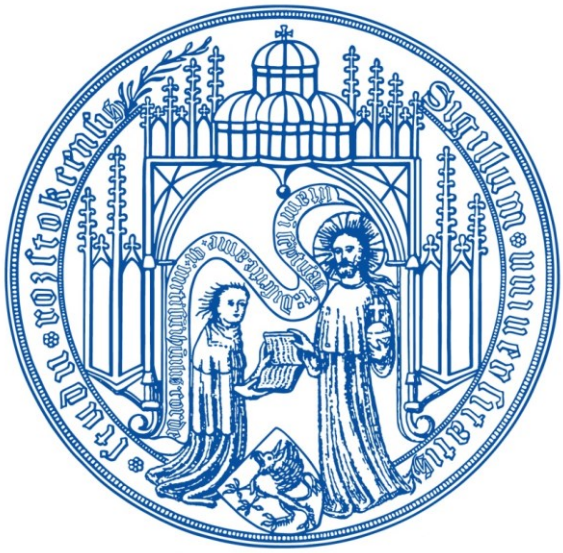

Untersuchung des Einsatzes von Web Service-Technologien in Automationsnetzwerken

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

vorgelegt von

Altmann, Vlado, geb. am 01.02.1983 in Taganrog,

aus Rostock

urn:nbn:de:gbv:28-diss2015-0045-2

Gutachter:

- Prof. Dr. Dirk Timmermann (Universität Rostock, Fakultät für Informatik und Elektrotechnik, Institut für Angewandte Mikroelektronik und Datentechnik)
- Prof. Dr. Wolfgang Kastner (TU Wien, Fakultät für Informatik, Institut für Rechnergestützte Automation)
- Prof. Dr. Heiko Krumm (TU Dortmund, Fakultät für Informatik, Lehrstuhl für Praktische Informatik)

Tag der Einreichung: 02.10.2014

Tag der Verteidigung: 09.01.2015

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

6LoWPAN ..... IPv6 over Low power Wireless Personal Area Network
AES ..... Advanced Encryption Standard
ANSI ..... American National Standards Institute
AP ..... Access Point
ARCNET ..... Attached Resource Computer Network
ARM ..... Advanced RISC Machines
ARP ..... Address Resolution Protocol
ASCII ..... American Standard Code for Information Interchange
ASHRAE ..... American Society of Heating, Refrigeration, and Air-Conditioning Engineers
AV ..... Audio/Video
AXI ..... Advanced Micro Controller Bus Architecture Advanced eXtensible Interface
BACnet ..... Building Automation and Control Networks
BCD ..... Binary Coded Decimal
BIBB ..... BACnet Interoperability Building Blocks
BSI ..... Bundesamt für Sicherheit in der Informationstechnik
BSS ..... Basic Service Set
CCo ..... Central Coordinator
CEA ..... Consumer Electronics Association
CoAP ..... Constrained Application Protocol
COSEM ..... Companion Specification for Energy Metering
COV ..... Change of Value
CR ..... Carriage Return
CSMA/CA ..... Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD ..... Carrier Sense Multiple Access/Collision Detection
DES ..... Data Encryption Standard
DHCP ..... Dynamic Host Configuration Protocol
DHT ..... Distributed Hash Table
DIF ..... Data Information Field
DIN ..... Deutsches Institut für Normung
DLMS ..... Device Language Message Specification
DPWS ..... Devices Profile for Web Services
DSP ..... Digital Signal Processor
EEP ..... EnOcean Equipment Profile
EHS ..... European Home Systems
EIA ..... Electronic Industries Alliance
EIB ..... Europäischer Installationsbus
EN ..... Europäische Norm
ETS ..... Engineering Tool Software
EXI ..... Efficient XML Interchange
FFD ..... Full Function Devices
FIFO ..... First In First Out
FPGA ..... Field Programmable Gate Array
FTP ..... File Transfer Protocol
GSM ..... Global System for Mobile Communications
HDLC ..... High Level Data Link Control Protocol
HTTP ..... Hypertext Transfer Protocol
HTTPS ..... Hypertext Transfer Protocol Secure
IEC ..... International Electrotechnical Commission

