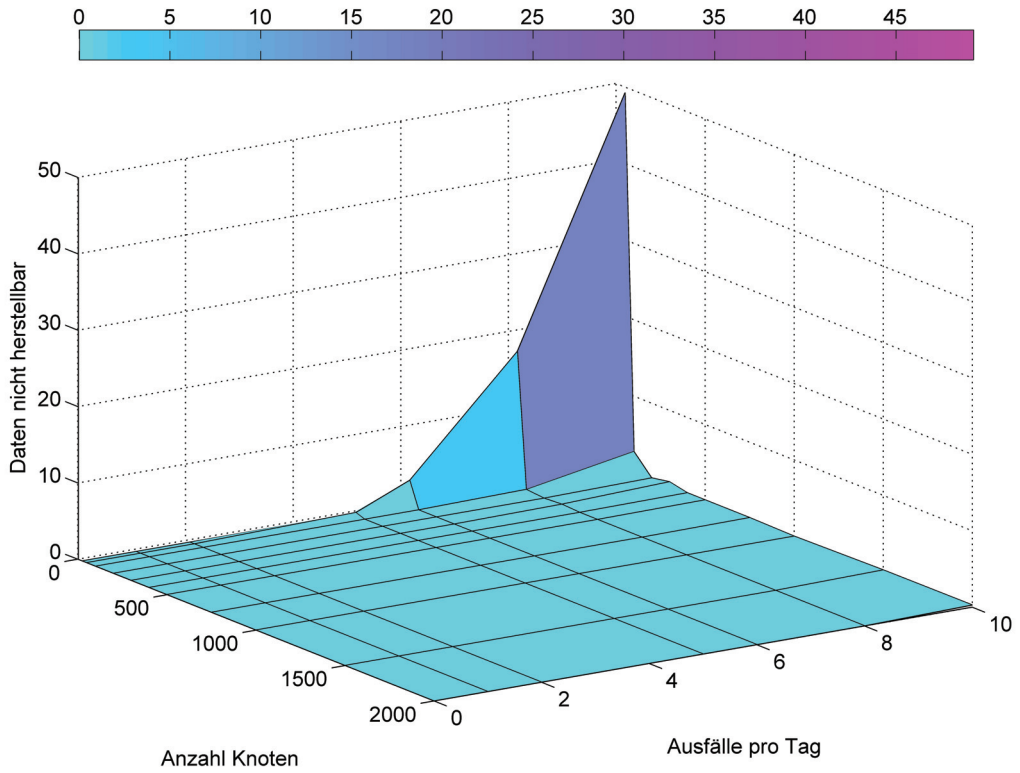

Abbildung 3.9.: Anzahl nicht wiederherstellbarer Datensätze im Verlaufe eines Tages.

3.2.6. Fazit PSP in der Automatisierungstechnik

Es wurde ein System namens PSP-Auto, welches einen verteilten Speicher in der Automatisierung mittels P2P-Technologie realisiert, entwickelt und mittels eines erweiterten Simulators mit erhöhter Zeitauflösung simuliert. Die Anpassungen an das Automatisierungsumfeld und die Ermittlung der Grenzen des Systems geben einen sehr guten Aufschluss über die Performance. Die mittels eines Prototyps als eingebettetes System ermittelten Parameter sind direkt in die Simulation eingeflossen. Die Ergebnisse zeigen eine hohe Zuverlässigkeit auch bei hohen Ausfallraten. Nur bei kleinen Netzen und sehr pessimistischen Ausfallraten war PSP-Auto nicht mehr in der Lage, Datensätze aus dem verteilten Speicher zu rekonstruieren. Das simulierte System besteht aus tausenden

Instanzen und weist eine weiche Echtzeitfähigkeit auf, was am neu verwendeten Prototypen liegt. Der Prototyp wird zur Bestimmung einzelner Simulationsparameter verwendet und weist durch das verwendete Betriebssystem sogar harte Echtzeiteigenschaften auf. Jedoch ist die Kommunikation der Teilnehmer untereinander durch die Verwendung von Standard-Ethernet noch nicht deterministisch.

Einordnung in den Gesamtkontext der Arbeit: PSP-Auto nutzt im Gegensatz zu PSP ausschließlich UDP zur Kommunikation im Kad-Netzwerk und zur Datenübertragung. Im Umfeld der Automatisierung lässt sich dies realisieren, da hier kleinere Datenmengen übertragen werden. PSP-Auto stellt die oberste Anwendung zur ausfallsicheren Datenhaltung in der Automatisierung dar (siehe Abbildung 3.10) und besitzt bereits weiche Echtzeitfähigkeit. Da nur UDP verwendet wird, ist es nicht möglich ohne zusätzlichen Aufwand größere zusammenhängende Datenmengen zu übertragen. Daher sollten diese in UDP-Pakete aufteilbar sein, um sie anschließend wieder zusammenführen zu können. Dieser Ansatz wurde bei PSP-Auto mittels der RS-Codes impliziert umgesetzt.

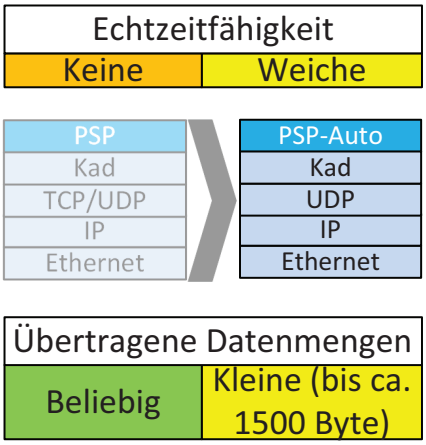

Abbildung 3.10.: Darstellung des PSP-Auto-Protokollstacks und Einordnung in den Gesamtkontext.

3.3. Kapitelzusammenfassung

In diesem Kapitel wurde das bereits prototypisch umgesetzte P2P-System zum verteilten Speichern namens PSP vorgestellt. Das Szenario der Zugangsnetze umfasst mehrere

tausend Knoten. Für dieses derart hochskalierte Netzwerk wurde die Charakteristik des Systems untersucht. Dies konnte erstmals mittels einer umfangreichen Simulation auf einem eigenen Simulator umgesetzt werden. Dabei konnten die Datenmengen und die Funktionalität von bis zu 8000 Knoten über einen simulierten Zeitraum von einem Jahr untersucht werden.

Im zweiten Teil des Kapitels wird untersucht, inwieweit sich PSP in der Automatisierung einsetzen lässt. Das neu entstandene System PSP-Auto wurde ebenfalls simuliert. Mit Hilfe der realen Parameter eines Prototyps konnten realistische Ergebnisse gewonnen werden. PSP-Auto selbst besitzt durch den Prototypen das Potential, mit weicher Echtzeit betrieben zu werden. Inwiefern sich P2P-Technologie auch mit harter Echtzeit für die Automatisierung umsetzten lässt, ist Thema des folgenden Kapitels.

Kapitel 4.

HaRTKad - P2P-Technologie mit harter Echtzeit

Dieses Kapitel startet mit einem umfassenden Überblick über bestehende Industrial Ethernet (IE)-Lösungen. IE wird eingesetzt, um Standard-Ethernet-Technologien im Automatisierungsumfeld nutzen zu können. Dabei werden die Vor- und Nachteile der einzelnen Systeme aufgeführt und jedes System wird entsprechend seiner Eigenschaften klassifiziert, um es vergleichbar machen zu können. Der nachfolgende Teil befasst sich mit dem neu vorgeschlagenen Verfahren HaRTKad (Hard Real-Time Kademlia), welches versucht, die Eigenschaften eines Echtzeitsystems durch einen nicht proprietären Ansatz deutlich zu verbessern. Im Vorfeld des HaRTKad-Ansatzes wird der Aspekt der Synchronisation behandelt, die nötig ist, um HaRTKad als TDMA-basierten Ansatz durchführen zu können. Abschließend wird das eigentliche Verfahren vorgestellt, wie sich der TDMA-Ansatz ohne jegliche zentrale Instanz in einem P2P-Netzwerk umsetzten lässt. Durch die Implementierung auf einer Zielplattform mit einem Echtzeitbetriebssystem entsteht ein hochskalierbares, ausfallsicheres und flexibles System zur Übertragung von Daten mit harten Echtzeitanforderungen.

4.1. Industrial Ethernet-Lösungen

4.1.1. Einleitung

Für echtzeitfähige Geräteverbindungen im industriellen Umfeld wurde ursprünglich eine Vielzahl an Feldbuslösungen eingesetzt. Allerdings weisen Feldbusse starke Einschrän-

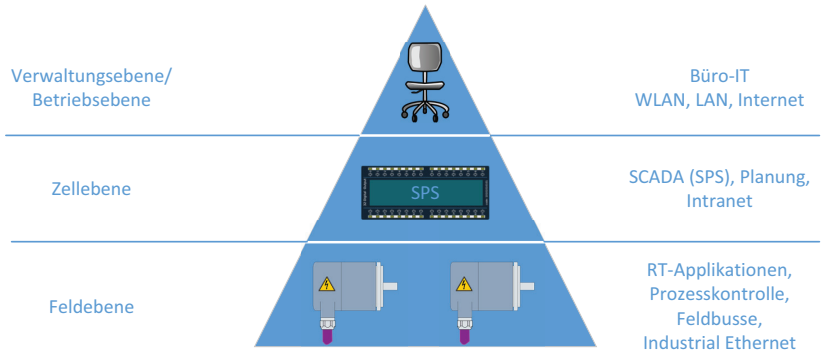

Abbildung 4.1.: Darstellung der Topologie eines Automatisierungsnetzwerkes [SL11].

kungen auf, wie die geringe Anzahl an Geräten in einem Netzwerk (geringe Skalierbarkeit) und eine niedrige Ausfallsicherheit, weil oft nur Linienstrukturen zugelassen sind. Außerdem ist eine Interoperabilität zwischen den verschiedenen Lösungen oftmals nicht gegeben, was ein Problem der Standardisierung ist [GJF13]. Daher setzt sich derzeit im Bereich der Automatisierung die weit verbreitete Ethernet-Technologie gegen Feldbusse durch [Pan13].

Die Anwendung von Ethernet zum Verbinden von Feldgeräten bietet verglichen mit Feldbussen erhebliche Vorteile. Sie erlauben eine vertikale und horizontale Integration eines Automatisierungssystems (siehe Abbildung 4.1) von der Feldebene bis zur Betriebsleitebene bzw. Unternehmensleitebene [Sau05, Sau10].

Feldgeräte können nun direkt mit herkömmlichen PCs eines Büros zusammenarbeiten, ohne ein Gateway zwischen Ethernet und anderen Feldbussen zu verwenden, wobei viele Herausforderungen entstehen. Eine davon ist z.B. die zukünftige Implementation zur Sicherstellung der korrekten Darstellung und Verarbeitung von Daten von der untersten Feldebene bis zur obersten Verwaltungsebene [SL11]. Über Ethernetverbindungen zwischen den Feldgeräten werden zeitkritische Prozessdaten für Kontrollaufgaben gesendet, während nicht-kritische IT-Daten an verschiedene IT-Services der Anlage gesendet werden [KOV11]. Auf diese Weise können Statusinformationen von Feldgeräten ausgelesen oder ferngesteuert werden. Für die Übertragung der IT-Daten werden Standardprotokolle wie TCP/IP oder UDP/IP genutzt, wobei zeitkritische Prozessdaten u.U. die Verwendung von speziellen Protokollen erfordern. Die Kommunikation mittels Ethernet geschieht aber auf einer einheitlichen Hardwarebasis, welche weltweit durch IEEE 802.3

standardisiert ist. Verschiedene Verbindungen und Kommunikationsmedien werden für definierte Verwendungszwecke spezifiziert. Ein weiterer Vorteil von Ethernet ist die hohe Performance gegenüber konventionellen Feldbussen, welche sich zu Schwachpunkten zwischen hoch performanten Computersystemen entwickeln.

Allerdings verwendet das heute genutzte Standard-Ethernet verschiedene Mechanismen, welche einen deterministischen Datenverkehr und somit den Einsatz in einer Umgebung mit Echtzeitanforderung verhindern. Das erste Problem von Standard-Ethernet ist die Verwendung des CSMA/CD-Zugriffsverfahrens. Aufgrund von möglichen Kollisionen kann die Datenübertragung unterbrochen werden und zu einem späteren Zeitpunkt stattfinden, was nicht deterministisch ist, da sich auch diese Kollisionen wiederholen können. Mit der Verwendung von Full-Duplex Switched Ethernet kann dieses Problem gelöst werden, wobei es trotzdem zu Überlastsituationen, die berücksichtigt werden müssen, kommen kann. In Switches werden Daten gepuffert, was zu einer zusätzlichen Verzögerung oder im schlimmsten Fall zu Paketverlusten bei zu hohen Verkehrslasten führen kann. Wenn z.B. Netzwerkteilnehmer zu viele Daten an das gleiche Ziel senden, kann es in den Switches zu Pufferüberläufen kommen, sodass mit Paketverlusten zu rechnen ist. Dies widerspricht der deterministischen Datenübertragung, da ggf. Pakete neu gesendet werden müssen, bei denen ebenfalls keine deterministische Übermittlung garantiert werden kann. Auf dem IP-Layer oberhalb von Ethernet gibt es zusätzlich das Problem der nicht-statischen Routen, sodass die Übertragungszeiten von Paketen nicht präzise vorhersagbar sind.

Ebenso ist die Wahl des Transportprotokolls für hohe Echtzeitanforderungen herausfordernd, da auch der Transport großer Datenmengen in zukünftigen Applikationen eine größere Rolle spielen wird und trotzdem deterministische vorhersagbare Übertragungszeiten zuzusichern sind. Dies wird in Zukunft immer wichtiger. Die IEEE Time Sensitive Networking (TSN) Task Group z.B. ist bestrebt, zeitsynchronisierte Streaming-Services mit geringer Latenz in 802-Netzwerken zu ermöglichen [IEE14]. Auch Automobilingenieure zeigen Interesse an dieser Arbeit, um z.B. verteilte Streams in Autokommunikationssystemen umzusetzen [SLK+12]. Daher kann diese Technologie in Zukunft an Verbreitung gewinnen. Während UDP-Nachrichten auf Grund der Sendung einzelner unabhängiger Pakete als echtzeitfähig angesehen werden können, ermöglichen sie nicht die Interpretation von großen Datenmengen, da dies den darüber liegenden Schichten überlassen wird. Im Gegensatz dazu ist TCP darauf ausgelegt, große Datenmengen zu

übertragen und damit eine Deklarierung der Datenreihenfolge zu ermöglichen. Aber auf Grund der variablen Übertragungsgeschwindigkeit, welche durch die Überlastkontrolle und Fehlerkorrektur entsteht, ist TCP grundsätzlich nicht echtzeitfähig.

Eine Vielzahl an echtzeitfähigen IE-Systemen wurden bereits entwickelt, welche die Defizite von Standard-Ethernet und TCP/IP oder UDP/IP-basierter Kommunikation auf verschiedenste Arten behandeln [Fel10]. „Industrial“ verweist dabei auf die Eignung der Lösungen für das raue Industrieumfeld [Wil11]. Daher werden Zertifizierungen angegeben, um die Konformität mit festgelegten Anforderungen im Industrieumfeld nachzuweisen. Da Standard-Ethernet den CSMA/CD-Mechanismus zur Kontrolle des Medienzugriffes nutzt, kann weder die Übertragung eines Ethernet-Frames noch die Zulieferzeit garantiert werden. Um Ethernet allerdings in automatisierten Anlagen etc. verwenden zu können, müssen die IE-Lösungen dies zusichern. Kontrollsysteme müssen als Echtzeitsystem derart konzipiert und implementiert sein, dass Daten innerhalb eines fixen Zeitlimits übertragen werden. Weder Switched noch Full-Duplex Ethernet können fixe Zeitlimits sicherstellen oder den Jitter von Ethernet-Frames eingrenzen. Der Jitter beschreibt dabei die zeitliche Abweichung bei der Paketzustellung zu einem Teilnehmer. Für Multimediaapplikationen wurde dieses Problem durch Priorisierung der Datenflüsse und Verwendung vom virtuellen LAN-Standard gelöst [IEE04], [IEE05]. Dabei konnte der Jitter reduziert und eine Zulieferzeit von 10 ms im Falle von hoch priorisiertem Datenverkehr erreicht werden. Allerdings sind dies nur statistische Abschätzungen. Um niedrige deterministische Zulieferzeiten zu garantieren, muss entweder das Ethernet-Protokoll selbst modifiziert werden und/oder Netzwerkkomponenten/-geräte müssen Maßnahmen ergreifen, um die Kommunikation zu verwalten. Echtzeit mittels Ethernet kann auch durch den Einsatz eines Tokenprinzips erreicht werden [MCB+11]. Dies hat jedoch auch Nachteile, die durch das Prinzip selbst gegeben sind (siehe Abschnitt 2.2.1).

Daher wird bei den meisten IE-Systemen ein Zeitmultiplexmechanismus verwendet, bei dem jedes Gerät in einem Zeitschlitz kommunizieren darf. Dies setzt eine einheitliche Zeitbasis für alle Geräte voraus, d.h. die Synchronisation aller Geräte ist erforderlich.

In diesem Unterkapitel werden die Vor- und Nachteile verschiedener Lösungen bezüglich ihrer Zukunftsfähigkeit analysiert:

- Echtzeitfähigkeit: Welche Echtzeit kann im besten Fall erreicht werden (weiche oder harte)?

- Zuverlässigkeit: Besitzt das Netzwerk einen Single Point of Failure (SPoF)?
- Skalierbarkeit: Wie viele Geräte können Daten in Echtzeit austauschen? Perspektifisch wird es einige tausend Geräte geben, die es in der Zukunft zu vernetzen gilt [EA12].
- Selbstkonfiguration: Besitzt das System Selbstkonfigurationseigenschaften wie das dynamische Anpassen bei Änderungen in der Netzwerktopologie oder muss man statisch/manuell konfigurieren?
- Hardwarevoraussetzungen: Ist eine spezielle Hardware nötig, wie z.B. spezielle Router?

Die Hauptbeiträge dieses Unterkapitels sind in [DSA+14] publiziert. Der Rest dieses Abschnitts ist wie folgt oganisiert: Zuerst werden in Abschnitt 4.1.2 Definitionen vorgenommen, anhand derer der spätere Vergleich der IE-Systeme vorgenommen wird. In Abschnitt 4.1.3 werden etablierte IE-Systeme präsentiert. In den darauffolgenden Abschnitten 4.1.4 und 4.1.5 wird vor dem Hintergrund der enorm steigenden Anzahl an Geräten ein Ausblick und der Stand aktueller Entwicklungen im wissenschaftlichen Umfeld gegeben. Zum Abschluss folgt eine Übersichtstabelle zum Vergleich aller präsentierten Lösungen.

4.1.2. Echtzeitfähige Industrial Ethernet-Systeme

Ein echtzeitfähiges System kann eine Antwort auf eine Anfrage innerhalb einer Zeitschranke gewährleisten. Die Zykluszeit in einem Echtzeitsystem definiert die Zeit, in der die Kommunikationsprozesse des Gesamtsystems einmal durchlaufen wurden, sodass i.d.R. jeder Netzwerkteilnehmer einmal kommunizieren konnte. In [Neu07] werden drei Klassen von Echtzeitsystemen definiert, abhängig von zugesicherten Antwortzeiten durch Datenpakete innerhalb des Systems. Entsprechend ihrer Echtzeitfähigkeit können sie für drei verschiedene Kontrollaufgaben definiert sein [Fel05]: Human Control, Process Control oder Motion Control.

- Klasse 1: weiche Echtzeit: variable Zykluszeiten (typisch ca. 100 ms), Human Control.
- Klasse 2: harte Echtzeit: Zykluszeiten von 1 bis 10 ms, Process Control.

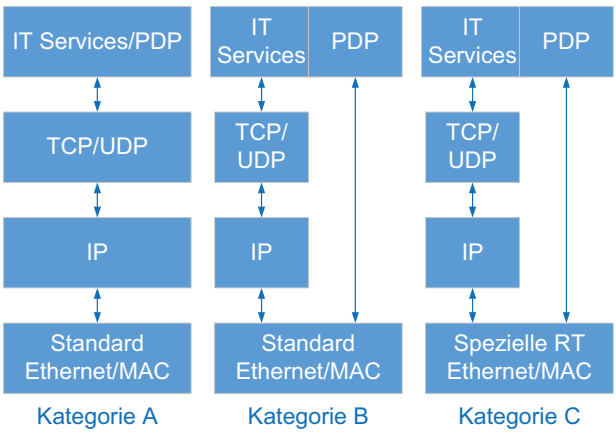

Abbildung 4.2.: Kategorisierung von IE-Systemen in Bezug auf Software- und Hardwareanforderungen für die Endgeräteimplementation [KOV11, Ros11].

- Klasse 3: isochrone Echtzeit: Zykluszeiten von 250 μs bis 1 ms; Jitter geringer als 1 μs, Motion Control.

Diese Klassifizierung wird im Folgenden verwendet und jedes IE-System wird nach seiner besten erreichbaren Echtzeitfähigkeit eingeordnet. Des Weiteren wird ein Überblick über die besten Zulieferzeiten, welche die entsprechenden Systeme erreichen können, gegeben. Die Zulieferzeit ist dabei als Zeit für das Zustellen einer Information von einer Instanz zu einer beliebigen anderen Instanz im Netzwerk definiert.

4.1.3. Etablierte IE-Systeme

Dieser Abschnitt gibt einen Überblick der derzeitig etablierten IE-Systeme. Zusätzlich zu der bereits eingeführten Klassifizierung in Bezug auf die Echtzeitfähigkeit wird jedes System nach den Software- und Hardwareanforderungen kategorisiert (siehe Abbildung 4.2) [KOV11, Ros11]:

- Kategorie A basiert komplett auf TCP/UDP/IP und nutzt Standard-Ethernet-Controller und Switches.
- Kategorie B nutzt gewöhnliche Ethernet-Controller und Switches, aber ein spezielles proprietäres Prozessdatenprotokoll (PDP) wird oberhalb von Ethernet eingesetzt.

- Kategorie C verwendet ein dediziertes PDP wie Kategorie B und benötigt zusätzlich spezielle echtzeitfähige Ethernet-Controller und/oder Switches.

4.1.3.1. Modbus-TCP

Modbus-TCP wurde ausschließlich für Applikationen mit weicher Echtzeit konzipiert und fällt damit in die Klasse 1. Es ist oberhalb von TCP/IP auf der Applikationsebene realisiert und basiert komplett auf Standard-Ethernet-Komponenten (unter der Verwendung von Ethernet als Medium) und stellt damit einen der wenigen Kategorie-A-Ansätze dar [Ros11]. Modbus-TCP verwendet den eigens zugesicherten TCP Port 502 zur Übertragung von Daten und kann damit auch per Fernzugriff kontrolliert werden [MI06]. Da es auf der Applikationsebene arbeitet, kann das Protokoll auf alle möglichen Kommunikationsgeräte angewandt werden. Zusätzlich ist es leicht zu konfigurieren, weil es einen hohen Grad an Selbstkonfiguration aufzeigt. Modbus-TCP ist als Client-Server-Architektur implementiert, wobei jedes Gerät Client oder Server sein kann, was für eine hohe Zuverlässigkeit sorgt. Der Server verarbeitet Anfragen der Clients und bestätigt diese mit Erfolgs- oder Fehlermeldungen. Die Anzahl der Geräte, die mittels Modbus-TCP verbunden werden können, ist quasi nur durch die Rechenleistung der Geräte selbst beschränkt.

4.1.3.2. Ethernet Powerlink

Ethernet Powerlink erlaubt generell eine Echtzeitkommunikation mit Standard-Ethernet-Hardware, da das Protokoll komplett in Software implementiert werden kann (Kategorie B) [EPS12]. Allerdings ist für Applikationen mit hohen Anforderungen an die Zykluszeit der Echtzeitkommunikation, wie der isochronen Echtzeit (Klasse 3), eine Implementierung in Hardware nötig (Kategorie C) [KOV11]. Das Powerlink-Protokoll ist zwischen Ethernet und IP einzuordnen. Das entwickelte Zeitschlitzzugriffsverfahren bzgl. der Ethernetverbindung basiert auf dem Master-Slave-Prinzip und wird *Slot Communication Network Management* genannt, das einen zyklischen Prozessdatenaustausch sowie azyklische Services und Kontrolldaten unterstützt. Der Master, „Powerlink Managing Node“ genannt, erteilt die Erlaubnis an die Slaves, sequentiell ihre Daten während eines Zyklus zu senden. Daher ist der Master maßgeblich für die erfolgreiche Ausführung von

Powerlink-Operationen verantwortlich, und ein Ausfall des Masters würde den Komplettausfall des Systems zur Folge haben.

Die Skalierbarkeit in Bezug auf die Anzahl der verbundenen Geräte ist stark limitiert, da Powerlink 8-Bit-Adressen verwendet. Die Adressen müssen manuell konfiguriert werden, Powerlink aber ist hot-plug-fähig, d.h. Geräte können zur Laufzeit des Systems hinzugefügt und entfernt werden, was einen gewissen Grad an Selbstkonfiguration darstellt. Auf der Applikationsschicht nutzt Powerlink den CANopen-Standard für die echtzeitfähige Datenübertragung, welcher eher für die Übertragung von Prozessdaten anstatt großer Datenmengen vorgesehen ist.

4.1.3.3. EtherCAT

EtherCAT ist ein IE-System, welches auf dem Master-Slave-Prinzip basiert und die Verarbeitung von zyklischen Prozessdaten von Feldgeräten (Slaves) ermöglicht [Eth13b, KOV11].

Für die Kommunikation wird ein Prozessimage beim Master erstellt, welches den aktuellen Zustand der verschiedenen Ein- und Ausgänge des Gesamtsystems, bestehend aus den Slaves, abbildet. Um den Zustand eines bestimmten Ausgangs eines Slaves zu ändern, muss der entsprechende Teil des Prozessabbildes zusammen mit dem Änderungsbefehl übertragen werden. Slaves selbst können während des zyklischen Datenaustausches Teile des Prozessdatenabbildes zum Master senden, um ihre Statusinformationen der Eingänge zu aktualisieren. Die Zuteilung der Teile des Prozessdatenabbildes der Ein- und Ausgänge eines einzelnen Slaves geschieht durch logische Adressen, welche in physikalische Adressen einzelner angeschlossener Geräte im EtherCAT Slave Controller (ESC) übersetzt werden [Eth13b, KOV11, Eth13a]. Teile des Prozessdatenabbildes und entsprechende Kommandos zur Änderung von Ausgängen werden direkt durch Ethernet-Frames oder als UDP-Payload übertragen. Diese EtherCAT-Frames werden zyklisch vom Master gesendet und passieren dabei die Slaves sequentiell in einer Ring-Struktur. Im Gegensatz zu Standard-Ethernet-Controllern, welche Frames puffern, verarbeiten und einen neuen Frame generieren, findet die Verarbeitung bei den ESCs komplett in Hardware statt, wenn die Frames einen ESC passieren (Kategorie C) [Eth13b].

EtherCAT ermöglicht daher ein hoch performantes System für die Echtzeitdatenübertragung mittels Ethernet. Aufgrund der schnellen Datenverarbeitung in den ESCs und

dem geringen Overhead sind Zykluszeiten weit unter 1 ms möglich (Klasse 3).

EtherCAT ist darauf ausgelegt, einfach Diagnosen und Konfigurationen durchführen zu können. Daher werden Konfigurationstools unterstützt, welche es ermöglichen, die Netzwerktopologie darzustellen und automatisch Geräte im Netzwerk zu konfigurieren. Allerdings ist die Anwesenheit eines Masters, welcher alle administrativen Aufgaben übernimmt und alle Daten und Adressen des Netzwerkes speichert, unabdingbar. Die Skalierbarkeit ist aufgrund des logischen Prozessdatenabbildes limitiert. Der Vorteil des geringen Overheads durch die Aggregierung der Daten verschiedener Empfänger wird stark gemindert, sobald das Netzwerk dafür genutzt wird, große Mengen an Daten zu übertragen - sofern dies möglich ist.

4.1.3.4. TCnet

TCnet, entwickelt von Toshiba, erweitert die ursprüngliche Zugriffsmethode von IEEE 802.3 bei Ethernetverbindungen und führt vier Kommunikationsklassen für verschiedene Prioritäten ein.

Die vier Kommunikationsklassen sind (sortiert nach der Übertragungspriorität mit der höchsten beginnend):

- Zyklische Daten mit hohen Zeitanforderungen (Klasse 3)
- Zyklische Daten mit normalen Zeitanforderungen
- Azyklische Daten
- Zyklische Daten mit geringen Zeitanforderungen

Die zyklische Datenübertragung wird von Echtzeitapplikationen genutzt und findet nur in einem Subnetz statt, da die Daten direkt in den Ethernet-Frames eingekapselt sind.

Um die neue Zugriffsmethode zu ermöglichen, muss jeder TCnet-Teilnehmer einen speziellen TCnet-Ethernet-Controller besitzen (Kategorie C). Vor der Inbetriebnahme muss eine Stationsnummer für jeden Teilnehmer manuell konfiguriert werden, wodurch die Übertragungsreihenfolge für den zyklischen Datenaustausch festgelegt wird [Tos13,Fel05]. Ein weiteres Prinzip, das von TCnet genutzt wird, ist der gemeinsam genutzte Speicher der TCnet-Stationen, die sich an der Echtzeitkommunikation beteiligen. Jede Station besitzt eine eigene Kopie des gemeinsamen Speichers und kann folglich auf alle Prozessdaten

der anderen Stationen zugreifen. Es handelt sich dabei um eine Echtzeitkommunikation, welche für Prozessdaten ausgelegt ist, die in der Größe des Speichers von 256 KByte integrierbar sind (definiert durch Toshiba). Derartige Speichergrößen sind nicht länger praktikabel, wenn größere Mengen an Daten übertragen werden müssen.

4.1.3.5. TTEthernet

TTEthernet ist ein echtzeitfähiges IE-System, welches generell mit Standard-Ethernet-Systemen kombiniert werden kann und daher eine Vielfalt an Applikationen ermöglicht.

TTEthernet definiert drei Nachrichtentypen (time triggered, rate constrained, best effort), wobei time triggered Nachrichtentypen für isochrone Echtzeitapplikationen designt wurden (Klasse 3) [TG13]. Um die Daten mittels der genannten Nachrichtentypen zu senden, sind spezielle Switches erforderlich, welche das TTEthernet unterstützen (Kategorie C). Um die TTEthernet-Pakete zum richtigen Zeitpunkt abzusenden und Übertragungskanäle für andere Nachrichtentypen zu blockieren, ist eine einheitliche Zeitbasis der Switches und verbundenen Geräte unabdingbar.

Hierbei können bei der Konfiguration des Netzwerkes ein oder mehrere Master-Geräte definiert werden, welche die Synchronisierung der Uhren übernehmen. TTEthernet unterstützt eine deterministische Übertragungsmethode, basierend auf den untersten beiden OSI-Schichten. Daher ist die Übertragung auf höheren Schichten durch TTEthernet komplett transparent und es existiert dafür keine Spezifikation, welche Art von Daten übertragen werden. Generell kann demnach eine Echtzeitdatenübertragung für eine große Menge an Daten integriert werden.

4.1.3.6. CC-Link IE Field

Das von Mitsubishi Electric entwickelte CC-Link IE Field basiert auf Full-Duplex Gigabit-Ethernet, womit eine nahezu beliebige Topologie aufgebaut werden kann, welche auch harte Echtzeit erreicht (Klasse 2 [Ros11]). Die Feldgeräte werden mittels einer Stationsnummer von 1 bis 254 adressiert, was eine starke Limitierung der Anzahl der Geräte zur Folge hat [KOV11]. Ähnlich wie bei anderen IE-Systemen nutzt CC-Link IE Field das Master-Slave-Prinzip. Der Master ist zuständig für die Initialisierung des gesamten Netzwerkes und kontrolliert die Datenübertragungen. Zusätzlich speichert der Mas-

ter alle Ein- und Ausgänge aller Geräte. Der Informationsspeicher des Masters ist auf 32.768 Bits und 16.384 Worte begrenzt. Die Datenübertragung findet zyklisch im Sinne eines Token-Mechanismus statt. Angefangen beim Master wird das Token im Netzwerk weitergereicht, was dem aktuellen Token-Besitzer das Senden von Daten erlaubt. Um CC-Link IE Field realisieren zu können, muss entweder der Slave spezielle Hardware besitzen (Mitsubishi CP220 Chip) oder eine Implementierung des Seamless Message Protocols wird vorausgesetzt (Kategorie C) [CC-13a, CC-13b]. Wie in [Ros11] vermerkt, ist es für Dritte schwierig, CC-Link IE Field zu implementieren, da keine Einführung außerhalb von Mitsubishi existiert.

4.1.3.7. Profinet

Die Kommunikation findet in Profinet zyklisch statt und ist in verschiedene Phasen unterteilt [PRO12]. Jeder Zyklus startet mit der isochronen Phase, in welcher Isochronous Real-Time(IRT)-Frames übertragen werden (Klasse 3). Die Übertragung und das Routen von IRT-Frames wird bereits während der Installation des Netzwerkes konfiguriert. Mittels synchronisierter Clocks weiß jedes Gerät den genauen Zeitpunkt, wann es IRT-Frames senden kann. Die Synchronisation wird durch einen Master realisiert. Trotz der Datenübertragung mittels Ethernet-Frames findet die Adressierung nicht anhand der MAC-Adressen statt, sondern die Frames werden entsprechend der geforderten Übertragungszeit durch Switches auf fixen Routen weitergeleitet. Daher sind spezielle Profinet-Switches erforderlich (Kategorie C). Nach der isochronen Phase folgt eine weitere Echtzeitphase und eine finale Phase, in der nicht zeitkritische Daten mittels UDP oder TCP erlaubt sind [Fel05]. Wie in [Ros11] beschrieben, ist der entscheidende Aspekt von Profinet IRT die komplexe Systemplanung. Trotzdem erreicht Profinet einen bemerkenswert hohen Marktanteil (14,5% geschätzt für 2015) auf Grund von Siemens' großer Unterstützung und Support für die eigene Entwicklung.

4.1.3.8. EtherNet/IP

Ethernet/IP basiert auf dem Common Industrial Protocol (CIP), welches oberhalb von TCP/IP und UDP/IP sitzt (Kategorie A). Für die Echtzeitübertragung der Prozessdaten wird UDP/IP genutzt, wobei eine direkte Kommunikation zwischen allen Geräten möglich ist. Ethernet/IP kann mit allen Protokollen auf der Applikationsebene arbeiten

und die Anzahl der Geräte ist generell nicht limitiert. Allerdings ist in der Praxis eine offensichtliche Limitierung vorhanden, wenn isochrone Echtzeit erreicht werden soll. Des Weiteren ist eine Zeitsynchronisation in Hardware erforderlich (Klasse 3). Dies bezieht sich auf spezielle Switches mit integrierter IEEE 1588 Zeitstempelunterstützung (Kategorie C), da die Software Stack Performance nicht ausreichend wäre. Erschwerend kommt hinzu, dass Router Multicast/Broadcast-Kontrollfeatures besitzen müssen und dass es keinen Standard zur Implementation oder Konfiguration dieser Features gibt [Ros11].

4.1.3.9. SERCOS III

In SERCOS III werden die Geräte in einer Doppelringstruktur organisiert und mit Hardwareredundanz ausgestattet. Alternativ lässt sich SERCOS III in einer Linienstruktur ohne Hardwareredundanz realisieren [Ser12]. Pro Ring/Linie ist die maximale Anzahl an Geräten auf 511 beschränkt, was nachteilig bzgl. der Skalierbarkeit ist. SERCOS III bewahrt seine Kommunikationseigenschaften auch im Falle eine Fehlers wie Kabelbruch oder ausgefallener Knoten. Zusätzlich können neue Geräte zur Laufzeit eingebunden werden, womit SERCOS III einen hohen Grad an Selbstkonfiguration und Flexibilität aufweist. Die Kommunikation in SERCOS III basiert auf einem Zeitschlitzverfahren mit zyklischer Telegrammübertragung auf Basis des Master-Slaves-Prinzips. Ein zentraler Master sendet sogenannte *Master Data Telegramme* an die Slaves, welche aber auch direkt miteinander kommunizieren können (*Cross Kommunikation-* und *Controller-to-Controller*-Kommunikationsprofile genannt). Die Telegramme basieren auf Standard-Ethernet-Frames und es wird nur eine kleine Menge an Prozessdaten übertragen. Der SERCOS III Master kann komplett in Software realisiert werden (Kategorie B, SERCOS III SoftMaster). Um aber eine isochrone Echtzeit (Kategorie C) zu erreichen, muss er mittels spezieller Hardware realisiert sein.

4.1.3.10. Vergleich etablierter IE-Systeme

Alle untersuchten IE-Systeme zeigen generell ähnliche Prinzipien, welche lediglich auf verschiedene Arten implementiert wurden. Einige Systeme verwenden einen Shared Memory und alle Systeme benötigen einen Master (bei TCnet nicht angegeben) oder ein vergleichbares Managementsystem, welches die Kontrolle über die Kommunikation hat. Ein weiterer Aspekt ist, dass die meisten Systeme manuell konfiguriert werden müssen. Der

Aufwand der manuellen Konfiguration, welcher angewendet werden muss, wenn neue Geräte integriert oder geändert werden müssen, unterscheidet sich je nach IE-System. Jedoch sollte der Aufwand generell gering sein, um eine hohe Flexiblität zur Laufzeit erreichen zu können.

Alle isochronen echtzeitfähigen Lösungen haben gemeinsam, dass neue Geräte von einem Master erkannt werden müssen, um den Kommunikationsmechanismus anzupassen und den neuen Geräten Zeitschlitze oder eine Polling- bzw. Tokenprozedur zuzuweisen. Der Master als zentrale Instanz stellt einen SPoF und Flaschenhals dar und schränkt das System in der Zuverlässigkeit und Skalierbarkeit ein. Ein sich komplett selbstorganisierendes Netzwerk, in dem jedes Gerät autonom agiert, existiert bis dato nicht. Vielmehr benötigen alle Realisierungen dedizierte und teure Hardware, um ein isochrones Echtzeitverhalten zu realisieren.

Eine weitere Gemeinsamkeit der Systeme ist die Optimierung bzgl. Echtzeitübertragung von geringen Prozessdatengrößen. Keines der IE-Systeme unterstützt ein Transportprotokoll, um in der Lage zu sein, größere Mengen an Daten wie z.B. Streams in Fahrzeugkommunikationssystemen zu unterstützen [SLK+12] und dabei eine deterministische vorhersagbare Übertragungszeit zu garantieren.

TTEthernet bildet eine Ausnahme, da es eine Echtzeit-Ethernet-Lösung darstellt, welche keine Einschränkungen der Datenübertragung auf höheren Schichten aufweist, allerdings auch kein Protokoll spezifiziert.

4.1.4. Ausblick Industrial Internet of Things: Abseits von Master-Slave-Ansätzen und spezieller Hardware

Zusammenfassend gibt es derzeit bei den etablierten Echtzeit-Ethernet-Technologien keine Lösung ohne Master und spezieller Hardware.

Zusätzlich erlaubt keines der existierenden Systeme ein nahtloses Skalieren mit steigendem Datenaufkommen bei einer zuverlässigen Echtzeitübertragung. Diese Kernaspekte werden zukünftig immer mehr an Bedeutung gewinnen. In [EA12] wird bereits erwähnt, dass die Zukunft der Industrieanlagen aus intelligenteren Geräten bestehen wird. Die Geräte steigen nicht nur in ihrer Leistungsfähigkeit, sondern auch enorm in ihrer Anzahl. Trotzdem müssen die Geräte dynamisch miteinander agieren können. Fabrikanlagen

als ein Hauptgebiet für zukünftige Applikationen werden somit aus mehreren tausend Geräten bestehen, welche alle ein Echtzeitverhalten benötigen. Des Weiteren ist eine hohe Flexibilität eine immer größere Herausforderung und kann nicht mittels hierarchischer und zentralisierter Systeme auf Grund ihres ausgeprägten statischen Verhaltens realisiert werden. Verteilte dezentralisierte Systeme können helfen, diesen Umstand entgegenzuwirken [BS11]. Daher werden derzeitige Lösungen schnell an ihre Grenzen stoßen und sind nur eingeschränkt für zukünftige Anforderungen in Bezug auf Zuverlässigkeit, Skalierbarkeit und Flexibilität gewappnet. Sie benötigen entweder eine komplette Überarbeitung oder eine neue Lösung muss entwickelt werden.

4.1.5. Derzeitige Entwicklungen

Es gibt bereits einige Entwicklungen, die die Schwächen der etablierten IE-Systemen lösen wollen.

Distributed Real-Time Protocols for Industrial Control Systems (DRTP): Schmidt et al. [SS12] schlagen eine Echtzeit-Ethernet-Lösung vor, welche verteilt arbeiten kann und die Klasse 3 erreicht. Die Bandbreite des geteilten Ethernet-Mediums kann dynamisch allokiert werden. Das Konzept basiert auf zwei zusätzlichen Schichten oberhalb der Ethernet-Schicht, um den Medienzugriff und einen Master zu managen, welcher die Synchronisation der Slaves mittels des IEEE 1588-Synchronisationsprotokolles vornimmt (Kategorie B). Das Ergebnis ist ein TDMA-Ansatz, welcher Zeitschlitze nutzt. Allerdings gibt es keine Aussage über die mögliche Anzahl an Teilnehmern oder aber den Einsatz in hochskalierten Netzwerken. Des Weiteren werden TCP/UDP und IP nicht unterstützt, was eine vertikale und horizontale Integration ohne Mehraufwand unmöglich macht. Die Arbeit der Autoren klingt vielversprechend. Besonders die dynamische Bandbreitenallokation ist eine interessante Selbstorganisationseigenschaft. Allerdings sind viele Implementationsaspekte noch ungeklärt. Zudem wird immer noch eine zentrale Instanz zur Synchronisation benötigt.

Design and application of a real-time industrial Ethernet protocol (DARIEP): In [HLZL13] wurde ein Echtzeit-IE-Protokoll entwickelt, welches das Master-Slave-Prinzip verwendet (Klasse 3). Der Master wurde unter Linux entwickelt und verwendet das Time Application Interface und koordiniert den Echtzeitdatenaustausch der Knoten über präzise Zeitzyklen. Der Slave basiert auf einem ARM-Chip und FPGA, sodass spezielle

Hardware benötigt wird (Kategorie C). Der Ansatz der Autoren erscheint als eine hoch performante Alternative gegenüber den bestehenden IE-Systemen mit dem Nachteil der benötigten dedizierten Hardware.

Ein Vergleich aller präsentierten etablierten IE-Systeme und neuen Entwicklungen ist in Tabelle 4.1 aufgeführt. Der geschätzte Marktanteil der einzelnen IE-Systeme für 2015 wird angegeben als Indikator für die entsprechende Marktdurchdringung. Die maximale Anzahl der Geräte wurde aus [Fel10] entnommen, falls nicht anders spezifiziert. Zusätzlich wird die Zulieferzeit für die Systeme bei ihrer höchsten zu erreichenden Echtzeitklasse angegeben. Einige Daten waren dabei nicht verfügbar (n/a) oder in den betreffenden Beschreibungen nicht spezifiziert (n/s).

IE-System	Modbus-TCP	Ethernet Powerlink	EtherCAT
Zulieferzeit [ms]	1-15	0,4	0,15
Klasse/Kategorie	1/A	3/C	3/C
SPoF?	nein	ja	ja
Skalierbarkeit	Unlimitiert [KOV11]	4	180
Selbstkonfiguration	ja	ja	ja
Spezielle HW?	nein	optional	ja (Slaves)
Marktanteil [%]	6,4	4,2	3,1
IE-System	TCnet	TTEthernet	CC-Link IE Field
Zulieferzeit [ms]	2	n/a	1,6
Klasse/Kategorie	2/C	2/C	2/C
SPoF?	n/s	ja	ja
Skalierbarkeit	24	n/s	254 [KOV11]
Selbstkonfiguration	nein	nein	nein
Spezielle HW?	ja (TCnet Controller)	ja (TTEthernet Switch)	optional
Marktanteil [%]	n/a	n/a	0 außerhalb Mitsubishi
IE-System	Profinet	EtherNet/IP	SERCOS III
Zulieferzeit [ms]	1	0,13	0,0398
Klasse/Kategorie	3/C	3/C	3/C
SPoF?	ja	nein	ja
Skalierbarkeit	60	90	9
Selbstkonfiguration	nein	nein	ja
Spezielle HW?	ja (Profinet Switch)	ja (IEEE 1588 Switch)	optional
Marktanteil [%]	14,5	13,9	2,1
IE-System	DRTP	DARIEP	Eig. System
Zulieferzeit [ms]	5-10	0,1-0,3	?
Klasse/Kategorie	3/B	3/C	?
SPoF?	ja	ja	?
Skalierbarkeit	n/s	n/s	?
Selbstkonfiguration	ja	nein	?
Spezielle HW?	nein	ja (FPGA sync. Slaves)	?
Marktanteil [%]	0 (Entwicklung)	0 (Entwicklung)	?

Tabelle 4.1.: Vergleich etablierter IE-Systeme und deren geschätztem Marktanteil 2015 [boo13] mit aktuellen Entwicklungen.

4.1.6. Fazit

Es wird ersichtlich, dass keines der etablierten IE-Systeme alle Anforderungen bzgl. hoher Zuverlässigkeit, Skalierbarkeit, Flexibilität hinsichtlich Selbstkonfiguration und der Verwendung von kostengünstiger Standard-Ethernet-Hardware erfüllt. Keine der derzeitigen Lösungen erreicht eine isochrone Echtzeitdatenübertragung ohne spezielle Hardware und SPoF zur gleichen Zeit. Die Protokolle auf Applikationsebene, welche eingeführt oder durch die etablierten IE-Systeme angewendet werden, sind optimiert auf die Übertragung von Prozessdaten und erlauben keine zuverlässige Echtzeitübertragung mit steigenden Datenmengen. Dies liegt daran, dass die Systeme nicht nahtlos mit den steigenden Datenmengen mitskalieren bzw. sich dynamisch anpassen oder per se keine großen Datenmengen unterstützen. Nur TTEthernet bietet einen Grad an Freiheit in Bezug auf die Verwendung eines anderen Protokolls für die deterministische Übertragung jeglicher Daten, aber definiert selbst keins. Derzeitige Entwicklungen erreichen isochrone Echtzeit, benötigen jedoch alle spezielle Hardware oder besitzen einen SPoF, was deren Zuverlässigkeit und Skalierbarkeit einschränkt.

Folglich ist ein neuer IE-Ansatz zu entwickeln, um mit der steigenden Anzahl an Geräten und steigenden Datenmengen Schritt zu halten, welche in Echtzeit übertragen werden müssen. Im folgenden Abschnitt 4.2.1 und Unterkapitel 4.3 wird ein neues Verfahren namens HaRTKad vorgestellt, welches die zuvor beschriebenen Schwachstellen der bestehenden IE-Systeme und Entwicklungen vermeidet. Dabei wird versucht, alle Aspekte eines IE-Systems abzudecken, was die Synchronisation, den Medienzugriff und potentielle Applikationen umfasst. Ziel ist es, ein echtzeitfähiges System für hochskalierte Netzwerke zu entwickeln, welches ohne zentrale Kontrolle auskommt.

4.2. Synchronisation innerhalb von Kad

4.2.1. Einleitung

Bevor das HaRTKad-Verfahren vorgestellt wird, ist eine Grundvoraussetzung zu erfüllen. Diese beinhaltet, dass alle Knoten auf einer einheitlichen Zeitbasis arbeiten, da sonst ein zeitschlitzbasiertes (TDMA) Verfahren mittels Kad nicht realisierbar ist.

Die vorgestellte Synchronisation soll direkt auf Applikationsebene realisiert werden und Teil von Kad sein, um eine konsistente Lösung vorstellen zu können. Zusätzlich soll ein Grad an Parallelität in der Kommunikation der Teilnehmer erlaubt werden, um die Zeit für die Synchronisation von großen Netzwerken anhand von helfenden Knoten zu verringern. Der daraus entstehende potentiell größere Zeitfehler auf Grund der parallelen Kommunikation muss dabei berücksichtigt werden. Es ist daher ein sinnvoller Kompromiss zu finden.

Zusätzlich wird aufgezeigt, dass der individuelle Clock Drift der Knoten eine fortwährende Re-Synchronisation des Systems erfordert. Daraus resultiert, dass die Dauer einer Synchronisation unter Berücksichtigung des Zeitfehlers durch die Parallelität möglichst klein gehalten werden sollte.

Folgende Aspekte werden behandelt:

- Ein neues Konzept zur Synchronisation des P2P-Netzwerks Kad.
- Ein neuer Synchronisationsalgorithmus.
- Theoretische Performanceanalyse des Konzepts.
- Praktische Performanceanalyse des Konzepts.
- Vergleich der theoretischen und praktischen Ergebnisse.

Die Hauptbeiträge dieses Unterkapitels sind in [SDAT13b] und [SADT14] publiziert. In Abschnitt 4.2.2 wird ein Vergleich des vorgestellten Ansatzes mit bestehenden Arbeiten vorgenommen. Abschnitt 4.2.3 beschreibt alle Schritte, um eine Synchronisation in Kad-basierten Netzwerken durchzuführen. Ein Algorithmus mit dem Namen *KaDisSy*, welcher die Synchronisation durchführt, wird erläutert. Anschließend wird in Abschnitt 4.2.4 die mögliche Performance der entwickelten Lösung mathematisch analysiert. Praktische Ergebnisse des Konzepts anhand eines Prototyps werden in 4.2.5 vorgestellt. Dies

umfasst auch den Vergleich zwischen theoretischen und praktischen Ergebnissen. Das Unterkapitel endet mit einem Fazit in 4.2.6.

4.2.2. Stand der Technik

Dieser Abschnitt behandelt bereits existierende Lösungen zur Synchronisation von Netzwerken. Das Network Time Protocol (NTP) und das Precision Time Protocol (PTP) sind hierbei die am weitesten verbreiteten Protokolle, um Geräte in einem Netzwerk zu synchronisieren. Dabei erreichen sie eine Genauigkeit von einigen Millisekunden (NTP) [WDHP+06] und 100 Mikrosekunden für eine einzelne Verbindung zwischen zwei Teilnehmern bei PTP [See07]. Die Genauigkeit ist die maximale Abweichung, die von der Referenzzeit erlaubt ist. Des Weiteren kann PTP eine bessere Performance erreichen, wenn spezielle Hardware genutzt wird. Diese generiert dabei hochgenaue Zeitstempel, womit die Zeitauflösung erhöht und der Zeitfehler bei der Synchronisation verringert wird.

In dieser Arbeit wird dagegen ein softwarebasierter Ansatz verwendet, der keine spezielle Hardware benötigt. Verglichen mit den hierarchischen Strukturen von NTP und PTP [WDHP+06, Mil92, LE02] basiert das verwendete Kad-Netzwerk auf einem dezentralen Netzwerk.

Ein alternativer Synchronisationsansatz für P2P-Netzwerke, namens *PariSync*, wird in [BBMP09] präsentiert. Er ist in Java implementiert und erreicht eine Genauigkeit von mehreren hundert Millisekunden über das Internet. Parisync ist für den Einsatz von großen Netzwerken mit hohem Churn vorgesehen. Der Informationsaustausch und der Medienzugriff werden allerdings nicht kontrolliert und sind für den Einsatz in Automatsierungsumgebungen mit deterministischen Anforderungen folglich nicht geeignet. Der resultierende Fehler durch die unkontrollierte parallele Kommunikation der Netzwerkteilnehmer und durch das Puffern der Pakete in den Switches wurde in dieser Arbeit nicht untersucht. Außerdem benötigt das System mehrere Sekunden, um ein stabiles Netzwerk aufzubauen.

In [LXN04] wird ein P2P-System vorgestellt, welches die Routingpfade und Nachbarschaftsinformationen hinsichtlich der Verkehrskosten optimiert. Verkehrskosten sind die verwendete Bandbreite, welche durch das entstehende Datenaufkommen genutzt wird. Im Ergebnis konnte die verwendete Bandbreite deutlich gesenkt werden.

Das Hauptaugenmerk wird hierbei auf große Netzwerke mit hohem Churn gerichtet. NTP wird genutzt, um die einzelnen Knoten miteinander zu synchronisieren. Allerdings werden die mögliche Zeitgenauigkeit und der während der Synchronisation entstehende Zeitfehler nicht erörtert. Zusätzlich ist der Paketaustausch nicht gesteuert, was zu weiteren Zeitfehlern führt.

In [BPQS08] wird ein System vorgestellt, welches auf einem „Gossip"-Ansatz basiert. Ein „Gossip"-basierter Ansatz verbreitet Informationen durch „Tratschen" im Netzwerk. Jeder Teilnehmer gibt die Information selbst an seine Nachbarn und Kontakte weiter. Der Fokus liegt auf der Beurteilung des Einflusses von korrupten Prozessen auf die Effektivität der Zeitsynchronisation. Dabei gibt es keinen formalen Beweis der Korrektheit der Konvergenz der Zeitsynchronisation.

[MJB04] präsentiert ein System, welches Aggregation verwendet, um einen Durchschnittswert, Produkt oder Extremwert in einem verteilten P2P-Netzwerk zu berechnen. Die Generierung basiert auf einem epidemischen Ansatz. Um diese globalen Werte im System zu erzeugen, werden Echtzeitintervalle, genannt *cycles*, benötigt. Um die cycles zu realisieren, ist eine Synchronisation der Knoten erforderlich. Der negative Effekt der parallelen Kommunikation wird nicht berücksichtigt und auf den Clock Drift sowie die Paketverzögerungen wird nur informell eingegangen.

Einige Parameter, wie z.B. die Anzahl der Informationsweiterleitung einer Instanz an andere, müssen bestimmt werden, um eine hohe Wahrscheinlichkeit für einen erfolgreichen Informationsaustausch zu garantieren. Daher stellt ein „Gossip"-basierter oder epidemischer Ansatz wie in [BPQS08] und [MJB04] immer auch ein Kompromiss zwischen Skalierbarkeit und Zuverlässigkeit dar [EGKM04]. Da der „Gossip"-basierte Ansatz auf Wahrscheinlichkeit basiert und nicht für ein System mit harter Echtzeit geeignet ist, wird in dieser Arbeit darauf verzichtet.

Das Ziel dieses Beitrages ist es, eine bessere oder vergleichbare Zeitauflösung und Zeitfehler wie NTP oder softwarebasiertes PTP zu erreichen. Es wird ein effektiver Kompromiss zwischen exklusiven Medienzugriffen und Zeitfehlern bei gleichzeitiger Beschleunigung des Synchronisationsprozesses durch eine kontrollierte und parallele Kommunikation dargestellt. Das Besondere ist, dass der Ansatz keine zentrale Instanz aufweist, aber dass trotzdem eine hohe Performance bei der Synchronisation von mehreren tausend Knoten bei einem niedrigen Zeitfehler ermöglicht wird.

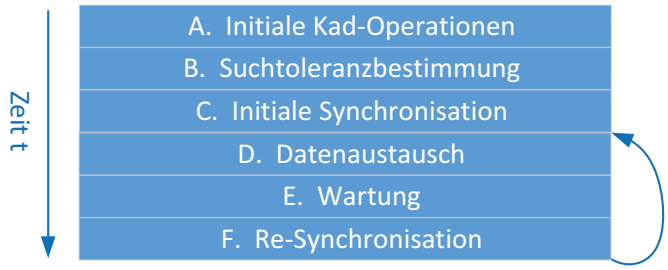

Abbildung 4.3.: Darstellung der Synchronisationsphasen.

4.2.3. Grundlagen und Designkonzept

Es wird ein deterministischer Ansatz zur Zeitsynchronisation von Echtzeitsystemen mittels eines Kad-Netzwerkes präsentiert. Das Konzept besteht aus sechs Phasen, welche in Abbildung 4.3 dargestellt sind. Das auf dem Kad-Netzwerk basierende System führt die Phasen im Betrieb aus, um seine Teilnehmer auf eine einheitliche Zeitbasis zu setzen. Kad wird gewählt, da es keinen SPoF besitzt und es voll dezentralisiert und strukturiert ist. Zusätzlich bietet es die beste Lookup-Performance unter den DHT-basierten P2P-Lösungen mit $O(log_{2^b}(N))$ [SW05]. Bei den Betrachtungen wird angenommen, dass die Realisierung des vorgestellten Ansatzes in einem dedizierten Netzwerk läuft, sodass es keinen konkurrierenden Datenverkehr durch andere Komponenten gibt. Im Folgenden werden die Phasen näher erläutert.

4.2.3.1. Initiale Kad-Operationen

In der ersten Phase führt das Kad-Netzwerk initiale Operationen aus. Dies umfasst das Bootstrapping neuer Knoten und die Wartung des Netzwerkes, wobei die Routingtabellen der Knoten mit Kontakten gefüllt werden. Zusätzlich wird die MAC-Adresse zur IP-Adresse der Knoten in der Routingtabelle gespeichert, um einen zusätzlichen Overhead durch das Address Resolution Protocol (ARP) in IPv4- oder das Neighbor Discovery Protocol (NDP) in IPv6-Netzwerken zu vermeiden. Dies ist notwendig, da der im Betriebssystem vorhandene ARP-Cache in der Größe limitiert ist und die Einträge nur eine gewisse Zeit gespeichert werden. Sollte kein gültiger Eintrag im ARP-Cache bzgl. einer IP-Adresse vorhanden sein, wird ein ARP-Broadcast im gesamten Netzwerk notwendig, was zu hohen Performanzeinbußen und erhöhten Zeitfehlern führen kann.

4.2.3.2. Suchtoleranzbestimmung

Um einen erfolgreichen Lookup-Prozess zu garantieren, wurde die dynamische Suchtoleranz (DST) entwickelt [DSA+15b]. Der DST-Algorithmus modifiziert die Suchtoleranz derart, dass mindestens ein Knoten für jeden beliebigen Hashwert zuständig ist. Dabei kann die DST die Performanz des Netzwerkes steigern, weil es zu weniger Timeouts durch nicht beantwortete Lookup-Prozesse kommt. Der DST-Algorithmus kann von jedem Knoten im Netzwerk gestartet werden, wenn er z.B. einen definierten Trigger erhält. Der Knoten, der den Trigger erhält, wird hierbei als *First Triggered-Knoten* (*FT*-Knoten) bezeichnet.

4.2.3.3. Initiale Synchronisation

Nachdem der *FT*-Knoten den DST-Algorithmus ausgeführt hat, wird der initiale Synchronisationsprozess ausgeführt. Der *FT*-Knoten sendet zuerst ein Broadcast in das Netzwerk und dann muss so lange gewartet werden, bis die Broadcast-Nachricht den kritischen Pfad des Netzwerkes durchlaufen hat, damit jeder Knoten die Nachricht erhält. Ab diesem Moment schweigen alle Knoten und der *FT*-Knoten hat den exklusiven Zugriff auf das Ethernet-Medium.

Kad führt eine iterative Lookup-Strategie durch und in der Originalimplementierung ist ein aktiver Knoten in der Lage, α parallele Lookup-Requests durchzuführen [SR06]. Um einen exklusiven Ethernet-Zugriff zu gewährleisten, sollte zu einem Zeitpunkt nur ein Lookup-Request erlaubt sein. Daher wird α auf 1 gesetzt. Während des Synchronisationsprozesses werden die Wartungsoperationen angehalten und die Routingtabellen werden nur durch die Lookup-Prozesse modifiziert. Dies stellt allerdings kein Problem dar, weil in der Originalimplementierung die Wartungsoperationen nur ca. alle 10 bis 20 Sekunden durchgeführt werden. Die Synchronisation innerhalb des Kad-Netzwerkes wird durch den Kademlia Discovery und Synchronisations (KaDisSy)-Algorithmus, welcher eine erweiterte Version des Kademlia Discovery (KaDis)-Algorithmus darstellt [SDA+11a], durchgeführt. KaDis wurde hierbei um die Funktionalität der Synchronisation erweitert. Der KaDis- und somit auch der KaDisSy-Algorithmus nutzen die Standardsuche vom Kad-Netzwerk, um mit den gefundenen Knoten zu interagieren. Es wird dabei angenommen, dass die Geräte im Kad-Netzwerk bestimmte Benennungsvorschriften einhalten. In

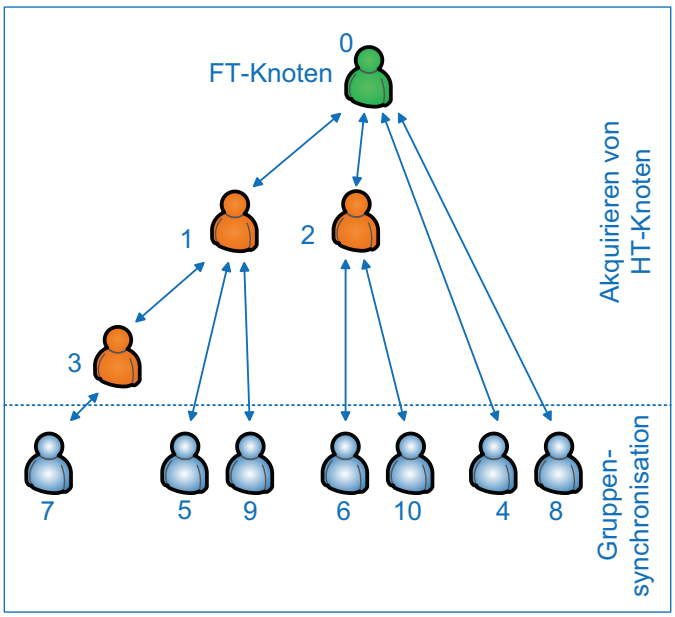

Abbildung 4.4.: Darstellung des mittels KaDisSy erstellten Synchronisationsbaumes.

diesem Fall ist es so, dass jeder Hashwert eines Knotens aus der Verbindung eines definierten Strings und einer Zählvariable i als Parameter für den MD5-Algorithmus erstellt wird. Dies ermöglicht eine effektive Suche aller Knoten im Netzwerk, unabhängig von der Hashwertverteilung.

Mittels des KaDisSy-Algorithmus ist es möglich, einen Synchronisationsbaum aufzubauen. Dieser ist schematisch in Abbildung 4.4 dargestellt. Es werden in dem gezeigten Beispiel in zwei Iterationen helfende Knoten (HT-Knoten) akquiriert, um in einem zweiten Schritt den Rest des Netzwerkes zu synchronisieren. Es ist ersichtlich, dass mit jeder Iteration die Anzahl der aktiv synchronisierenden Knoten verdoppelt wird. Die zwei Schritte des KaDisSy-Algorithmus werden im Folgenden näher erörtert:

HT-Knoten akquirieren: Im ersten Schritt werden die HT-Knoten akquiriert, was in der Abbildung 4.5 dargestellt ist. Dabei wird mit jedem Schritt die Anzahl der HT-Knoten verdoppelt, da jeder HT-Knoten wieder einen neuen HT-Knoten akquiriert. Zuerst (1) werden die Zähler auf ihren initialen Wert gesetzt.

Die Laufvariable s zählt dabei die Anzahl der Schritte, um HT-Knoten zu akquirieren. Der Parameter J definiert die Gesamtanzahl der Akquirierungen von HT-Knoten. Dabei

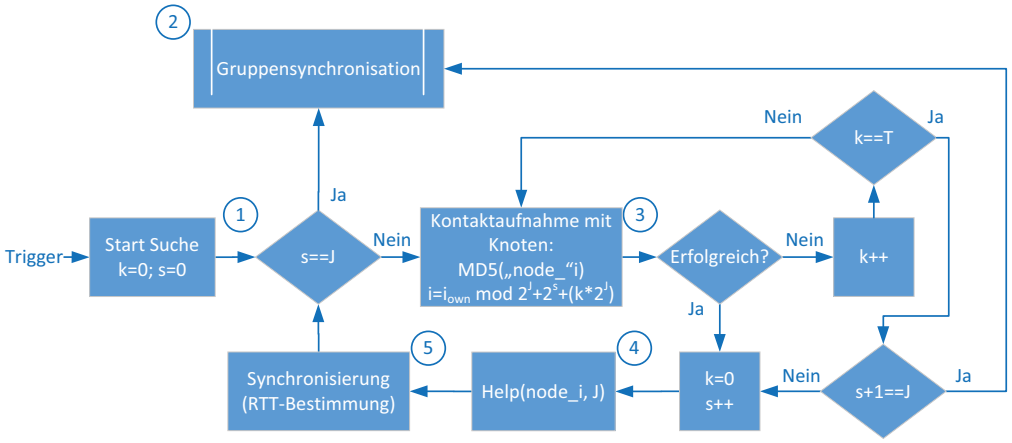

Abbildung 4.5.: KaDisSy-Algorithmus: Akquirieren von HT-Knoten.

ist es auch möglich, den Parameter J auf Null zu setzen. In diesem Fall wäre der FT-Knoten alleine für die Synchronisation zuständig, sodass keine HT-Knoten hinzugezogen werden. Somit kann direkt mit dem zweiten Schritt der Gruppensynchronisation (2), welche später erörtert wird, fortgesetzt werden. Allerdings wird im Normalfall bei einer größeren Anzahl von Knoten ein höherer Wert für J verwendet. Daher beträgt die Gesamtanzahl an Knoten, die für die Synchronisation zuständig sind, den Wert 2^J. Solange s kleiner ist als J, wird der FT-Knoten und ggf. HT-Knoten versuchen, weitere Knoten als HT-Knoten anzusprechen (3). Der Hashwert als Basis für den Lookup-Prozess wird dabei aus der nutzerspezifizierten Verbindung der Strings, z.B. „Node_“ und einem Integerwert i, berechnet. Die Berechnung des Hashwertes erfolgt mittels MD5-Algorithmus. Der aktuelle Wert für i wird aus i_{own}, welcher der eigene i-Wert des anfragenden Knotens ist, J, s und dem Zähler k bestimmt (siehe Formel 4.1).

$$i = i_{own} \bmod 2^J + 2^s + (k * 2^J) \tag{4.1}$$

Sollte der verantwortliche Knoten für den Wert i nicht durch den Lookup-Prozess gefunden werden, gilt dieser im Kad-Netzwerk als nicht existent und der Offsetzähler k wird erhöht. Ist k kleiner als der Schwellwert T, wird ein neuer Lookup-Prozess (3) ausgeführt. Andererseits überprüft der Knoten, ob $s + 1$ gleich J ist, was darauf hinweisen könnte, dass dies die letzte Iteration war, um neue HT-Knoten zu akquirieren. Sollte es sich um die letzte Iteration gehandelt haben, startet der Algorithmus die Gruppen-

synchronisation (2). Sollte es sich nicht um die letzte Iteration gehandelt haben, wird k auf Null zurückgesetzt und s um 1 erhöht, um eine neue Suche nach HT-Knoten für die nächste Gruppe fortzuführen. Im Gegenzug wird bei erfolgreich vorausgegangenem Lookup-Prozess mit dem Zählwert i, k auf Null zurückgesetzt und s um eins erhöht. Der neu angeforderte Knoten wird nun zum HT-Knoten und bekommt zusätzlich den Parameter J (4) mitgeteilt. Es folgt nun der Synchronisationsprozess (5), bei dem die sogenannte Round Trip Time (RTT), welche die Zeit für den Hin- und Rückweg eines Paketes von einem Knoten zu dem anderen darstellt, bestimmt wird. Aufgrund der angenommenen Kanalsymmetrie in einem Automatisierungs-Kad-Netzwerk wird davon ausgegangen, dass die Laufzeitverzögerung in Hin- und Rückrichtung zwischen zwei Knoten gleich ist.

Die ermittelte Verzögerung $RTT/2$ wird zu der aktuellen Zeit des FT- oder HT-Knotens addiert und durch ein UDP-Paket zu dem zu synchronisierenden Knoten gesendet. Der neue HT-Knoten muss nun seinen initialen Wert für die Laufvariable s für die nächste Iteration bestimmen. Dieser Wert muss nicht von dem im Vorfeld anfragenden Knoten mitgeliefert werden, sondern kann stattdessen mit Hilfe der Formel 4.2 bestimmt werden.

$$s = \lfloor log_2(i_{own} \bmod 2^J) + 1 \rfloor \tag{4.2}$$

Nun wird durch den Algorithmus versucht, weitere HT-Knoten zu finden, solange s kleiner als J (genug Iterationen) ist. Das zweite Abbruchkriterium ist, wenn k gleich T ist (zu viele Fehlschläge) und es die letzte Iteration ist, weitere HT-Knoten zu akquirieren. Nachdem der erste Teil des Algorithmus durchlaufen ist, wird mit der Gruppensynchronisation (2) fortgeführt.

Gruppensynchronisation: Der zweite Schritt der Synchronisation mittels KaDisSy ist die Synchronisation der eigenen Gruppe (siehe Abbildung 4.6). Der FT-Knoten und jeder HT-Knoten synchronisiert seine Knoten, welche sich potentiell in seiner Gruppe befinden.

Ähnlich dem ersten Teil, dem Akquirieren von HT-Knoten, kontaktieren der FT- und jeder HT-Knoten die Knoten in ihrer jeweiligen Gruppe abhängig von ihrem Wert i_{own} plus einem Offset, abhängig von m (1). Sollte ein Lookup-Prozess nicht erfolgreich sein, werden die Laufvariablen q und m erhöht, solange q kleiner als der Schwellwert Z ist. Sollte dies der Fall sein, gilt der Synchronisationsprozess als beendet (3), da angenommen

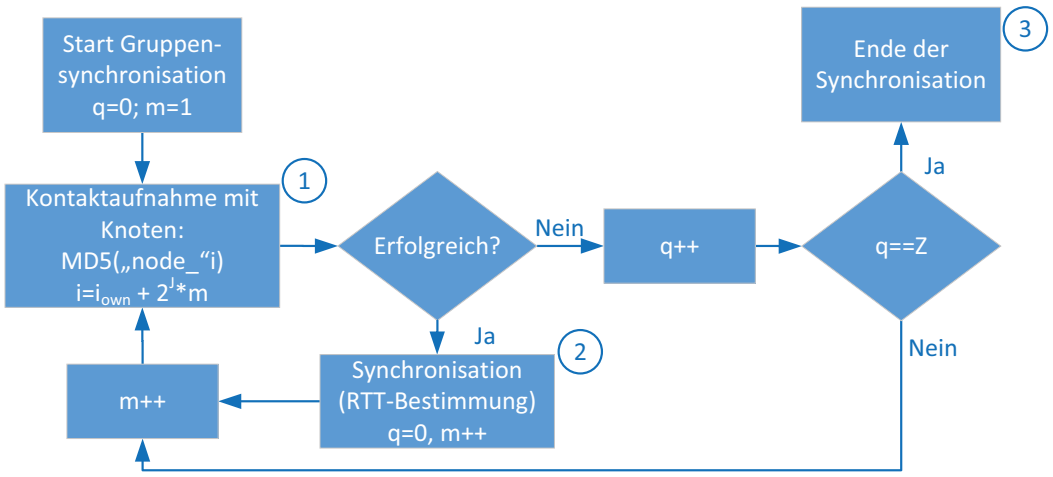

Abbildung 4.6.: KaDisSy-Algorithmus: Darstellung der Gruppensynchronisation.

wird, dass keine weiteren Knoten mehr in der Gruppe existieren. Ist der Lookup-Prozess erfolgreich, wird die Laufvariable q auf Null zurückgesetzt und die eigentliche Synchronisation zwischen dem FT- oder HT-Knoten und dem angefragten Knoten durchgeführt (2), wobei die Synchronisation mit dem ersten Schritt der Akquirierung der HT-Knoten identisch ist.

4.2.3.4. Datenaustausch

Während des Datenaustausches zwischen den Knoten ist es nicht erlaubt, dass neue Knoten dem neuen Kad-Automatisierungsnetzwerk beitreten, weil dies zu einem höheren Zeitfehler führen würde. Die Phase des Datenaustausches sollte dabei den größten zeitlichen Anteil haben, da in dieser Zeit die eigentliche Applikation im Kad-Automatisierungsnetzwerk aktiv sein kann. Die Kommunikation, die den Austausch von Daten zwischen den Knoten regelt, muss dabei durch ein separates Protokoll gesichert sein. Ein Beispiel für ein solches Protokoll ist in Abschnitt 5.2 gegeben.

4.2.3.5. Wartung

Wartungsoperationen und das Eintreten neuer Knoten in das Kad-Netzwerk sind in den vorherigen Phasen verboten bzw. unterbunden, da dies zur Erhöhung des Zeitfehlers füh-

ren könnte. Um allerdings die Routingtabellen der Knoten aktuell zu halten und neuen Knoten den Zugang zum Netzwerk zu ermöglichen, wurde die Wartungsphase eingeführt. Den Knoten ist es erlaubt, dem Netzwerk beizutreten, indem sie ihren Bootstrap-Knoten kontaktieren, nachdem sie ein Jamming-Signal erhalten haben. Dieses Jamming-Signal ist ein Broadcast, das vom *FT*-Knoten ausgesendet wird.

4.2.3.6. Re-Synchronisation

Aufgrund des spezifischen Clock Drifts der einzelnen Knoten wird eine Neusynchronisierung des Netzwerkes nötig, was als *Re-Synchronisation* bezeichnet wird. Die Re-Synchronisation muss periodisch ausgeführt werden. Definiert wird in diesem Zusammenhang die Re-Synchronisierungsperiode T_{ReSyn}, welche in Formel 4.3 beschrieben wird. T_{ReSyn} repräsentiert die Zeit zwischen zwei Synchronisierungen und sollte möglichst groß sein, um mehr Zeit für die anderen Phasen zu haben. Die maximale Zeitabweichung von einem Referenzknoten und damit auch Genauigkeit ist durch $T_{MaxError}$ gegeben. $T_{SynError}$ ist der zeitliche Fehler bei der Synchronisation zwischen zwei Knoten und wird später detaillierter erörtert. D_{Clk} ist der Clock Drift einzelner Teilnehmer im Netzwerk. T_{ReSyn} hängt auch direkt von der Synchronisationsdauer des gesamten Netzwerkes $T_{SynComp}$ ab, da der erste bereits synchronisierte Knoten durch seinen spezifischen Drift vom Referenztakt abweicht, bis der letzte Knoten synchronisiert wird.

$$T_{ReSyn} < \frac{T_{MaxError} - T_{SynError}}{2 * D_{Clk}} - T_{SynComp} \tag{4.3}$$

Eine detaillierte Analyse der Performance ist in Abschnitt 4.2.5.2 aufgeführt. Die Re-Synchronisierung ist technisch mit der initialen Synchronisation identisch. Nachdem die Re-Synchronisierung durchgeführt wurde, kann wieder mit dem Datenaustausch fortgefahren werden, woraufhin die zwei folgenden Phasen wiederholt werden.

4.2.4. Performance-Analyse

Da der KaDisSy-Algorithmus für den Einsatz in einer harten Echtzeitumgebung gedacht ist, ist es notwendig, das Worst Case-Verhalten zu bestimmen.

Synchronisationperformance: Zuerst wird die Zeit T_{Syn}, welche die Zeit zur Synchronisation zwischen zwei Knoten darstellt, durch die Formel 4.4 definiert. N ist die

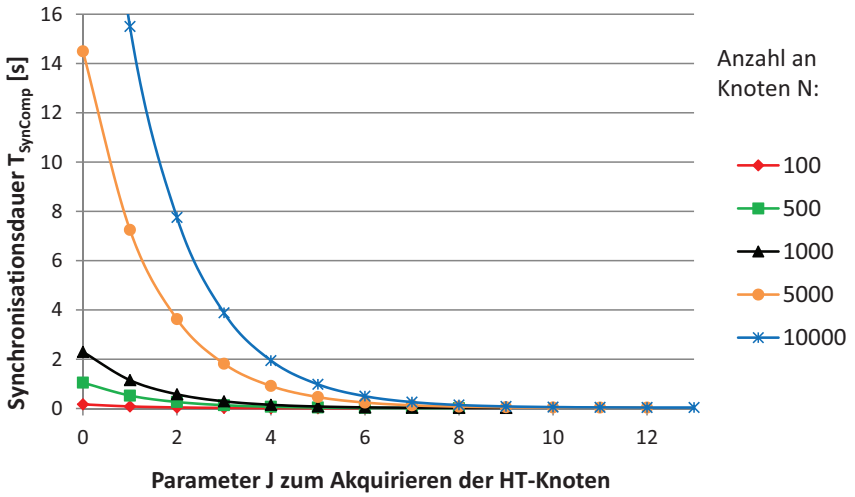

Abbildung 4.7.: Benötigte Zeit zum Synchronisieren des Kad-Netzwerkes.

Anzahl der Knoten im Kad-Netzwerk. b beschreibt die Anzahl an Bits, die bei jedem Lookup-Schritt übersprungen werden können und die für die Worst Case-Analyse auf den Wert „1“ gesetzt wird. $T_{SynComp}$ in Formel 4.5 ist die Zeit, um den gesamten Synchronisationsprozess für alle Knoten durchzuführen. Sie hängt direkt von N ((1) zum Synchronisieren der Knoten) und der Anzahl an Iterationen J ((2) zum Akquirieren der HT-Knoten) ab. Für RTT wurde ein Wert von 200 μs angenommen.

$$T_{Syn} = \lceil log_{2^b}(N) \rceil * RTT + 1.5RTT \tag{4.4}$$

$$T_{SynComp} = T_{Syn} * (\overbrace{J}^{(1)} + \frac{\overbrace{N}^{(2)}}{2^J} - 1) \tag{4.5}$$

Die benötigte Zeit $T_{SynComp}$ wurde theoretisch bestimmt und ist in Abbildung 4.7 dargestellt.

Bestimmung des Fehlers der Synchronisation: Der Fehler $T_{SynError}$ bei der Synchronisation zwischen zwei Knoten ist durch die Formel 4.6 gegeben. Sollte es keine HT-Knoten geben, hängt der Prozess der Synchronisation linear von der Anzahl der Knoten N im Netzwerk ab, sodass die Performance der Synchronisation für große Netzwerke nicht adäquat skaliert. Wenn allerdings $T_{SynComp}$ auf Grund von Applikationsanforderungen

niedrig sein muss, kann dies durch das Hinzuziehen von HT-Knoten erreicht werden. Es muss jedoch beachtet werden, dass sich durch die Verwendung von zusätzlichen HT-Knoten der generelle Zeitfehler bei der Synchronisation zwischen zwei Knoten im Worst Case auf Grund des parallelen Datenverkehrs im Netzwerk erhöht. Durch die parallele Kommunikation kann es passieren, dass andere Pakete durch Pufferung in Switches die Synchronisation verfälschen. Dies stellt den ersten Fehleranteil bei der Synchronisation dar (durch (3) in der Formel 4.6 gekennzeichnet). In dem präsentierten System sind die größten Pakete Kad-Pakete und werden daher für eine Worst Case-Analyse verwendet. T_{Pkt} repräsentiert die Zeit, die benötigt wird, damit ein Kad-Paket durch einen Switch weitergeleitet wird. Der zweite Anteil am Fehler entsteht durch die Abweichung bei der Bestimmung des RTT-Wertes, welcher als ΔRTT definiert ist. Dieser beinhaltet auch den Fehler durch die vorherige Synchronisation, die z.B. ein nun aktiver HT-Knoten durch seinen Vorgänger bei der Synchronisation erhält (durch (4) in der Formel 4.6 gekennzeichnet). Die Abweichung beim Bestimmen des RTT-Wertes ΔRTT wird mit 30 μs angenommen, was als Worst Case-Messwert bei praktischen Versuchen ermittelt wurde. Hierzu wurde die maximale Abweichung beim Ping bestimmt, der zwischen zwei Rechnern mit einem Linux Betriebssystem und dem Programm Wireshark [Wir14] gemessen wurde. Die Rechner waren über einen Gigabit-Switch miteinander verbunden.

$$T_{SynError} = \overbrace{((2^J - 1) * T_{Pkt})}^{(3)} + \overbrace{\Delta RTT * (J + 1)}^{(4)} \tag{4.6}$$

Eine Priorisierung in den Switches würde an dieser Stelle keinen Vorteil bringen, da die Kommunikationsprozesse, welche alle der Synchronisation dienen, die gleiche Priorität haben sollten. Die Zeit T_{ReSyn} ist in Formel 4.3 definiert und sollte möglichst groß gewählt werden, um genügend Zeit für die laufende Applikation zu gewährleisten. Einige Parameter müssen im Vorfeld definiert werden, um gültige Werte abzuleiten. Der maximal erlaubte Zeitfehler im Netzwerk für alle Instanzen ist durch $T_{MaxError}$ vorgegeben. Für die folgenden Betrachtungen wird der Wert auf 1 ms gesetzt, was folglich der zu erreichenden Genauigkeit jedes Teilnehmers entspricht. Es wird angenommen, dass das betrachtete Netzwerk aus Switches und Teilnehmern besteht und es somit keine Vorgaben für die Strukturen gibt. Ein Kad-Response-Paket stellt mit einer Größe von 80 Byte das größtmögliche Paket mit einer festen Anzahl an Kontakten dar. Dieses benötigt 640 ns, um einen Gigabit-Switch zu durchlaufen. Der Wert 640 ns ergibt sich als Zeit,

die 80 Byte zum Passieren einer 1-GBit/s-Schnittstelle benötigt. Jedes Paket muss warten, bis ein zuvor eingetroffenes Paket im Switch verarbeitet wird. Daher wird T_{Pkt} für die Worst Case-Betrachtung auf 640 ns gesetzt.

Abschließend ist es notwendig, aus den gegebenen Parametern die Zeitperiode T_{ReSyn} zu bestimmen. Übersichtshalber sind alle relevanten Zeiten in der Tabelle 4.2 aufgeführt.

Parameter	Beschreibung
T_{Syn}	Zeit zur Synchronisierung zwischen zwei Knoten
$T_{SynComp}$	Zeit, um alle Knoten zu synchronisieren
$T_{SynError}$	Synchronisationsfehler zwischen zwei Knoten
T_{Pkt}	Zeit für ein Paket, um einen Gigabit-Switch zu durchlaufen
$T_{MaxError}$	Maximal erlaubter Zeitfehler im Netzwerk
T_{ReSyn}	Zeitperiode zwischen zwei Synchronisationen

Tabelle 4.2.: Zeitparameter für die Performance-Evaluierung.

Zusätzlich muss hinzugefügt werden, dass der Takt, der von einem elektronischen Bauteil erzeugt wird, nicht ideal ist. Jeder Takt besitzt demnach einen spezifischen Drift. Als Evalutionszielplattform ist das ZedBoard vorgesehen [Avn]. Der interne Oszillator liefert einen stabilen Takt mit einer Drift D_{Clk} von $\pm$ 50 ppm [Avn].

In Abbildung 4.8 ist die Re-Synchronisierungsperiode T_{ReSyn} in Abhängigkeit von der Anzahl der Knoten N und von den HT-Knoten dargestellt. Dabei wurden alle zuvor bestimmten Parameter berücksichtigt. Es wird deutlich, dass es von Vorteil ist, mehr HT-Knoten zu verwenden, um eine bessere Synchronisationsperformance zu erreichen und damit eine größere Anzahl an Knoten zu ermöglichen. So ist es möglich, auch größere Netzwerke mit mehr als 10.000 Knoten in einer adäquaten Zeit zu synchronisieren.

Sollte T_{ReSyn} Null sein, da z.B. J zu klein gewählt wurde, bedeutet dies, dass nach der Synchronisation bereits die neue Synchronisation starten muss. Dies führt de facto zu einem permanenten Synchronisieren und es bleibt keine Zeit für eine Applikation, das Medium zu nutzen. Sollte hingegen J zu groß gewählt sein, wird der Fehler durch parallele Kommunikation zu groß, sodass es ebenfalls öfter bzw. permanent synchronisiert werden muss. Zu sehen ist, dass in großen Kad-Automatisierungsnetzwerken ein FT-Knoten nicht ausreichend ist, da T_{ReSyn} größer sein muss als $T_{SynComp}$ und folglich

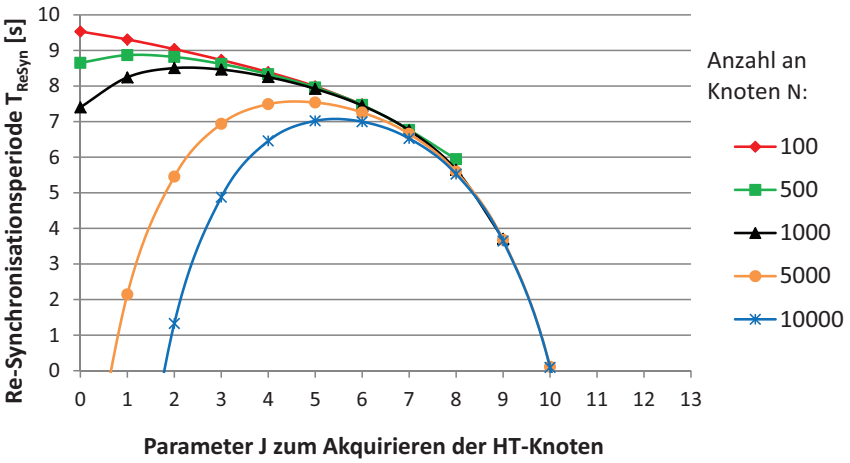

Abbildung 4.8.: Minimale Synchronisationperiode bei einer Genauigkeit bzw. bei einem max. erlaubten Fehler von 1 ms.

zusätzliche *HT*-Knoten erforderlich sind. Das Optimum, das hierbei gewählt werden kann, ist, dass J so gewählt wird, dass möglichst selten synchronisiert werden muss und T_{ReSyn} möglichst groß wird. Dies berücksichtigt aber nicht direkt die Zeit $T_{SynComp}$. Es muss daher entsprechend der gegebenen Anforderungen überprüft werden, ob $T_{SynComp}$ diesen genügt und ggf. J erhöht werden muss, um T_{ReSyn} zu verringern. So kann es passieren, dass man zwar selten eine Re-Synchronisation durchführt, die Synchronisation an sich aber sehr lange dauert.