IEEE ..... Institute of Electrical and Electronics Engineers
IOB ..... Interoperabilitätsbereich
IoT ..... Internet of Things
IP. ..... Internet Protocol
IR ..... Infrarot
ISM ..... Industrial, Scientific and Medical
ISO ..... International Organization for Standardization
ISP ..... Internet Service Provider
ITU ..... International Telecommunication Union
ITU-T ..... International Telecommunication Union Telecommunication Standardization Sector
LF ..... Line Feed
LN ..... Logical Name
LON ..... Local Operation Network
LSB ..... Least Significant Bit
LUT ..... Lookup Table
LWL ..... Lichtwellenleiter
M2M ..... Machine-to-Machine
MAC ..... Media Access Control
M-Bus ..... Meter Bus
MD5 ..... Message-Digest Algorithm 5
MIME ..... Multipurpose Internet Mail Extensions
MIMO ..... Multiple Input Multiple Output
MMIO ..... Memory Mapped Input/Output
MSB ..... Most Significant Bit
MTU ..... Maximum Transmission Unit
NFC ..... Near Field Communication
NID ..... Network Identifier
NIST ..... National Institute of Standards and Technology
NMK ..... Network Membership Key
OASIS ..... Organization for the Advancement of Structured Information Standards
OBIS ..... Object Identification System
OSI ..... Open Systems Interconnection
P2P ..... Peer-to-Peer
PAN ..... Personal Area Network
PC ..... Personal Computer
PICS ..... Protocol Implementation Conformance Statement
PL ..... Powerline
PLC ..... Power Line Communication
QName ..... Qualified Name
QoS ..... Quality of Service
RAM ..... Random Access Memory
REST ..... Representational State Transfer
RF ..... Radio Frequency
RFC ..... Request for Comments
RFD ..... Reduced Function Device
RISC ..... Reduced Instruction Set Computer
ROM ..... Read Only Memory
RTS/CTS ..... Request to Send / Clear to Send
SCPT ..... Standard Configuration Property Type
SFPT ..... Standard Functional Profile Type

SML ..... Smart Message Language
SN ..... Short Name
SNVT ..... Standard Network Variable Type
SOA ..... Service-Oriented Architecture
SOAP ..... Simple Object Access Protocol
SoC ..... System-on-Chip
SPoF ..... Single Point of Failure
SRD ..... Short Range Devices
TCP ..... Transmission Control Protocol
TDMA ..... Time Division Multiple Access
TLS ..... Transport Layer Security
TP ..... Twisted Pair
UDDI ..... Universal Description, Discovery and Integration
UDP ..... User Datagram Protocol
UPnP ..... Universal Plug and Play
URI ..... Uniform Resource Identifier
URL ..... Uniform Resource Locator
UUID ..... Universally Unique Identifier
VHDL ..... Very High Speed Integrated Circuit Hardware Description Language
VIF ..... Value Information Field
VLAN ..... Virtual Local Area Network
W3C ..... World Wide Web Consortium
WLAN ..... Wireless Local Area Network
WS ..... Web Service
WSDL ..... Web Services Description Language
WWW ..... World Wide Web
XML ..... Extensible Markup Language
XSD ..... XML Schema Definition
ZCL ..... ZigBee Cluster Library
ZDO ..... ZigBee Device Object

Danksagung

Ich möchte an dieser Stelle die Gelegenheit ergreifen mich bei all denen zu bedanken, die mich bei der Entstehung dieser Arbeit unterstützt haben. Dabei möchte ich meiner Familie danken, die mir immer beigestanden und mich ermutigt hat. Ohne diese Unterstützung wäre die Verfassung dieser Forschungsarbeit nicht möglich gewesen.

Mein großer Dank gilt weiterhin meinem Doktorvater Professor Dirk 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, gab mir wichtigen inhaltlichen Ratschlag und trug damit maßgeblich zu dem Gelingen dieser Forschungsarbeit bei.

Darüber hinaus danke ich Jan Skodzik, Dr. Peter Danielis und Dr. Frank Golatowski, die mir große Unterstützung während der Arbeit am Institut für Angewandte Mikroelektronik und Datentechnik sowie beim Entstehen dieser Forschungsarbeit geleistet haben. Ich möchte mich weiterhin bei allen Kollegen und Studenten, die an dieser Arbeit beteiligt waren, danken. Dabei möchte ich besonders Jens Rohrbeck, Johannes Müller, Nam Pham Van, Sebastian Kruse, Christoph Schulz und Achim Dörre 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.