Im Folgenden wird das Optimum derart theoretisch bestimmt, dass T_{ReSyn} möglichst groß ist und möglichst selten synchronisiert werden muss. Der korrespondierende Optimumwert für J ist J_{Opt} und wurde analytisch mittels des Programms *Mathematica* ermittelt [Wol14]. J_{Opt} hängt von der Anzahl der Knoten N im Netzwerk, T_{Syn} (Zeit, um einen beliebigen Knoten durch einen anderen beliebigen Knoten zu synchronisieren) und T_{Pkt} ab.

Zusätzlich ist es wichtig, das Datenaufkommen durch den präsentierten Ansatz zu bestimmen. Um das Datenvolumen bestimmen zu können, müssen die Pakete, die ausgetauscht werden, bekannt sein. Dies umfasst das Kad-Response-Paket mit 80 Byte, das bereits im Vorfeld für T_{Pkt} genutzt wird. Zusätzlich sind Kademlia-Requests (77 Byte), zwei Pakete für das Pinging (je 60 Byte) und zwei Pakete (je 60 Byte) zum Austausch des

neuen 4-Byte-Zeitwertes mit entsprechender Bestätigung (siehe Schritt (5) in Abbildung 4.5) in den Synchronisationsprozess involviert.

In der Tabelle 4.3 sind für die numerisch ermittelten J_{Opt}-Werte charakteristische Werte für verschiedene Netzwerkgrößen aufgeführt. Alternativ werden die Werte für J angegeben, um eine Synchronisationsdauer von 100 ms zu unterschreiten (als J_{100ms} deklariert). Für alle in der Tabelle aufgeführten Ergebnisse gilt eine Genauigkeit von 1 ms, was dem erlaubten Fehler $T_{MaxError}$ entspricht.

KNOTEN	100	500	1000	5000	10000
Vorgabe	Max(T_{ReSyn})				
J_{Opt}	0	1	2	5	5
$T_{SynComp}$ [ms]	168,30	525,00	577,30	464,72	981,15
T_{ReSyn} [s]	9,53	8,86	8,50	7,54	7,02
Vorgabe	$T_{SynComp} < 100$ ms				
J_{100ms}	1	4	5	8	9
$T_{SynComp}$ [ms]	85,00	71,93	81,08	76,94	85,35
T_{ReSyn} [s]	9,30	8,33	7,92	5,59	3,64
Datenverkehr [KB]	127	822	1.802	10.843	23.259

Tabelle 4.3.: Performanceauswertung für J_{Opt} und $T_{SynComp} < 100$ ms bei einer Genauigkeit von 1 ms.

Wie ersichtlich ist, ist die Performance mit der zentralisierten hierarchischen Lösung NTP, welche eine Genauigkeit von mehreren Millisekunden ermöglicht, konkurrenzfähig, da eine Synchronisation für die Genauigkeit von 1 ms effizient erreichbar ist. PTP ermöglicht eine Genauigkeit von 100 Mikrosekunden als Softwarerealisierung für eine Verbindung. In der präsentierten Lösung kann für 10.000 Knoten bereits eine Genauigkeit von maximal 300 Mikrosekunden erreicht werden, bevor eine dauerhafte Synchronisation ($T_{ReSyn} = 0$) durchgeführt wird, was eine vergleichbare Leistungsfähigkeit darstellt.

In dem beschriebenen Szenario ist die Genauigkeit $T_{MaxError}$ auf 1 ms gesetzt und es ist möglich, dies in adäquater Synchronisationsdauer zu erreichen. Außerdem basiert der präsentierte Ansatz nicht auf einem „Gossip“-Ansatz wie die Lösungen [BPQS08] und [MJB04], sondern auf einem deterministischen Ansatz. Daher ist der Ansatz be-

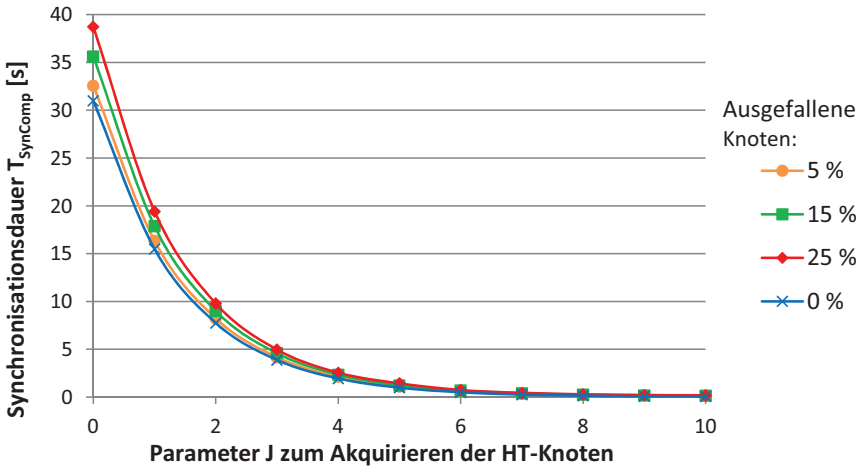

Abbildung 4.9.: Synchronisationsdauer unter Berücksichtigung von ausgefallenen Knoten.

sonders für harte Echtzeitszenarien geeignet. Die Ergebnisse wurden durch funktionale Simulationen für die Werte $T_{SynComp}$ und T_{ReSyn}, welche im Vorfeld berechnet wurden, bestätigt. Auch wird deutlich, dass nur T_{ReSyn} als betrachtetes Optimum u.U. eine hohe Synchronisationsdauer zur Folge haben kann. Aber durch Vorgabe von Anforderungen kann J speziell für eine Anwendung gesetzt werden, wie in Tabelle 4.3 mit der Vorgabe $T_{SynComp} < 100$ ms ersichtlich ist.

Der Einfluss von ausgefallenen Knoten: Wenn man von Kad-Netzwerken bzw. P2P-Netzwerken spricht, ist oftmals die Rede von Churn, was die Häufigkeit des Ein- und Austretens von Knoten darstellt. Für Automatisierungsszenarien wird davon ausgegangen, dass der Churn relativ gering ist. Trotzdem wird untersucht, wie sich beim Ausfall einer bestimmten Anzahl an Knoten die Synchronisationsperformance verändert. Dazu wurden mehrere Fehlerraten von 1 % bis 25 % eingeführt, wobei 25 % einen sehr pessimistischen Wert darstellt. Die Größe des Netzwerks wird für die folgende Betrachtung auf 10.000 Knoten festgesetzt. Sollte der FT- oder ein HT-Knoten einen Knoten nicht synchronisieren können, wird ein automatischer Timeout nach 6200 μs generiert, was die doppelte Zeit darstellt, um einen Knoten durch einen anderen im Kad-Netzwerk zu synchronisieren. Die Parameter T und Z des KaDisSy-Algorithmus sind auf den Wert 10 gesetzt. Unter diesen Umständen wurden alle verfügbaren Knoten erfolgreich synchronisiert. Beispielhafte Simulationsergebnisse für die Fehlerraten 0%, 5%, 15% und 25%

sind in Abbildung 4.9 dargestellt. Zusätzlich wurde die absolute Abweichung der Synchronisationsdauer $T_{SynComp}$ in Bezug auf ein ideales Netz ohne Ausfälle durch Abbildung 4.10 verdeutlicht. Ersichtlich ist, dass die Fehlerrate einen nennenswerten Einfluss auf die Synchronisationsdauer hat und daher berücksichtigt werden sollte, insofern die Netzwerkcharakteristik bekannt ist. Wird eine Fehlerrate von 25 %, 10.000 Knoten und $J = 0$ angenommen, wird $T_{SynComp}$ um 25 % erhöht. Sollte ein höherer Wert für J gewählt werden (z.B. $J = J_{Opt} = 5$), dann erhöht sich der Zeitbedarf im Vergleich um 45 %.

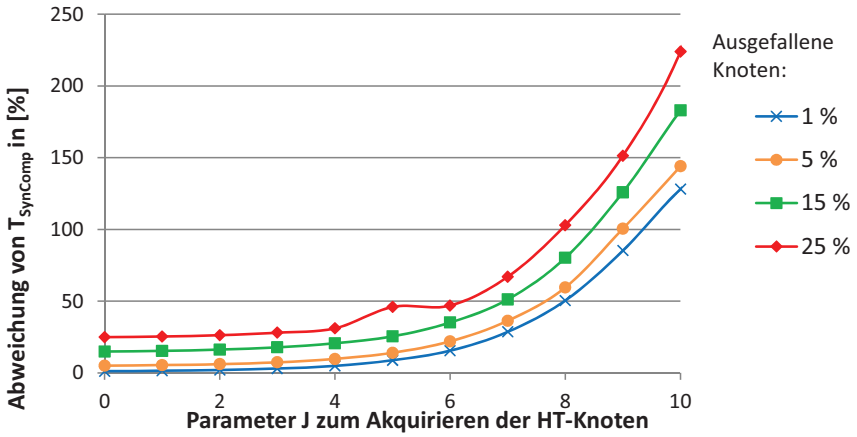

Abbildung 4.10.: Abweichung von $T_{SynComp}$ für verschiedene Ausfallraten.

4.2.4.1. Weitere Verbesserungen

Es werden nun noch zwei weitere Ansätze vorgeschlagen, um den Zeitfehler der Synchronisation zu senken und die Zuverlässigkeit des Systems zu erhöhen.

Das Gruppieren von Netzen: Besteht das P2P-Netzwerk aus mehreren Subnetzen, ist es möglich, entsprechende *HT*-Knoten direkt in den Subnetzen zu wählen, damit diese die Synchronisation in den Subnetzen übernehmen. Dies sollte die Genauigkeit der Synchronisation erhöhen, da Zeitfehler auf Grund physikalischer Nähe geringer ausfallen sollten, weil nur Pakete im Subnetzwerk ausgetauscht werden. Um die Adressierung der Knoten in jedem einzelnen Subnetzwerk zu ermöglichen, ist eine konsistente Namensgebung erforderlich. Allerdings sollten die Subnetze unabhängig von anderen Subnetzen

arbeiten, da es sonst passieren kann, dass kleine Subnetze lange auf große Subnetze warten müssen. Dies wäre der Fall, wenn Knoten vom kleinen Subnetz mit Knoten vom großen Subnetz über gemeinsame Applikationen miteinander interagieren müssen.

Integration eines Backup-Systems: Ein weiterer Fokus liegt auf der Zuverlässigkeit in Automatisierungsumgebungen.

In bisherigen Systemen müssen eine Anzahl an Parametern, Algorithmen oder nutzerspezifische Vorgabe ausgetauscht werden, um einen Backup-Knoten zu bestimmen. Dies führt zu weiterem Kommunikationsoverhead. Im Gegensatz dazu wird das Problem eines ausfallenden initialen *FT*- oder *HT*-Knotens in dieser Arbeit ohne zusätzlichen Overhead gelöst. Sollte der *FT*-Knoten, welcher jeder Knoten im Kad-Netzwerk sein kann, ausfallen, wird der Knoten mit der geringsten XOR-Distanz zum Hashwert Null gewählt. Dieser ist dann zuständig für die initiale Synchronisationsphase. Jeder Knoten ist in der Lage, eine Worst Case-Zeit zu berechnen, in der er hätte kontaktiert werden sollen. Wird diese Zeit überschritten, wird der Knoten aktiv und startet die (Re-)Synchronisation selbst.

4.2.5. Praktische Ergebnisse

In diesem Abschnitt wird ein Prototyp und ein Experimentsetup erörtert, welches genutzt wurde, um die zuvor ermittelten theoretischen Erkenntnisse zu überprüfen. Hierdurch wird der funktionale Nachweis des KaDiSy-Algorithmus geführt sowie dessen Performance anhand realer Knoten ermittelt.

4.2.5.1. Der Prototyp

Der entwickelte Prototyp arbeitet auf der OSI-Schicht 7, der Applikationsschicht, und ist daher flexibel auf anderen Plattformen einsetzbar. Bei der gewählten Plattform handelt es sich um das ZedBoard, welches einen ARM-Prozessor enthält, der mit 667 MHz betrieben wird [Avn]. Keine zusätzliche proprietäre Hardware wie FPGA-Ressourcen wurde für die KaDisSy-Funktionalität verwendet. Es wird ein Kern des ARM-Prozessors genutzt, womit auch das Profiling durch die Tools vereinfacht wird und alle Zeiten für die einzelnen Prozesse bestimmt werden können. Das verwendete Betriebssystem FreeRTOS

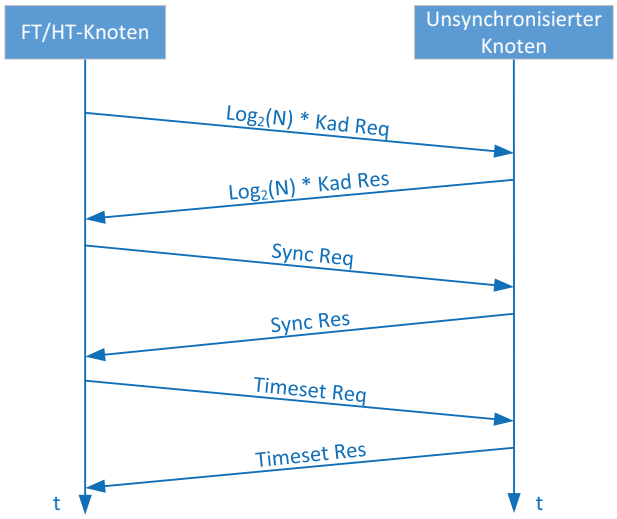

Abbildung 4.11.: Praktische Synchronisation zwischen zwei Knoten.

ist ein Echtzeitbetriebssystem, welches lwIP als TCP/IP-Stack nutzt [Rea] [Fre]. Die Implementierung von Kademlia, Kad genannt [BB06], wurde in dieser Arbeit für FreeRTOS portiert und existierende Threads wurden priorisiert. Eine ausführlichere Beschreibung zum verwendeten Prototypen ist in Abschnitt 4.3.3 zu finden. Während der Ausführung des KaDisSy-Algorithmus müssen sich die Knoten untereinander synchronisieren. Dies ist direkt innerhalb der Kad-Applikation realisiert (siehe Abbildung 4.11).

Zuerst müssen Knoten im Kad-Netzwerk gefunden werden. Dafür sind im Worst Case $log_2(N)$ Suchschritte notwendig, wobei N wieder die Anzahl aller Kad-Knoten im Netzwerk darstellt. Die Suche wird durch Kad-Request- und Kad-Response-Pakete durchgeführt. Ist der unsynchronisierte Knoten gefunden, initiiert der FT- oder HT-Knoten die Synchronisation mittels eines Sync-Request-Paketes. Zusätzlich nimmt der FT/HT-Knoten seinen ersten Zeitstempel, um später die Paketumlaufzeit (RTT) bestimmen zu können. Auf das Sync-Request-Paket folgt als Bestätigung ein Sync-Response-Paket, bei dessen Erhalt der FT/HT-Knoten den zweiten Zeitstempel nimmt. Nun kann der RTT-Wert bestimmt werden. Der FT/HT-Knoten sendet den neu zu setzenden Zeitwert, welcher der aktuellen Zeit des FT/HT-Knotens plus dem halben RTT-Wert entspricht, in einem Timeset-Request-Paket. Der vormals unsychronisierte Knoten speichert die Differenz zu seinem eigenen Zeitwert und ist nun synchronisiert. Zusätzlich bestätigt er seine

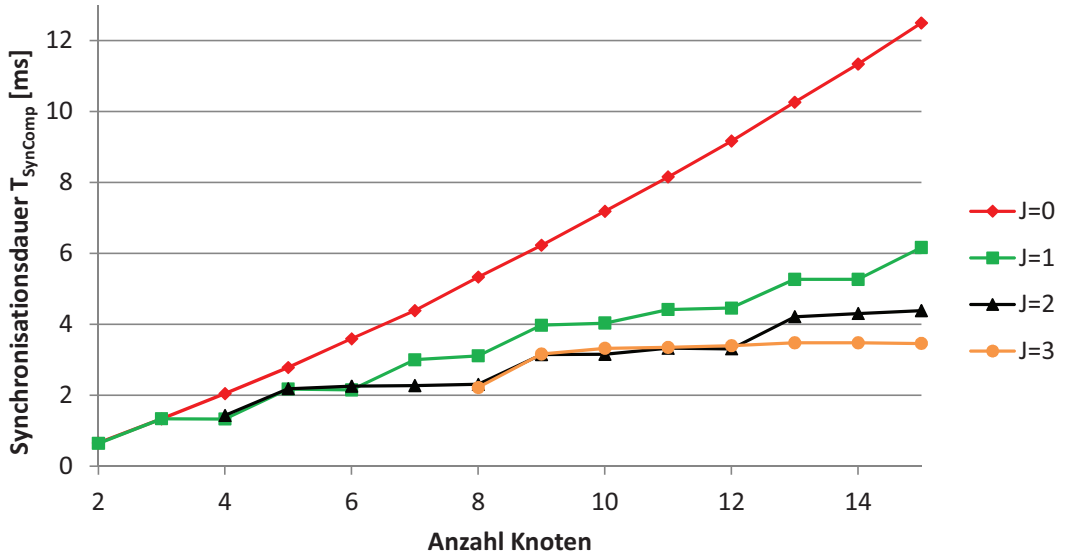

Abbildung 4.12.: Synchronisationsperformance für verschiedene J-Werte und Anzahl an Knoten.

Synchronisation mittels eines Timeset-Response-Paketes. Dieser Prozess findet bei jeder Synchronisation zwischen zwei Knoten statt, die den KaDisSy-Algorithmus ausführen.

4.2.5.2. Experimentelle Messungen

Die experimentellen Ergebnisse wurden mittels 15 ZedBoards, welche über ein 1-GBit/s-Ethernet-Netzwerk verbunden waren, erzeugt. Jedes ZedBoard führt dabei die modifizierte Kad-Software aus. Mit Hilfe eines Host-PC wird ein Triggersignal und der Wert für J an ein ZedBoard gesendet, woraufhin der Synchronisationsprozess angestoßen wird. Der Knoten, welcher das Triggersignal erhält, wird zum FT-Knoten und startet die Synchronisation des ersten Knotens. Es wird ein erster Zeitstempel $Stamp_{Start}$ genommen, um später die Gesamtsynchronisationsperformance bestimmen zu können. Die Auflösung des Zeitwertes beträgt 1 μs. Zuerst akquiriert der FT-Knoten, abhängig vom gesetzten J-Wert, optional weitere HT-Knoten.

Im Experimentaufbau senden alle synchronisierten Knoten ihr Timeset-Response-Paket an den FT-Knoten. Hierdurch ist der FT-Knoten in der Lage, zu bestimmen, wann alle anderen 14 Knoten synchronisiert wurden. Wenn alle Knoten ihre Antwort gesendet

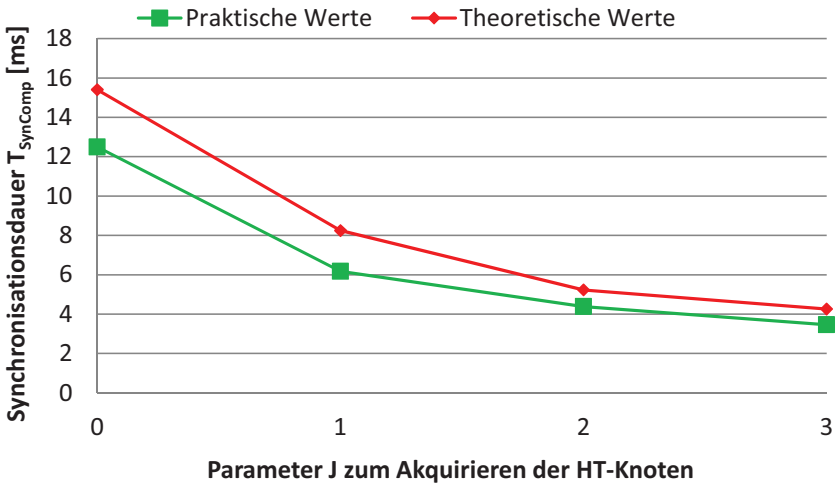

Abbildung 4.13.: Synchronisationsperformance für 15 Knoten bei unterschiedlichen J-Werten.

haben, wird der zweite Zeitstempel $Stamp_{Finish}$ genommen. Die Synchronisationsdauer $T_{SynComp}$ ist folglich die Differenz der beiden Zeitstempel $Stamp_{Start}$ und $Stamp_{Finish}$.

Synchronisationsperformance: Die Zeit $T_{SynComp}$ für eine unterschiedliche Anzahl an Knoten N ist in Abbildung 4.12 dargestellt. Wie ersichtlich, synchronisiert nur ein Knoten das Netzwerk, wenn $J = 0$ ist. Die benötigte Zeit zum Synchronisieren ist linear abhängig von der Anzahl der Knoten, welche zu synchronisieren sind, wohingegen eine Erhöhung des Parameters J eine signifikante Verringerung von $T_{SynComp}$ zur Folge hat. Mit dem höchst möglichen Wert für $J = 3$ ist $T_{SynComp}$ für insgesamt 15 Knoten geringer als 3,5 ms, was eine Verbesserung um 72,30 % gegenüber der Synchronisierung durch einen einzelnen FT-Knoten darstellt.

In Abbildung 4.13 wurde der Parameter J auf der x-Achse dargestellt und die Anzahl der Knoten N ist konstant 15. Es wird deutlich, dass $T_{SynComp}$ proportional zu $1/log_2(2^J)$ ist. Die gemessenen Werte bestätigen das Verhalten, welches zuvor theoretisch ermittelt wurde. Es ist außerdem ersichtlich, dass der Wert für $T_{SynComp}$ in die Sättigung geht und es daher nicht immer sinnvoll ist, den höchstmöglichen Wert für J zu wählen. Ein hoher Wert für J führt zusätzlich zu einer höheren Fehlerfortpflanzung, da bei der Akquirierung weiterer HT-Knoten in der nächsten Iteration durch andere HT-Knoten deren eigener Fehler weitergegeben wird. Alternativ könnte auch $T_{MaxError}$ auf einige wenige 100 μs

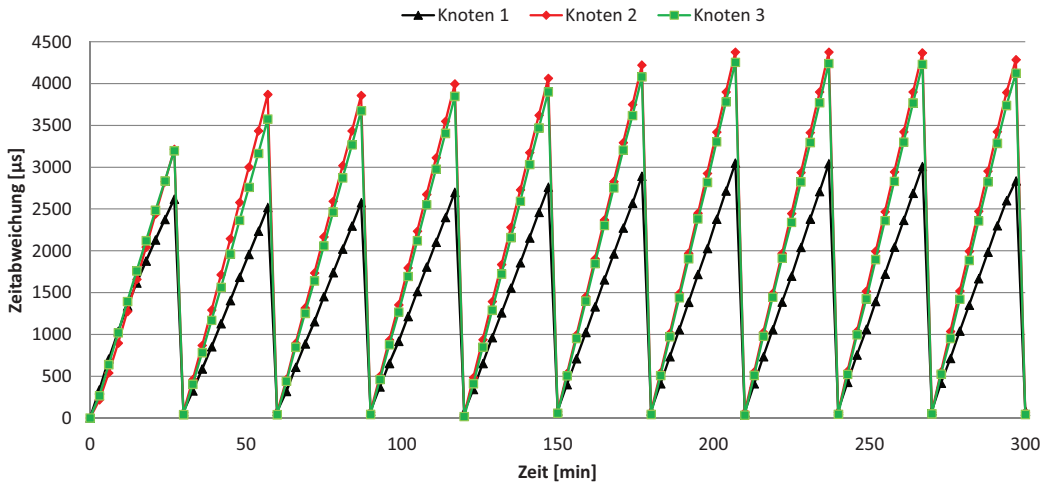

Abbildung 4.14.: Vier Knoten führen den KaDisSy-Algorithmus mit $J = 0$ aus.

reduziert werden. Die Einstellungen des KaDisSy-Algorithmus hängen somit stark von der Zielapplikation ab.

Bestimmung von ΔRTT: Die Re-Synchronisierung hängt auch vom Fehler ab, der bei der Synchronisierung zweier Knoten entsteht. Wie in Formel 4.6 deutlich ist, hängt der Fehler direkt von der Abweichung ΔRTT ab. Daher wurden im praxisnahen Einsatz die maximale Abweichung vom kleinsten und größtem RTT-Wert sowie der Einfluss von Switches auf ΔRTT ermittelt. Zuerst wurde der RTT-Wert unter Verwendung eines Gigabit-Switches gemessen. Im zweiten Szenario wurde ein zweiter Switch zwischen zwei Teilnehmern geschaltet. In beiden Fällen war der gemessene RTT-Wert kleiner als 152 μs. Die maximale Differenz zwischen dem geringsten und höchsten RTT-Wert betrug 14 μs, was als ΔRTT bezeichnet wird. Des Weiteren ließ sich der Einfluss weiterer Switches ermitteln. Ein zusätzlicher Switch in Reihe hatte im Versuchsaufbau nur einen geringen Einfluss, da er den ΔRTT-Wert nur um 1 μs verschlechterte.

Bestimmung des Clock Drifts: Der Hauptgrund für den Bedarf einer Re-Synchronisierung ist der spezifische Drift jedes Knotens durch seinen Oszillator. Daher ist es sinnvoll, den Clock Drift der einzelnen Knoten zu ermitteln und ihn mit dem vom Hersteller gegebenen Parameter zu vergleichen. Das ZedBoard als Zielplattform hat einen angegebenen Clock Drift von ± 50 ppm.

In der Abbildung 4.14 wurde die Funktionalität des KaDisSy-Algorithmus aus Über-

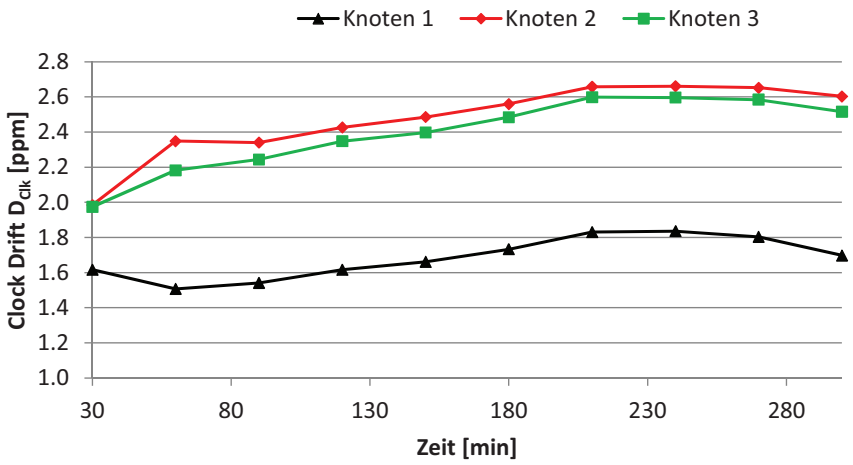

Abbildung 4.15.: Gemessener Clock Drift D_{Clk} von drei Knoten.

sichtsgründen exemplarisch für vier Knoten über fünf Stunden dargestellt. Knoten 0 ist der Referenzknoten und alle 5 Minuten senden die anderen drei Knoten ihren aktuellen Zeitwert an den Referenzknoten. Es ist erkennbar, dass jeder Knoten konstant vom Takt des Referenzknotens durch den spezifischen Drift abweicht. Die Drifts werden als absolute Differenz in der Abbildung dargestellt. Alle 30 Minuten werden die Knoten neu synchronisiert, was der sechsten Phase des Ansatzes entspricht (Re-Synchronisationsphase). Dies weist die funktionale richtige Arbeitsweise des Prototyps nach und erlaubt die beispielhafte Bestimmung des Clock Drifts der Knoten. Die große Re-Synchronisationsperiode erlaubt eine genaue Bestimmung von glaubwürdigen Werten der Clock Drifts. Verglichen mit dem für das ZedBoard spezifizierten Wert von ± 50 ppm, wurde ein deutlich besserer Wert von ± 3 ppm für den Clock Drift gemessen. Der gemessene Drift der drei Knoten über fünf Stunden, verglichen zum Referenzknoten 0, ist in Abbildung 4.15 dargestellt. In den Tests war der gemessene Drift stets unter ± 3 ppm.

Wenn man zusätzlich einen geringeren $T_{MaxError}$-Wert und damit Genauigkeit von 100 μs annimmt, ist man immer noch in der Lage, ein Netzwerk mit 10.000 Knoten synchron zu halten. Dafür müsste allerdings eine permanente Synchronisierung ($T_{ReSyn} = 0$) des Netzwerkes durchgeführt werden. Ein noch geringerer Wert führt zu nicht ausreichender Zeit zur Synchronisation des Netzwerkes. Mit diesen Merkmalen ist diese rein auf Software basierende Lösung eine Alternative zu NTP und PTP, welche eine Genauigkeit

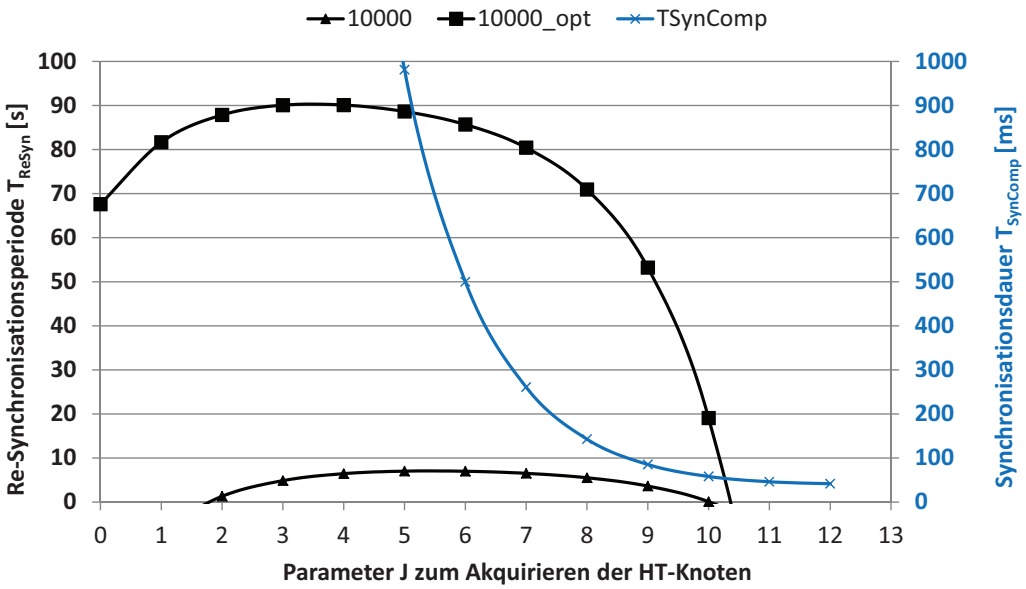

Abbildung 4.16.: Vergleich zwischen theoretischen und praktischen Ergebnissen.

von einigen Millisekunden (NTP) und einigen hundert Mikrosekunden (PTP) angeben. Zusätzlich wird keine zentrale Instanz zur Realisierung benötigt, was das System robust sowie flexibel macht und für den Einsatz in hochskalierten Netzwerken prädestiniert.

4.2.5.3. Vergleich mit theoretischen Werten

In diesem Abschnitt werden die Ergebnisse des Prototypenszenarios mit den theoretischen Werten aus der Simulation verglichen.

Wie in Formel 4.4 ersichtlich, haben ΔRTT und der Clock Drift einen immensen Einfluss auf die Re-Synchronisierungsperiode. Wenn die ermittelten praktischen Werte für ΔRTT und die Clock Drifts verwendet werden, wird in Abbildung 4.16 deren signifikanter Einfluss auf T_{ReSyn} deutlich. Der maximal erlaubte Zeitfehler ist durch $T_{MaxError}$ definiert. Der Wert wird für alle weiteren Betrachtungen auf 1 ms gesetzt. Die Kennlinie mit der Kennzeichnung „_opt“ berücksichtigt die neuen Werte für ΔRTT und den Clock Drift, während die andere Kennlinie aus den theoretischen Werten abgeleitet ist, welche in Abschnitt 4.2.4 vorgestellt wurden. Für 10.000 Knoten muss die Re-Synchronisation ca. alle 10 Sekunden durchgeführt werden. Mit den neu gemessenen Werten kann für

10.000 Knoten (dargestellt als 10,000_opt-Kennlinie) eine Re-Synchronisierung auf alle 90 Sekunden reduziert werden. Damit hat der Nutzer effektiv 800 % mehr Zeit für den Datenaustausch durch eine Applikation, verglichen mit den zuvor angenommenen Werten. Der höchste Wert für T_{ReSyn} kann als Optimum interpretiert werden, da hier die Zeit zwischen zwei Synchronisationen am größten ist.

Allerdings ist es auch nötig, $T_{SynComp}$ zu berücksichtigen, weil mit einem höheren Wert für J der $T_{SynComp}$-Wert exponentiell verringert werden kann. Damit muss ein applikationsabhängiger Kompromiss zwischen T_{ReSyn} und $T_{SynComp}$ definiert werden.

4.2.5.4. Weitere Verbesserungen

Einige Schritte könnten ausgeführt werden, um eine bessere Performance zu erreichen. Es ist möglich, den RTT-Wert bereits durch den Austausch des Kad-Request- und Kad-Response-Paketes zu ermitteln. Als Resultat könnten die beiden Synch-Request- und Synch-Response-Pakete eingespart werden. Für eine bessere Vergleichbarkeit wurde dies nicht auf dem Prototypen umgesetzt, ist aber denkbar.

Ein weiterer vielversprechender Optimierungsschritt wäre es, Informationen vom Underlay zu verwenden. Der als Missmatch bezeichnete Unterschied zwischen logischem Overlay (Kad) und dem physikalischen Underlay (z.B. Ethernet-Infrastruktur) müsste dafür verringert werden bzw. es müssten zusätzliche Informationen generiert werden.

Wenn Wissen über die Infrastruktur vorhanden wäre, könnten parallele nicht konkurrierende Pfade simultan genutzt werden, ohne einen negativen Einfluss durch das Puffern in den Switches zu erhalten. In Automatisierungsszenarien, bei denen in Bezug auf den Churn von Knoten in der Infrastruktur keine hohe Dynamik vorherrscht, existiert an dieser Stelle ein hohes Optimierungspotential. Abschließend ist eine aktive Kompensation des Drifts möglich, indem die Knoten diesen durch den synchronisierenden Knoten mitgeteilt bekommen könnten. Dies würde die Re-Synchronisationsperiode erheblich vergrößern, sodass mehr Zeit für die anderen Phasen wäre. In Bezug auf die Synchronisationsdauer wäre es sinnvoll, im Echtzeiteinsatz die Synchronisation in mehreren Schritten durchzuführen, um nicht einen zu großen Zeitblock zu beanspruchen. Somit wäre es möglich, die Synchronisation auch in Echtzeitsystemen mit Anforderungen an die Zykluszeit von 10 ms nicht zu verletzen. Inwiefern dies die Anzahl der potentiellen Knoten im Gesamtsystem beschränkt, gilt dabei noch zu untersuchen.

4.2.6. Fazit

Ein Ansatz zur Zeitsynchronisation des DHT-basierten P2P-Netzwerkes Kad wurde vorgestellt, um Operationen im Echtzeitumfeld zu ermöglichen. Eine Worst Case-Analyse zeigt, dass die Performance mit bestehenden hierarchischen Lösungen wie NTP und PTP vergleichbar ist, aber auf jeglichen SPoF in Form einer zentralen Instanz verzichtet. Zusätzlich wird ein Optimum für eine Anzahl an helfenden Knoten (HT-Knoten) definiert, um so möglichst wenige Re-Synchronisationen durchführen zu müssen. Hierbei zeigt sich, dass die Anzahl der *HT*-Knoten stark von den gegebenen Anforderungen abhängt und auch die Synchronisationszeit zu berücksichtigen ist. Die Idee der Gruppierung innerhalb eines Netzwerks kann potentiell zu besseren Ergebnissen führen. Die kann durch einen verringerten Zeitfehler innerhalb der Subnetze erreicht werden. Außerdem wird der Einfluss ausfallender Knoten untersucht, sowie ein Ansatz vorgestellt, damit umzugehen, ohne zusätzlichen Overhead zu erzeugen. Neben dem Konzept wurden reale Messungen mittels eines Aufbaus von 15 Prototypen präsentiert. Der Einfluss verschiedener Parameter wie *RTT*, Clock Drifts oder Anzahl der Switches wurde ebenfalls untersucht. Abschließend wurde ein Vergleich zwischen theoretischen und praktischen Werten vorgenommen, wobei die Ergebnisse des Prototyps überzeugend sind und theoretische Annahmen bzgl. der Performance bestätigen. Somit ist eine Grundlage geschaffen worden, um den angestrebten TDMA-Ansatz innerhalb Kads umsetzen zu können, ohne eine Synchronisation mittels eines hierarchischen bzw. zentralisierten Ansatzes durchführen zu müssen.

4.3. Das HaRTKad-Verfahren

4.3.1. Einleitung

In diesem Unterkapitel wird das Verfahren *HaRTKad* (Hard Real-Time Kademlia) vorgestellt. HaRTKad soll die bereits in Unterkapitel 4.1 beschriebenen Nachteile der bestehenden Industrial Ethernet (IE)-Lösungen mittels eines neuen Ansatzes beseitigen. Gewöhnlich besitzen die IE-Lösungen eine zentrale Instanz, welche einen SPoF oder Flaschenhals darstellt, da meistens ein Master-Slave- oder Server-Client-Ansatz verwendet wird. Andere Realisierungen benötigen zusätzliche dedizierte Hardware, um ein hartes Echtzeitverhalten garantieren zu können. Zusätzlich sind die IE-Lösungen oftmals in ihrer Flexibilität eingeschränkt. Diese Aspekte gewinnen jedoch in Zukunft immer mehr an Bedeutung und wurden in Unterkapitel 4.1 untersucht. Die Schlussfolgerung ist, dass keine der bestehenden Lösungen für die Herausforderungen der Zukunft bzgl. Skalierbarkeit, Flexibilität oder Robustheit per se vorbereitet ist. P2P-Netzwerke hingegen sind eine Alternative zum typischen Master-Slave- oder Client-Server-Ansatz. Die typischen P2P-Netze befinden sich direkt in der Applikationsschicht, sodass keine dedizierte Spezialhardware erforderlich ist. Daher wird ein P2P-basierter Ansatz vorgestellt, welcher Kad nutzt, um ein dezentrales Netzwerk für Echtzeitapplikationen zu realisieren. Das Hauptaugenmerk liegt dabei auf der hohen Robustheit, Skalierbarkeit und Flexibilität des Netzwerkes. Das Kad-Protokoll wurde modifiziert, um einen arbitrierten Medienzugriff, welcher nötig ist, um einen deterministischen Kommunikationsablauf einer Echtzeitanwendung zu realisieren, zu ermöglichen. Zusätzlich werden die Ergebnisse eines HaRTKad-Prototypenknotens präsentiert.

Die Hauptbeiträge sind:

- Das modifizierte Kad-Protokoll.
- Performanceanalyse eines HaRTKad-Knotens.
- Bestimmung der Performance des modifizierten Kad-Protokolls bestehend aus mehreren HaRTKad-Knoten.
- Vergleich von HaRTKad mit einem proprietären System.

Die Hauptbeiträge sind in [SDAT14] publiziert. Der folgende Teil des Unterkapitels ist wie folgt organisiert: In Abschnitt 4.3.2 werden die nötigen Grundlagen kurz erläutert

und der HaRTKad-Ansatz erörtert. Abschnitt 4.3.3 beschreibt einen HaRTKad-Knoten, dessen Realisierung als Prototyp und dessen Performanceanalyse. In Abschnitt 4.3.5 wird ein Optimierungsvorschlag präsentiert, um die Anzahl der HaRTKad-Teilnehmer deutlich zu erhöhen, ohne dabei Echtzeitvorgaben zu verletzen. Ein Vergleich mit einer eigens entwickelten proprietären Lösung ist in Abschnitt 4.3.7 aufgeführt.

4.3.2. Grundlagen und Designkonzept

Kad wurde entwickelt, um ein vollkommen dezentrales strukturiertes Netzwerk aufzubauen. Jeder Knoten hat einen einzigartigen Hashwert, welcher ID genannt wird. Dieser Hashwert wird üblicherweise durch eine Hashfunktion generiert, wie z.B. mittels des MD4- oder MD5-Algorithmus [BB06]. Jeder Knoten ist verantwortlich für einen Satz an Daten bzw. Informationen. Die Verantwortlichkeit wird durch die Suchtoleranz ST gegeben. Kad generiert Hashwerte für Daten oder Informationen und bestimmt zuständige Knoten mittels der XOR-Distanz D zwischen dem Daten/Informations-Hashwert und dem Knoten-Hashwert. Ist die Distanz D kleiner als ST, ist ein Knoten für die Daten/Informationen zuständig. Formel 2.5 zeigt die Korrelation, wann ein Knoten zuständig ist. Somit beeinflusst der Hashwert direkt, welche Daten/Informationen einem Knoten zugeordnet werden [SW05].

In derzeitigen IE-Lösungen wird das Problem der Einhaltung von harten Echtzeitbedingungen gelöst, indem ein deterministischer Datenaustausch zugesichert wird. Dies kann durch einen TDMA-Ansatz, welcher aber meist mittels eines Masters bzw. einer zentralen Instanz realisiert wird, gelöst werden. Jedem Teilnehmer wird ein Zeitschlitz, in dem er das Medium nutzen darf, zugeordnet. Kad hingegen besitzt keine zentrale Instanz, welche den Medienzugriff steuern kann. Daher muss ein Knoten autonom wissen, wann er Zugriff auf das Medium erlangen darf. Der Hashwert eines Knotens $Hash_{Node}$ ist einzigartig und kann daher als Information genutzt werden, um einen Zeitschlitz zu bestimmen, in der der Knoten einen erlaubten Zugriff auf das Medium hat. Ähnlich der Relation zwischen Hashwerten der Daten/Informationen und denen der Knoten wird vorgeschlagen, eine Korrelation zwischen den Zeitschlitzen und den Hashwerten der Knoten zu erzeugen, da beide einzigartig sind.