1. Einführung

Die rasante Entwicklung und Popularität des Internets hat zu einer Verbreitung und Durchdringung der Kommunikationstechnologien in nahezu allen Bereiche des Lebens beigetragen. Die Internettechnologien werden allmählich auch in den Bereichen eingesetzt, in denen früher noch proprietäre Kommunikationstechnologien vorherrschten. Eine solche Technologie stellen die Web Services dar. Ein wichtiger Vorteil der Internet-Technologien ist, dass sie ohne Lizenzkosten eingesetzt werden können. In vielen Bereichen wie z.B. der Gebäudeautomation finden derzeit proprietäre Busprotokolle Verwendung. Diese sind dadurch gekennzeichnet, dass sie keine Interoperabilität untereinander, aber oft auch zwischen verschiedenen Herstellern einer Technologie aufweisen. So hat die Entwicklung von Smart Metering in letzter Zeit dazu geführt, dass mehrere neue proprietäre Protokolle ohne Interoperabilität zueinander entwickelt wurden (wie z.B. METERS&MORE, SML und andere).

Im Rahmen dieser Arbeit wird die Einsatzfähigkeit von Web Services in Automationsszenarien, insbesondere im Bereich der Gebäudeautomation bzw. ihrer Teilbereiche wie Smart Home und Smart Metering, untersucht. Hierbei soll die Machbarkeit des Ansatzes sowie die Konkurrenzfähigkeit der Web Services im Vergleich zu den bestehenden Kommunikationstechnologien der Gebäudeautomation ermittelt werden.

1.1. Zielsetzung der Arbeit

Einsatz von Web Services in Gebäudeautomation [1]**:**

Die aktuelle Situation in der Gebäudeautomation ist durch eine Koexistenz einer großen Anzahl von proprietären und offenen Standards sowohl für eine drahtgebundene als auch eine drahtlose Kommunikation gekennzeichnet. Die Standards zeigen eine geringe oder gar keine Interoperabilität zueinander. Es ist deswegen äußerst schwierig, eine herstellerübergreifende Lösung zu realisieren und dabei einen nachhaltigen ganzheitlichen Ansatz zu verfolgen. In dieser Arbeit wird der Einsatz von Web Services als einer offenen und weit verbreiteten Internet-Technologie für die Erstellung von heterogenen Netzwerken in der Gebäudeautomation untersucht. Es werden darüber hinaus Mechanismen vorgeschlagen, die die Konkurrenzfähigkeit der Web Services im Vergleich zu den bestehenden Technologien verbessern, sodass die Umsetzung des vorgeschlagenen Ansatzes mit keinen Nachteilen verbunden ist.

Einsatz von Web Services in Automationsnetzwerken mit Echtzeitanforderungen [2]**:**

Die Nutzung von Extensible Markup Language (XML) ist in den meisten Computer-Systemen zum Standard geworden. Die Vorteile der XML-Dokumente sind einfache

Handhabung, dynamische Anpassung auf nahezu alle Anforderungen, Verfügbarkeit von Parsern für die meisten Programmiersprachen und die Menschenlesbarkeit. Trotz vieler Vorteile von XML findet es vorwiegend in softwarebasierten Systemen Verwendung. Der entscheidende Nachteil ist hierbei die Größe der XML-Dokumente sowie ein textbasiertes Parsen. In Automationssystemen, insbesondere mit Echtzeitanforderungen, werden oft hohe Verarbeitungsgeschwindigkeiten gefordert. Dies setzt eine kompakte Darstellung der Daten und schnelles Parsen voraus. In solchen Szenarien schaffen hardwarebasierte Lösungen oft Abhilfe und beschleunigen die Datenverarbeitung. Efficient XML Interchange (EXI) wurde dafür entwickelt, den Einsatz von XML in Deeply Embedded-Systemen zu ermöglichen. EXI erlaubt hohe Kompressionsraten, ohne zusätzlich die Rechenlast zu erhöhen. Die EXI-Verarbeitung wurde bis jetzt jedoch nur in softwarebasierten Systemen untersucht. Da eine sehr hohe Verarbeitungsgeschwindigkeit im Rahmen dieser Arbeit angestrebt wird, um den Einsatz von Web Services in Systemen mit harten Echtzeitanforderungen wie z.B. Motion Control zu ermöglichen, wird eine Untersuchung und prototypische Implementierung eines hardwarebasierten EXI-Parsers vorgenommen und mit softwarebasierten Lösungen verglichen. Dadurch soll gezeigt werden, dass Web Services zukünftig sogar für Anwendungen mit harten Echtzeitanforderungen und kurzen Reaktionszeiten eingesetzt werden können.