Aus Effektivitäts- und Skalierbarkeitsgründen wird der in [DSA+15b] vorgestellte Algorithmus zur Bestimmung der dynamischen Suchtoleranz (DST) verwendet, um die

Netzwerkgröße zu bestimmen und die Suchtoleranz *ST* so einzustellen, dass für jeden Hashwert im Netzwerk **mindestens** ein Knoten zuständig ist. Der DST-Ansatz wird nicht nur verwendet, um das Netzwerk sinnvoll vorzukonfigurieren (siehe Abschnitt 4.2.3.1), sondern wird auch im Sinne von HaRTKad erweitert. Die Abbruchbedingung wird derart geändert, dass **höchstens** ein Knoten pro Hashwert zuständig ist. Dieser neue Algorithmus wird als inverse dynamische Suchtoleranz (IDST) bezeichnet. Ziel ist es, Hashbereiche zu ermitteln, die exklusiv durch einen Knoten besetzt sind. Abbildung 4.17 repräsentiert einen DHT-Ring, bei dem die neue zweite Suchtoleranz ST_{Slot} derart mit dem IDST-Algorithmus bestimmt wurde, dass höchstens ein Knoten pro Hashwert zuständig ist. Der neue Wert für ST_{Slot} wird konsistent mittels des IDST-Algorithmus bei jedem Knoten im Netzwerk gespeichert.

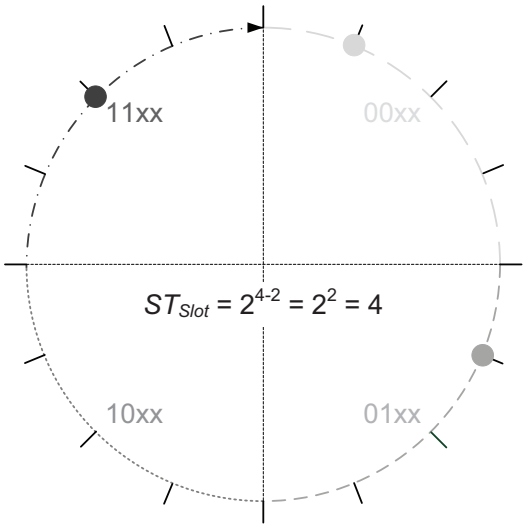

Abbildung 4.17.: Resultat des ISDT-Algortihmus. Der gesamte 4-Bit-Adressraum ist als Ring dargestellt. Drei Knoten sind beispielhaft auf dem Ring platziert. Es existieren vier Hashbereiche, die höchstens durch einen Knoten besetzt sind.

Die Voraussetzung für das TDMA-basierte Zeitschlitzverfahren ist eine einheitliche Zeitbasis aller teilnehmenden Knoten. Daher müssen alle Knoten im Kad-Netzwerk synchronisiert werden. Dies kann mittels des NTP- oder PTP-Protokolls erfolgen oder direkt in Kad mittels des in Abschnitt 4.2 beschriebenen Ansatzes.

Nachdem der IDST-Algorithmus ausgeführt wurde und alle Knoten synchronisiert wurden, ist jeder Knoten in der Lage, zu bestimmen, wann er kommunizieren darf (Zugriff auf das geteilte Ethernet-Kommunikationsmedium). Es wird empfohlen, den IDST-Algorithmus direkt nach dem DST-Algorithmus auszuführen, da nur das Abbruchkriterium ein anderes ist und die vorherigen Berechnungen identisch sind. Auf diese Weise ließe sich u.U. ein Großteil an Ausführungszeit einsparen.

Es ist nötig, eine Korrelation zwischen den Hashbereichen des Kad-Netzwerkes und dem Zeitraum herzustellen, um einen Zusammenhang zwischen HaRTKad-Knoten-Hashwerten und Zeitschlitzen zu bilden. Aus den Hashbereichen werden somit eindeutige Zeitschlitze. Daher wird ein Ansatz vorgestellt, welcher den arbitrierten Medienzugriff auf eine dezentrale Art löst. Jeder Knoten weiß selbst, wann er einen Medienzugriff vornehmen darf, da er sich selbstständig einem Zeitschlitz zuordnen kann. Die Anzahl an Schlitzen bzw. Hashbereichen, in denen maximal ein Knoten enthalten ist, ist in Formel 4.7 dargestellt und hängt von der Bitbreite der Hashwerte und der neuen Suchtoleranz ST_{Slot} ab. Zusätzlich muss jeder Netzwerkteilnehmer die HaRTKad-Applikation ausführen, weil ein unkontrollierter Medienzugriff zu möglichen Pufferüberläufen in den Switches und damit zu Paketausfällen führen kann. Somit wäre kein Determinismus und Echtzeitverhalten garantiert. Allerdings ist es denkbar, dass Nicht-HaRTKad-Teilnehmer gesonderte Zeiten zugesichert bekommen, um z.B. nicht isochrone Daten zu übertragen.

$$N_{Slots} = \frac{2^b}{ST_{Slot}} \tag{4.7}$$

Die Periode T_{Cyc} repräsentiert die Zykluszeit des Systems, in der jeder Knoten einmal kommunizieren darf, was einem Umlauf auf dem Hashring entsprechen soll. T_{Cyc} ist in Formel 4.8 definiert. T_{Del} ist die Zulieferzeit eines Knotens. Dies umfasst das Finden eines Zielknotens und eine Aktion mit diesem Knoten, wie die Übermittlung von Daten.

$$T_{Cyc} = T_{Del} * N_{Slots} \tag{4.8}$$

Die aktuelle Zeit auf dem Ring t_{Ring} kann aus der absoluten aktuellen Zeit t_{Now} und der Zykluszeit T_{Cyc} ermittelt werden (siehe Formel 4.9).

$$t_{Ring} = t_{Now} \bmod T_{Cyc} \tag{4.9}$$

Nun ist es wichtig, zu wissen, welchem Hashbereich und somit Zeitschlitz $Slot_{Node}$ ein Knoten zugeordnet ist. Hierfür kann Formel 4.10 verwendet werden.

$$Slot_{Node} = \frac{Hash_{Node}}{ST_{Slot}} \tag{4.10}$$

Da jeder Knoten den Wert für ST_{Slot} und die Zeit t_{Now} kennt, kann er unabhängig entscheiden, ob er senden darf. Das Entscheidungskriterium wird in Formel 4.11 repräsentiert.

$$(t_{Ring} > Slot_{Node} * T_{Del}) \wedge (t_{Ring} << Slot_{Node} * T_{Del} + T_{Del}) \tag{4.11}$$

Unter der Verwendung der zuvor genannten Formeln ist man in der Lage, durch Korrelation des Hash- und Zeitbereiches eine auf Applikationsebene mittels Kad realisierte TDMA-basierte Zeitschlitzkommunikation umzusetzen. Aus den Hashbereichen, die mittels des IDST-Algorithmus ermittelt werden, und dem Bezug zur Zeit werden Zeitschlitze generiert. Nach der Verwendung des IDST-Algorithmus muss nur noch die Zulieferzeit T_{Del} bestimmt werden. T_{Del} ist ein technischer Parameter, der nur schwer theoretisch zu ermitteln ist, da er auch von vielen anderen Parametern, wie der verwendeten Hardware, abhängt. Deshalb wurde ein Prototyp entwickelt, welcher in Abschnitt 4.3.3 vorgestellt wird.

Ein Grund, warum man nicht direkt IP-Adressen für die Zuordnung zu den Zeitschlitzen verwenden sollte, ist, dass HaRTKad unabhängig von jeglichen Protokollen in den tieferen Schichten sein soll. Daher erfolgt die Zuordnung der Zeitschlitze in der Applikationsschicht, sodass andere und auch proprietäre Protokolle neben IP unterstützt werden können. Außerdem sind die IP-Adressen nicht derart gleich verteilt wie die durch den MD4- oder MD5-Algorithmus generierten Hashwerte.

4.3.3. Der HaRTKad-Prototypenknoten

Da der präsentierte Ansatz für Automatisierungsszenarien gedacht ist, ist es nicht nur notwendig, den deterministischen Datenaustausch zwischen den Knoten zu garantieren. Vielmehr ist es auch nötig, das Echtzeitverhalten der Kad-Knoten in Bezug auf die Paketverarbeitung sicherzustellen. Voraussetzung ist ein Echtzeitbetriebssystem, um einen

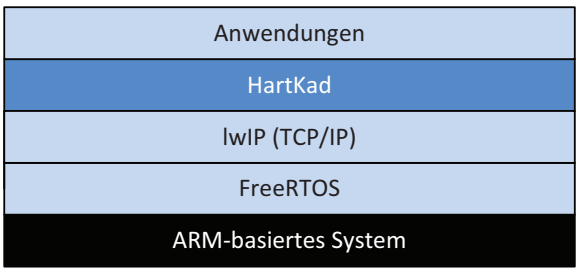

Abbildung 4.18.: Software-Stack eines Kad-Knotens.

HaRTKad-Knoten zu realisieren. Unter diesen Bedingungen muss die gewählte Zielplattform von einem Echtzeitbetriebssystem unterstützt werden. Eine Umsetzung der Idee in Form eines ersten Prototyps erfolgte in der Masterarbeit von Robert Gubitz [Gub13], welche mit dem Ludwig-Bölkow-Nachwuchspreis 2013 ausgezeichnet wurde [lud13]. Aufgrund des Einsatzszenarios in der industriellen Automatisierung wurde ein eingebettetes System ausgewählt. Als Zielplattform dient das ZedBoard mit einem 667-MHz getakteten ARM-Prozessor [Avn]. Mittels des entwickelten HaRTKad-Prototyps ist es möglich, alle Zeiten bzgl. der Erstellung, Sendung, des Empfangs und der Verarbeitung der ausgetauschten UDP-Pakete zu ermitteln, was T_{Del} entspricht. Der Software-Stack des Prototyps ist in Abbildung 4.18 dargestellt.

Software-Stack: Als Basis für die Software dient das ZedBoard als ARM-basierendes System. FreeRTOS wurde als Betriebssystem ausgewählt, da es ein hartes Echtzeitverhalten der Kad-Knoten ermöglicht [Rea]. Zusätzlich wird lwIP als Teil von FreeRTOS als leichtgewichtige Implementierung des TCP/IP-Stacks verwendet, um die Kommunikation über Ethernet zu ermöglichen [Fre]. In der nächsten Schicht befindet sich die HaRTKad-Applikation, welche den Medienzugriff steuert und somit die echtzeitfähige Kommunikation durch die Realisierung der Zeitschlitze ermöglicht. In dieser Schicht wird der neue Ansatz realisiert. HaRTKad versteht sich hierbei nicht nur als Applikation, sondern vielmehr auch als Middleware für weitere Applikationen oberhalb von HaRTKad. Zwei mögliche Applikationen werden in Kapitel 5 vorgestellt.

Threadaufbau: Der HaRTKad-Client besteht aus mehreren Threads. Alle Threads sind mit einer kurzen Beschreibung in Tabelle 4.4 aufgeführt. Die Threads sind bereits nach ihrer Priorität in FreeRTOS sortiert und es wird mit dem höchsten begonnen.

Der Main-Thread ist notwendig, da er die anderen Threads startet, und hat daher die

Thread	Priorität	Beschreibung
Main	5	Startet die anderen Threads
Externe Kontrolle	4	Empfang von externen Kommandos
Kad-Kommunikation	3	Verarbeitung der Kad-Pakete
Suche	3	Überwacht/zerstört Suchobjekte
Netzwerk	2	Paketverarbeitung im Netzwerkinterface
Wartung	1	Wartungsthread
Idle	OS	Verwendet, um neue Threads zu generieren

Tabelle 4.4.: Aktive Threads des Kad-Clients.

höchste Priorität. Nachdem er die anderen Threads gestartet hat, geht er in den Idle-Zustand. Die externe Kontrolle bekommt die zweithöchste Priorität, um auf externe Trigger wie einen durch eine Person ausgelösten Feueralarm reagieren zu können. Dabei können externe Trigger auch dedizierte Leitungen/Geräte darstellen, die für hochkritische Prozesse wie z.B. angeschlossene Sensoren an einem HaRTKad-Knoten verwendet werden. Der Thread für die Kad-Kommunikation hat die nächstniedrigere Priorität und ist verantwortlich für die Verarbeitung der Kad-Pakete. Nachfolgend gibt es maximal drei Threads, die die Suchobjekte von Kad überwachen und löschen, wenn die entsprechenden Bedingungen erfüllt sind. Da drei Suchobjekte aktiv unterstützt werden, können diesbezüglich maximal drei Threads existieren. Der Netzwerkthread buffert die Pakete vom Netzwerk-Interface und leitet diese an die HaRTKad-Applikation weiter. Drei Wartungs-Threads sind dafür verantwortlich, das Netzwerk auf dem aktuellsten Stand zu halten. Abschließend kommt der Idle-Thread, welcher zur Generierung neuer Threads dient. Dieser hat eine vom Betriebssystem abhängige Priorität.

4.3.4. Performance-Evaluierung

Die Verwendung des modifizierten Kademlia-Protokolls und der echtzeitfähigen Kad-Knoten erlaubt die Realisierung von Applikationen mit harten Echtzeitanforderungen auf Basis von P2P-Technologie. Es ist nötig, ein Prototypenszenario mit HaRTKad-Knoten aufzubauen, um die Performance des Systems zu ermitteln. Daher wurde ein Setup von vier Komponenten zusammengestellt, welches es ermöglicht, die nötigen Messwer-

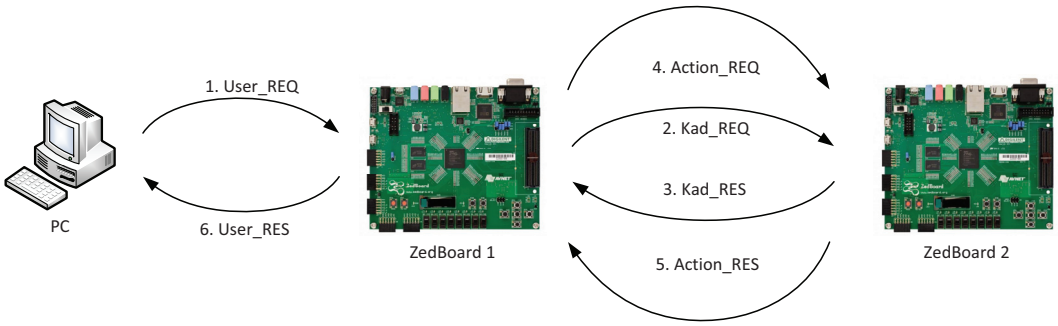

Abbildung 4.19.: Experimentalaufbau für die Evaluierung.

te aufzunehmen. Ein PC, zwei ZedBoards und ein 8-Port Gigabit-Switch der Firma Netgear [Net] stellen den Experimentalaufbau dar. Die zwei ZedBoards repräsentieren das Kad-Netzwerk und sind logisch durch das Kad-Protokoll verbunden. Das Setup ist in Abbildung 4.19 dargestellt. Die zwei Knoten bilden genau den diskreten Zeitraum eines Zeitschlitzes ab, bei dem sie den exklusiven Zugriff auf das Medium besitzen. Somit ist es möglich mit zwei Instanzen pro diskreten Zeitschlitz auf das Gesamtsystem bestehend aus N_{Slots} Zeitschlitzen zu schlussfolgern. Derart können alle nötigen Parameter ermittelt werden. Zwei Operationen zwischen zwei ZedBoards werden in diesem Setup unterstützt. Es handelt sich um Lese- und Schreibeoperationen, welche innerhalb der HaRTKad-Applikation ausgeführt werden können.

Leseoperation: Wenn ein Nutzer eine Leseoperation durchführt, wird eine Anzahl an Integerwerten angefordert. Die Anzahl der Werte wird durch den Nutzer im User-Request-Paket angegeben. Zusätzlich wird der Hashwert des Knotens, welcher die Integerwerte liefern soll, angegeben. Das erste ZedBoard empfängt und verarbeitet das User-Request-Paket. Da das erste ZedBoard nicht für die Leseanfrage zuständig ist, sucht es im Kad-Netzwerk den Knoten, welcher laut Hashwert im User-Request-Paket für die Anfrage verantwortlich ist. Daher kontaktiert das erste ZedBoard das zweite mittels eines Kad-Requests, weil es für Anfragen zuständig ist, und überprüft, ob dieser Knoten noch existiert. Das zweite ZedBoard antwortet durch ein Kad-Response-Paket. Wenn das erste ZedBoard das Kad-Response-Paket erhält, kann es das zweite Board wieder kontaktieren und die Leseaktion ausführen. Die Aktion wird mittels eines Action-Request-Paketes durchgeführt, welches in diesem Falle ein Read Action-Request-Paket darstellt. Das zweite ZedBoard antwortet durch ein Action-Response-Paket, was in diesem Fall durch ein

Read Action-Response-Paket realisiert wird. Das Read Action-Response-Paket enthält alle vom Nutzer angeforderten Integerwerte, wobei auf dem sendenden Knoten zufällig Integerwerte erzeugt werden. Nachdem das erste ZedBoard das Read Action-Response-Paket erhalten hat, leitet es die Integerwerte zum Nutzer mittels eines User-Response-Paketes weiter.

Schreibeoperation: Wenn eine Schreibeoperation ausgeführt wird, wird eine Anzahl an Integerwerten im User-Request-Paket übertragen. Wie bei der zuvor beschriebenen Leseoperation wird der verantwortliche Knoten für die Schreibeoperation im Kad-Netzwerk gesucht. Das ist in diesem Fall ebenfalls das zweite ZedBoard. Dieses erhält nun das Action-Request-Paket, welches ein Write Action-Request-Paket vom ersten ZedBoard darstellt. In dem Paket sind die Integerwerte enthalten, die auf dem zweiten ZedBoard gespeichert werden sollen. Das zweite ZedBoard sendet ein Write Action-Response-Paket als Bestätigung zurück, welches vom ersten ZedBoard zum Nutzer in Form eines User-Response-Paketes weitergeleitet wird.

Die Zeiten wurden für beide Operationen gemessen, was auch die Kad-Operationen und dessen Verarbeitung der Pakete mit einschließt. Der erste Messpunkt wird genommen, wenn das erste ZedBoard das User-Request-Paket erhält. Der zweite Zeitwert wird genommen, wenn das User-Response-Paket zurück zum PC gesendet wird. Das Ergebnis der Zeitmessung für beide Operationen ist in Abbildung 4.20 zu sehen. Es ist deutlich, dass Ergebnisse von unter einer Millisekunde erreicht werden können. Außerdem ist ein lineares Verhalten bzgl. der Anzahl an angeforderten bzw. gesendeten Integerwerten zu erkennen. Die Ergebnisse stellen T_{Del} dar, was der Erstellung eines Suchobjektes, einem Suchschritt (Lookup-Schritt) und dem Austausch von Integerwerten entspricht. Diese Ergebnisse können für weitere Betrachtungen verwendet werden. Es wurde auch eine Alternative zum ZedBoard untersucht. In Anhang B ist als Alternative zum ZedBoard das Raspberry Pi für den Einsatz mit HaRTKad beschrieben. Auf Grund der Performance und Echtzeitfähigkeit wurde das ZedBoard ausgewählt.

4.3.5. Optimierung der Kanalauslastung

Durch die Nutzung des TDMA-Ansatzes und der IDST kann es vorkommen, dass einzelne Slots keinen Teilnehmer haben. Daher wird ein Ansatz vorgeschlagen, der eine effektivere Auslastung des Mediums erzeugt. Zudem wird bei besetzten Zeitschlitzen das Medium

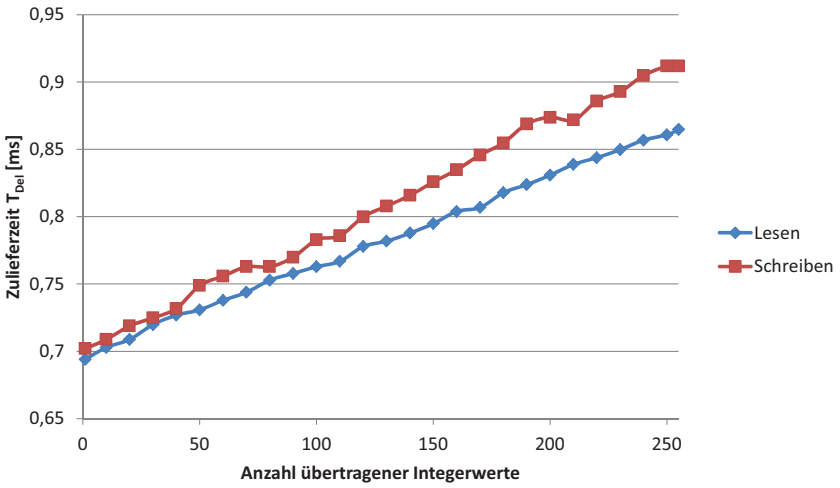

Abbildung 4.20.: Performance-Evaluierung der HaRTKad-Kommunikation und Paketverarbeitung.

nur geringfügig ausgelastet, worin noch Optimierungsmöglichkeiten bestehen.

Ausnutzung der Verarbeitungszeit auf einem Knoten: Ist die schnellste Verarbeitungszeit eines Knotens bekannt, kann diese Zeit genutzt werden, um eine höhere Auslastung des Kanals zu erreichen. Wenn als Zielplattform für HaRTKad das ZedBoard genommen wird, ist es möglich, die Kanalauslastung CU zu bestimmen (siehe Formel 4.12).

$$CU = \frac{Data_{packets}}{T_{Del} * 1GBit/s} \tag{4.12}$$

Die Auslastung hängt direkt von der Datenmenge ab, die übertragen wird. Die Pakete, die verwendet werden, sind bereits aus dem im Vorfeld vorgestellten Szenario ersichtlich. Die Pakete wurden zusammengefasst in der Tabelle 4.5 mit ihren Größen aufgelistet, sodass es möglich ist, die Kanalauslastung mittels der Gesamtheit an Daten $Data_{packets}$, die ausgetauscht wurden, zu berechnen. Für die folgenden Ergebnisse wurden nur die Pakete verwendet, die innerhalb des Kad-Netzwerkes übertragen wurden. Die Pakete vom und zum PC wurden nicht berücksichtigt. Um die Ergebnisse vergleichbar zu machen, wurde angenommen, dass bei einer Lese- oder Schreibeoperation ein Integerwert (4 Bytes) übertragen wurde.

Paket	Sub-Typ	Grösse [Byte]
User_REQ		64
User_RES		64
Kad_REQ		89
Kad_RES		96
Action_REQ	read	88
	write	88
Action_RES	read	72
	write	72

Tabelle 4.5.: Auflistung der Paketgrößen.

Abbildung 4.21.: Interleaving des Medienzugriffes zur Erhöhung der Auslastung.

Wenn ein Knoten mit dem Medienzugriff an der Reihe ist, benötigt er die meiste Zeit für das Verarbeiten von Paketen, anstatt für die Datenübertragung. Daher ist es möglich, einen parallelen Zugriff auf das Medium zu gewähren, um die effektive Auslastung zu erhöhen. Dieses Interleaving ist beispielhaft in der Abbildung 4.21 dargestellt. Der Peer mit dem Hashwert „0000“ kontaktiert den Peer mit dem Hashwert „0011“. Aufgrund der Zeit für die Verarbeitung des Paketes kann der Peer mit dem Hashwert „0001“ den Peer mit dem Hashwert „0010“ ohne negative Einflüsse auf das System kontaktieren. Zu beachten ist, dass ein Knoten nicht parallel in mehrere Prozesse eingebunden ist, damit es nicht zu unbeabsichtigten Blockierungen kommt. Dies sollte auf Applikationsebene geregelt sein, da HaRTKad i.d.R. nicht über die nötigen Informationen verfügt.

Mit den zuvor bestimmten Performanceparametern ist es nun möglich, die Anzahl der Knoten, welche theoretisch das Medium bei einer theoretischen Kanalauslastung von 100 % in einem Zeitschlitz nutzen können, zu bestimmen.

Um eine sinnvolle Aussage über die Performance bzw. Anzahl unterstützter Knoten vorzunehmen, wurde die Arbeit von Mark Felser [Fel05] verwendet. In dieser Arbeit werden drei Klassen definiert: Human Control, Process Control und Motion Control. Diese drei Klassen haben verschiedene Vorgaben in Bezug auf die Zykluszeit T_{Cyc}. Zur besseren Vergleichbarkeit wird für die weiteren Betrachtungen angenommen, dass ein Informationsaustausch von einem Integerwert (4 Bytes) erfolgt. Das Ergebnis des Prototyps zeigt, dass das Finden eines Knotens und der Austausch von 4 Byte ca. 700 μs benötigt, was bereits einen Suchschritt von 150 μs beinhaltet. Jeder weitere Suchschritt dauert zusätzliche 150 μs, was als T_{Step} gekennzeichnet ist. Die Anzahl der Schritte hängt logarithmisch von der Anzahl der Knoten im Kad-Netzwerk ab. 550 μs werden benötigt, um ein Suchobjekt zu erstellen und ein Action-Request- und Reponse-Paket zu senden, zu verarbeiten, was insgesamt als T_{Action} bezeichnet wird.

Es wurde berücksichtigt, dass die Zeit zum Finden anderer Knoten im Kad-Netzwerk ansteigt, was durch Formel 4.13 ausgedrückt wird.

$$T_{Del} = T_{Action} + (log_2(Nodes) * T_{Step}) \tag{4.13}$$

Mit Hilfe der Formel 4.14 kann die Anzahl der Knoten ohne Interleaving ermittelt werden. Dies muss berücksichtigt werden, da man durch mehr Knoten auch mehr Zeit zum Finden der Knoten benötigt.

$$Nodes = \frac{T_{Cyc}}{T_{Del}} \tag{4.14}$$

Die Lösung von Formel 4.14 nach dem Parameter $Nodes$ ist in Formel 4.15 gegeben, wobei W die Lambert-Funktion ist.

$$Nodes_{Max} = \left\lfloor \frac{T_{Cyc} * log(2)}{T_{Step} * W(\frac{2^{\frac{T_{Action}}{T_{Step}}} * T_{Cyc} * log(2)}{T_{Step}})} \right\rfloor \tag{4.15}$$

In Tabelle 4.6 ist die mögliche Anzahl an Knoten gegeben, die mit dem vorgestellten Ansatz realisiert werden kann.

Wenn keine Optimierung vorgenommen wurde, ist man in der Lage, eine geringe Anzahl an Knoten zu unterstützen. Das Datenaufkommen, bestehend aus den Kad-Paketen und den Action-Paketen, ist angegeben, um die Kanalauslastung zu berechnen. Es ist ersichtlich, dass die daraus resultierende Ethernet-Kanalauslastung sehr gering ist ($< 1\%$). Wenn hingegen die Optimierung in Form von Interleaving berücksichtigt wird und eine parallele Kommunikation erlaubt ist, kann die Anzahl der Knoten stark gesteigert werden. Dabei wird eine theoretische Auslastung des Mediums von 100 % angenommen und numerisch die maximale Anzahl an Knoten bestimmt. Selbst wenn die Steigerung geringer ist, ist ein enormes Steigerungspotential zu erwarten.

Attribute	Human	Process	Motion
Zykluszeit T_{Cyc} [ms]	100	10	1
Ohne Interleaving			
$Nodes_{Std}$	68	9	1
Zulieferzeit T_{Del} [μs]	1600	1150	700
Daten pro T_{Cyc} [KB]	98,94	8,1	0,345
Kanalauslastung [%]	0,73	0,63	0,39
Mit Interleaving			
$Nodes_{Opt}$	4.873	621	85 (8)
Zulieferzeit T_{Del} [μs]	2650	2200	1750 (1000)
Daten pro T_{Cyc} [KByte]	12.499,95	1.248,21	123,68 (5,586)
Kanalauslastung [%]	99,99	99,85	98,94 (4,576)

Tabelle 4.6.: Zusammenfassung der HaRTKad-Performance.

Wenn ein Human Control-Szenario gewählt wird, kann eine Kanalauslastung von 0,73 % erreicht werden, falls kein Interleaving verwendet wird. Aufgrund der niedrigen Kanalauslastung ist es möglich und auch von Vorteil, parallele Kommunikationen einzusetzen. Man kann dabei aber nicht ohne Weiteres im Human Control-Szenario die Anzahl der Knoten um den Faktor 136 erhöhen, um von 0,73 % auf 100 % zu kommen, da eine höhere Anzahl an Knoten auch die Anzahl der Suchschritte erhöht und dies ein erhöh-

tes Datavolumen zur Folge hat. So kann z.B. beim Human Control-Szenario die Anzahl der Knoten von 68 auf 4873 erhöht werden, was eine Steigerung um den Faktor von 71 entspricht, wenn die Abhängigkeit von der Anzahl der Knoten berücksichtigt wird.

Zu beachten sind dabei die Werte für Interleaving beim Motion-Szenario. Würde man eine 100%ige Auslastung des Mediums zulassen, führt dies bei 85 Knoten zu einer zu hohen Zeit T_{Del} zum Finden eines Knoten und einer Interaktion. Die Zeit T_{Del} würde 1750 μs betragen. Dies liegt vor allem an der größeren Anzahl an Hops bei 85 Knoten. Da für die Anzahl der Hops der Worst Case angenommen wird, wurde alternativ in Klammern in der Tabelle 4.6 die Anzahl an Knoten angegeben, ohne die Vorgaben des Motion-Szenarios ($T_{Cyc} = 1$ ms) zu verletzen. Hier ist ein großes Potential zu erkennen, da vom Worst Case ausgegangen wurde. Ergebnisse der Arbeit [Kap14] zeigen, dass die durchschnittliche Anzahl an Hops bei 1,4 liegt. Dies würde zu einer geringeren Zeit T_{Del} führen, womit die Anzahl von 85 Knoten potentiell möglich wäre.

Um die Optimierung (Interleaving) einzuführen, ist es nur nötig, den IDST-Algorithmus, welcher in Abschnitt 4.3.2 beschrieben ist, leicht zu verändern. Der IDST-Algorithmus berechnet ST_{Slot} derart, dass höchstens ein Knoten in einem Zeitschlitz erlaubt ist. Die Anzahl der Schlitze N_{Slots} wird beibehalten und die Anzahl der Knoten $Nodes_{Opt}$, die im Vorfeld numerisch ermittelt wurden, wird nun verwendet. Der neue approximierte Wert für die Knotenanzahl pro Schlitz $Nodes_{Slot_{Opt}}$ ist in Formel 4.16 gegeben. Der Vorteil durch die Nutzung des MD5-Algorithmus ist, dass nahezu das theoretische Maximum erreicht wird, da die Hashwerte gleich verteilt sind.

$$Nodes_{Slot_{Opt}} = \frac{Nodes_{Opt}}{Nodes_{Std}} \tag{4.16}$$

Der neue Suchtoleranzwert $ST_{Slot_{Opt}}$ wird durch den IDST-Algorithmus unter der Verwendung von $Nodes_{Slot_{Opt}}$ erzeugt und im Netzwerk verteilt. Die Wechselwirkung der Kommunikation durch die erhöhte Netzwerkauslastung sollte dabei noch durch Simulationen untersucht werden. Inwieweit die theoretische Kanalauslastung von nahezu 100 % erreicht werden kann, könnte derart bestimmt werden.

4.3.6. Umgang mit Kollisionen

Vereinzelte Kollisionen sollten direkt in der Applikation abgefangen werden, indem überprüft wird, ob bereits ein Gerät mit dem Hashwert im Netzwerk existiert. Nun kann es vorkommen, dass z.B. Geräte anhand ihrer Funktion beschrieben werden, wie z.B. Feuermelder, und dies als Parameter für die Hashfunktion genutzt wird. Bei vielen Feuermeldern in einer Anlage würde dies zu massiven Hashkollisionen führen, sodass nicht mehr garantiert werden kann, dass alle Geräte gefunden werden, da nur ein Gerät pro Hashwert in der Routingtabelle eines Knotens gespeichert wird. Dieses Problem wurde bereits durch ein sogenanntes MultiKad-System gelöst. Durch die Einführung eines zweites Hashwertes können Subringe gebildet werden, die alle den gleichen ursprünglichen Hashwert besitzen. Dies ermöglicht einen Aufbau aus Subringen, die z.B. alle die gleiche Funktionalität anbieten. Eine detaillierte Beschreibung findet sich in [ASD+14a].

4.3.7. Vergleich mit einem proprietären System

Wie bereits in Abschnitt 4.1 ersichtlich ist, kann hingegen mit teilweise nicht standardkonformen, spezialisierten Lösungen auch eine höhere Performance erreicht werden. In Zusammenarbeit mit dem Max-Planck-Institut für Plasmaforschung in Greifswald wurde solch ein z.T. proprietäres System für das Wendelstein 7-X (W7-X)-Fusions-Experiment entwickelt, das sogenannte Time-Trigger-Event(TTE)-System. Das TTE-System wird an dieser Stelle vorgestellt, um die Vor- und Nachteile einer proprietären Lösung mit höchsten Anforderungen aufzuzeigen. Zudem ist damit ein Vergleich mit dem HaRTKad-System möglich. Der W7-X ist ein Fusions-Experiment, basierend auf dem Stellarator-Konzept [GWXT98]. Ziel des Projektes ist es, die Dauerbetriebsmöglichkeiten der Stellarator-Fusions-Anlage zu erforschen. Es sind Datenraten von 30 GByte/s und eine Gesamtmenge von 50 TByte pro Langzeitexperiment geplant. Aus diesem Grund sind erweiterte Konzepte für die Echtzeitkontrolle des Plasmas, der kontinuierlichen Datenerfassung und Datenarchivierung nötig. Die wichtigste Voraussetzung für die Kontrolle und Datenerfassung ist es, alle Messungen mit präzisen Zeitstempeln zu versehen.

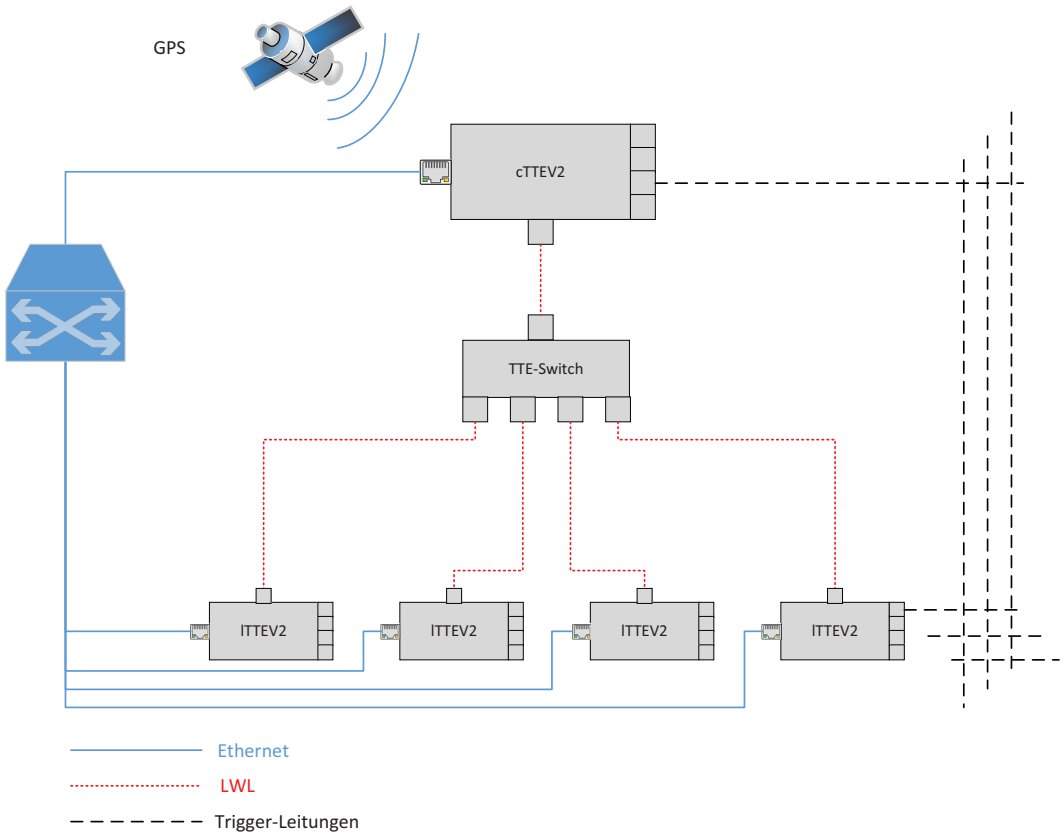

Abbildung 4.22.: TTE-Systemübersicht.

4.3.7.1. Genereller Aufbau des TTE-Systems

Abbildung 4.22 stellt das gesamte TTE-System des W7-X dar. Das TTE-System besteht aus vier Komponenten: der zentralen cTTE-Karte, den lokalen lTTE-Karten, den TTE-Switches und den Verbindungen zwischen den Komponenten.

Um den hohen Anforderungen der präzisen Zeitstempel mit einer Auflösung 10 bis 20 ns im W7-X-Umfeld gerecht zu werden, wurde das TTE-System entwickelt. Die Hauptaufgabe des TTE-Systems ist die Synchronisation aller Takte mit einer ausreichenden Genauigkeit [SNW+02]. Des Weiteren muss das System in der Lage sein, Event-Nachrichten über Ereignisse senden, empfangen und verarbeiten zu können. Auf Basis der präzisen Zeiten bauen weitere Funktionalitäten auf, wie z.B. Zeitstempelgenerierung oder die Erzeugung von elektrischen Pulsen. Die Verbindung zwischen der cTTE-Karte und den

lTTE-Karten wird über ein optisches Lichtwellenleiter(LWL)-Netzwerk hergestellt, welches auch TTE-Switches beinhaltet, um einen Nachrichtenversand zur cTTE-Karte, die ebenfalls einen LWL-Eingangsport besitzt, zu ermöglichen. Der Datentransfer von Zeit- und Event-Nachrichten auf dem LWL basiert auf einem proprietären Protokoll, aufbauend auf Bitstuffing und Manchester-Kodierung. Zusätzlich wird der Takt kodiert über LWL mitübertragen. Er wird direkt aus dem Manchester-kodierten Signal gewonnen (Details siehe in [SLN05, SLN06]). Die TTE-Funktionalität umfasst, dass jede lTTE- und cTTE-Karte in der Lage ist, Trigger-, Zeit- und Event-Nachrichten zu verarbeiten und zu erzeugen. Nachrichten von der cTTE-Karte zu den lTTE-Karten wurden als Broadcast gesendet, wohingegen Nachrichten von einer lTTE-Karte an eine cTTE-Karte als Unicast-Nachrichten verschickt sind, da nur die cTTE-Karte als Empfänger bestimmt ist. Trigger-Signale mit sehr hohen Anforderungen an die Antwortzeit und Zuverlässigkeit werden über zusätzliche Trigger-Leitungen realisiert. Die Anzahl der TTE-Einheiten ist flexibel und nicht eingeschränkt. Die erste Generation der TTE-Einheiten wird im Folgenden mit dem Kürzel „V1“ vermerkt. Die aktuell entwickelte Version mit dem Kürzel „V2“ wird in [ScSk12] detaillierter vorgestellt. Ebenfalls Bestandteil des Funktionsumfangs des Systems ist eine Delaymessung der Pfade zwischen den Instanzen, welche mit einer Genauigkeit von ca. 2 ns durchgeführt werden kann. Dies zeigt ebenfalls die hohen Anforderungen an das System. Im laufenden Betrieb besteht das System aus zwei cTTE-Karten, wobei eine als Backup dient. Die Anzahl der lTTE-Karten beträgt ca. 100 Einheiten.

4.3.7.2. Vergleich TTE-System mit HaRTKad

Das HaRTKad-System und das TTE-System weisen beide Vor- und Nachteile auf. Je nach Einsatzgebiet ist die Wahl zu treffen, welches System zu bevorzugen ist. Als Entscheidungsgrundlage werden im Folgenden die Vor- und Nachteile aufgeführt.

Vorteile des TTE-Systems: Der größte Vorteil des TTE-Systems liegt in der Performance. Benachrichtigungen über das LWL-Netzwerk werden nahezu ohne Verzögerung an die Teilnehmer verteilt. Die einzige variable Verzögerung tritt durch die Leitungslängen auf. Darum muss das Delay zwischen der cTTE-Karte und den lTTE-Karten bestimmt werden. Die lokale Auflösung der Zeit beträgt 10 ns, was hochperformante und präzise lokale Funktionalitäten ermöglicht. Das System selbst nutzt mehrere Übertra-

gungsmedien. Neben dem LWL-Netzwerk werden auch Ethernet und dedizierte Kabel zur direkten Verdrahtung verschiedener Geräten mit lTTEV2-Karten oder lTTEV2-Karten untereinander verwendet. Durch den Einsatz von Ethernet und Standardprotokollen, wie Raw-Ethernet oder UDP/IP, ist es möglich, die Systeme auch als autonome Einheiten aufzufassen. Sie lassen sich somit bei der Verwendung von UDP/IP in bestehende Ethernet-basierte Netzwerke integrieren, was eine Wartung/Interaktion mit dem System von einer beliebigen Stelle im Netzwerk ermöglicht.

Nachteile des TTE-Systems: Der Nachteil des Systems ist der komplexe Aufbau durch die Verwendung mehrerer Übertragungsmedien. Zusätzlich ist der Aufbau durch die hierarchische Struktur eingeschränkt, da ein Master zur Synchronisation zwingend notwendig ist. Dieser Master, welcher durch die cTTEV2-Karte repräsentiert ist, stellt auch eine potentielle Schwachstelle in Form eines SPoFs dar. Sollte die cTTEV2-Karte ausfallen, ist eine Synchronisation des gesamten Netzwerkes nicht möglich. Dieser Umstand wird ein wenig abgeschwächt, da die lTTEV2-Karten auch in der Lage sind, autonom zu arbeiten. Jedoch kann ihr synchronisierter Zustand durch z.B. Clock Drifts der lokalen Taktgeber nicht zugesichert werden. Die cTTEV2-Karte stellt auch einen Flaschenhals bzgl. der Kommunikation mittels LWL dar. Da alle lTTEV2-Karten zu der zentralen cTTE-Karte senden können, kann schnell eine Überlastsituation entstehen. Als Gegenmaßnahme werden TTE-Switches eingesetzt, um die Daten zur cTTEV2-Karte zu puffern. Bei den drei TTE-Komponenten handelt es sich allerdings um proprietäre Lösungen, was hohe Anschaffungskosten gegenüber Standardkomponenten zur Folge hat. Durch den Einsatz von z.T. proprietären Protokollen (Datenübertragung bei LWL) und Hardware ist auch die Interoperabilität eingeschränkt. Ein weiterer Nachteil ist, dass Ethernet nicht für harte Echtzeitanforderungen verwendet werden kann, da der Medienzugriff nicht gesteuert wird.

Vorteile des HaRTKad-Systems: Das HaRTKad-System als reine Realisierung auf der Applikationsschicht verwendet keine proprietären Protokolle und versteht sich als Middleware für andere Applikationen. Zusätzlich wird nur Standard-Hardware zur Verarbeitung und Kommunikation genutzt, was die Herstellung von Geräten kostengünstig macht. Da die Kommunikation nur UDP/IP als Voraussetzung hat, lassen sich HaRTKad-Systeme auf alle Technologien unterhalb von UDP/IP abbilden. Hierdurch wird ein hoher Grad an Integration erreicht. Da HaRTKad zusätzlich auf Kad als P2P-Netzwerk basiert, erbt es auch dessen intrinsische Eigenschaften. Es wird keine zentrale

Instanz zur Verwaltung des Netzwerkes benötigt, womit potentielle Schwachstellen in Form von SPoFs vermieden werden. Dies ist eine Grundvoraussetzung für ein Höchstmaß an Flexibilität. Das Netzwerk ist von jedem Gerät aus wartbar, da nur ein IP-Zugang zum Netzwerk erforderlich ist. Auch kann es zu keiner Überlastsituation von Geräten kommen, weil das Netzwerk sehr gut mit der Anzahl der Knoten bzgl. der Synchronisation und des Datenaustausches skaliert.

Nachteile des HaRTKad-Systems: Als Nachteil ist die geringere Performance gegenüber dem TTE-System festzustellen. Die für die Ergebnisse verwendete eingebettete Zielplattform erreicht bereits beachtliche Werte von unter 1 ms für das Finden und den Austausch von Daten, ist aber für Einsätze im Nanosekundenbereich derzeitig nicht geeignet. Mit Steigerung der Leitungsfähigkeit der Systeme und auch der Netzwerkverbindungen ist aber ein großes Potential in der Zukunft zu sehen, um HaRTKad weiter zu verbessern.

Es wird deutlich, dass die Zeitanforderungen das zu wählende System maßgeblich bestimmen. Das HaRTKad-System bietet auf Grund seiner hohen Flexibilität, Skalierbarkeit, Wartbarkeit, niedrigen Kosten und Integrationsfähigkeit die höchste Zukunftsfähigkeit, besonders im Umfeld des Internet der Dinge.

4.3.8. Fazit

In diesem Abschnitt wurde ein Ansatz zur Realisierung eines komplett dezentralen Kad-Netzwerkes, welcher Operationen in harter Echtzeit ermöglicht, vorgestellt. Zusätzlich wurde ein Prototyp vorgestellt, der den HaRTKad-Ansatz realisiert. Die Kombination aus dem präsentierten modifizierten Kad-Protokoll und dem laufenden Prototypen erlaubt die Realisierung von Applikationen mit harten Echtzeitanforderung, hoher Ausfallsicherheit, Skalierbarkeit und Flexibilität ohne einen SPoF. Des Weiteren besitzt das System eine hohe administrative Skalierbarkeit, da es von allen Punkten im Netz aus gewartet werden kann. Das Hinzufügen und Entfernen von Instanzen geschieht flexibel ohne den Bedarf einer zentralen Instanz. Jeder Knoten agiert autonom anhand seiner Daten und gegebenen Parameter und steuert seinen Medienzugriff, was ein hartes Echtzeitverhalten mit hoher Flexibilität und Skalierbarkeit ermöglicht. Weitere Optimierungen könnten durch Verwendung eines neueren 10-GBit/s-Ethernet-Standards erreicht werden. Zudem würde das Einbeziehen von Topologiewissen einen enormen Vorteil bringen,

weil derzeit die Worst Case-Annahme getroffen wird, dass es einen Punkt im Netzwerk geben könnte, den der gesamte Datenverkehr passieren muss. Dies ist z.B. der Fall, wenn alle Daten an einen Endknoten gehen. Ist dies nicht der Fall, könnte eine gesteigerte Parallelität einen Vorteil bzgl. der Anzahl der unterstützten Knoten bringen. Ein abschließender Vergleich mit dem mitentwickelten teils proprietären TTE-System erörtert die Vor- und Nachteile beider Ansätze. Es wird deutlich, dass die Zeitanforderungen das zu wählende System maßgeblich bestimmen. Das HaRTKad-System bietet auf Grund seiner hohen Flexibilität, Skalierbarkeit, Wartbarkeit, niedrigen Kosten und Integrationsfähigkeit die höchste Zukunftsfähigkeit, besonders im Umfeld des Internet der Dinge.

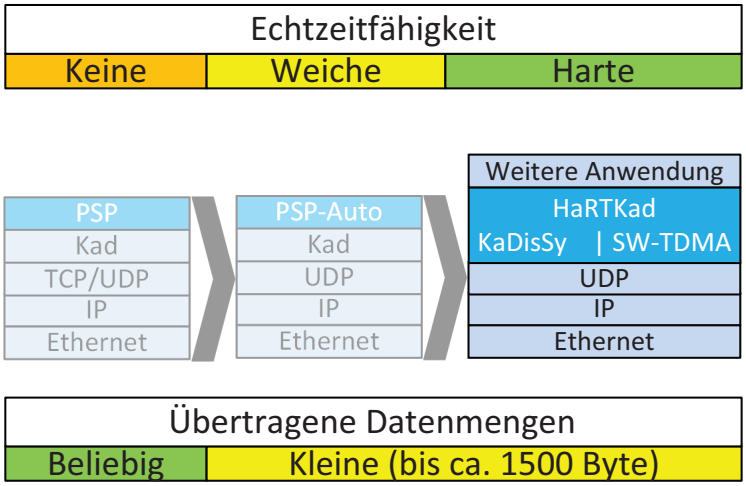

Abbildung 4.23.: Darstellung des HaRTKad-Protokollstacks und Einordnung in den Gesamtkontext.

Einordnung in den Gesamtkontext der Arbeit: HaRTKad wurde entwickelt, um das Defizit der deterministischen Datenübertragung mittels Ethernet zu lösen. Dieses Defizit ist der Grund, weshalb PSP-Auto (siehe Unterkapitel 3.2) noch nicht über harte Echtzeit verfügt. Mit dem Fokus auf die Automatisierung wurden weitere Anforderungen wie die Interoperabilität, Ausfallsicherheit, Flexibilität und Skalierbarkeit abgeleitet. HaRTKad nutzt hierfür P2P-Technologie und basiert konsistent auf UDP. Neben dem eingeführten softwarebasierten TDMA-Verfahren ist es ebenso möglich, mittels des vorgestellten KaDisSy-Algorithmus das Netzwerk komplett dezentral zu synchronisieren. Oberhalb von HaRTKad können weitere Applikationen folgen, die auf den übertragenen

Daten arbeiten können. Eine Darstellung ist in Abbildung 4.23 zu sehen. Mit Hilfe von HaRTKad lassen sich so erstmals in dieser Arbeit Systeme mit harter Echtzeit umsetzen. Zusammenhängende Daten sind jedoch noch auf die maximale Payload von UDP-Paketen (1500 Bytes) begrenzt, sodass weitere Mechanismen eingeführt werden müssen, um dieses Defizit zu beheben.

4.4. Kapitelzusammenfassung

In diesem Kapitel wurden die Grundlagen aufgezeigt, um ein System, basierend auf Kad, zu realisieren, welches für den Einsatz in der Automatisierung mit harter Echtzeit geeignet ist. Die Anforderungen an ein solches System wurden durch einen Überblick auf bestehende Lösungen abgeleitet. Es ist ersichtlich, dass derzeitige Systeme den hohen Anforderungen und Ansprüchen nicht ohne Änderungen gerecht werden. Das hier vorgestellte Verfahren, HaRTKad, realisiert ein Echtzeitsystem und die Echtzeitkommunikation mittels Kad. Zusätzlich kann auf Grund des verwendeten TDMA-Ansatzes auch die Synchronisation innerhalb von HaRTKad erfolgen. HaRTKad bietet somit ein eigenständiges System, das in Software realisiert ist und auf Standard-Protokollen basiert. Es weist durch die Verwendung von P2P-Technologie bereits einen Großteil der abgeleiteten und geforderten Eigenschaften auf. Ein Prototyp beweist bereits, dass ein Austausch von Daten unter 1 ms möglich ist. Als weitere Herausforderung bleibt die Begrenzung des Jitters, welcher zu untersuchen ist. Das System lässt sich bereits im Process Control-Umfeld einsetzen. Für einen sinnvollen Einsatz im Motion Control-Umfeld bzw. in isochroner Echtzeit sind weitere Verbesserungen durchzuführen, die als Optimierungen erwähnt wurden, um auch die Anzahl der Knoten steigern zu können. HaRTKad besitzt als Alleinstellungsmerkmal, dass es die erste rein softwarebasierte P2P-Realisierung im Umfeld der Automatisierung darstellt. Wie HaRTKad als Middleware dienen kann, wird im folgenden Kapitel 5 anhand von zwei Beispielen gezeigt.

Kapitel 5.

Harte Echtzeitapplikationen basierend auf HaRTKad

In diesem Kapitel sollen Applikationen vorgestellt werden, die in Verbindung mit HaRTKad genutzt werden können. Es werden zwei Applikationen vorgestellt, die harte Echtzeiteigenschaften besitzen oder durch HaRTKad zur harten Echtzeit gelangen, auch in Bezug auf die Kommunikation.

Die erste Applikation in Abschnitt 5.1 ist im Bereich der Web Services angesiedelt. Es wird eine Verbesserung des sogenannten WS-Eventing vorgestellt, welches zum Verteilen von Events an viele Teilnehmer gedacht ist. Zum einen wird ein neuer skalierbarer Ansatz vorgestellt und zum anderen eine geeignete Plattform, die harte Echtzeit erreicht. Mittels des HaRTKad-Ansatzes kann die maximale Größe und Anzahl der Eventingsysteme bestimmt werden, bei der auch eine deterministische Kommunikation zugesichert werden kann.

Die zweite Applikation ist eine Erweiterung des aus Unterkapitel 3.2 vorgestellten PSP-Auto-Systems, welches für Automatisierungsanforderungen angepasst wurde. Die PSP-Auto-Applikation, die in Abschnitt 3.2 für weiche Echtzeit definiert wird, wird um die zusätzliche Eigenschaft zur Übertragung von größeren Datenmengen erweitert. Das Gesamtsystem weist bezüglich der Plattform und der Kommunikation harte Echtzeitfähigkeiten auf.

5.1. Ein hochskalierbares echtzeitfähiges WS-Eventing

5.1.1. Einleitung

Web Services sind durch eine lose Kopplung von Geräten und die Verwendung von standardisierten Protokollen eine etablierte und verbreitete Art, Services in einer vernetzten Welt anzubieten (siehe Abschnitt 2.1.4). Als Enterprise-Lösung oder als externe API sind sie bereits bei Firmen wie Amazon oder Google stark akzeptiert [Ama] [Goo].

Mit der Idee des Internet der Dinge (IoT) wird die Anzahl der Geräte in den verschiedensten Netzwerken exponentiell ansteigen [Eva11]. IoT ist ebenfalls ein wichtiger Faktor in der Automatisierungsindustrie, wobei hier ein Echtzeitverhalten verlangt wird [EA12]. Dies umfasst ebenso Geräte mit limitierten Ressourcen wie eingebetteten Systemen. Mit dem ursprünglich 2006 veröffentlichten Devices Profile for Web Services (DPWS) von Microsoft können diese Geräte miteinander über einen gemeinsamen Standard kommunizieren und sind somit leicht in bestehende Netzwerke integrierbar. DPWS, welches im Jahre 2009 ein OASIS-Standard wurde, ist ein Verbund verschiedener Standards, welcher Web Services für eingebettete Systeme einheitlich definiert und nutzbar macht [OAS09a]. Ein Überblick über die Standards und die verwendeten Protokolle ist in Abbildung 5.1 zu sehen.

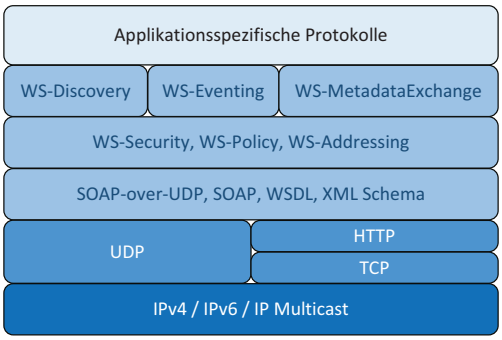

Abbildung 5.1.: Darstellung des DPWS-Stacks [ZMTG10].

Typische Standards sind WS-Discovery und WS-Eventing, welche essentiell zum Ausführen verschiedenster Operationen sind [OAS09c], [DMWC11].

WS-Discovery: WS-Discovery wird verwendet, um Services im Netzwerk zu finden, was mittels zwei verschiedener Modi geschehen kann - dem Managed-Modus oder dem

Ad-Hoc-Modus. Im Managed-Modus wird die Suche nach Services mittels eines Service Proxies gelöst. Der Service Proxy besitzt das Wissen über alle verfügbaren Services im Netzwerk und kann bei einer Anfrage direkt vermitteln. Der Ad-Hoc-Modus hingegen wird bevorzugt, da er keine zentrale Instanz benötigt, welche einen Single Point of Failure (SPoF) und Flaschenhals darstellt. Jedes Gerät agiert autonom, wenn es dem Netzwerk beitritt. Dies hat allerdings in Bezug auf die Skalierbarkeit Nachteile, wenn neue Geräte bzw. Services gesucht werden. Wenn ein Gerät mittels eines Multicasts nach allen Services eines Typs sucht, erhält es alle Antworten innerhalb eines kurzen Zeitfensters. Dies kann zu einer immensen Überlastsituation beim suchendem Gerät und somit zu Verwürfen von Antworten führen. Zusätzlich stellt dies auch ein Sicherheitsrisiko auf Grund von DDoS-Attacken dar. Ein Ansatz zur Lösung dieses Problems ist bereits in [ASD+14a] beschrieben und erfolgreich getestet worden. Die Lösung basiert auf der Verwendung eines DHT-basierten Discovery-Algorithmus. Der Algorithmus weist durch die aktive Kontrolle der Suchkommandos nach Services an einzelne Knoten eine sehr gute Skalierung auf und vermeidet damit die Nutzung von Multicast. Der DHT-basierte Algorithmus wurde mittels Kad als Softwaregrundlage realisiert. Basierend auf dieser Lösung sollen nach dem Finden von Services weitere Applikationen, die auch den Fokus auf Skalierbarkeit und Effizienz legen, realisiert werden. Eine Applikation ist ein Eventing-System, welches im Vorfeld das optimierte WS-Discovery [ASD+14a] zum Finden von Eventquellen nutzt. Ein Eventing-System basiert auf einem Publish-Subscribe-Mechanismus und bietet eine gute Möglichkeit der asynchronen Kommunikation.

WS-Eventing: Das WS-Eventing wird genutzt, um z.B. sogenannte Publish-Subscribe-Konzepte umzusetzen. Hierdurch kann auf ein Polling von z.B. Sensorwerten verzichtet werden. Zusätzlich erlaubt das Eventing einen hohen Grad an entkoppelten Geräten, was die Skalierbarkeit erhöhen kann sowie eine unabhängige Kommunikation zwischen den Geräten ermöglicht [EFGK03]. Es existieren zwei Instanzen, die in Interaktion zueinander stehen - Eventquellen und Eventsenken. Ein System basierend auf dem Publish-Subscribe-Konzept ist bereits im WS-Eventing-Standard beschrieben, hat aber einen großen Nachteil in Bezug auf die Skalierbarkeit, da nur ein Gerät die Event-Benachrichtigungen an alle bei ihm angemeldeten Knoten senden muss. Dies ist nicht für Echtzeitszenarien geeignet, weil die Benachrichtigungen über Events bei einer steigenden Anzahl an angemeldeten Geräten sehr spät eintreffen können. In diesem Beitrag wird ein neuer Ansatz präsentiert, um dieses Problem zu beheben und um ein Eventing-System

aufzubauen, das für hochskalierte Netzwerke geeignet ist. Dies gelingt durch einen neuen Benachrichtigungsalgorithmus, im Gegensatz zu dem im WS-Eventing-Standard verwendeten. Zusammengefasst sind die Hauptbeiträge folgende:

- Vorstellung eines Ansatzes für ein optimiertes Eventing-System.
- Eine prototypische Implementierung des vorgeschlagenen Ansatzes.
- Messergebnisse von einem experimentalen Aufbau.
- Vergleich des neuen Benachrichtigungsalgorithmus mit dem Standard-WS-Eventing-Verteilungsalgorithmus.

Die Hauptbeiträge dieses Unterkapitels sind in [SAD+14] publiziert. Der Rest ist wie folgt gegliedert: Abschnitt 5.1.2 beinhaltet die Beschreibung des Standes der Technik. In Abschnitt 5.1.3 wird ein kurzer Überblick über den WS-Eventing-Standard gegeben. Verschiedene Optimierungen und der neue skalierbare Benachrichtigungsalgorithmus werden in Abschnitt 5.1.4 beschrieben.

In Abschnitt 5.1.5 wird der Experimentalaufbau bestehen aus mehreren Prototypen vorgestellt. In den folgenden Abschnitten werden aufgenommene Messergebnissen aufgeführt, welche den Standard- und optimierten Benachrichtigungsalgorithmus zur Verteilung der Events abbilden.

5.1.2. Stand der Technik

Es gibt viele Ansätze, ein Eventing mittels des Publish-Subscribe-Ansatzes zu realisieren [EFGK03]. Diese Arbeit konzentriert sich auf das standardisierte WS-Eventing und listet relevante Arbeiten zur Verbesserung des WS-Eventing in Form von besserer Skalierbarkeit und Zuverlässigkeit auf. Diese Aspekte sind essentiell für den Einsatz in hochskalierten Netzwerken mit Echtzeitanforderungen, wie Automatisierungsszenarien.

In [TGK08] werden WS-Eventing und der Java Message Service (JMS) als eine Möglichkeit zur Realisierung eines Eventing-Systems im Umfeld der Automatisierung präsentiert. Der Java-basierte JMS erhöht dabei die Komplexität der Implementierung und den Overhead des Systems. Zusätzlich wird eine hierarchische/zentralisierte Struktur des JMS-basierten Publish/Subscrib-Systems verwendet, was im Widerspruch zum Gedanken des Ad-Hoc-Netzes und der Vermeidung eines SPoFs/Flaschenhalses steht. Als

zentrales Element stellt sich der Message Broker oder Application Server heraus. WS-Eventing als Alternative zu JMS wird wegen seiner geringen Skalierbarkeit auf Grund der alleinigen Benachrichtigung der Eventsenken durch die Eventquelle durch [TGK08] kritisiert. Im Gegenzug wird WS-Eventing als Teil von DPWS für seine einfache Integration durch die Standardisierung hervorgehoben. Zusätzlich wird das schnelle Verstehen und Einarbeiten in den WS-Eventing-Standard positiv erwähnt, was auf die Einfachheit und Modularität zurückzuführen ist.

Des Weiteren ist aus dem Beitrag nicht ersichtlich, welche Performance Geräte in diesem Umfeld haben müssen und ob die vorgeschlagenen Lösungen dann noch sinnvoll sind. Außerdem argumentierten die Autoren gegen die Verwendung von geteilten Medienkanälen und schlagen die Verwendung von proprietären Protokollen auf unteren Geräteebenen vor. Allerdings nutzt Industrial Ethernet ein geteiltes Medium und die Verwendung von proprietären Protokollen bedeutet auch, die gewünschte horizontale und vertikale Integration der Anlagen von der Geräte- bis zur Leitebene zu erschweren bzw. unmöglich zu machen [Sau05].

Das Problem der schlechten Skalierbarkeit von WS-Eventing wird auch in [Gre11] behandelt. Als Lösung wird hierbei UDP-basiertes Multicast verwendet. Dies hat allerdings den Nachteil, dass Multicast über alle Router und Switches hinweg unterstützt werden muss, was über große IP-basierte Netzwerke schwer zu realisieren ist. Zudem wird zu Lasten der Zuverlässigkeit auf Acknowledgements (ACKs) der Benachrichtigungen verzichtet, damit die Eventquelle nicht überlastet wird. Zusätzliche prototypische Ergebnisse sind leider nicht aufgezeigt worden. Im Gegensatz dazu nutzt der später in diesem Unterkapitel beschriebene Ansatz keine Multicasts, sondern basiert auf einfachen Unicast-Nachrichten. Damit kann die Lösung unbegrenzt verwendet werden, solange IP und UDP unterstützt werden. Zusätzlich weist die präsentierte Lösung eine höhere Zuverlässigkeit auf, da jede Benachrichtigung durch ein ACK abgesichert werden kann, ohne dabei die Eventquelle selbst zu überlasten.

Huang et al. stellen in [HG06] und [HSHG06] einen WS-Messenger vor, um eine bessere Skalierbarkeit zu erreichen. WS-Messenger unterstützt gleichzeitig WS-Eventing und den konkurrierenden Standard WS-Notification, welcher auch einen Publish-Subscribe-Mechanismus realisiert. Sie können zwar ein konventionelles WS-Notification-System schlagen, setzen aber trotzdem auf ein zentralisiertes System mit einem auf JMS-basierenden Broker. Auch die praktischen Ergebnisse zeigen eine nur lineare Skalierbar-

keit, was für zukünftige Anforderungen mit hoher Geräteanzahl nicht ausreichen wird. Zudem gibt es keine Aussage darüber, ob es auch im „embedded"-Bereich eingesetzt werden kann, geschweige denn in einem Echtzeitumfeld.

In [JGP10] werden aufbauend auf WS-Messenger Optimierungen vorgeschlagen, die direkt die Skalierbarkeit erhöhen sollen. Der vorgeschlagene Aspekt der Nachrichtenzustellung mit Hilfe von einzelnen Consumern basiert dabei auf der Idee, Jobs mit Queues zu verteilen, die die entsprechenden zu verteilenden Benachrichtigungen enthalten. Hierbei ist ein Management der Jobs durchzuführen, welcher einen zusätzlichen Overhead darstellt. Nach wie vor ist das vorgestellte System mit seinem Broker und somit mit SPoF sowie Flaschenhals ein zentralisiertes System, das in Ad-Hoc-Netzwerken zu vermeiden gilt.

Der hier vorgestellte generalisierte Ansatz benötigt keine zentralisierte Verwaltung der Benachrichtigungen. Die Verwendung von Message Brokern oder anderen zentralen Elementen ist nicht nötig, was der Skalierbarkeit und Zuverlässigkeit zu Gute kommt. Zusätzlich weist keines der vorgestellten Systeme, im Gegensatz zu dem in dieser Arbeit präsentierten System, das Potential auf, auch in Umgebungen mit harter Echtzeit agieren zu können, da viele Aspekte wie die Zielplattform oder der Medienzugriff nicht geklärt sind. Der später vorgestellte Prototyp erreicht aufgrund seiner hohen Performance bereits Anforderungen für Echtzeitapplikationen mit Anforderungen im niedrigen Millisekundenbereich.

5.1.3. Grundlagen

Bevor WS-Eventing genutzt werden kann, muss eine entsprechende Eventquelle im Netzwerk gefunden werden. Dies wird mittels WS-Discovery durchgeführt. Dabei wurde eine optimierte und hochskalierbare Version des WS-Discovery genutzt, welche auf dem bereits erwähnten DHT-basierten Discovery-Algorithmus basiert [ASD+14a]. Die verbesserte Version verwendet Kad als P2P-Netzwerk, um in der Lage zu sein, alle Services mittels UDP Unicast-Nachrichten in Ad-Hoc-Netzwerken zu finden. Durch den Verzicht auf TCP und Multicast kann ein Determinismus erreicht werden, wodurch das Discovery selbst bereits für das Umfeld der Automatisierung mit Echtzeitanforderungen qualifiziert ist. Weitere Applikationen, die darauf aufbauen, wie z.B. ein WS-Eventing-System, profitieren von diesen Vorbedingungen. Deshalb wird dieses bestehende Framework genutzt,

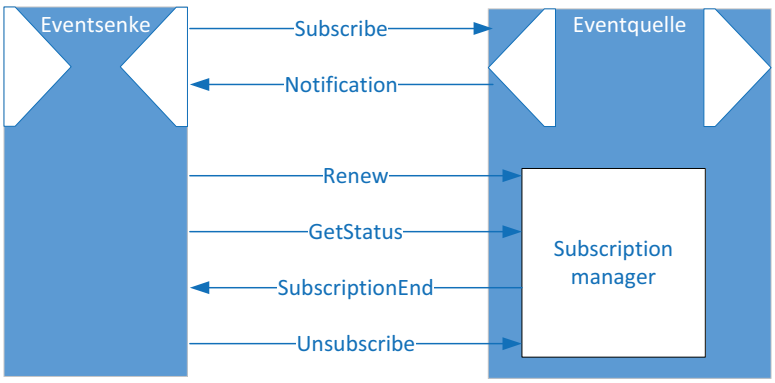

Abbildung 5.2.: WS-Eventing-Operationen.

um ein hochskalierbares WS-Eventing zu realisieren, nachdem eventuelle Eventquellen mittels des Discovery gefunden wurden. Es existieren verschiedene Operationen, um mit der gefundenen Eventquelle zu interagieren, welche in Abbildung 5.2 dargestellt sind.

Um Benachrichtigungen von einer Eventquelle zu erhalten, muss sich ein potentieller Konsument bei der Eventquelle anmelden (engl. Subscribe) und wird dadurch zu einer Eventsenke. Die Eventquelle besteht aus einem Subscription Manager-Objekt, welches die angemeldeten Eventsenken speichert und verwaltet. Um den Status der Anmeldung zu managen, kann die Eventsenke Renew-Kommandos an die Eventquelle senden, da die Anmeldung einen Time-to-Live-Wert besitzt. Zusätzlich kann die Eventsenke den Subscription Manager nach dem aktuellen Status der Anmeldung mittels einer GetStatus-Operation befragen. Ist eine Eventsenke nicht mehr an Benachrichtigungen der Eventquelle interessiert, führt diese eine Unsubscribe-Operation durch. Sollte eine Anmeldung unerwartet unterbrochen werden, kann der Subscription Manager eine SubscriptionEnd-Nachricht an die betreffenden Eventsenken senden. Alle die bisher genannten Nachrichten bzw. Operationen sind nicht zeitkritisch, da sie der Verwaltung dienen.

Soll ein Event an die angemeldeten Eventsenken verteilt werden, versendet die Eventquelle die Benachrichtigungen seriell (siehe Abbildung 5.3). Da dies seriell geschieht, skaliert die Verteilung im besten Fall nur linear mit der Anzahl der angemeldeten Eventsenken. Sollte eine hohe Anzahl an Eventsenken angemeldet sein, kann die Eventquelle überlastet werden, sodass ihre Funktionalität nicht gewährleistet werden kann. Benachrichtigungen über Events könnten somit nur mit sehr hohen Zeitverzögerungen zugestellt werden.

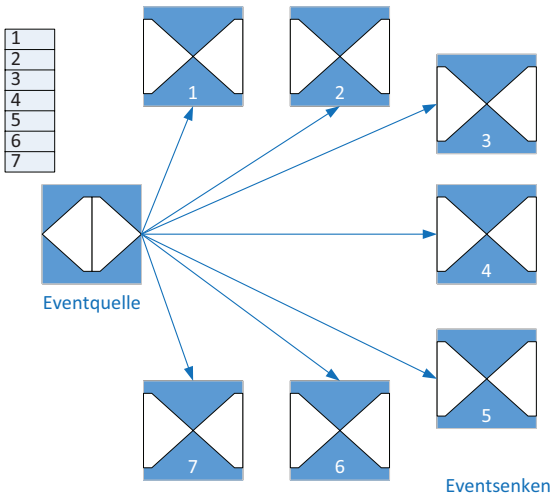

Abbildung 5.3.: WS-Eventing: Standard-Benachrichtigung der Eventsenken.

Da die Benachrichtigungen aber eine zeitkritische Information in einem Echtzeitszenario darstellen, muss diese innerhalb einer spezifizierten Zeit versendet sein. Es kann natürlich ein Zeitraum gewählt werden, der bei einer linearen Skalierbarkeit groß genug wäre, was allerdings zu enorm großen Zeitintervallen führen und somit die Einsatzbereiche stark einschränken würde. Daher ist der Standard-Benachrichtigungsmechanismus nicht für Echtzeitapplikationen im Umfeld mit vielen Geräten geeignet. Deshalb wurde ein alternativer Ansatz entwickelt, welcher eine hohe Skalierbarkeit und Zuverlässigkeit aufzeigt.

5.1.4. Optimierungen

In diesem Abschnitt werden drei Optimierungen bzgl. des WS-Eventing vorgeschlagen. Die erste betrifft das Optimieren der Benachrichtigung durch eine parallelisierte Verteilung mit Hilfe von involvierten Eventsenken. Der zweite Aspekt bezieht sich auf die Verwendung von XML-Kompression, um die Effektivität der Datenverarbeitung zu steigern und einen geringeren Overhead zu erhalten. Abschließend wird die Möglichkeit der Modifizierung der Reihenfolge der Anmeldeliste im Subscription Manager diskutiert und wie dies geschehen sollte.

5.1.4.1. Parallelisierter Benachrichtigungsmechanismus

Um das Problem der geringen Skalierbarkeit zu lösen, wird ein neuer Mechanismus zum Benachrichtigen vorgestellt. Anstatt nur die Eventquelle zum Verteilen zu nutzen, werden nun Eventsenken, die beim Verteilen der Benachrichtigungen helfen, akquiriert. Bei den gewählten Geräten handelt es sich gewöhnlich um Eventsenken, welche auch für dasselbe Event angemeldet sind. Um kompatibel mit dem WS-Eventing-Standard zu bleiben, wurde XML für die Beschreibung der Payload in den Nachrichten verwendet. Die Anmelde- und Abmeldemethode ist dieselbe wie beim Standard-WS-Eventing-Prozess. Wenn ein Gerät, welches eine Eventquelle ist, ein Event erzeugt, hat es eine Liste mit allen Eventsenken, die sich für dieses Event angemeldet haben. Die Eventquelle kontaktiert nun den ersten Knoten und sendet ihm die erste Hälfte der Liste mit weiteren Kontakten. In einem zweiten Schritt wird die zweite Eventsenke kontaktiert, welche die zweite Hälfte der Liste erhält. Nachdem diese beiden Schritte durchgeführt sind und sie ein ACK von den beiden kontaktierten Eventsenken erhalten hat, geht die Eventquelle bereits in den Ruhezustand. Die ACKs sichern die Zuverlässigkeit ab, da die Kommunikation nur mittels UDP stattfindet. Die zwei Eventsenken fahren anhand ihrer erhaltenen Liste genauso wie die Eventquelle fort. Ein Beispiel ist in Abbildung 5.4 zu sehen. Die sich nun aufbauende Verteilungsstruktur wird als Benachrichtigungsbaum bezeichnet.

Die neue Verteilung der Benachrichtigungen führt eine Beschleunigung herbei, die theoretisch zu einer logarithmischen Geschwindigkeit führt. Unter Verwendung der logarithmischen Funktion 5.1 müssen die Parameter a und b bestimmt werden, um eine theoretische Vorhersage über die Verteilungsperformance T_{Dist} für die Zeit zum Verteilen eines Events durchführen zu können. N bestimmt die Anzahl der Instanzen (einschließlich der Eventquelle).

$$T_{Dist} = a * \overbrace{\lfloor log_B(N) \rfloor}^{(E)} \tag{5.1}$$

Ein Knoten wird immer nur von maximal zwei Folgeknoten belastet. Als theoretische Schlussfolgerung muss mit jeder Verdopplung der Anzahl an Eventsenken nur ein weiterer Zeitschritt mit einkalkuliert werden. Daher ist die Basis $B = 2$. Erst wenn eine weitere Ebene im Benachrichtigungsbaum aufgebaut wird, wird mehr Zeit für die Benachrichtigungen benötigt. Daher lässt sich die benötigte Zeit auch mittels der Ebenen

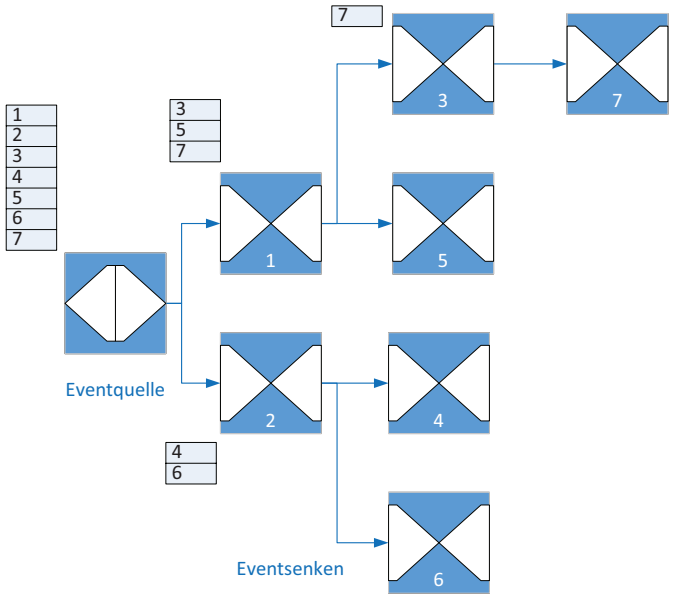

Abbildung 5.4.: WS-Eventing: Optimiertes Verteilen der Benachrichtigungen.

E bestimmen. Dabei gilt die Eventsenke als Wurzel und noch nicht als Ebene. Der Parameter a bestimmt die Zeit, die benötigt wird, um jeweils die nachfolgenden zwei Knoten zu benachrichtigen. a kann direkt bestimmt werden, wenn die Zeit für das Verteilen von Events von der Eventquelle zu den ersten beiden Eventsenken bekannt ist.

5.1.4.2. Verwendung von Efficient XML Interchange

Da XML menschenlesbar ist, ist der Umgang aus Sicht des Nutzers stark erleichtert. Dies geschieht aber auf Kosten eines hohen Overheads. Weil die angestrebte Lösung jedoch eine hohe Effektivität des Datenaustausches anstrebt, wird eine Optimierung bzgl. der Daten nötig. Um die übertragenen Daten minimal zu halten, wurde sich entschieden, Efficient XML Interchange (EXI) zu verwenden [SKPK14]. EXI ist ein binärkodiertes XML-Format, womit hohe verlustfreie Datenkompressionsraten erreicht werden können. Wie vorherige Arbeiten zeigen, kann die Verwendung von EXI den Datenoverhead reduzieren, wodurch die Effektivität der Datenübertragung gesteigert werden konnte [ASGT12]. Da Kad als Hintergrundsystem gewählt wurde, um das DHT-basierte Discovery zu realisieren [ASD+14a], sind die Discovery-Daten in den Kad-Paketen als

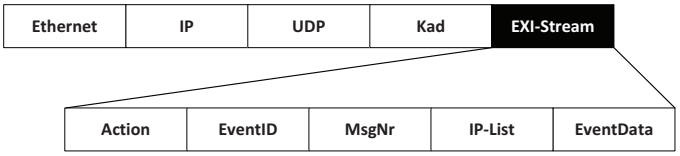

Abbildung 5.5.: Benachrichtigungspaket.

Payload integriert. Dies gilt auch für die EXI-kodierten Daten von WS-Eventing. Alle Kad-Pakete sind UDP-Pakete und eignen sich für die Realisierung von Echtzeitapplikationen. Der Aufbau eines Beispiel-Benachrichtigungspaketes ist in Abbildung 5.5 dargestellt.

Der EXI-Stream als Payload des Kad-Paketes besteht je nach Nachrichtentyp aus verschiedenen Elementen, um WS-Eventing zu realisieren. Diese Elemente sind in Tabelle 5.1 aufgeführt.

Element	Beschreibung
Action	Enthält Nachrichtentyp
EventID	ID zur eindeutigen Zuordnung zum Event
ReplyTo	IP-Zieladresse für An/Abmelde-Bestätigungen
MsgNr	Nummer der Nachricht
RelatesTo	Nummer, zu der die Nachricht gehört
Sink-IP	IP-Adresse einer Eventsenke in einer Anmeldenachricht
Subscription-IP	IP-Adresse des Subcription Managers
Subscription-ID	ID der Anmeldung
IP-List	Beinhaltet die IP-Eventsenkenadressen für die Verteilung
EventData	Payload mit Sensorwerten

Tabelle 5.1.: EXI Elemente.

Durch das neue Konzept und der aufgeführten Elemente ist es möglich, WS-Eventing mit einer hohen Performance in limitierten Umgebungen, wie z.B. in eingebetteten Systemen in Automatisierungsszenarien, einzusetzen. Dies gelingt durch das zusätzliche und neue IP-Listenelement *IP-List*.

5.1.4.3. Sortierung der Eventsenken

Durch die Verwendung der internen Liste des Subscription Managers ist es möglich, einen Einfluss auf die Verteilung der Benachrichtigung eines Events zu haben. Eine Strategie könnte die Sortierung der Liste derart sein, dass Geräte mit hohen Zeitanforderungen die Benachrichtigungen bevorzugt als erstes erhalten. Alternativ wäre ein auf Zuverlässigkeit optimierter Ansatz vorteilhaft, bei dem die Liste mit Eventsenken nach deren Zuverlässigkeit sortiert wird. Sollte eine Eventsenke ausfallen, muss die vorherige Instanz den nächsten Knoten aus der Liste der Eventsenken fragen. Die nun angefragte Eventsenke ist dann zuständig für die Benachrichtigung der anderen Eventsenken der Liste. In diesem Falle sollten Geräte mit einer hohen Zuverlässigkeit und damit hohen Robustheit die Benachrichtigung zuerst erhalten. Dies ist wichtig, da die Wahrscheinlichkeit, dass ein Gerät am Anfang der Benachrichtigungskette und somit nahe der Wurzel des Benachrichtigungsbaums ausfällt, sehr gering ist. Damit können hohe Delays auf eine große Submenge an Eventsenken durch Timeouts vermieden werden. Ein Gerät, das später und demnach auf einer tieferen Ebene des Benachrichtigungsbaums ausfällt, hat einen geringeren negativen Einfluss, da es im schlimmsten Fall für weniger Knoten indirekt durch seine weiterzugebende Liste/Benachrichtigung zuständig ist. Auch eine höhere Belastung der Eventquelle ist damit unwahrscheinlicher. Aus diesem Grund wird die zweite Strategie zum Sortieren der Liste empfohlen.

5.1.5. Implementation und Evaluation

Als Zielplattform wird das ZedBoard verwendet. Der Prototyp basiert im Wesentlichen auf dem in Abschnitt 4.3.3 vorgestellten Prototypen.

Die EXI-Verarbeitung erfolgt beim Prototypen statisch, da er weiß, wo er im EXI-Paket seine Daten einzutragen hat. Es stellt sich zusätzlich die Frage nach der Verarbeitung der EXI-Daten, wenn diese ebenfalls geparst werden müssen. Diesem Thema widmet sich die Arbeit [ASD+14b], welche die Verarbeitung in Echtzeit mittels Software oder Hardware-Software-Co-Design untersucht. Somit ist es möglich, die EXI-Daten selbst mit hoher Geschwindigkeit und Determinismus zu verarbeiten. Für weitere Details wird auf die Veröffentlichung selbst verwiesen.

Die Applikation umfasst den Kad-Client, den DHT-basierten Discovery-Algorithmus

und das umgesetzte optimierte WS-Eventing. Zusätzlich wurde der Standard-Benachrichtigungsalgorithmus umgesetzt, um später direkte Vergleiche durchführen zu können. Der Experimentalaufbau besteht aus 19 Geräten, die mittels Gigabit-Switches miteinander verbunden sind. Zuerst startet die Eventquelle und wartet auf Anmeldenachrichten. Nachdem die Eventsenken gestartet haben, melden sie sich automatisch bei der Eventsenke an. Diese Schritte sind nicht zeitkritisch und werden daher nicht zeitlich gemessen. Wenn alle 18 Eventsenken sich bei der Eventquelle angemeldet haben, warten sie auf die Benachrichtigung. Nachrichten, die über einen seriellen COM-Port ausgegeben werden können, werden erst nach den kompletten Messungen dargestellt, da derartige Ausgaben das Ergebnis stark verfälschen. Mittels eines PCs wird ein Trigger (UDP-Paket) für die Eventquelle erzeugt, welche nun das Senden der Benachrichtigungen an die Instanzen, angemeldet beim Subscription Manager, beginnt. In diesem Augenblick wird der erste Zeitstempel $t_{trigger}$ genommen. Die Zeitauflösung beträgt 1 μs, was der höchsten unterstützten Auflösung im Betriebssystem FreeRTOS auf dem ZedBoard entspricht. Nun wird in Abhängigkeit vom Benachrichtigungsalgorithmus, welcher durch den Trigger des PCs vorgegeben ist, die Benachrichtigung durchgeführt.

Zwei Szenarien werden für den Standard- und den optimierten Benachrichtigungsalgorithmus evaluiert. Der Unterschied liegt im Grade der Zuverlässigkeit. Im ersten Fall wird kein ACK von der Eventsenke erwartet. So ist eine bessere Performance zu erwarten, da die Eventquelle nicht von jeder Eventsenke eine Antwort erhält und diese verarbeiten muss. Dies geschieht allerdings zu Lasten der Zuverlässigkeit. Im zweiten Szenario werden ACKs von den Eventsenken erwartet, sodass ein höherer Grad an Zuverlässigkeit erreicht wird. Vollständigkeitshalber werden beide Szenarien untersucht und verglichen. Bei allen Nachrichten handelt es sich um UDP-Unicast-Nachrichten.

5.1.6. Szenario Eins: Keine ACKs

Nachdem der Trigger von der Eventquelle empfangen wurde, informiert diese mittels des Standard-Benachrichtigungsalgorithmus die angemeldeten Eventsenken. Allerdings geschieht dies seriell. Nur die letzte Eventsenke wird eine ACK-Nachricht an die Eventquelle senden. Der zweite Zeitstempel t_{last} wird genommen, wenn die Eventquelle die ACK-Nachricht entgegennimmt.