Untersuchung der Skalierbarkeit von Web Services in Automationsszenarien [3] [4] [5]**:**

Um den vollen Funktionsumfang von Web Services in Automationsszenarien nutzen zu können, müssen diese für große Netzwerke gut skalieren. Ein Skalierbarkeitsproblem kann entstehen, wenn in einem Netzwerk eine gleichzeitige Kommunikation zwischen mehreren Netzwerkteilnehmern notwendig ist, die von der Anzahl dieser abhängig ist. Ein solches Verhalten weist die Service Discovery aus. Beim Service Discovery handelt es sich um die Suche nach Services in einem Netzwerk. Dieses ist ein essentieller Bestandteil der Web Service-Technologie und eine wichtige Funktion für Automationsnetzwerke. Es ermöglicht, automatisiert gewünschte Geräte bzw. Services in einem unbekannten Netzwerk zu finden. Da die Anzahl der Service Provider mit der Netzwerkgröße ansteigt, sollen Mechanismen gefunden werden, die Service Discovery unabhängig von der Anzahl der Geräte im Netzwerk skalieren lässt. Hierfür wird im Rahmen dieser Arbeit das Verhalten von Service Discovery in großen Netzwerken untersucht. Dabei werden die kritischen Abläufe identifiziert und analysiert. Anschließend werden mehrere Ansätze zur Optimierung des Service Discovery-Prozesses erarbeitet. Diese werden sowohl in einer Simulation als auch praktisch umgesetzt und evaluiert.

1.2. Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt gegliedert. In Kapitel 2 werden die Grundlagen erklärt, die das Verständnis der Arbeit erleichtern sollen. In Kapitel 0 wird der Einsatz von Web Services in der Gebäudeautomation untersucht. Daraufhin folgt in Kapitel 4 eine Evaluierung, ob die Web Service-Nachrichten in Echtzeit mit Hardware-Unterstützung verarbeitet werden können. Kapitel 5 ist der Optimierung des Geräte- und Service-Discovery für große Netzwerke gewidmet. Die vorgeschlagenen Optimierungen basieren auf dem Discovery-Standard und sind rückwärtskompatibel. Im Kapitel 6 wird ein alternativer Discovery-Mechanismus, basierend auf Distributed Hash Table, vorgeschlagen. Die Arbeit wird in Kapitel 7 zusammengefasst und ein Ausblick wird gegeben. In Abbildung 1 ist die Struktur dieser Arbeit mit den hervorgehobenen wesentlichen eigenen Forschungsbeiträgen dargestellt.

Abbildung 1: Struktur der Arbeit. Die wesentlichen eigenen Forschungsbeiträge sind fett hervorgehoben.

2. Grundlagen

In diesem Kapitel werden die Grundlagen erklärt, die für das Verständnis der vorgelegten Arbeit notwendig sind.

2.1. ISO/OSI-Referenzmodell

Die International Organization for Standardization (ISO) hat mit dem Open Systems Interconnection (OSI)-Referenzmodell ein Schichtenmodell entwickelt, welches die Netzwerkkommunikation in 7 Schichten aufteilt. Das Modell beschreibt eine allgemeingültige Weise, wie die Rechner vernetzt werden sollen. Jedes Protokoll kann demnach einer bestimmten Schicht im Referenzmodell zugeordnet werden [6]. Da Rechnerkommunikation oft über das globale Netzwerk Internet abläuft und dabei nur 5 Schichten des ISO/OSI-Referenzmodells umfasst, wurde ein reduziertes Referenzmodell – das Internet-Modell – eingeführt [7]. Die beiden Referenzmodelle sind in Abbildung 2 gegenübergestellt.