Bevor die Ergebnisse für den optimierten Benachrichtigungsalgorithmus aufgenommen

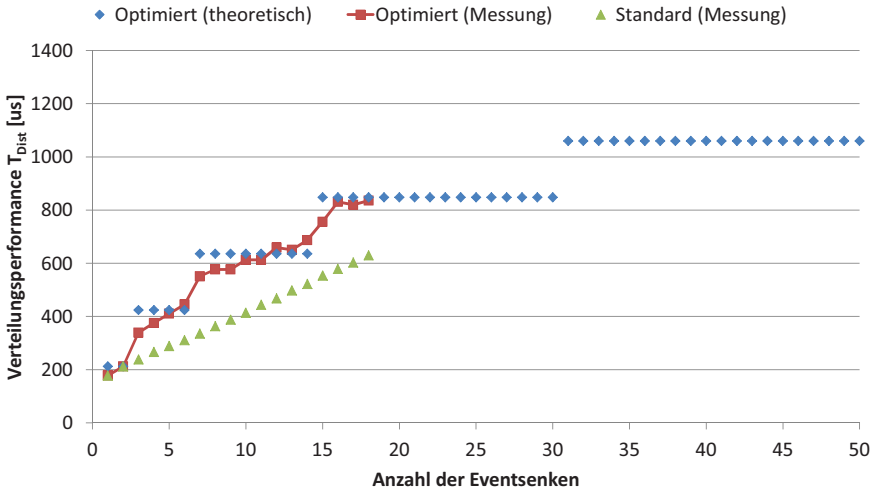

Abbildung 5.6.: Szenario Eins: Ergebnisse ohne ACKs.

wurden, wurde im Vorfeld der Parameter a aus Formel 5.1 für den optimierten Ansatz ermittelt. Hierzu wird zuerst die Zeit gemessen, die benötigt wird, um von der Eventquelle zwei Eventsenken die Benachrichtigung zukommen zu lassen. Der Wert für a ist bei diesem Szenario 212 μs. Die resultierende theoretische Funktion wird zum Vergleich mit den gemessenen Werten in Abbildung 5.6 genutzt. Der optimierte Benachrichtigungsalgorithmus arbeitet ähnlich, nutzt aber zur Beschleunigung des Benachrichtigungsprozesses andere Eventsenken. Auch hier sendet die letzte Eventsenke ein ACK an die Eventquelle, woraufhin der zweite Zeitstempel t_{last} genommen wird. Die letzte Eventsenke ist die Senke, welche sich auf der tiefsten Ebene und dort am weitesten rechts im Benachrichtigungsbaum befindet. Die Differenz zwischen $t_{trigger}$ und t_{last} ergibt die Verteilungsperformance der Benachrichtigung T_{Dist}. Das Ergebnis des Experimentalaufbaus ist in Abbildung 5.6 zu sehen.

Es wird ersichtlich, dass ein einzelner Knoten immer noch eine bessere Performance aufweist, als wenn er helfende Knoten hinzuziehen würde. Die gemessenen Werte für 18 Knoten stimmen nahezu mit der zuvor theoretisch ermittelten Performancevorhersage überein. Es ist erkennbar, dass nach der Verdopplung der Eventsenken auch ein eindeutig stärkerer Anstieg der benötigten Zeit für die Benachrichtigung erkennbar ist. Nachdem die ersten beiden Eventsenken die Benachrichtigung erhalten haben, ist ein signifikanter Anstieg zu erkennen, bis die nächste Eventsenke antwortet. Der nächste erkennbare

Zeitsprung ist deutlich, nachdem weitere vier Eventsenken benachrichtigt wurden, was der nächsten Ebene des Benachrichtigungsbaumes entspricht. Der letzte erkennbare signifikante Anstieg ist nach weiteren acht Eventsenken erkennbar.

Mit jeder Ebene des Benachrichtigungsbaumes ist somit ein größerer signifikanter Zeitsprung ersichtlich, was immer dann der Fall ist, wenn die Anzahl der benachrichtigten Eventsenken verdoppelt wird. Die benötigte Zeit zwischen Eventsenken auf derselben Ebene ist deutlich geringer. Der serielle Standard-Benachrichtigungsalgorithmus hingegen zeigt einen linearen Verlauf. Aus diesem Grund wird der optimierte Benachrichtigungsalgorithmus eine bessere Performance für eine größere Anzahl an Eventsenken aufweisen. Der neue parallelisierte Ansatz ist demnach nicht besser als der Standardansatz, wenn nur wenige Eventsenken verwendet werden, was auf die Komplexität der neuen Funktionalität zurückzuführen ist. Approximiert man die lineare Funktion des Standardansatzes und bildet den Schnittpunkt mit der theoretisch ermittelten Funktion für den optimierten Ansatz, ergibt sich eine Eventsenkenanzahl von 28. Ab ca. 28 Eventsenken ist der optimierte Ansatz in der Performance besser als der Standardansatz. In Anbetracht der Vorgabe einer steigenden Anzahl an Geräten und somit potentiellen Eventsenken ist der neue Ansatz im Vorteil und kann zur Verbesserung der Benachrichtigung eingesetzt werden. Beide Algorithmen erreichen eine Benachrichtigungsperformance von weniger als einer Millisekunde und sind damit für Automatisierungsszenarien geeignet, und dies mit harter Echtzeit auf Grund der gewählten Plattform und des Betriebssystems.

5.1.7. Szenario Zwei: Mit ACKs

Da nur UDP-Nachrichten für den Ansatz verwendet werden, um ein deterministisches Verhalten für Echtzeitszenarien zu ermöglichen, muss trotzdem die Zuverlässigkeit sichergestellt werden.

Folglich wurden auch die Ergebnisse aufgenommen, bei denen eine Eventsenke immer auch mit einer ACK-Nachricht den Erhalt der Benachrichtigung bestätigen muss. In der Standardversion sendet die Eventquelle die Benachrichtigungen seriell und erhält auch die ACKs seriell. Sie wartet allerdings nicht auf die ACKs und kontaktiert dann erst die nächste Eventsenke, sondern sendet sofort an die nächste Eventsenke weiter. Wenn die Eventquelle die letzte ACK-Nachricht erhält, wird auch hier der zweite Zeitstempel t_{last}

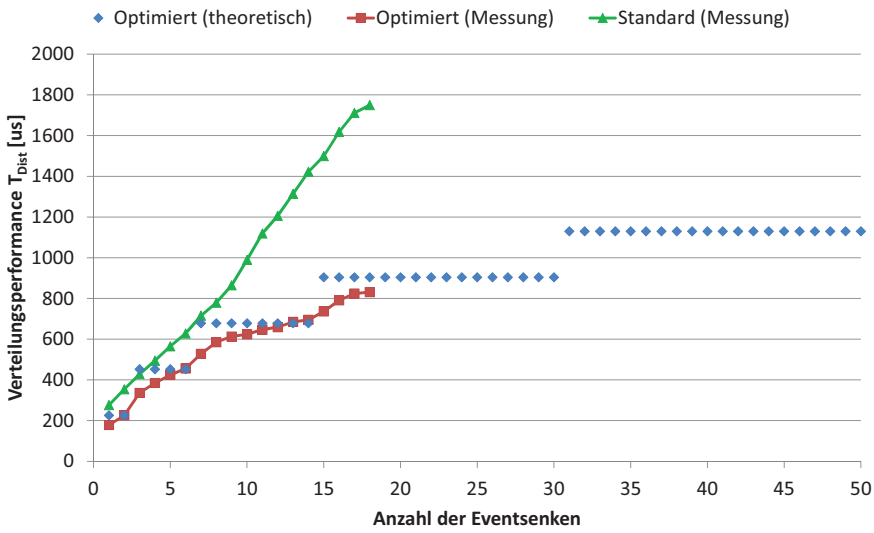

Abbildung 5.7.: Szenario Zwei: Ergebnisse mit ACKs.

genommen und die Verteilungsperformance T_{Dist} bestimmt.

Auch hier wurde im Vorfeld der Bestimmung der Performance des optimierten Benachrichtungsalgorithmus der Faktor a aus Formel 5.1 ermittelt. Es ergibt sich ein Wert von 226 μs, welcher zur Ermittlung der theoretischen Funktion, die zum Vergleich mit den Messwerten genutzt wird, dient. Bei dem optimierten Algorithmus antworten nur die letzten beiden Eventsenken der Eventquelle, damit T_{Dist} bestimmt werden kann. Die anderen Eventsenken senden ihre ACK-Nachrichten nur an ihren direkten Vorgänger, welcher ihnen im Vorfeld die Benachrichtigung zukommen ließ. Die Ergebnisse für T_{Dist} sind in Abbildung 5.7 zu sehen.

Hierbei wird noch deutlicher, dass der neue Ansatz besser als der Standardansatz skaliert, was bereits im vorherigen Szenario (für eine höhere Anzahl an Eventsenken) ersichtlich war. Auch stimmen die Werte von den 18 Eventsenken aus dem prototypischen Szenario mit der theoretisch ermittelten Funktion überein. Erkennbar wird auch, dass der optimierte Benachrichtigungsalgorithmus dem Standard von Anfang an überlegen ist. Steht die Zuverlässigkeit im Fokus, ist der neue Ansatz eine sehr gute Alternative, da er bereits für eine geringe Anzahl an Eventsenken sehr effizient ist. Vergleicht man die Ergebnisse für den optimierten Benachrichtigungsalgorithmus mit und ohne ACKs der Eventsenken, erkennt man, dass die ACKs kaum einen Einfluss haben. Die benötigte Zeit T_{Dist} ist bei

den Messungen bis 18 Knoten unter einer Millisekunde. Im Gegensatz dazu haben ACKs einen signifikanten negativen Einfluss auf den Standard-Benachrichtigungsalgorithmus, da die Eventquelle jeden ACK verarbeiten muss. Der optimierte Ansatz ist für 18 Eventsenken bereits um 52,51 % besser als der Standardansatz und besitzt eine logarithmische Tendenz, was ihn für den Einsatz in hochskalierten Netzwerken prädestiniert.

5.1.8. Realisierung mittels HaRTKad

Es ist sinnvoll, das WS-Eventing-System mit HaRTKad zu verwenden. Der größte Vorteil ist, dass HaRTKad auf derselben Plattform wie das WS-Eventing-System basiert - Kad als Implementierung von Kademlia. HaRTKad ermöglicht es außerdem, mehr als eine Eventquelle zu verwenden und trotzdem die harten Echtzeitbedingungen zu garantieren, weil die Kommunikation der Eventquellen durch HaRTKad gesteuert wird. Mit Hilfe ihres Hashwertes können die Eventquellen den Zeitpunkt einer zu sendenden Benachrichtigung auf dem Medium autonom ermitteln. Eine Suche des Empfängers mittels Kad hingegen ist nicht nötig, da die Eventsenken bekannt sind.

Wird ein Netzwerk mit 200 Eventsenken pro Eventquelle angenommen, muss man die Datenmengen, die daraus resultieren, betrachten, um eine Aussage über die Performance innerhalb von HaRTKad treffen zu können. Es wird sich dabei auf das Szenario konzentriert, bei dem alle Eventsenken auch eine Bestätigung auf eine Benachrichtigung senden müssen. Zudem wird aus Performancegründen direkt der optimierte Benachrichtungsalgortihmus vorausgesetzt. Wichtig ist im Vorfeld zu wissen, wie viel Zeit benötigt wird, um 200 Eventsenken zu benachrichtigen. Für 200 Eventsenken wird eine Benachrichtungszeit T_{Dist} von ca. 1,6 ms benötigt, wenn man die theoretische Performance aus Abbildung 5.7 als Grundlage wählt. Daher eignet sich das System für die Process-Klasse mit $T_{Cyc} = 10\ ms$.

Daraufhin muss sich die Frage gestellt werden, wie viele Eventquellen im Durchschnitt verwendet werden können, ohne eine Überlastsituation zu erzeugen. Hierzu muss man die Gesamtdatenmenge betrachten, welche das 1-GBit/s-Medium von im Durchschnitt für einen Zyklus nicht überschreiten darf.

Bestimmung der übertragenen Datenmenge: Die Gesamtdatenmenge hängt von der Anzahl der Benachrichtigungen und von den ACKs ab. Die Datenmenge, die durch

die Benachrichtigung entsteht, ist hierbei gesondert zu berechnen, da zusätzliche Teillisten versendet werden müssen. Zuerst wird die Anzahl der vollbesetzten Ebenen E_{Voll}, die der Benachrichtigungsbaum hat, bestimmt (siehe Formel 5.2).

$$E_{Voll} = \lfloor log_2(N+1) - 1 \rfloor \tag{5.2}$$

Formel 5.3 gibt folglich die Anzahl der Kontakte an, die per Liste übertragen werden müssen.

$$LE = N(E_{Voll} - 1) - \sum_{i=1}^{E_{Voll}-1} 2^i (E_{Voll} - 1) \tag{5.3}$$

Formel 5.4 repräsentiert das Ergebnis ohne Summenformel und kann direkt angewendet werden, um die Anzahl der zu übertragenen Listenelemente LE zu bestimmen.

$$LE = N(E_{Voll} - 1) - 2(-E_{Voll} + 2^{E_{Voll}} - 1) \tag{5.4}$$

Das Beispiel in Abbildung 5.4 besteht aus vier Ebenen. Die Summe der übertragenen Listenelemente, wobei ein Listenelement eine Adresse darstellt, ist in der Summe 6. Von der Eventquelle müssen fünf Listenelemente an die zweite Ebene übertragen werden, in der sich Knoten 1 und 2 befinden. Zur dritten Ebene muss nur noch ein Listenelement an Knoten 3 übertragen werden.

Für das vorher beschriebene Beispiel mit 200 Knoten ergibt sich eine zu übertragene Anzahl an Listenelemente von $LE = 891$. Jedes LE stellt eine 4-Byte-IP-Adresse dar. Diese Daten werden zusätzlich zu den 200 Benachrichtungspaketen (88 Byte) und ACKs-Paketen (64 Byte) gesendet. Die Datenmenge, die dabei entsteht, beträgt 33.964 Byte.

Bestimmung maximal unterstützter Eventquellen: Die Datenmenge 33.964 Byte und die prognostizierte Zeit von ca. 1,6 ms ergeben eine Datenrate $R_{Eventquelle}$ von 169,82 MBit/s. Als Grundlage dient die theoretisch ermittelte Funktion für den optimierten Verteilungsansatz mit ACKs. Berücksichtigt man die Zykluszeit $T_{Cyc} = 10\ ms$ und die maximale Datenrate von $R_{Ethernet}$ von 1-GBit/s, können 36 Eventquellen jeweils

200 Eventsenken benachrichtigen und halten dabei die harte Echtzeit ein. Die Berechnung der Knoten ist in Formel 5.5 zu sehen.

$$Eventquellen = \frac{R_{Ethernet}}{R_{Eventquelle}} * \frac{T_{Cyc}}{T_{Dist}} \tag{5.5}$$

Betrachtet man die nächsthöhere Human-Klasse mit $T_{Cyc} = 100\ ms$, könnten 368 Eventquellen mit je 200 Eventsenken unterstützt werden.

5.1.9. Fazit

In diesem Abschnitt wurde ein optimierter Algorithmus zum verteilten Benachrichtigen in WS-Eventing vorgestellt. Der serielle Standard-Benachrichtigungsalgorithmus überfordert die Eventquelle in Ad-Hoc-Netzwerken bei einer großen Anzahl an Eventsenken. Der neue Algorithmus hingegen nutzt die Eventsenken, um die Benachrichtigungen parallel mit einer Geschwindigkeit in logarithmischer Abhängigkeit von der Anzahl der Eventsenken zu verteilen, wodurch die Eventquelle stark entlastet wird. Der neue Ansatz hat eine hohe Skalierbarkeit und ermöglicht die Benachrichtigung von Knoten mit einer hohen Performance.

Der vorgestellte Prototyp zeigt eine hohe Performance und beweist, dass eine Benachrichtigung von vielen Eventsenken mit einigen wenigen Millisekunden erreichbar ist. Zusammen mit dem erwähnten DHT-basierten Discovery-Algorithmus, welcher auch potentielle Echtzeitfähigkeit besitzt, kann das System sogar Web Services in Automatisierungsszenarien ermöglichen, welche das WS-Eventing umsetzen. Als Ausblick soll die praktische Verwendung von mehreren Eventsenken untersucht werden, was zu Überlastsituationen führen kann und damit zu Paketverlusten. Zusätzlich wurde beschrieben, wie mehrere Eventquellen unterstützt werden, indem der Medienzugriff durch das in Kapitel 4 vorgestellte HaRTKad-Verfahren gesteuert wird. Die Kombination aus dem optimierten WS-Eventing und HaRTKad ergibt ein System mit harter Echtzeit, um effizient ein Benachrichtigungssystem mit einer hohen Skalierbarkeit zu erstellen. Es lässt sich bereits sinnvoll in der Process-Klasse (Zykluszeit = 10 ms) sowie in der Human-Klasse (Zykluszeit = 100 ms) einsetzen. Weitere Verbesserungen von HaRTKad in zukünftigen Arbeiten sollten sich direkt positiv auf die Anzahl der unterstützten Eventquellen und Eventsenken auswirken.

Einordnung in den Gesamtkontext der Arbeit: Das durch Verteilung optimierte Eventing stellt die erste Applikation oberhalb von HaRTKad dar (siehe Abbildung 5.8). Das System basiert weiterhin komplett auf UDP und durch die Verwendung von HaRTKad und einer geeigneten Plattform besitzt das System harte Echtzeiteigenschaften. Innerhalb des Eventing-Systems lassen sich weiterhin nur zusammenhängende Daten übertragen, die als Payload in ein UDP-Paket integrierbar sind. Auf Grund der geringen zu übertragenen Datengrößen ist dies jedoch möglich. Bei steigenden Datengrößen lassen sich diese jedoch aufteilen, da die Daten vorwiegend aus einzelnen Adressen der Senken bestehen und damit nicht zusammenhängend sind. Oberhalb des verteilten Eventings können beliebige weitere Anwendungen folgen, die auf den ausgetauschten Daten arbeiten.

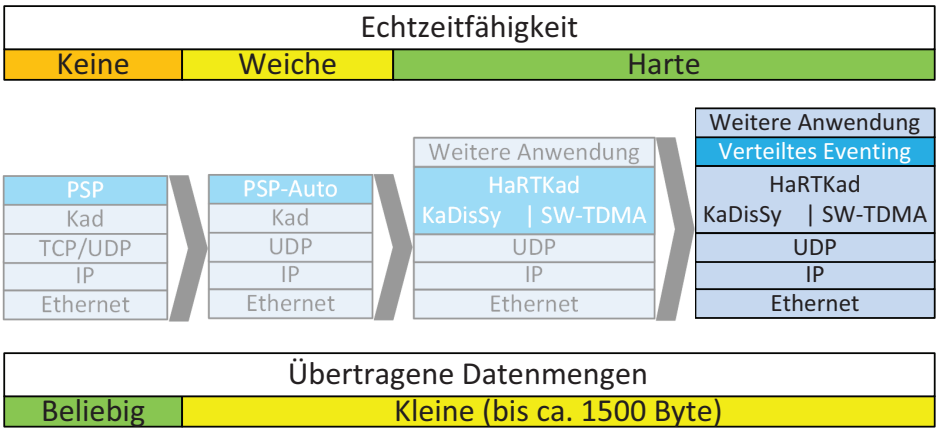

Abbildung 5.8.: Darstellung des Protokollstacks des verteilten Eventing-Verfahrens und Einordnung in den Gesamtkontext.

5.2. CoHaRT - Ein System zur deterministischen Übertragung großer Datenmengen mittels CoAP und HaRTKad

5.2.1. Einleitung

In diesem Abschnitt wird eine zweite Applikation, welche das HaRTKad-System als Middleware nutzt, vorgestellt. Eine Herausforderung in der Automatisierung mit hohen Echtzeitanforderungen sind hohe Datenmengen. Derzeitig sind vorrangig kleine Datenmengen von wenigen Bytes zu übertragen. Jedoch sind die Anforderungen in der heutigen Zeit stark gestiegen, z.B. im Automobilbereich, wo ein Bedarf an deterministischer Übertragung auch von großen Datenmengen angestrebt wird [SLK+12]. Bei der Verwendung von IP in Verbindung mit TCP ist eine deterministische Datenübertragung nicht gegeben und z.T. proprietäre oder veraltete Protokolle werden verwendet (siehe Abschnitt 4.1). Das hier vorgestellte System CoHaRT (**Co**ap **HaRT**Kad) ermöglicht durch die Verwendung von HaRTKad, welches in Kapitel 4 beschrieben ist, und CoAP eine deterministische Übertragung von größeren zusammenhängenden Datenmengen. HaRTKad selbst unterstützt Daten von bis zu 1020 Byte, da diese noch in ein UDP-Paket als Payload passen und genügend Platz für den Headerteil bieten. Man kann zwar mehr Daten über mehrere Zeitschlitze übertragen, jedoch werden diese nicht als zusammenhängende Daten von HaRTKad interpretiert. HaRTKad sichert die deterministische Übertragung der UDP-Pakete zu und durch optionale zusätzliche Redundanz, wie im PSP-Auto-Ansatz (siehe Unterkapitel 3.1), kann eine erhöhte Ausfallsicherheit der Daten garantiert werden. CoAP als leichtgewichtiger Ansatz zur Ressourcennutzung bietet die Möglichkeit, Datenblöcke, die mittels UDP gesendet werden, als zusammenhängende Datenmengen zu interpretieren. CoHaRT ist als Weiterentwicklung des PSP-Auto-Ansatzes gedacht und erweitert dieses um die deterministische Datenübertragung und die Unterstützung der Übertragung von großen Datenmengen.

Die Hauptbeiträge sind:

- Das CoHaRT-Konzept, welches HaRTKad und CoAP verbindet.
- Ein experimenteller CoHaRT-Prototyp.
- Performanceanalyse anhand eines Prototypensetups.

Die Hauptbeiträge dieses Unterkapitels sind in [SDA+15a] publiziert. Der Rest des Unterkapitels ist wie folgt gegliedert: Abschnitt 5.2.2 gibt den Stand der Technik wieder. In den Abschnitten 5.2.3 und 5.2.4 werden das Konzept und die Grundlagen erörtert. Der Prototyp wird in Abschnitt 5.2.5 vorgestellt. In Abschnitt 5.2.6 werden die Messergebnisse eines Prototypensetups dargelegt und ausgewertet. Danach wird der Einsatz von RS-Codes in Abschnitt 5.2.7 diskutiert. Anschließend erfolgt in Abschnitt 5.2.8 eine Einschätzung der Anzahl unterstützter Knoten innerhalb von CoHaRT, bevor das Unterkapitel mit einem Fazit in Abschnitt 5.2.9 endet.

5.2.2. Stand der Technik

Im Bereich der IE-Umgebung wird eine deterministische Datenübertragung der Systeme meist durch proprietäre Hardware oder Protokolle gelöst (siehe Abschnitt 4.1). De facto gibt es keine Implementierung für ein System, das dynamisch mit der Anzahl der Geräte und den Datenmengen skaliert. Da HaRTKad selbst nur kleine Datenmengen überträgt bzw. interpretiert, wird CoAP als höher liegendes Protokoll verwendet.

Zurzeit wird CoAP bei stark ressourcenbeschränkten Systemen, wie Sensornetzwerken, verwendet. Allerdings setzen erste Forschungsprojekte CoAP in medizinischen und industriellen Applikationen ein. Ein deterministisches Zeitverhalten gerade in Bezug auf die Kommunikation ist in diesen Bereichen von essentieller Bedeutung, da erhöhte Anforderungen an Echtzeit vorliegen. In [KRDS14] werden mittels CoAP und des IEEE-Standards 802.15.4 mehrere medizinische Sensoren kabellos miteinander verbunden [IEE11, IEE12]. Innerhalb von 802.15.4 werden reservierte Zeitschlitze für die Echtzeitkommunikation verwendet, wobei durch die Verwendung von WLAN als Medium eine harte Echtzeit nicht garantiert werden kann. In [TWP+13] wird ein System spezifiziert, bei dem CoAP-Instanzen mittels mehrerer 802.15.4e Subnetze und eines Backbone-Netzwerkes verbunden sind. Der Medienzugriff wird mittels eines TDMA-Ansatzes realisiert. Durch die Verwendung eines Hopping-Algorithmus wird die Anfälligkeit gegenüber Interferenzen minimiert. Allerdings wird eine zentrale Instanz benötigt, um einen optimalen Datenfluss zu garantieren. Diese zentrale Instanz stellt allerdings einen SPoF im System dar.

Der hier vorgestellte Ansatz CoHaRT basiert auf einem kabelgebundenen Netzwerk und hat das auf Kad-basierende HaRTKad-System als Grundlage. Es wird keine zentrale

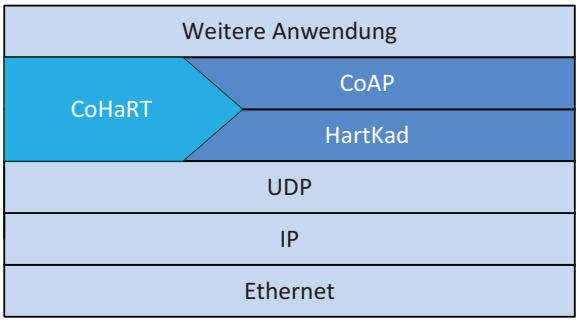

Abbildung 5.9.: Protokollaufbau CoHaRT.

Instanz zur Verwaltung benötigt, was zu einer hohen Zuverlässigkeit des Systems führt. Zusätzlich ermöglicht die Kombination aus HaRTKad und CoAP eine deterministische Datenübertragung.

5.2.3. Genereller Aufbau

Ein Überblick über den Protokollaufbau von CoHaRT ist in Abbildung 5.9 zu sehen. Der Aufbau ist mit dem Protokollaufbau von HaRTKad identisch. Jedoch wird oberhalb der HaRTKad-Schicht eine weitere CoAP-Schicht hinzugefügt, welche die Übertragung von großen Datenmengen erlaubt. TCP wird nicht verwendet, da es ohne Modifikationen nicht echtzeitfähig ist. Ein weiteres Problem bei TCP ist der Stream-basierte Ansatz, große Daten zu übertragen. In Verbindung mit HaRTKad muss die Übertragung aber zu jeder Zeit unterbrochen werden können und das auch über eine längere Zeit, was bei TCP nur mit Modifikationen möglich wäre. UDP bietet diese Möglichkeit, weil nur einzelne Pakete übertragen und kein durchgängiger Datenstream gesichert werden muss. Jedoch bietet UDP nicht die Möglichkeit, zuverlässig große Datenmengen zu interpretieren. Diese Aufgabe übernimmt CoAP. Trotzdem bleibt das darunterliegende Protokoll UDP. CoHaRT selbst stellt bereits eine Applikation zur Übertragung von großen Datenmengen dar, kann aber auch wie HaRTKad einer darüber liegenden Anwendung Daten bereitstellen. Ein weiterer bemerkenswerter Aspekt ist, dass CoAP durch eine geeignete Plattform und HaRTKad z.T. harte Echtzeiteigenschaft erlangt.

5.2.4. Interpretation großer Datenmengen mittels CoAP-Blockoptionen

Die Interpretation von großen Datenmengen mittels UDP ist per se nicht möglich. Als Lösung hierfür wird CoAP verwendet. CoAP bietet mit den Blockoptionen die Möglichkeit, auch größere Datenmengen zu übertragen und zu interpretieren (Siehe Abschnitt 2.1.5). Daten werden hierzu in kleinere Datenmengen unterteilt. In der Praxis sind dies 1024 Bytes, um genügend Raum für die Protokollheader und Optionen zu garantieren. Die Daten können beim Empfänger durch Blocknummern wieder zusammengesetzt werden. Jedem Datenteil ist eine der Blocknummern zugeordnet. Je nach Szenario muss man die Blockoption 1 oder Blockoption 2 verwenden. Blockoption 1 wird verwendet, wenn eine PUT-Operation ausgeführt wird. Dies entspricht dem Abspeichern von Daten auf einer anderen Instanz. Zur Durchführung der GET-Operation, um Daten zu erhalten, wird Blockoption 2 verwendet. Für beide Szenarien wurden Beispiele in Anhang C.1 und Anhang C.2 angegeben. Beide Fälle wurden implementiert und kommen zum Einsatz, wenn zusammenhängende Daten nicht mehr in die Payload eines CoAP-Paketes passen.

5.2.5. Erstellen eines Prototyps

Bei der Applikation in Unterkapitel 5.1 ist es ausreichend, die Performance der Verteilung von Daten innerhalb eines Slots für den Prototypen zu ermitteln. Bei CoHaRT ist dies für die Analyse des Systems nicht ausreichend, da Aktionen über mehrere Zeitschlitze hinweg stattfinden. Um dies zu realisieren, wird eine Queue im Prototypen integriert, welche die Kommunikation steuert. Die Basis des Prototyps ist in Abschnitt 4.3.3 beschrieben. Das Senden der Daten über mehrere Zeitschlitze mit der Queue ist in Abbildung 5.10 dargestellt. Ein Senden von Daten kann aus mehreren Funktionen bzw. Threads erfolgen. Die erste zu treffende Entscheidung ist, ob das Paket ein Request oder Response darstellt. Bei einem Response kann das Paket sofort gesendet werden, da es eine Antwort auf ein Request ist und somit eine andere Instanz den aktuellen Zeitschlitz besitzt und auf eine direkte Antwort wartet. Requests hingegen sind initiale Anfragen von einem Knoten, welche innerhalb der eigenen Zeitschlitze zu versenden sind. Daher müssen Requests zuerst in eine Queue kopiert werden, da ein sofortiges Senden nicht möglich ist. In Abbildung 5.10 ist dieser Vorgang durch Thread 1 definiert. Thread 2 beschreibt das Auslesen der Queue in Abhängigkeit vom eigenen Zeitschlitz. Zuerst wird der eigene Zeitschlitz berechnet (siehe Abschnitt 4.3). Im Folgenden wird gewartet, bis der eigene

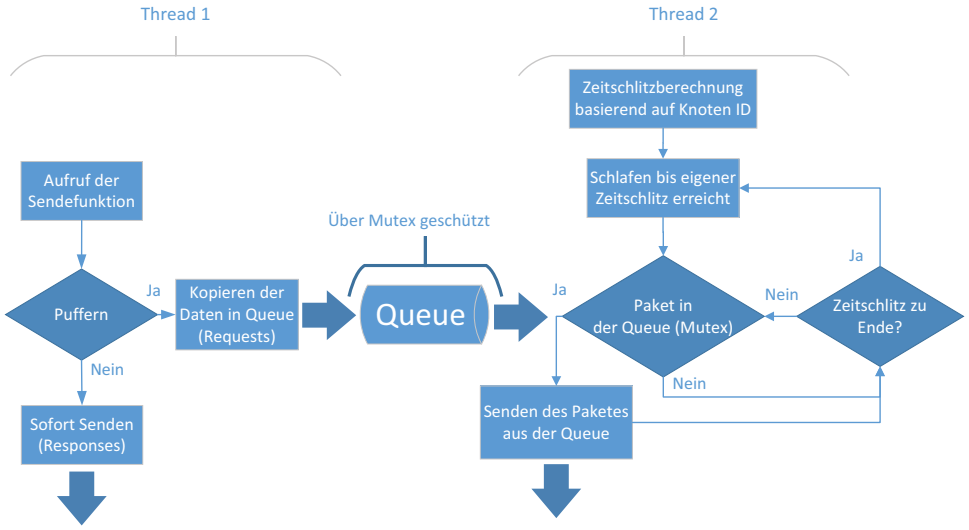

Abbildung 5.10.: Puffern von Daten mittels Queue.

Zeitschlitz erreicht wurde. Ist der eigene Zeitschlitz erreicht, wird geprüft, ob sich ein zu sendendes Paket in der Queue befindet. Sollte dies der Fall sein, wird es sofort gesendet und aus der Queue gelöscht. Anschließend wird geprüft, ob der eigene Zeitschlitz zu Ende ist. Ist dieser nicht zu Ende, kann ein weiteres ggf. vorhandenes Paket gesendet werden. Das Abfragen der Queue und der Zugriff auf die Queue wird durch blockierende Mutexe geregelt. Der Vorteil dieser Methode ist die Vermeidung von Polling. Sollte der Zeitschlitz zu Ende sein, legt sich Thread 2 bis zum nächsten eigenen Zeitschlitz schlafen. Der gesamte Vorgang wird periodisch durchgeführt und erlaubt die Übertragung von großen und zusammenhängenden Datenmengen über mehrere Zeitschlitze.

5.2.6. Messergebnisse und Auswertung

Das Prototypensetup besteht aus zwei ZedBoards, welche jeweils einen CoAP-Client und Server darstellen. Dies genügt, da bei der Datenübertragung immer nur zwei Teilnehmer exklusiv miteinander kommunizieren. Ein zusätzlicher PC sendet Trigger zu einem der beiden Instanzen, um eine Interaktion, wie das Auslösen einer PUT- oder GET-Anfrage, zu starten. Verbunden sind alle Geräte mittels eines Gigabit-Switches. Als Performanceindikator für CoHaRT wird die Durchsatzrate eines Teilnehmers gesehen. Die Durchsatzrate ist charakterisierend, wenn es um die Übertragung von großen Da-

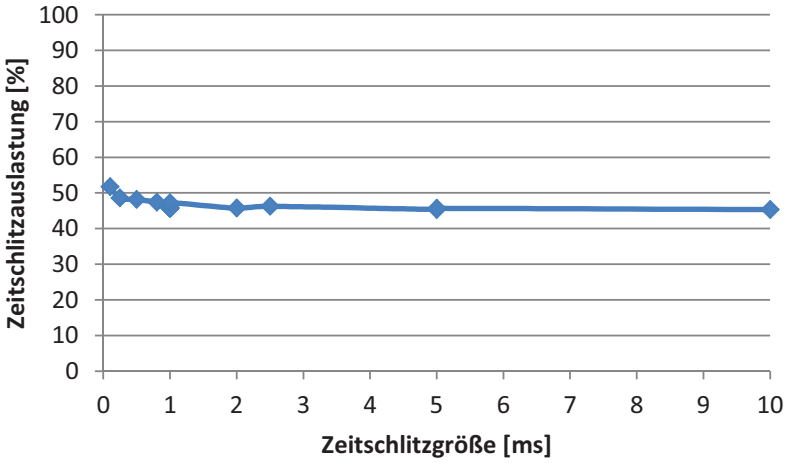

Abbildung 5.11.: Zeitschlitzauslastung beim Senden von UDP-Dummy-Paketen.

tenmengen geht, und ist definiert als Datenmenge pro Zeit bzw. Zeitschlitz. Es wurden verschiedene Szenarien definiert und gemessen, um den Einfluss verschiedener Funktionalitäten auf die Durchsatzrate zu untersuchen.

5.2.6.1. 1. Szenario: UDP-Übertragung

Im Vorfeld wurde der Datendurchsatz mittels UDP-Dummy-Paketen ermittelt. Hierzu sendet ein Knoten kontinuierlich UDP-Dummy-Pakete zu einem anderen Knoten. Dies dient dazu, den Datendurchsatz ohne einen Beeinflussung durch CoAP zu ermitteln. Da die Dummy-Pakete aus dem Kad-Client heraus gesendet werden, wird auch der Overhead durch Kad berücksichtigt.

In Abbildung 5.11 wird die erreichte Auslastung des Mediums aufgezeigt, welche nahezu konstant 45 % beträgt. Als Auslastung ist die Auslastung des Zeitschlitzes definiert. Es wurden 50 Zeitschlitze genutzt und der Knoten selbst darf nur innerhalb eines Zeitschlitzes aktiv sein. Variiert wurde dabei die Größe der Zeitschlitze. Die Größe der Daten beträgt 1000 Byte. Die Auslastung war während der Tests unabhängig von der Anzahl und Größe der Zeitschlitze. Nur bei sehr kleinen Zeitschlitzen zeigt sich, dass die Auslastung etwas ansteigt. Dies liegt daran, dass zwar das Senden eines UDP-Paketes innerhalb eines Zeitschlitzes beginnt, aber nicht endet. Die Übertragung wird in den folgenden Zeitschlitz übergehen, was zu einer höheren Auslastung führt. Zusammenfassend

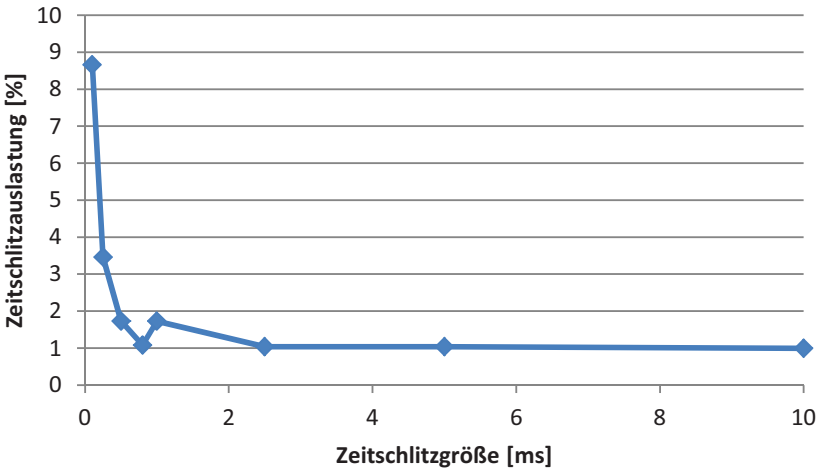

Abbildung 5.12.: Zeitschlitzauslastung bei der Verwendung von CoAP ohne Blockoptionen.

zeigt das Ergebnis eine hohe Performance, die durch den Queuing-Mechanismus erreicht wird. HaRTKad und die benötigte Queue zur Realisierung des TDMA-Ansatzes stellen als zusätzlichen Overhead somit keinen limitierenden Faktor.

Bei den vorgestellten Ergebnissen mit Dummy-Paketen dürften nur zwei Knoten pro Zeitschlitz parallel kommunizieren, wenn eine 1-GBit/s-Verbindung gegeben ist. Dies kann nicht gesteigert werden, da als Worst Case angenommen wird, dass kein Netzwerktopologiewissen vorliegt. So könnte es sein, dass ein Punkt im Netzwerk existiert, durch den der gesamte Datenverkehr durchgeleitet wird.

5.2.6.2. 2. Szenario: CoAP ohne Blockoptionen

Im zweiten Szenario werden von einem CoAP-Client zu einem CoAP-Server Daten mittels PUT-Operationen übertragen. Jede Instanz läuft auf je einem ZedBoard. Allerdings kann auf einem ZedBoard ein CoAP-Client und Server gleichzeitig laufen, sodass diese Rollenaufteilung keinen Nachteil darstellt. Die Anzahl der Zeitschlitze pro Zyklus ist mit 50 definiert. Wieder wurde die Größe der Zeitschlitze variiert. Es wurden nur unabhängige Daten in der Payload, die wieder 1000 Byte beträgt, übertragen, wobei keine Verwendung der Blockoptionen stattfand. Das Ergebnis ist in Abbildung 5.12 dargestellt. Die Auslastung tendiert bei steigender Zeitschlitzgröße gegen 1 %. Für kleine Zeitschlit-

ze steigt die Auslastung an. Dies war zu erwarten, da bei sehr kleinen Zeitschlitzen die Responses nicht mehr in den Zeitschlitz passen. Effektiv werden die Zeitschlitze immer kleiner, jedoch passt das Request weiterhin in den Zeitschlitz, was die Auslastung pro Zeitschlitz effektiv erhöht.

Dies liegt an der verwendeten Antwortstrategie. Ein angefragter Teilnehmer sendet sein Response direkt zurück. Wenn z.B. ein CoAP-Client eine PUT-Operation mit einem CoAP-Server ausführt, sendet der Client zuerst ein Request im eigenen Zeitschlitz. Der Server erhält das Request und generiert ein Response. Da der Zeitschlitz des CoAP-Clients sehr klein ist, wird das Response nicht mehr im selben Zeitschlitz zurückkommen. Daher wird die Antwort innerhalb des Zeitschlitzes eines anderen Teilnehmers übertragen, der nicht in die Interaktion mit eingebunden ist. Dies macht es schwer, die Kommunikation bei sehr kleinen Zeitschlitzen zu steuern bzw. nachzuvollziehen. Eine einfache Lösung wäre es, die Responses über die eigene Queue zu versenden. In dem beschriebenen Beispiel würde der CoAP-Server so lange warten, bis sein Zeitschlitz aktiv ist und das Response an den CoAP-Client senden.

Allerdings ist die Strategie lediglich für sehr kleine Zeitschlitze geeignet und sollte bei großen Zeitschlitzen vermieden werden. Hier sollte die ursprüngliche Strategie beibehalten werden. Ist es erforderlich, dass alles in einem Zeitschlitz abgehandelt wird, sollte man beim Senden des Requests überprüfen, ob genügend Zeit für das Response bleibt. Ein Puffern der Antwort in der Queue würde zu einer sehr schlechten Performance führen, da die großen Zeitschlitze sehr ineffektiv ausgenutzt werden. Das liegt daran, dass man u.U. sehr lange auf ein Response warten würde.

Zu berücksichtigen gilt, dass die Zeit für die Suche einer Instanz im Kad-Netzwerk bei den Ergebnissen nicht einbezogen wird. Die Suche würde auch nur einmalig durchgeführt und ist abhängig von der Anzahl der Knoten. Typische Werte liegen zwischen 550 μs und einigen wenigen Millisekunden (siehe Abschnitt 4.3) und sollten innerhalb eines Zeitschlitzes abgehandelt werden.

5.2.6.3. 3. Szenario: CoAP mit Blockoptionen

Das letzte Szenario umfasst auch die Verwendung der CoAP-Blockoptionen. Dabei wird die Blockoption 1 gewählt, welche bei der PUT-Operation verwendet wird. Es werden zusammenhängende Daten in Form einer Datei der Größe 10 KB oder 100 KB übertra-

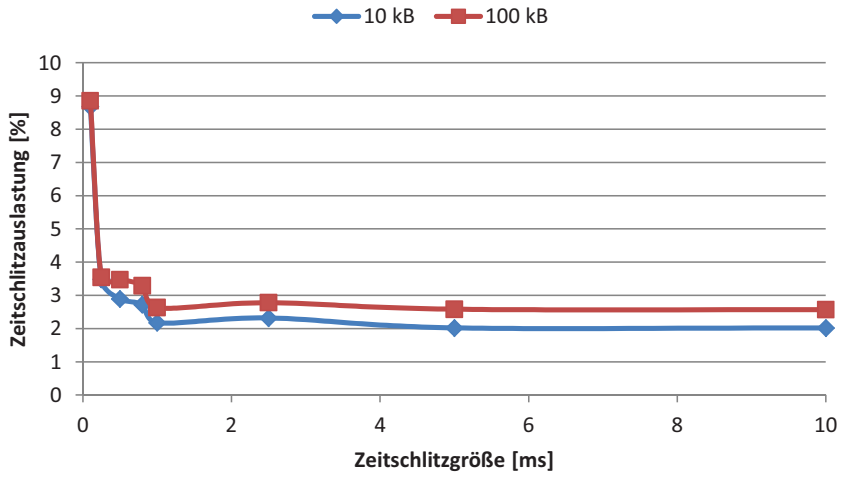

Abbildung 5.13.: Zeitschlitzauslastung bei der Verwendung von CoAP mit Blockoptionen.

gen. Die Übertragung der Dateien wurde mehrfach hintereinander ausgeführt, um einen aussagekräftigen Durchschnittswert ermitteln zu können. Die Daten werden in maximal 1000-Byte große Stücke aufgeteilt und übertragen. Die Anzahl der Zeitschlitze ist erneut 50 und die Größe der Zeitschlitze wurde wieder variiert. Das Ergebnis ist in Abbildung 5.13 ersichtlich. Es ist zu erkennen, dass die Übertragung effizienter ist, da die Pakete nicht jedes Mal neu konstruiert werden müssen. Nur Teile des CoAP-Headers, wie die Blocknummer, müssen ausgetauscht werden. Durch diesen Umstand kann die Auslastung auf ca. 2 % für 10 KB- und ca. 2,6 % für 100 KB große Dateien gesteigert werden. Dies ergibt eine Steigerung gegenüber dem ersten Szenario um mehr als 100 %. Die Steigerung nimmt mit wachsenden Dateigrößen nicht mehr so stark zu, da der negative Effekt durch das Neugenerieren der Pakete kaum noch wahrnehmbar wird. Werden die Zeitschlitze sehr klein (<1 ms) gesetzt, ist der gleiche Effekt wie im 2. Szenario zu sehen. Das Request passt weiterhin in den Zeitschlitz, aber das Response wird außerhalb des eigenen Zeitschlitzes gesendet. Für die Berechnung der Auslastung wird hingegen das Response in einem „cycle“ mit berücksichtigt. Hierdurch entsteht bei der Berechnung eine höhere Auslastung je Zeitschlitz.

5.2.7. Verwendung von RS-Codes

In IE-Umgebungen wird die erfolgreiche Datenübertragung normalerweise durch den Medienzugriff garantiert. Es ist möglich, wie auch bei PSP-Auto (siehe Abschnitt 3.2), die Datenzuverlässigkeit mittels ERCs zu steigern. Dies würde den Ausfall durch physikalische Defekte im Netzwerk kompensieren. Zur Umsetzung der ERCs wurden die RS-Codes verwendet. Nehmen wir an, das Datenvolumen beträgt 1000 Byte, dann benötigt man ca. 430 ms für das Enkodieren und 620 ms für das Dekodieren im Worst Case. Für Szenarien mit harter Echtzeit und geforderten Antwortzeiten im geringen Millisekundenbereich kann dies einen signifikanten Overhead bedeuten und RS-Codes sollten nicht verwendet werden. In diesem Fall sollte die Zuverlässigkeit des Netzwerkes selbst gesteigert werden. Zwei weitere Möglichkeiten wären, die RS-Codes in der Performance zu steigern, z.B. durch die Verwendung von Hardware, was aber eine proprietäre Lösung darstellen würde. Die zweite einfache Lösung könnte die Verwendung von einfacher Datenreplikation sein, was aber ein erhöhtes Datenvolumen bei gleicher Zuverlässigkeit der Daten zur Folge hätte. Ist das Enkodieren und Dekodieren ein nicht zeitkritischer Aspekt, können RS-Codes ohne Einschränkungen eingesetzt werden.

5.2.8. Abschätzung der Anzahl an Knoten

Wenn man nur die Datenübertragung selbst betrachtet und wenn die Suche bereits durchgeführt wurde, dann kann man die Anzahl der möglichen Knoten über die verfügbare Bandbreite abschätzen. Voraussetzung ist eine 1-GBit/s-Verbindung zwischen allen Teilnehmern. Für die Tabelle 5.2 wurde zusätzlich angenommen, dass die Anzahl der Zeitschlitze 50 und die Zeitschlitzgröße 1 ms beträgt, um einen direkten Vergleich zu ermöglichen. Bei steigender Anzahl an Zeitschlitzen pro Zyklus wird folglich die effektive Datenrate pro Knoten gesenkt. In dem gewählten Szenario bei einer Zykluszeit von 50 ms können mittels CoHaRT bereits deultich über 2000 Knoten miteinander kommunizieren bei einer zugesicherten Datenraten von ca. 50 KByte/s. Sollte ein Topologiewissen über das Netzwerk vorhanden sein, könnte man die parallele Kommunikation deutlich steigern und deutlich mehr Knoten unterstützen.

Attribute	Szenario 1	Szenario 2	Szenario 3
Beschreibung	UDP-Dummy-Pakete	CoAP ohne Blockoptionen	CoAP (10 KB/100 KB) mit Blockoptionen
Datenrate pro Knoten [KB/s]	1142,75	43,25	54,33/65,72
Zeitschlitzauslastung in [%]	45,71	1,73	2,02/2,62
Max. Knoten pro Zeitschlitz	ca. 2	ca. 57	ca. 49/39
Max. Knoten pro Zyklus	ca. 100	ca. 2850	ca. 2450/1950

Tabelle 5.2.: Zusammenfassung der CoHaRT-Performance bei 50 Zeitschlitzen pro Zyklus und einer Zeitschlitzgröße von 1 ms. Die Zykluszeit beträgt 50 ms.

5.2.9. Fazit

In diesem Abschnitt wurde die Kombination von HaRTKad und CoAP, welche den Namen CoHaRT trägt, vorgestellt. Das Defizit der Übertragung und Interpretation von großen Datenmengen in HaRTKad konnte durch die Verwendung von CoAP und den CoAP-Blockoptionen beseitigt werden. Es ist dabei gelungen, die Datenraten zu bestimmen, die auch Aufschluss über die unterstützte Anzahl an Knoten geben. Neben dem in Abschnitt 5.1 vorgestellten optimierten WS-Eventing-Ansatz verdeutlicht CoHaRT sehr gut die Leistungsfähigkeit und das Potential der Gesamtkomposition. Der Prototyp, der während der Bachelorarbeit von Herrn Schweißguth entwickelt wurde [Sch13b], wurde bereits mit dem Prof. Dr. Werner Petersen-Preis der Technik für den 2. Platz bei den Bachelorarbeiten ausgezeichnet [Thi14].

Einordnung in den Gesamtkontext der Arbeit: CoHaRT bildet den Abschluss im Gesamtkontext (siehe Abbildung 5.14) und ist gegenüber dem durch Verteilung optimierten Eventing-Ansatzes aus Unterkapitel 5.1 anders einzuordnen. Dies liegt daran, dass sich das verwendete CoAP-Protokoll nicht nur oberhalb von HaRTKad befindet, sondern es auch in seiner Eigenschaft verbessert. Es ist nun möglich, mittels dem Standard

CoAP und den dazugehörigen im Draft definierten Blockoptionen große Datenmengen mittels HaRTKad mit harter Echtzeit zu übertragen. Oberhalb können ebenfalls weitere Anwendungen eingesetzt werden, wobei CoHaRT für diese transparent ist und sich somit auch in bestehende Systeme integrieren lässt.

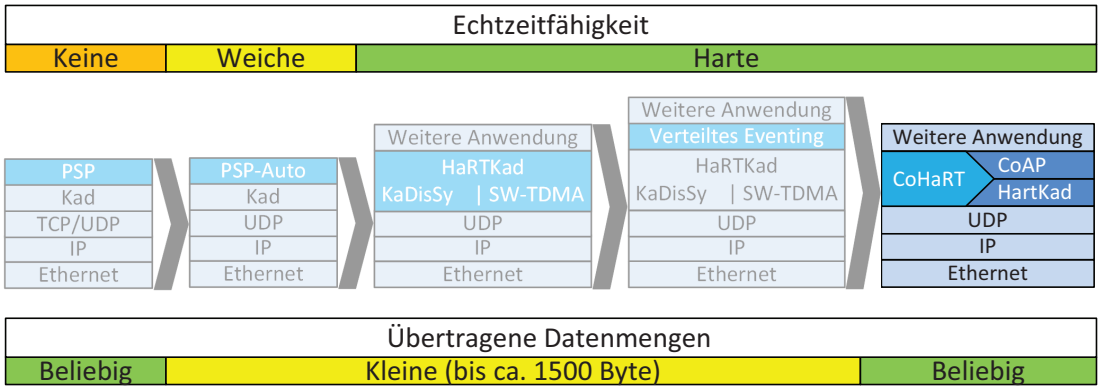

Abbildung 5.14.: Darstellung des CoHaRT-Protokollstacks und Einordnung in den Gesamtkontext.

5.3. Kapitelzusammenfassung

Es wurden zwei Applikationen vorgestellt, die mit harter Echtzeit realisierbar sind. Das WS-Eventing-Beispiel konnte aufzeigen, wie durch Verwendung von P2P-Technologie eine bessere Skalierbarkeit erreicht werden kann und Eventquellen entlastet werden. In Kombination mit HaRTKad ist auch die Kommunikation deterministisch und es wird aufgezeigt, wie viele Eventquellen mit Eventsenken im Worst Case parallel arbeiten könnten. In einem zweiten Beispiel wird HaRTKad mittels des CoAP-Protokolls erweitert, um größere Datenmengen übertragen und interpretieren zu können. Beide Applikationen wurden als Prototypen umgesetzt, sodass deren Machbarkeit bewiesen werden konnte. Diese beiden Beispiele zeigen die Leistungsfähigkeit der verwendeten Plattform, HaRTKad und der daraus resultierenden Kommunikation.

Kapitel 6.

Zusammenfassung

Ausgehend vom PSP-System zum verteilten Speichern wurde in Abschnitt 3.1 verdeutlicht, dass man mittels des P2P-Netzwerkes Kademlia auch Anwendungen, die über das Filesharing hinausgehen, realisieren kann. PSP wurde anschließend in Abschnitt 3.2 für die Automatisierung weiterentwickelt. Das neue PSP-Auto-System zeigt die Grenzen des Systems in Bezug auf die Zuverlässigkeit der Daten auf. PSP-Auto besitzt durch die verwendete Plattform bereits weiche Echtzeitfähigkeiten. Harte Echtzeit kann noch nicht garantiert werden, da die Ethernet-basierende Kommunikation nicht geregelt ist. Ausgehend von PSP-Auto wird die Frage gestellt, inwieweit man ein auf P2P-Technologie basierendes System mit harter Echtzeit umsetzen kann, um sie im Bereich der Automatisierung einsetzen zu können.

Wie dies erreicht werden kann, wird in Kapitel 4 untersucht. Dabei werden zuerst die existierenden Lösungen im Industrial Ethernet-Bereich, welche in der Automatisierung in der Industrie eingesetzt werden, untersucht. Jedoch gibt es keine rein softwarebasierende Lösung ohne einen zentralisierten Ansatz, welche die zukünftigen Anforderungen an Determinismus bei Netzwerken, bestehend aus tausenden Geräten, garantiert und mit steigender Geräteanzahl skaliert. Die Idee der Korrelation vom Hashbereich Kademlias und dem Zeitbereich ermöglicht einen TDMA-basierten Ansatz auf der Grundlage von P2P-Technologie. Zuerst wird das Problem der Synchronisation der Knoten behandelt. Da das System konsistent nach den Prinzipien von unstrukturierten dezentralisierten Netzwerken gestalten werden soll, sollte es möglich sein, die Synchronisation innerhalb von Kad umzusetzen. Dies konnte durch den entwickelten KaDisSy-Algorithmus erreicht werden und durch einen Prototypen abgeleitete Ergebnisse zeigen eine Synchronisationsperformance von deutlich unter 100 ms für 10.000 Knoten.

Der anschließende HaRTKad-Ansatz beschreibt, wie man anhand der Hashwerte der Knoten eindeutige Zeitschlitze zur Realisierung des TDMA-Ansatzes ermittelt. Dies geschieht ohne zentrale Instanz und lässt sich generisch anwenden. Eine weitere Besonderheit ist, dass HaRTKad in der Applikationsebene ohne proprietäres Protokoll oder Hardware realisiert wurde. Ein entwickelter HaRTKad-Prototyp zeigt, dass bereits Anforderungen an harte Echtzeit von unter 10 ms Zykluszeit sinnvoll erreicht werden. Für wenige Prototypen ist sogar eine Zykluszeit von unter 1 ms realisierbar. Abschließend erfolgt ein Vergleich mit einem proprietären System. Es zeigt sich, dass mit steigenden Anforderungen auch andere Herausforderungen auftreten. Zusammenfassend wurden jeweils die Vor- und Nachteile beider Ansätze erörtert.

Aufbauend auf HaRTKad, wurden zwei Applikationen implementiert (siehe Kapitel 5). Bei dem ersten Beispiel wurde ein hochskalierbares WS-Eventing-System entwickelt, das von den positiven Eigenschaften Kads profitiert. Damit konnte eine starke Verbesserung der Benachrichtigungsperformance erreicht werden. In einem realen Prototypensetup konnte bei der Verwendung von nur 18 Eventsenken bereits eine Verbesserung von bis zu 52 % erzielt werden.

Ein zweites Beispiel zeigt ebenfalls den direkten Synergieeffekt, der durch die Verwendung eines Protokolls oberhalb von HaRTKad entsteht. Bei dem Protokoll handelt es sich um CoAP. Dies ist ein leichtgewichtiges REST-Protokoll, das speziell für den Einsatz auf ressourcenbeschränkten Geräten gedacht ist. CoAP wird genutzt, um eine Unterstützung von deterministischen Streams bzw. eine Übertragung großer Datenmengen zu ermöglichen. Das daraus entstandene System namens CoHaRT konnte bereits als Prototyp realisiert werden. Die Ergebnisse der Prototypen zeigen eine hohe Performance des Systems durch die spezielle Verwendung der Blockoptionen von CoAP. CoHaRT ermöglicht ein nahtloses Skalieren der zu übertragenen Datenmengen, da es sich dynamisch an gegebene Anforderungen anpassen lässt.

In dieser Arbeit konnte gezeigt werden, dass auch P2P-Technologie sinnvoll in Einsatzgebieten mit hohen Anforderungen an ein Echtzeitverhalten eingesetzt werden kann. Die zwei entwickelten Applikationen, die HaRTKad als Middleware nutzen, zeigen, dass HaRTKad bestehende Protokolle um harte Echtzeit erweitern kann. Der nicht-proprietäre und standardkonforme Ansatz HaRTKad selbst bleibt für den Nutzer transparent und kann in bestehende Systeme integriert werden. Folglich ist P2P-Technologie nicht nur sinnvoll in der Automatisierung einsetzbar, sondern vermeidet auch einen Großteil der

Probleme derzeitiger Systeme wie eine zentralisierte Struktur.

6.1. Ausblick

Diese Arbeit bezieht sich auf kabelgebundene Netzwerke. Es wäre interessant, die gewonnenen Erkenntnisse auf den Bereich der kabellosen Netzwerke auszuweiten. Dies sollte möglich sein, da HaRTKad oberhalb von UDP direkt in Software umgesetzt wurde und das darunterliegende Medium aus Sicht von HaRTKad transparent ist. Ein weiterer Ansatzpunkt, der auf Hardware basiert, ist die Verwendung von neuen Übertragungsstandards bei kabelgebundenen Medien wie 10-GBit/s. Hierdurch könnte eine theoretische Steigerung der unterstützten Knotenanzahl und auch deren Performance erreicht werden. Auf Seite der Software bietet es sich vor allem an, Wissen über die Netzwerktopologie einfließen zu lassen. Durch dieses Wissen könnte der Grad der parallelen Kommunikation enorm gesteigert werden, da derzeitige Betrachtungen vom Worst Case ausgehen, der besagt, dass im Netzwerk ein Knotenpunkt mit 1-GBit/s existiert. Alle Bestrebungen sollten darauf hinauslaufen, dass HaRTKad für den Bereich der isochronen harten Echtzeit mit einer Zykluszeit von 1 ms weiterentwickelt wird, weil dies bereits prototypisch für zwei Knoten realisiert wurde. Das Potential ist gegeben und gerade mit zukünftigen Ansätzen, wie Software Defined Networks (SDNs), bieten sich Möglichkeiten, z.B. den geforderten Jitter von unter $1\mu s$ zu erreichen. HaRTKad würde folglich stark von neuen leistungsstarken Übertragungsmedien und neuen Ansätzen wie SDN profitieren und sein enormes vorhandenes Potential entfalten. Abschließend soll HaRTKad in einem größeren praktischen Umfeld, bestehend aus mehreren Geräten, getestet werden, um weitere Aussagen über die Skalierbarkeit, insbesondere mit dem Interleaving und der daraus resultierenden parallelen Kommunikation, treffen zu können.

Anhang A.

Abstrakter Aufbau des PSP-Simulators

In der Abbildung A.1 ist der Aufbau des Simulators abstrakt dargestellt. Es handelt sich dabei um einen selbst entwickelten diskreten Simulator, geschrieben in C++. Dieser ermöglicht es durch Verzicht auf reale Topologie, mehrere tausend Knoten über einen langen Zeitraum von mehreren Monaten zu simulieren. Entwickelt wurde der Simulator in Qt [Qt], der zudem eine grafische Oberfläche bietet, welche in der Abbildung A.2 zu sehen ist. Der Simulator besteht im Wesentlichen aus zwei Threads. Der erste Thread ist für die GUI zuständig, startet den zweiten Thread, der für die Simulation verantwortlich ist, und reicht die nötigen Parameter an diesen weiter. Der zweite Thread besteht aus einer Hauptschleife, die die einzelnen Sekunden hoch zählt. Das Objekt „Knotenliste“ enthält dabei die Objekte „Knoten“, welche nun für jeden Zeitwert in der Schleife überprüft werden. Sind alle Knoten überprüft, ist ein Schleifendurchlauf geschafft. Die Knoten selbst stellen ebenfalls Objekte dar, die je nach Applikation verschiedene Attribute besitzen, wie z.B. die gespeicherten Chunks der PSP-Applikation. Jeder Knoten verfügt über eine Zustandsmaschine, mittels der die Übergänge definiert sind. Als Beispiel sei hier der Übergang aus einem Ruhemodus in den Sendemodus genannt, damit ein Knoten seine Chunks im Netzwerk verteilen kann.

Mittels der GUI aus Abbildung A.2 kann ein Nutzer von extern die Parameter, wie Knotenanzahl, Simulationsdauer und RS-Parameter, justieren. Dies ermöglicht eine intuitive und effektive Steuerung des Simulators.

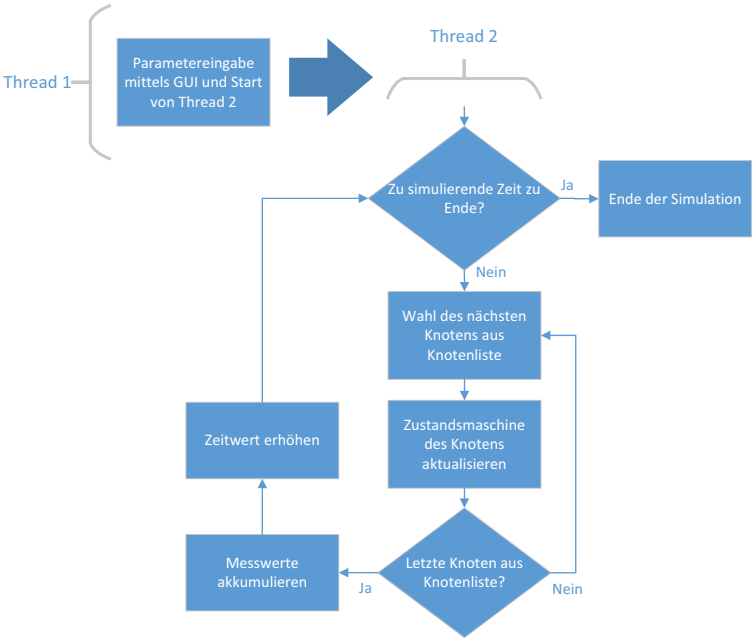

Abbildung A.1.: Abstrakte Darstellung des PSP-Simulators.

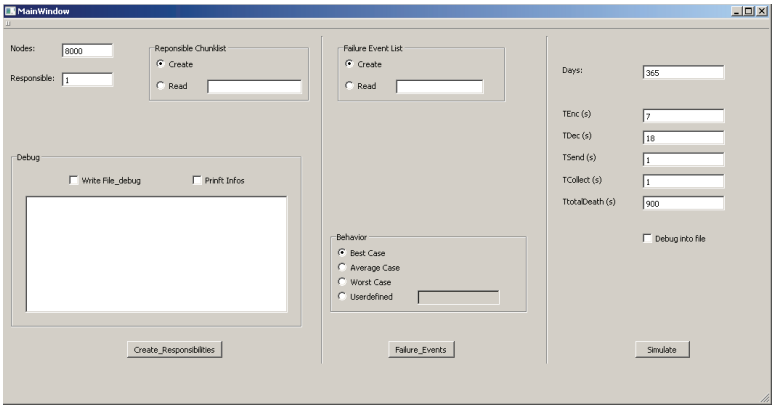

Abbildung A.2.: Darstellung der GUI für die Simulation von PSP-Auto.

Anhang B.

Alternative Zielplattform Raspberry Pi

Es wurde auch eine alternative Zielplattform für HaRTKad untersucht. Hierbei handelt es sich um das Raspberry Pi Model B [ras14]. Es wurde ermittelt, inwiefern man in der Lage ist, harte Echtzeitbedingungen mit dem Raspberry Pi umzusetzen. Die ist auf Grund der Limitierung der Hardware der Raspberry Pi-Plattform bedingt nur möglich. Ein Nachteil ist, dass das Ethernetmodul keinen direkten Interrupt besitzt, sondern über den USB-Controller angeschlossen wurde. Auch die Daten von und zum Ethernetcontroller gehen über den USB-Controller. Generell gibt es es mehrere Möglichkeiten, um das Raspberry Pi unter Echtzeitanforderungen laufen zu lassen. Eine Möglichkeit ist die Verwendung von Xenomai und RTAI, die als Mikrokernel fungieren und den normalen Linux-Threads mit niedriger Priorität ausführen [xen14]. Da allerdings für das Raspberry Pi kein echtzeitfähiger Treiber für Ethernet zur Verfügung stand, wurde diese Möglichkeit nicht weiter verfolgt. Der Treiber ist nötig, da ansonsten normale nicht echtzeitfähige Linuxfunktionen genutzt werden müssten.

Die zweite umgesetzte Möglichkeit ist die Verwendung von RT-Preempt-Patch, um den Linux-Kernel selbst echtzeitfähig zu machen [pre14]. Als Grundlage dient die Raspbian-Distribution von Linux mit dem durch RT-Preempt gepatchen Linux Kernel 3.2. Als Messwerte für die Performance vom Raspberry Pi wurde für diese Plattform HaRTKad portiert. Gemessen wurden die Zeiten von der Erstellung eines Suchobjektes und ein dazugehöriger Suchschritt, bestehend aus einem Kad Req/Res-Paar. Der Prototypenaufbau umfasst zwei Raspberry Pis und ein 1-GBit-Switch. Effektiv wurden aber auf Grund der Einschränkungen des Ethernetcontrollers des Raspberry Pis nur 100-MBit-Verbindungen genutzt.

In Abbildung B.1 ist die Zeit aufgeführt, die benötigt wird, um auf einem Raspberry Pi

Profil	Normal	Übertaktet
CPU	700 MHz	1 GHz
GPU	250 MHz	500 MHz
SD-RAM	400 MHz	600 MHz
Spannung	1,2V	1,35V

Tabelle B.1.: Raspberry Pi-Einstellungen.

innerhalb HaRTKads ein Suchobjekt zu erstellen, ein Kad Req an ein zweites Raspberry Pi zu senden und die Antwort in Form eines Kad Res auf dem ersten Raspberry Pi zu erkennen. Dieser Vorgang wurde 100.000 mal durchgeführt und die Verteilung angegeben. Es wird zusätzlich unterschieden, ob das Raspberry Pi zudem übertaktet wurde. In der Tabelle B.1 sind die Werte für das Raspberry Pi mit und ohne Übertaktung aufgezeigt. Der Mittelwert ohne Übertaktung beträgt 1752 μs und mit Übertaktung 1753 μs. Der Unterschied ist marginal, aber es fällt auf, dass mit Übertaktung keine so starken Ausreißer entstehen. Eine Erklärung hierfür könnte sein, dass Standard-Hintergrundtasks schneller abgearbeitet werden können und es so zu weniger Delays im Vorfeld kommen kann.

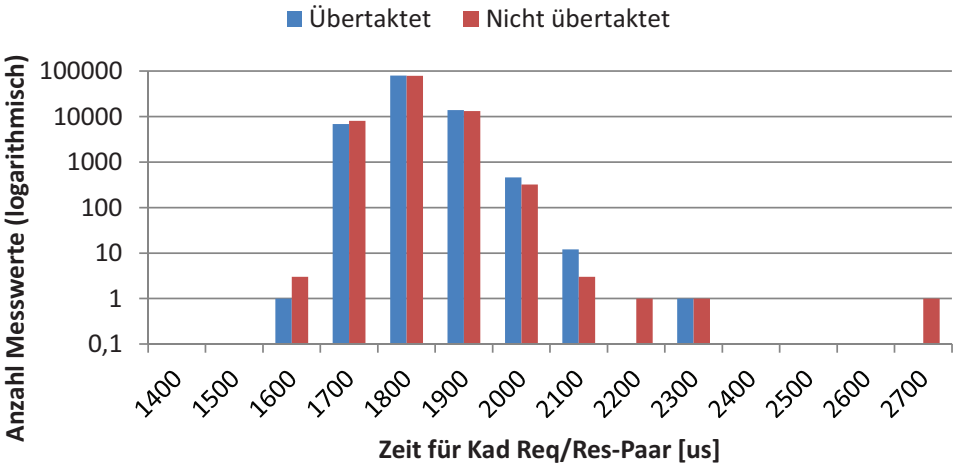

Abbildung B.1.: Zeit für ein Kad Req/Req-Paar mit Suchobjekterstellung.

Alternativ wurden die Ergebnisse aufgenommen, die die Erstellung des Suchobjektes

nicht mit einbeziehen. Bei den Zeitwerten wird folglich nur das Senden, Empfangen und Verarbeiten des Kad Req/Res-Paares berücksichtigt. Die Ergebnisse sind in Abbildung B.2 dargestellt. Es wurden ebenfalls wieder 100.000 Messungen durchgeführt und die Verteilung dargestellt. Die Tendenz ist die gleiche wie vorher, nur dass sich die Werte um einen Offset verschieben. Hieraus lässt sich die durchschnittliche Zeit für die Erstellung des Suchobjektes ermitteln. Dieser Wert beträgt ohne Übertaktung ca. 167,5 μs. Mit Übertaktung ist ebenfalls kaum eine signifikante Verbesserung feststellbar. Hier liegt der Wert bei ca. 166 μs. Auch hier ist wieder zu erkennen, dass mit Übertaktung keine größeren Ausreißer vorkommen.

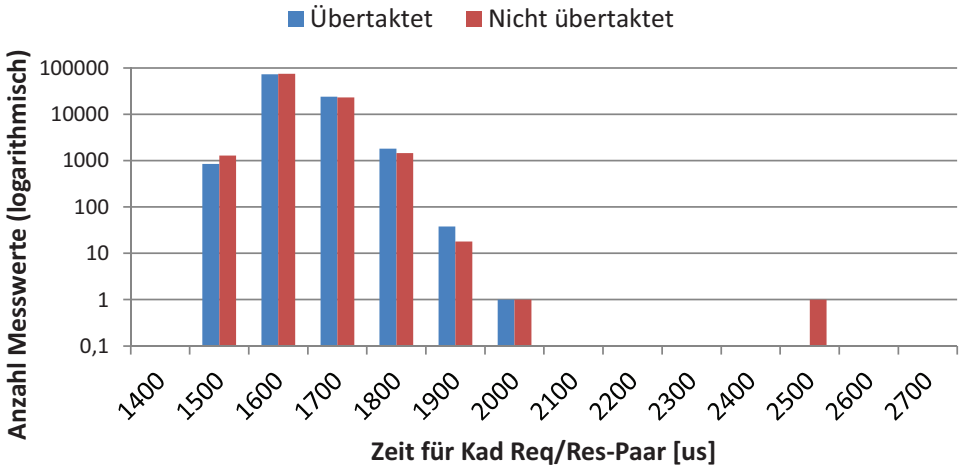

Abbildung B.2.: Zeit für ein Kad Req/Req-Paar ohne Suchobjekterstellung.

Mit dem Raspberry Pi können adäquate Performancewerte erreicht werden, die den Einsatz im Human-Umfeld ($T_{Cyc} = 100ms$) von Automatisierungsanlagen erlaubt. Es ist zu empfehlen, das Raspberry Pi im übertakteten Modus zu nutzen, um signifikante Ausreißer zu vermeiden. Im Vergleich zum ZedBoard ist das Raspberry Pi in der prototypischen Umsetzung ca. um den Faktor 3,2 schlechter, da das ZedBoard für eine Suchobjekterstellung und ein Kad Req/Res-Paar nur ca. 550 μs benötigt. Es ist zwar möglich, mittels des Raspberry Pis auch HaRTKad umzusetzen, jedoch mit den beschriebenen Einbußen in der Performance. Für die beschriebenen Projekte wurde daher das ZedBoard als Zielplattform beibehalten.

Anhang C.

CoAP-Kommunikation

C.1. Beispiel Kommunikationsablauf mittels Block1-Option

In der Abbildung C.1 ist der Ablauf der Datenübertragung mittels Block1-Option dargestellt. Ziel ist es, 3000 Bytes vom Client an den Server zu übertragen. Zunächst wird vom Client eine PUT-Methode als NON-Nachricht ausgeführt. Danach folgt die URI-Adresse der Repräsentation. In den eckigen Klammern bei der genannten Block1-Option befinden sich die in Abschnitt 2.1.5 beschriebenen Parameter der Blockoption. Nun wird mit der Null, die erste Blocknummer angegeben. Danach folgt die Ankündigung mindestens eines weiteren Blockes durch Setzen der 1 im M-Feld. Abschließend wird bestimmt, wie groß die übertragenen Bytes in diesem Paket sind. In diesem Fall beträgt der Wert 1024 Byte.

Der Server antwortet daraufhin mit dem Typ „Changed“, dass die Ressource erstellt oder geändert wurde. Durch die Blockoptionen ist gekennzeichnet, welcher Block und in welcher Größe dieser erfolgreich übernommen wurde. Zusätzlich ist das M-Feld gesetzt, da vom Server in diesem Beispiel noch weitere Blöcke in Paketen erwartet werden.

Im nächsten Request/Response-Schritt werden die nächsten 1024 Bytes erfolgreich übertragen. Abschließend werden die letzten Bytes im dritten Schritt übertragen. Das M-Feld kennzeichnet bereits, dass keine weiteren Blöcke folgen.

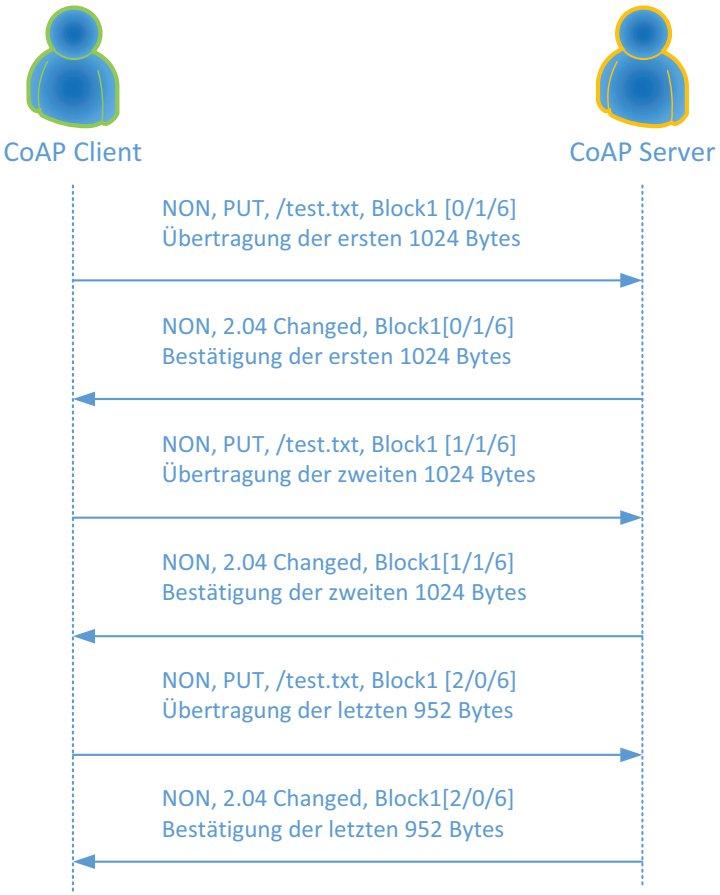

Abbildung C.1.: Darstellung der Datenübertragung mittels Block1-Option.

C.2. Beispiel Kommunikationsablauf mittels Block2-Option

In der Abbildung C.2 ist der Ablauf der Datenübertragung mittels Block2-Option dargestellt. Im zweiten Beispiel sollen 2000 Bytes übertragen werden. Zuerst sendet der Client eine GET-Anfrage in Form einer NON-Nachricht. Diese Anfrage erfolgt noch ohne Block2-Option. Der Server sendet darauf ein Request mit dem Typ „Content“. Der Typ „Content“ besagt, dass die angeforderten Daten erfolgreich als Daten im Paket vorliegen. Die mitübertragene Block2-Option gibt an, dass es sich um die Blocknummer 0 handelt und die Payload eine Größe von 1024 Bytes besitzt. Zusätzlich wird durch Setzen des M-Feldes gekennzeichnet, dass noch weitere Blöcke existieren. In diesem Beispiel

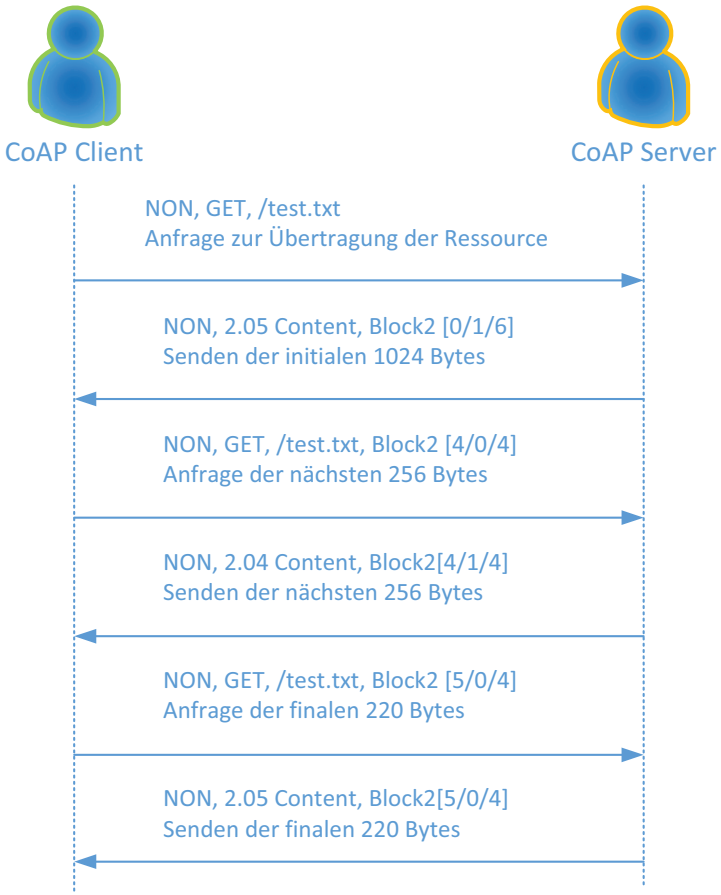

Abbildung C.2.: Darstellung der Datenübertragung mittels Block2-Option.

wurde eine Besonderheit eingeführt: das dynamische Anpassen der Größe der übertragenen Bytes in der Payload. Der Client erfragt nun den nächsten Block, jedoch mit der Nummer 4, da er die angefragte Bytegröße im Paket auf 256 Bytes reduziert haben möchte und sich daraus eine neue Nummerierung der Blöcke ergibt. Er erhält daraufhin die nächsten 256 Bytes. Abschließend werden im dritten Schritt die letzten 220 Bytes übertragen, was durch den Server mit Setzen der 0 im M-Feld gekennzeichnet wird.

Literaturverzeichnis

[Ama] http://aws.amazon.com/

[Avn] http://www.zedboard.org/

[BB06] Brunner, René ; Biersack, E: A performance evaluation of the Kad-protocol. In: *Institut Eurecom, France* (2006)

[BBMP09] Bertasi, P. ; Bonazza, M. ; Moretti, N. ; Peserico, E.: PariSync: Clock synchronization in P2P networks. In: *Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. International Symposium on*, 2009, S. 1 –6

[Bid03] Bidgoli, Hossein: *The Internet Encyclopedia*. Bd. 3. John Wiley & Sons Inc, 2003. – 944 S.

[boo13] book industrial e.: *The world market for Industrial Ethernet components*. http://www.iebmedia.com/index.php?id=8595&parentid=74&themeid=255&showdetail=true&bb=true. Version: 2013

[BPQS08] Baldoni, R. ; Platania, M. ; Querzoni, L. ; Scipioni, S.: A Peer-to-Peer Filter-Based Algorithm for Internal Clock Synchronization in Presence of Corrupted Processes. In: *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium on*, 2008, S. 64 –72

[BS11] Bratukhin, A. ; Sauter, T.: Functional Analysis of Manufacturing Execution System Distribution. In: *Industrial Informatics, IEEE Transactions on* 7 (2011), Nov, Nr. 4, S. 740–749. http://dx.doi.org/10.1109/TII.2011.2167155. – DOI 10.1109/TII.2011.2167155. – ISSN 1551–3203

[BS13] Bormann, C. ; Shelby, Z.: *Blockwise transfers in CoAP: draft-ietf-core-block-14*. http://tools.ietf.org/html/draft-ietf-core-block-14. Version: 2013

[CC-13a] CC-Link Partner Association: *CC-Link IE Field Brochure*. http://www.cclinkamerica.org/functions/dms/getfile.asp?ID=045000000000000001000009146100000. Version: 2013

[CC-13b] CC-Link Partner Association: *CC-Link IE Field White Paper (CLPA-2327)*. http://www.cclinkamerica.org/functions/dms/getfile.asp?ID=045000000000000001000007488300000. Version: 2013

[Com98] Comer, Douglas: *Computernetzwerke und Internets*. Prentice Hall, 1998

[Dan12] Danielis, Peter: *Peer-to-Peer-Technologie in Teilnehmerzugangsnetzen*, Universität Rostock, Diss., 2012

[DGT+10] Danielis, Peter ; Gotzmann, Maik ; Timmermann, Dirk ; Bahls, Thomas ; Duchow, Daniel: A Peer-To-Peer-based Storage Platform for Storing Session Data in Internet Access Networks. In: *Telecommunications: The Infrastructure for the 21st Century (WTC), 2010* (2010), sept., S. 1 –6

[DMWC11] Davis, Doug ; Malhotra, Ashok ; Warr, Katy ; Chou, Wu: *Web Services Eventing (WS-Eventing)*. W3C Recommendation. http://www.w3.org/TR/ws-eventing/. Version: 2011

[DR01] Druschel, P. ; Rowstron, A.: PAST: a large-scale, persistent peer-to-peer storage utility. In: *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, 2001, S. 75 – 80

[EA12] Evans, Peter C. ; Annunziata, Marco: Industrial Internet: Pushing the Boundaries of Minds and Machines / General Electric. 2012. – Forschungsbericht

[EFGK03] Eugster, Patrick T. ; Felber, Pascal A. ; Guerraoui, Rachid ; Kermarrec, Anne-Marie: The Many Faces of Publish/Subscribe. In: *ACM Comput. Surv.* 35 (2003), Juni, Nr. 2, 114–131. http://dx.doi.org/10.1145/857076.857078. – DOI 10.1145/857076.857078. – ISSN 0360–0300

[EGKM04] Eugster, P.T. ; Guerraoui, R. ; Kermarrec, A.-M. ; Massoulie, L.: Epidemic information dissemination in distributed systems. In: *Computer* 37 (2004), may, Nr. 5, S. 60 – 67. http://dx.doi.org/10.1109/MC.2004.1297243. – DOI 10.1109/MC.2004.1297243. – ISSN 0018–9162

[emu] *eMule*. http://www.emule-project.net/

[EPS12] EPSG: *Ethernet POWERLINK*. http://www.ethernet-powerlink.org. Version: 2012

[Erl09] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2009

[Eth13a] EtherCAT Technology Group: *EtherCAT - the Ethernet fieldbus*. http://www.ethercat.org/pdf/ethercat_e.pdf. Version: 2013

[Eth13b] EtherCAT Technology Group: *EtherCAT Technical Introduction and Overview*. http://www.ethercat.org/en/technology.html. Version: 2013

[Eva11] Evans, Cisco:Dave: The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. In: *Cisco Internet Business Solutions Group (IBSG)*, 2011

[Fel05] Felser, M.: Real-Time Ethernet - Industry Prospective. In: *Proceedings of the IEEE* 93 (2005), Nr. 6, S. 1118–1129. http://dx.doi.org/10.1109/JPROC.2005.849720. – DOI 10.1109/JPROC.2005.849720. – ISSN 0018–9219

[Fel10] Felser, M.: Real time ethernet: Standardization and implementations. In: *IEEE International Symposium on Industrial Electronics*, 2010

[Fie00] Fielding, Roy T.: *REST: Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Doctoral dissertation, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[Fre] http://savannah.nongnu.org/projects/lwip/

[FWW13] Forschungsunion Wirtschaft Wissenschaft, Promotorengruppe K.: *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. http://www.bmbf.de/pubRD/Umsetzungsempfehlungen_Industrie4_0.pdf. Version: April 2013

[GJF13] Gaj, P. ; Jasperneite, J. ; Felser, M.: Computer Communication Within Industrial Distributed Environment—Survey. In: *Industrial Informatics, IEEE Transactions on* 9 (2013), Nr. 1, S. 182–189. http://dx.doi.

org/10.1109/TII.2012.2209668. – DOI 10.1109/TII.2012.2209668. – ISSN 1551–3203

[GMR+09] Geoghegan, S.J. ; McCorkle, G. ; Robinson, C. ; Fundyler, G. ; Ramaswamy, S. ; Brown, J. ; Itmi, M.: A P2P, Agent-Based System of Systems Architecture for Cooperative Maritime Networks. In: *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, 2009, S. 1–2

[Goo] https://developers.google.com/maps/documentation/business/webservices/

[Goo14] Google: *IPV6 Statistiken*. http://www.google.de/ipv6/statistics.html. Version: Mai 2014

[Gre11] Gregorczyk, D.: WS-Eventing SOAP-over-UDP Multicast Extension. In: *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, S. 660–665

[GWXT98] Grieger, Günther ; W7-X Team the: The WENDELSTEIN 7-X Project. In: *Journal of Plasma and Fusion Research Vol. 1*, 1998

[HG06] Huang, Yi ; Gannon, D.: A comparative study of Web services-based event notification specifications. In: *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, 2006. – ISSN 1530–2016, S. 8 pp.–14

[HLZL13] Hu, Tianliang ; Li, Peng ; Zhang, Chengrui ; Liu, Riliang: Design and application of a real-time industrial Ethernet protocol under Linux using RTAI. In: *International Journal of Computer Integrated Manufacturing* 26 (2013), Nr. 5, S. 429–439. http://dx.doi.org/10.1080/0951192X.2012.731609. – DOI 10.1080/0951192X.2012.731609

[HSHG06] Huang, Yi ; Slominski, A. ; Herath, C. ; Gannon, D.: WS-Messenger: a Web services-based messaging system for service-oriented grid computing. In: *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on* Bd. 1, 2006, S. 8 pp.–173

[Huf03] Huffman, V. W. C. ; P. W. C. ; Pless: Fundamentals of Error-Correcting Codes., Cambridge University Press, 2003

[IEE] IEEE: *IEEE Standard for Ethernet 802.3-2012*

[IEE04] IEEE: *IEEE 802.1D Edition 2004 IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges*. New York, 2004

[IEE05] IEEE: *IEEE 802.1Q Edition 2005 IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks*. 2005

[IEE11] IEEE: *IEEE Std 802.15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. New York, 2011

[IEE12] IEEE: *IEEE Std 802.15.4e: Low-Rate Wireless Personal Area Networks (LR-WPANs) - Amendment 1: MAC sublayer*. New York, 2012

[IEE14] IEEE 802 LAN/MAN Standards Committee: *Time-Sensitive Networking Task Group*. `http://www.ieee802.org/1/pages/tsn.html`. Version: July 2014

[JGP10] Jayasinghe, R. ; Gamage, D. ; Perera, S.: Towards Improved Data Dissemination of Publish-Subscribe Systems. In: *Web Services (ICWS), 2010 IEEE International Conference on*, 2010, S. 520–525

[KBC+00] Kubiatowicz, John ; Bindel, David ; Chen, Yan ; Czerwinski, Steven ; Eaton, Patrick ; Geels, Dennis ; Gummadi, Ramakrishan ; Rhea, Sean ; Weatherspoon, Hakim ; Weimer, Westley ; Wells, Chris ; Zhao, Ben: OceanStore: an architecture for global-scale persistent storage. In: *SIGPLAN Not.* 35 (2000), Nr. 11, S. 190–201. `http://dx.doi.org/http://doi.acm.org/10.1145/356989.357007`. – DOI http://doi.acm.org/10.1145/356989.357007. – ISSN 0362–1340

[KGS11] Khan, S. ; Gani, A. ; Sreekandath, M.: The routing performance of logarithmic-hop structured P2P overlay. In: *Open Systems (ICOS), 2011 IEEE Conference on*, 2011, S. 202–207

[KOV11] Klasen, Frithjof ; Oestreich, Volker ; Volz, Michael: *Industrial Communication with Fieldbus and Ethernet*. VDE Verlag GmbH, 2011

[KRDS14] Khattak, H.A ; Ruta, M. ; Di Sciascio, E.: CoAP-based healthcare sensor networks: A survey. In: *Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on*, 2014, S. 499–503

[LCL04] Lin, W. K. ; Chiu, D. M. ; Lee, Y. B.: Erasure Code Replication Revisited, IEEE P2P, 2004, S. 90–97

[LE02] Lee, Kang ; Eidson, John: IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In: *In 34 th Annual Precise Time and Time Interval (PTTI) Meeting*, 2002, S. 98–105

[Loc] Locke, Dave: *MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*. `http://www.ibm.com/developerworks/webservices/library/ws-mqtt/ws-mqtt-pdf.pdf`

[LT10] Liu, Chung-Hsin ; Teng, Po-Cheng: The safety design of the laboratory monitoring system. In: *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on*, 2010, S. 509–513

[lud13] *LUDWIG-BÖLKOW-Nachwuchspreis Mecklenburg-Vorpommern*. `http://www.boelkowpreis.de/pdf/06_preistraeger.pdf`. Version: 2013

[LW06] Lüders, Christoph ; Winkler, Martin: Wie die TCP/IP-Flusskontrolle das Surf-Tempo bestimmt. In: *c't - Magazin für Computertechnik, 23/2006* (2006), S. 198

[LXN04] Liu, Y. ; Xiao, L. ; Ni, L.M.: Building a scalable bipartite P2P overlay network. In: *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, S. 46

[LZHZ09] Linying, Jiang ; Zhiliang, Zhu ; Hailong, Li ; Zhenhua, Guo: Embedded Laboratory Environment Monitor System. In: *Information Engineering, 2009. ICIE '09. WASE International Conference on* Bd. 2, 2009, S. 197–201

[MCB+11] Moraes, Ricardo ; Carreiro, Francisco B. ; Bartolomeu, Paulo ; Silva, Valter ; Fonseca, José A. ; Vasques, Francisco: Enforcing the timing behavior of real-time stations in legacy bus-based industrial Ethernet networks. In: *Computer Standards & Interfaces* 33 (2011), März, Nr. 3, S. 249–261. `http://dx.doi.org/10.1016/j.csi.2010.05.002`. – DOI 10.1016/j.csi.2010.05.002. – ISSN 0920–5489

[MI06] Modbus-IDA.ORG: *MODBUS Messaging on TCP/IP Implementation Guide V1.0b*. 2006

[Mil92] Mills, D.L.: *Network Time Protocol (NTP)*. RFC 1305. `http://www.ietf.org/rfc/rfc1305.txt`. Version: March 1992 (Request for Comments)

[MJB04] Montresor, A. ; Jelasity, M. ; Babaoglu, O.: Robust aggregation protocols for large-scale overlay networks. In: *Dependable Systems and Networks, 2004 International Conference on*, 2004, S. 19 – 28

[MKS08] Morariu, C. ; Kramis, T. ; Stiller, B.: DIPStorage: Distributed storage of IP flow records. In: *Local and Metropolitan Area Networks, 2008. LANMAN 2008. 16th IEEE Workshop on*, 2008, S. 108 –113

[MM02] Maymounkov, P. ; Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS, 2002

[ND85] Normung (DIN), Deutsches I.: *DIN 44300: Informationsverarbeitung - Begriffe*. Beuth-Verlag, 1985

[Net] `http://www.netgear.com/service-provider/products/switches/unmanaged-desktop-switches/GS108.aspx`

[Net12] Networks, HMS I.: *Industrial Ethernet - Status und Ausblick*. `http://www.feldbusse.de/trends/status-ethernet.shtml`. Version: 2012

[Neu07] Neumann, Peter: Communication in industrial automation—What is going on? In: *Control Engineering Practice* 15 (2007), Nr. 11, S. 1332–1347. `http://dx.doi.org/10.1016/j.conengprac.2006.10.004`. – DOI 10.1016/j.conengprac.2006.10.004. – ISSN 0967–0661

[Nin11] Ninnemann, M.: Freescale Semiconductor PowerQUICC II Pro - Performance and Utilization Broadband Access Division, Former Nokia Siemens Networks GmbH & Co. KG, November 2011

[OAS02] OASIS: *OASIS UDDI Specifications TC - Committee Specifications*. `http://www.w3.org/TR/wsdl.html`. Version: 2002

[OAS09a] OASIS: *Devices Profile for Web Services Version 1.1*. `http:`

//docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html. Version: 2009

[OAS09b] OASIS: *SOAP-over-UDP Version 1.1.* http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html. Version: 2009

[OAS09c] OASIS: *Web Services Dynamic Discovery (WS-Discovery) Version 1.1.* http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html. Version: 2009

[Pan13] Panel Building & System Integration: *Ethernet adoption in process automation to double by 2016.* http://www.pbsionthenet.net/article/58823/Ethernet-adoption-in-process-automation-to-double-by-2016.aspx. Version: 2013

[PKB10] Prowell, Stacy ; Kraus, Rob ; Borkin, Mike: *Seven Deadliest Network Attacks.* Syngress, 2010

[PLR13] Paiva, J. ; Leitao, J. ; Rodrigues, L.: Rollerchain: A DHT for Efficient Replication. In: *Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on*, 2013, S. 17–24

[pre14] https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

[PRO12] PROFIBUS Nutzerorganisation e.V.: *PROFINET.* http://www.profibus.com/technology/profinet/. Version: 2012

[QLZH10] Qiao, Yuliang ; Liu, Guo-Ping ; Zheng, Geng ; Hu, Wenshan: NCSLab: A Web-Based Global-Scale Control Laboratory With Rich Interactive Features. In: *Industrial Electronics, IEEE Transactions on* 57 (2010), Oct, Nr. 10, S. 3253–3265. http://dx.doi.org/10.1109/TIE.2009.2027924. – DOI 10.1109/TIE.2009.2027924. – ISSN 0278–0046

[Qt] http://qt-project.org/

[RA10] Ribeiro, H.B. ; Anceaume, E.: DataCube: A P2P Persistent Data Storage Architecture Based on Hybrid Redundancy Schema. In: *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010. – ISSN 1066–6192, S. 302 –306

[ras14] http://www.raspberrypi.org/

[Rea] `http://www.freertos.org/`

[rfc81] *RFC 791: Internet Protocol*. `http://tools.ietf.org/html/rfc791`. Version: 1981

[RFSC+05] Robinson, J.M. ; Frey, J.G. ; Stanford-Clark, A.J. ; Reynolds, A.D. ; Bedi, B.V.: Sensor networks and grid middleware for laboratory monitoring. In: *e-Science and Grid Computing, 2005. First International Conference on*, 2005, S. 8 pp.–569

[Riv92] Rivest, R.: *The MD5 Message-Digest Algorithm*. United States, 1992

[RL05] Rodrigues, Rodrigo ; Liskov, Barbara: High Availability in DHTs: Erasure Coding Vs. Replication. In: *Fourth International Workshop on Peer-to-Peer Systems*, 2005

[Ros11] Rostan, Martin: Industrial Ethernet Technologies: Overview / EtherCAT Technology Group. 2011. – Forschungsbericht

[Sau05] Sauter, T.: Integration aspects in automation - a technology survey. In: *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005.* Bd. 2, 2005, S. 255–263

[Sau10] Sauter, T.: The Three Generations of Field-Level Networks 2014;Evolution and Compatibility Issues. In: *Industrial Electronics, IEEE Transactions on* 57 (2010), Nov, Nr. 11, S. 3585–3595. `http://dx.doi.org/10.1109/TIE.2010.2062473`. – DOI 10.1109/TIE.2010.2062473. – ISSN 0278–0046

[Sch09] Schreiner, Rüdiger: *Computernetzwerke: Von den Grundlagen zur Funktion und Anwendung*. 3. Auflage. Carl Hanser Verlag, 2009

[Sch10] Schweda, Sebastian: *Federal Administrative Court Rejects Competitors' Claim for Access to Telekom's Dark Fibre*. `http://merlin.obs.coe.int/iris/2010/3/article14.en.html`. Version: 2010

[See07] Seely, Devin: AN-1728 IEEE 1588 Precision Time Protocol Time Synchronization Performance / Texas Instruments. Version: 2007. `http://www.ti.com/lit/an/snla098/snla098.pdf`. 2007. – Forschungsbericht

[sem07] semiconductors freescale: Integrated Communications Processors: MPC8360E PowerQUICC II Pro Family. 2007. – Forschungsbericht

[Ser12] Sercos international e.v.: *Sercos III*. http://www.sercos.com/technology/sercos3.htm. Version: 2012

[SHB14] Shelby, Z. ; Hartke, K. ; Bormann, C.: *RFC 7252: The Constrained Application Protocol (CoAP)*. https://tools.ietf.org/html/rfc7252. Version: Juni 2014

[SKPK14] Schneider, John ; Kamiya, Takuki ; Peintner, Daniel ; Kyusakov, Rumen: *Efficient XML Interchange (EXI) Format 1.0 (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/2014/REC-exi-20140211/. Version: 2014

[SL11] Sauter, T. ; Lobashov, M.: How to Access Factory Floor Information Using Internet Technologies and Gateways. In: *Industrial Informatics, IEEE Transactions on* 7 (2011), Nov, Nr. 4, S. 699–712. http://dx.doi.org/10.1109/TII.2011.2166788. – DOI 10.1109/TII.2011.2166788. – ISSN 1551–3203

[SLK+12] Steinbach, T. ; Lim, Hyung-Taek ; Korf, F. ; Schmidt, T.C. ; Herrscher, D. ; Wolisz, A.: Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 2012. – ISSN 1090–3038, S. 1–5

[SLN05] Schacht, J. ; Laqua, H. ; Niedermeyer, H.: Synchronization of processes in a distributed real time system exemplified by the control system of the fusion experiment WENDELSTEIN 7-X. In: *Real Time Conference, 2005. 14th IEEE-NPSS*, 2005, S. 6 pp.–

[SLN06] Schacht, J. ; Laqua, H. ; Niedermeyer, H.: Synchronization of Processes in a Distributed Real Time System Exemplified by the Control System of the Fusion Experiment WENDELSTEIN 7-X. In: *Nuclear Science, IEEE Transactions on* 53 (2006), Aug, Nr. 4, S. 2187–2194. http://dx.doi.org/10.1109/TNS.2006.877930. – DOI 10.1109/TNS.2006.877930. – ISSN 0018–9499

[SNW+02] Schacht, Jörg ; Niedermeyer, Helmut ; Wiencke, Christian ; Hildebrandt, Jens ; Wassatsch, Andreas: A trigger-time-event system for the

W7-X experiment. In: *Fusion Engineering and Design* Bd. 60, 2002, S. 373–379

[SR06] Stutzbach, D. ; Rejaie, R.: Improving Lookup Performance Over a Widely-Deployed DHT. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006. – ISSN 0743–166X, S. 1 –12

[SS12] Schmidt, K.W. ; Schmidt, E.G.: Distributed Real-Time Protocols for Industrial Control Systems: Framework and Examples. In: *Parallel and Distributed Systems, IEEE Transactions on* 23 (2012), Nr. 10, S. 1856–1866. http://dx.doi.org/10.1109/TPDS.2011.300. – DOI 10.1109/TPDS.2011.300. – ISSN 1045–9219

[Sta88] Stankovic, J.A.: Misconceptions about real-time computing: a serious problem for next-generation systems. In: *Computer* 21 (1988), Oct, Nr. 10, S. 10–19. http://dx.doi.org/10.1109/2.7053. – DOI 10.1109/2.7053. – ISSN 0018–9162

[Stu06] Stutzbach, Reza Daniel ; R. Daniel ; Rejaie: Improving Lookup Performance Over a Widely-Deployed DHT., INFOCOM, 2006, S. 1–12

[SW05] Steinmetz, Ralf ; Wehrle, Klaus: *Peer-to-Peer Systems and Applications*. Springer, 2005

[TG13] TTA-Group: *TTEthernet - A Powerful Network Solution for All Purposes*. http://www.ttagroup.org/ttethernet/doc/TTEthernet_Article.pdf. Version: 2013

[TGK08] Trifa, V.M. ; Guinard, D. ; Koehler, M.: Messaging methods in a service-oriented architecture for industrial automation systems. In: *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 2008, S. 35–38

[Thi14] Thiel, Wolfgang: *Sie machen den Datentransfer sicherer: Höchste Anerkennung für zwei Rostocker Studenten*. http://www.uni-rostock.de/detailseite/news-artikel/sie-machen-den-datentransfer-sicherer/. Version: 2014

[Tos13] Toshiba: *TCnet Technology*. http://www.toshiba.co.jp/sis/en/seigyo/tcnet/technology.htm. Version: 2013

[TW14] Tanenbaum, Andrew S. ; Wetherall, David J.: *Computernetzwerke*. 5. Auflage. Pearson, 2014

[TWP+13] Thubert, P. ; Watteyne, T. ; Palattella, M.R. ; Vilajosana, X. ; Wang, Qin: IETF 6TSCH: Combining IPv6 Connectivity with Industrial Performance. In: *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, 2013, S. 541–546

[Ves14] Vesper, Martin: *Wenn der Kühlschrank twittert: Das Zusammenspiel von Haushaltsgeräten in unserer vernetzten Welt*. http://re-publica.de/session/wenn-kuehlschrank-twittert-zusammenspiel-haushaltsgeraeten-unserer-vernetzten-welt. Version: 2014

[W3C01] W3C: *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl.html. Version: 2001

[W3C07] W3C: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. http://www.w3.org/TR/soap12-part1/. Version: 2007

[W3C08] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/. Version: 2008

[WDHP+06] Windl, Ulrich ; Dalton, David ; Hewlett-Packard ; Martinec, Marc ; Institute, Josef S. ; Worley, Dale R.: *The NTP FAQ and HOWTO*. http://www.ntp.org/ntpfaq/NTP-s-algo.htm. Version: 2006

[Wil11] Wilamowski, Bogdan M.: *Industrial Communication Systems*. Hoboken, NJ : CRC Press, 2011

[Wir14] Wireshark: (2014). http://www.wireshark.org/

[Wol14] Wolfram: *Mathematica*. http://www.wolfram.com/mathematica/. Version: 2014

[xen14] *Xenomai: Real-Time Framework for Linux*. http://www.xenomai.org/. Version: 2014

[XSL+12] Xu, Hai-Mei ; Shi, Yan-Jun ; Liu, Yu-Lin ; Gao, Fu-Bing ; Wan, Tao: Integration of cloud computing and P2P: A future storage infrastructure. In: *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on*, 2012, S. 1489–1492

[YYXT08] Ye, Yunqi ; Yen, I-Ling ; Xiao, Liangliang ; Thuraisingham, B.: Secure, Highly Available, and High Performance Peer-to-Peer Storage Systems. In: *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*, 2008. – ISSN 1530–2059, S. 383 –391

[Zöb08] Zöbel, Prof. Dr. D.: *Echtzeitsysteme: Grundlagen der Planung*. Springer-Verlag, 2008. http://dx.doi.org/10.1007/978-3-540-76396-3. http://dx.doi.org/10.1007/978-3-540-76396-3

[ZMTG10] Zeeb, E. ; Moritz, G. ; Timmermann, D. ; Golatowski, F.: WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services. In: *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010. – ISSN 1530–2016, S. 1–8

[ZWLW11] Zhao, Guangyuan ; Wang, Zhiwei ; Li, Wei ; Wang, Ke: An Embedded Laboratory Security Monitoring System. In: *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on* Bd. 1, 2011, S. 395–398

Liste der Veröffentlichungen und Fachvorträge auf Tagungen

[ADS+13] Altmann, Vlado ; Danielis, Peter ; Skodzik, Jan ; Golatowski, Frank ; Timmermann, Dirk: Optimization of Ad Hoc Device and Service Discovery in Large Scale Networks. In: *18th IEEE Symposium on Computers and Communications (ISCC)*, 2013. – ISBN 978–1–4799–3755–4, S. 833–838. – DOI 10.1109/ISCC.2013.6755052

[ARS+13] Altmann, Vlado ; Rohrbeck, Jens ; Skodzik, Jan ; Danielis, Peter ; Timmermann, Dirk ; Roennau, Maik ; Ninnemann, Matthias: SWIFT: A Secure Web Domain Filter in Hardware. In: *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, 2013. – ISBN 978–1–4673–6239–9, S. 678–683. – DOI 10.1109/WAINA.2013.15

[ASD+14a] Altmann, Vlado ; Skodzik, Jan ; Danielis, Peter ; Mueller, Johannes ; Golatowski, Frank ; Timmermann, Dirk: A DHT-based Scalable Approach for Device and Service Discovery. In: *2th IEEE International Conference on Embedded and Ubiquitous Computing (EUC14)*, 2014. – ISBN 978–0–7695–5249–1, S. 97–103. – DOI 10.1109/EUC.2014.23

[ASD+14b] Altmann, Vlado ; Skodzik, Jan ; Danielis, Peter ; Pham Van, Nam ; Golatowski, Frank ; Timmermann, Dirk: Real-Time Capable Hardware-based Parser for Efficient XML Interchange. In: *9th IEEE/IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP14)*, 2014. – ISBN 978–1–4799–2581–0, S. 395–400. – DOI 10.1109/CSNDSP.2014.6923861

[ASGT12] Altmann, V. ; Skodzik, J. ; Golatowski, F. ; Timmermann, D.: Investi-

gation of the use of embedded Web Services in smart metering applications. In: *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012. – ISBN 978–1–4673–2419–9, S. 6172–6177. – DOI 10.1109/IECON.2012.6389071

[BWS+13] Bosch, H.-S. ; Wolf, R.C. ; Schacht, Joerg ; Skodzik, Jan ; Timmermann, Dirk ; et. al.: Technical challenges in the construction of the steady-state stellarator Wendelstein 7-X. In: *Nuclear Fusion* 53 (2013), Nr. 12, 126001. `http://stacks.iop.org/0029-5515/53/i=12/a=126001`. – DOI 10.1088/0029-5515/53/12/126001

[DAS+15] Danielis, Peter ; Altmann, Vlado ; Skodzik, Jan ; Schweissguth, Eike B. ; Golatowski, Frank ; Timmermann, Dirk: Emulation of SDN-Supported Automation Networks. In: *20th IEEE International Conference on Emerging Technologies and Factory Automation (IEEE ETFA)*, 2015

[DKW+09] Danielis, P. ; Kubisch, S. ; Widiger, H. ; Rohrbeck, J. ; Altman, V. ; Skodzik, J. ; Timmermann, D. ; Bahls, Thomas ; Duchow, D.: Trust-by-Wire in Packet-Switched IPv6 Networks: Tools and FPGA Prototype for the IPclip System. In: *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, 2009. – ISBN 978–1–4244–2308–8, S. 1–2. – DOI 10.1109/CCNC.2009.4785006

[DSA+14] Danielis, Peter ; Skodzik, Jan ; Altmann, Vlado ; Schweissguth, Eike B. ; Golatowski, Frank ; Timmermann, Dirk ; Schacht, Jörg: Survey on Real-Time Communication Via Ethernet in Industrial Automation Environments. In: *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2014)*, 2014. – ISBN 978–1–4799–4845–1, S. 1–8

[DSA+15a] Danielis, Peter ; Skodzik, Jan ; Altmann, Vlado ; Kappel, Benjamin ; Timmermann, Dirk: Extensive Analysis of the Kad-based Distributed Computing System DuDE . In: *IEEE Symposium on Computers and Communications (IEEE ISCC)*, 2015

[DSA+15b] Danielis, Peter ; Skodzik, Jan ; Altmann, Vlado ; Lender, Lennard ; Timmermann, Dirk: Dynamic Search Tolerance at Runtime for Lookup Determinism in the DHT-based P2P Network Kad. In: *12th IEEE Consumer*

Communications & Networking Conference (CCNC), 2015. – ISBN 978–1–4799–6389–8, S. 357–362

[DSL+10] Danielis, Peter ; Skodzik, Jan ; Lorenz, Carsten ; Timmermann, Dirk ; Bahls, Thomas ; Duchow, Daniel: P-DONAS: A Prototype for a P2P-based Domain Name System in Access Networks. In: *DATE University Booth*, 2010

[DSR+11] Danielis, Peter ; Skodzik, Jan ; Rohrbeck, Jens ; Altmann, Vlado ; Timmermann, Dirk ; Bahls, Thomas ; Duchow, Daniel: Using Proximity Information between BitTorrent Peers: An Extensive Study of Effects on Internet Traffic. In: *International Journal on Advances in Systems and Measurements* 4 (2011), S. 212–221. – ISSN 1942–261x

[DSTB11] Danielis, Peter ; Skodzik, Jan ; Timmermann, Dirk ; Bahls, Thomas: Impacts of Improved Peer Selection on Internet Traffic in BitTorrent Networks. In: *International Conference on Internet Monitoring and Protection (ICIMP)*, 2011. – ISBN 978–1–61208–004–8, S. 8–13

[RAP+11] Rohrbeck, Jens ; Altmann, Vlado ; Pfeiffer, Stefan ; Danielis, Peter ; Skodzik, Jan ; Timmermann, Dirk ; Ninnemann, Matthias ; Rönnau, Maik: The Secure Access Node Project: A Hardware-Based Large-Scale Security Solution for Access Networks. In: *International Journal On Advances in Security* 4 (2011), S. 234–244. – ISSN 1942–2636

[SAD+14] Skodzik, Jan ; Altmann, Vlado ; Danielis, Peter ; Koal, Moritz ; Timmermann, Dirk: An Optimized WS-Eventing for Large-Scale Networks. In: *8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE)*, 2014. – ISBN 978–1–4799–4845–1, S. 1–6. – DOI 10.1109/ETFA.2014.7005248

[SADT14] Skodzik, Jan ; Altmann, Vlado ; Danielis, Peter ; Timmermann, Dirk: A Kad Prototype for Time Synchronization in Real-Time Automation Scenarios. In: *World Telecommunications Congress (WTC) 2014*, 2014. – ISBN 978–3–8007–3602–7, S. 1–6

[SAW+13] Skodzik, Jan ; Altmann, Vlado ; Wagner, Benjamin ; Danielis, Peter ; Timmermann, Dirk: A Highly Integrable FPGA-Based Runtime-Configurable Multilayer Perceptron. In: *Advanced Information Networking*

and Applications (AINA), 2013 IEEE 27th International Conference on, 2013. – ISBN 978–1–4673–5550–6, S. 429–436. – DOI 10.1109/AINA.2013.19

[SDA+11a] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Rohrbeck, Jens ; Timmermann, Dirk ; Bahls, Thomas ; Duchow, Daniel: DuDE: A Distributed Computing System using a Decentralized P2P Environment. In: *36th IEEE LCN, 4th International Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI)*, 2011. – ISBN 978–1–61284–926–3, S. 1060–1067. – DOI 10.1109/LCN.2011.6115162

[SDA+11b] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Rohrbeck, Jens ; Timmermann, Dirk ; Bahls, Thomas ; Duchow, Daniel: DuDE: A Prototype for a P2P-based Distributed Computing System. In: *36th IEEE LCN, 4th International Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI)*, 2011

[SDA+15a] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Konieczek, Björn ; Schweissguth, Eike B. ; Golatowski, Frank ; Timmermann, Dirk: CoHaRT: Deterministic Transmission of Large Data Amounts using CoAP and Kad. In: *2015 IEEE International Conference on Industrial Technology (IEEE ICIT)*, 2015. – ISBN 978–1–4799–7799–4, S. 1851–1856

[SDA+15b] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Schweissguth, Eike B. ; Timmermann, Dirk: PSP-Auto: An DHT-based Storage and Retrieve System for Automation. In: *9th Annual IEEE International Systems Conference (IEEE SysCon)*, 2015. – ISBN 978–1–4799–5927–3, S. 853–858

[SDAT13a] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Timmermann, Dirk: Extensive Analysis of a Kad-based Distributed Storage System for Session Data. In: *18th IEEE Symposium on Computers and Communications (IEEE ISCC)*, 2013. – ISBN 978–1–4799–3755–4, S. 489–494. – DOI 10.1109/ISCC.2013.6754994

[SDAT13b] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Timmermann, Dirk: Time Synchronization in the DHT-based P2P Network Kad for Real-Time Automation Scenarios. In: *2nd IEEE WoWMoM Workshop on the Internet of Things: Smart Objects and Services (IoT-SoS)*, 2013. – ISBN 978–1–4673–5828–6, S. 1–6. – DOI 10.1109/WoWMoM.2013.6583490

[SDAT14] Skodzik, Jan ; Danielis, Peter ; Altmann, Vlado ; Timmermann, Dirk: HaRTKad: A Hard Real-Time Kademlia Approach. In: *11th IEEE Consumer Communications & Networking Conference (CCNC)*, 2014. – ISBN 978–1–4799–2356–4, S. 566–571. – DOI 10.1109/CCNC.2014.6866588

[ScSk12] Schacht, J. ; Skodzik, J.: Multifunction-timing card lTTEV2 for CoDaC systems of Wendelstein 7-X. In: *Real Time Conference (RT), 2012 18th IEEE-NPSS*, 2012. – ISBN 978–1–4673–1082–6, S. 1–6. – DOI 10.1109/RTC.2012.6418207

Betreute studentische Arbeiten

[Gub13] Gubitz, Robert: *Entwurf eines echtzeitfähigen Peer-to-Peer-Clients*, Universität Rostock, Masterarbeit, 2013

[Kap14] Kappel, Benjamin: *Untersuchung der Skalierbarkeit eines P2P-basierten verteilten Rechensystems*, Universität Rostock, Bachelorarbeit, 2014

[Koa13a] Koal, Moritz: *Integration von WS-Eventing in eine dezentrale P2P-Umgebung*, Universität Rostock, Bachelorarbeit, 2013

[Koa13b] Koal, Moritz: *Synergie von Web Services und P2P*, Universität Rostock, Literaturarbeit, 2013

[Kru13] Kruse, Sebastian: *Evaluierung und Optimierung der WS-Discovery für hochskalierende Netzwerke*, Universität Rostock, Masterarbeit, 2013

[Kut11a] Kutzner, Jonas: *Umsetzung der Taktrückgewinnung aus eeine DMC codierten Signal in VHDL*, Universität Rostock, Großer Beleg, 2011

[Kut11b] Kutzner, Jonas: *UmsetzungsmöglUmsetzung der Taktrückgewinnung aus codierten Signalen in Hard- und Software*, Universität Rostock, Kleiner Beleg, 2011

[Len14] Lender, Lennard: *Performance-Analyse eines dynamischen Suchtoleranzalgorithmus für das Kad-Netzwerk*, Universität Rostock, Bachelorarbeit, 2014

[Mül13] Müller, Johannes: *Konzipierung und Implementierung eines dezentralen DHT-basierten Verfahrens für Geräte- und Service-Discovery*, Universität Rostock, Bachelorarbeit, 2013

[Nie15] Niemann, Christoph: *Konzeptionierung und prototypische Umsetzung eines integrierten Managements zur Zuverlässigkeitssteigerung eingebetteter Systeme*, Universität Rostock, Masterarbeit, 2015

[Put14] Puttnies, Henning: *Konzipierung und Implementierung eines Smart Metering-Systems zur automatisierten Zählerwerterfassung und Übermittlung*, Universität Rostock, Masterarbeit, 2014

[Sch13a] Schweissguth, Eike B.: *Echtzeitdatenübertragung in Automatisierungsumgebungen*, Universität Rostock, Literaturarbeit, 2013

[Sch13b] Schweissguth, Eike B.: *Umsetzung eines echtzeitfähigen Datenaustausches mit hoher Ausfallsicherheit in Kad-Netzwerken*, Universität Rostock, Bachelorarbeit, 2013

[Sch15] Schweissguth, Eike B.: *Nutzung von SDN in der MiniNet-Umgebung zur Optimierung einer Beispielapplikation*, Universität Rostock, Projektarbeit, 2015

[Van11] Van, Nam P.: *Entwicklung Künstlicher Neuronaler Netze und deren Umsetzung in Hardware*, Universität Rostock, Literaturarbeit, 2011

[Van12] Van, Nam P.: *Umsetzung eeine mehrlagigen Perzeptrons auf einem Virtex ML 605 Evaluation Board*, Universität Rostock, Bachelorarbeit, 2012

[Van13] Van, Nam P.: *Konzipierung und Implementierung eines Hardware-basierten EXI-Parsers für ein echtzeitfähiges Web Service-Profil*, Universität Rostock, Masterarbeit, 2013

[Wal13] Wall, Arne: *Umsetzung eines denzentralen Synchronisations- und arbitrierten Medienzugriffsmechanimus' für das Kad-Netzwerk*, Universität Rostock, Bachelorarbeit, 2013

[Zie12] Zierke, Robert: *Qualitätsverbessertes Peer-to-Peer-basiertes IPTV durch Bandbreitenersparnis im Upload*, Universität Rostock, Bachelorarbeit, 2012

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die Dissertation zum Thema

Peer-to-Peer-Technologie in der Automatisierung

vollkommen selbst verfasst habe und zu ihrer Anfertigung keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe.

Rostock, den 04.02.2015

Thesen

1. Der Einsatz von Peer-to-Peer (P2P)-Technologie in der Automatisierung ist möglich und sinnvoll, was das entwickelte PSP- und besonders das PSP-Auto-Projekt zum verteilten Speichern darlegen. Hierbei wird eine weiche Echtzeit erreicht, da die Kommunikation noch nicht echtzeitfähig ist.
2. Die entwickelten Systeme zum verteilten Speichern weisen eine sehr gute Skalierbarkeit und Zuverlässigkeit auf, da sie durch das verwendete Kademlia-Protokoll keinen zusätzlichen Single Point of Failure (SPoF) aufweisen. Ein zusätzlicher Vorteil ist die selbstverwaltende Eigenschaft der Netzwerke, wodurch Kontaktinformationen automatisch aktuell gehalten werden.
3. Im Bereich der industriellen Automatisierung etablieren sich Industrial Ethernet(IE)-Lösungen, welche die Feldbusse durch Standard-Ethernet-Komponenten ablösen, um eine totale horizontale und vertikale Integration zu ermöglichen.
4. Jedoch sind die IE-Lösungen nicht ausreichend für zukünftige Herausforderungen, bei denen immer mehr vernetzte Geräte in Echtzeit miteinander kommunizieren müssen, da sie z.T. schlecht skalieren, eine geringe Flexibilität aufweisen oder eine geringe Ausfallsicherheit bieten.
5. Es existiert keine IE-Lösung, die ohne zentrale Instanz und ohne proprietäre Protokolle oder Hardware harte/isochrone Echtzeit erreicht. Dies macht die Lösung teuer und aufwändiger in der Integration. Zusätzlich ist dadurch die Interoperabilität stark eingeschränkt.
6. Als neuer Ansatz ist es möglich P2P-Technologie derartig zu modifizieren, dass sie in der Automatisierung einsetzbar ist, was in einem komplett dezentralen System mit dem Namen HaRTKad (**Ha**rd **R**eal-**T**ime **Kad**emlia) resultiert.
7. HaRTKad ermöglicht harte Echtzeit mit den Attributen der Flexibilität, Ausfallsicherheit, Skalierbarkeit und der Vermeidung von Propritarität, da das Verfahren

direkt in der Applikationsschicht realisiert wurde.

8. Auch die Synchronisation, die HaRTKad als TDMA-Ansatz benötigt, lässt sich mittels P2P-Technologie realisieren und erreicht zu etablierten Verfahren wie dem Network Time Protocol (NTP) und dem Precision Time Protocol (PTP) konkurrenzfähige Zeiten, da es die Synchronisation mit Hilfe von anderen Knoten beschleunigt. Der Grad der Beschleunigung kann vom Nutzer gewählt werden und sollte an die Anforderungen der Applikation angepasst sein.
9. Die Synchronisation ist deterministisch und vermeidet einen sogenannten „Gossip"-Ansatz („Tratschen") und kann daher in der Automatisierung, welche harte Echtzeit benötigt, eingesetzt werden.
10. HaRTKad kann praktisch eingesetzt werden, da die praktischen Ergebnisse bereits die Anforderungen an verschiedene Echtzeitklassen wie z.B. eine Zykluszeit von 10 ms (Prozesssteuerung) mit bis zu 621 Knoten erfüllt. Dies wird durch parallele Kommunikation ermöglicht, bei der aber in der darüber liegenden Applikation sichergestellt werden muss, dass kein Knoten in zwei parallele Prozesse eingebunden ist.
11. HaRTKad kann auch als Middleware für andere Applikationen dienen. HaRTKad realisiert dabei die deterministische Datenübertragung und somit die Echtzeitkommunikation. Die Echtzeitfähigkeit des Systems wird mittels des verwendeten Echtzeitbetriebssystems (z.B. FreeRTOS) realisiert.
12. Als ein Beispiel wurde WS-Eventing verwendet, das HaRTKad als Middleware nutzt. Somit ist es möglich, den bestehenden WS-Eventing-Benachrichtigungsalrogithmus hochskalierbar und echtzeitfähig zu machen.
13. Ein Standard WS-Eventing ist für eine geringe Anzahl an Knoten dem optimierten Ansatz auf Grund seiner Komplexität in der zeitlichen Performance überlegen. Mit steigender Anzahl jedoch skaliert der optimierte Ansatz deutlich besser und ist dem Standard überlegen. Bei einem zuverlässigen (mit Acknowledgement) WS-Eventing ist der optimierte WS-Eventing-Ansatz von Anfang an überlegen.
14. Es ist sinnvoll, die Benachrichtigung der Knoten derart zu gestalten, dass zuerst die Knoten mit der geringsten Ausfallwahrscheinlichkeit informiert werden, da so nachfolgende Knoten mit höherer Wahrscheinlichkeit nicht durch einen Ausfall des benachrichtigenden Knotens betroffen sind.

15. Ein weiteres entwickeltes System namens CoHaRT ist ein Beispiel, bei dem HaRTKad ebenfalls als Middleware dient. CoAP sitzt oberhalb von HaRTKad und ermöglicht die Übertragung und Interpretierung von großen Datenmengen. Somit sind deterministische Datenströme im Umfeld der industriellen Automatisierung möglich.
16. Durch die Verwendung von den sogenannten CoAP-Blockoptionen kann eine Effizienzsteigerung erreicht werden, sodass diese Option bei größeren Datenmengen eingesetzt werden sollte.
17. Es hängt von der Anwendung und dessen Anforderungen ab, inwieweit HaRTKad einsetzbar ist. Dies wird z.B. durch den Vergleich mit einer eigens entwickelten proprietären Lösung gezeigt. Die proprietäre Lösung ist in Zeitauflösung und Synchronisation HaRTKad überlegen, jedoch auf Kosten der Propritärität und Komplexität, da bei der proprietären Lösung andere Probleme in den Vordergrund rücken.
18. HaRTKad und die aufbauenden Applikationen können durch zukünftige Weiterentwicklung in der Anzahl der unterstützten Knoten noch weiter optimiert werden. Einen wesentlicher Punkt stellen dabei die Software Defined Networks (SDNs) dar, welche durch ihr Netzwerk-Topologiewissen eine potentielle Vervielfachung der unterstützten Knoten ermöglichen könnten.

Kurzreferat

Peer-to-Peer (P2P)-Technologie hat sich seit ihrem ursprünglichen Einsatz für das Filesharing weiterentwickelt und Einzug in neue Einsatzgebiete, wie z.B. das Videostreaming, erhalten. Ein Szenario ist die Anwendung für das verteilte Speichern von Daten, welche für das Umfeld der Automatisierung angepasst wurde. Das System nutzt die positiven Eigenschaften der P2P-Technologie, wie die hohe Flexibilität, Skalierbarkeit und Ausfallsicherheit. Ob P2P-basierte Anwendungen auch für den Einsatz im Umfeld der industriellen Automatisierung sinnvoll sind, wurde in dieser Arbeit untersucht. Erhöhte Anforderungen, wie z.B. harte Echtzeit des Systems und der Kommunikation, gilt es mit Hilfe des alternativen Netzwerkparadigma der P2P-Netze zu realisieren. Gerade in Bezug auf die Zukunftsfähigkeit weisen etablierte Lösungen im industriellen Umfeld Mängel auf, die durch die intrinsischen Eigenschaften von P2P beseitigt werden könnten.

Das im Laufe dieser Arbeit entstandene Verfahren HaRTKad mit seinem für das Umfeld neue Netzwerkparadigma erfüllt diese Anforderungen, womit es sich in diesem Umfeld einsetzen lässt. Ein anschließender Vergleich mit einer proprietären Lösung zeigt, dass es je nach Anforderungen durch gegebenen Anwendungen bestimmt werden muss, was zu bevorzugen ist. Generell ist ein nicht proprietärer Ansatz zu bevorzugen, da sich dieser nahtlos in bestehende Strukturen einbinden lässt; insbesondere hinsichtlich der Interoperabilität. Zusätzlich entwickelte Applikationen, die HaRTKad nutzen, zeigen, dass HaRTKad auch als Middleware genutzt werden kann. Applikationen aus dem Umfeld der SOAP-basierten Web Services und des RESTful Protokolls CoAP wurden gewählt, um die positiven Eigenschaften von HaRTKad zu demonstrieren. HaRTKad erweitert die genannten Beispiele um harte Echtzeitfähigkeit, wobei der Funktionsumfang der Applikationen nicht eingeschränkt wird. Alle entwickelten Lösungen konnten als Prototypen realisiert werden, was die Machbarkeit der Konzepte basierend auf P2P-Technologie beweist. Zusätzlich untermauern sie die zuvor getroffenen theoretischen Annahmen.

Kurzreferat

Abstract

Beyond the original use of Peer-to-Peer (P2P) technology for file sharing, it has been further developed for other use cases, e.g., video streaming. First, a self-developed application for P2P-based distributed storage has been developed, which has been additionally modified for the field of automation. The system utilizes the positive properties of P2P technology like the high flexibility, high scalability, and high resilience. In this thesis, the operation of P2P technology in the industrial automation with higher requirements in real-time bhavior has been investigated as well. Established solutions in the field of industrial automation lack sustainability, which could be solved by the intrinsic features of P2P technology. Therefore, this thesis clarifies if and how P2P technology can be used in the field of automation with hard real-time requirements.

The resulting developed system HaRTKad with its new network paradigm for the domain of automation fulfills the mentioned requirements. In conclusion, a comparison with a proprietary solution shows that depending on the application the user has to decide, which solution has to be preferred. Generally, a non proprietary approach should be preferred as it can be integrated into existing structures without high effort also in terms of interoperability. Additional applications, which use HaRTKad as a middleware, have been developed. Two applications from the domain of SOAP-based Web Services and the RESTful protocol CoAP have been chosen to demonstrate the positive properties of HaRTKad. HaRTKad extends the mentioned examples by hard real-time capability without limiting the functionality of the applications.

All results are proven by prototypes, which show the feasibility of such a P2P-based system in practice. Additionally, it substantiates the theoretical assumptions.