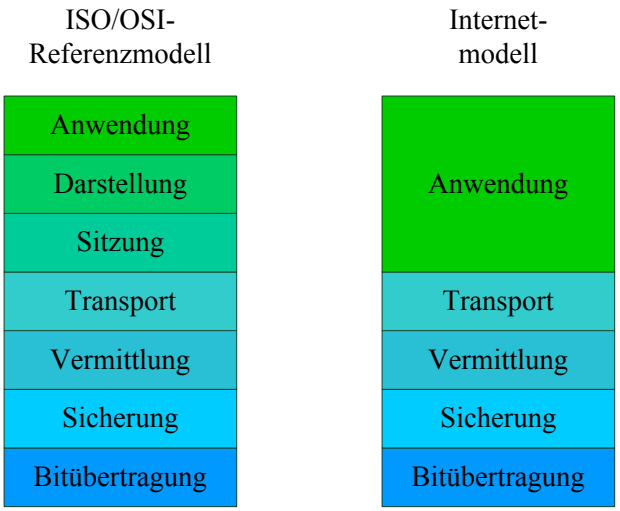

Abbildung 2: ISO/OSI-Referenzmodell und Internet-Modell im Vergleich

Die *Bitübertragungsschicht* (Schicht 1) steuert die Übertragung des Bit-Streams über das physikalische Medium. Sie beschäftigt sich mit den mechanischen und elektrischen Spezifikationen des Interfaces und des Übertragungsmediums. Die Bitübertragungsschicht definiert darüber hinaus die Repräsentation der Bits auf dem Übertragungsmedium (Codierung), die Datenrate, die Synchronisation des Senders und des Empfängers, die physikalische Topologie und den Übertragungsmodus (Simplex, Halbduplex, Vollduplex). Simplex ermöglicht die Übertragung auf einem Kanal nur in eine Richtung. Halbduplex ermöglicht die Übertragung in beide Richtungen, jedoch nicht gleichzeitig. Bei Vollduplex ist eine parallele Übertragung in beide Richtungen möglich.

Die *Sicherungsschicht* (Schicht 2) wandelt eine reine Bitübertragung in eine zuverlässige Verbindung um. Dabei wird der Bit-Stream in die besser handhabbaren Dateneinheiten

(Frames) unterteilt. Des Weiteren bietet die Sicherungsschicht eine Adressierung der Geräte an. Damit können die Sender und Empfänger identifiziert werden. Eine weitere Aufgabe dieser Schicht ist die Fehlerkontrolle. Dadurch können fehlerhafte oder verlorene Frames detektiert und gegebenenfalls neu übertragen werden. Zu den weiteren wichtigen Aufgaben der Sicherungsschicht gehört die Zugriffskontrolle. Wenn mehrere Sender gleichzeitig Daten übertragen wollen, bestimmt die Zugriffskontrolle, welcher Sender das Medium für sich beanspruchen kann. Darüber hinaus kann die Sicherungsschicht eine Flusssteuerung anbieten. Dadurch kann die Rate, mit der der Sender die Daten erzeugt, auf die maximal mögliche Rate, mit der der Empfänger Daten empfangen kann, angepasst werden. Dadurch lässt sich eine Überlastung des Empfängers verhindern.

Die *Vermittlungsschicht* (Schicht 3) ist für die Zustellung der Pakete von der Quelle zur Senke zuständig. Die Zustellung kann dabei über mehrere Netze hinweg stattfinden. Hierfür bietet die Vermittlungsschicht eine logische Adressierung an. Um das Ziel erreichen zu können, gehört das Routing zu den Aufgaben dieser Schicht. Das Routing bestimmt dabei den nächsten Hop (Zwischenstation) auf dem Pfad von der Quelle zur Senke. Geräte, die mehrere Netze verbinden, werden Router genannt.

Die *Transportschicht* (Schicht 4) ist für die Zustellung der Nachrichten zwischen den Prozessen zuständig. Ein Prozess ist ein Programm, welches auf einem Host ausgeführt wird. Zu den Aufgaben der Transportschicht gehört die Adressierung der Prozesse. Auf einem Host können gleichzeitig mehrere Programme ausgeführt werden, die über das Netzwerk mit verschiedenen Netzwerkteilnehmern kommunizieren. Die Adressierung der Prozesse ermöglicht eine Zuordnung der Pakete zu einem bestimmten Programm. Die Transportschicht unterstützt zwei Kommunikationsarten. Bei einer verbindungslosen Kommunikation wird jedes Paket als eine unabhängige Dateneinheit betrachtet. Bei einer verbindungsorientierten Kommunikation wird zunächst eine sichere Verbindung zwischen dem Sender und dem Empfänger aufgebaut. Anschließend findet der Datenaustausch statt. Nach dem Datenaustausch wird die Verbindung abgebaut. Die Transportschicht ermöglicht, die Daten auf mehrere Pakete im Sender aufzuteilen (segmentieren) und im Empfänger wieder zusammenzusetzen (reassemblieren). Für eine korrekte Datenwiederherstellung spielt die Reihenfolge der Pakete eine Rolle. Hierfür werden die Pakete mit einer Sequenznummer gekennzeichnet. Ähnlich der Sicherungsschicht bietet die Transportschicht eine Fehlererkennung und eine Flusssteuerung. Diese beziehen sich jedoch auf die Interprozesskommunikation und nicht auf die einzelnen physikalischen Verbindungen [7].

Die *Sitzungsschicht* (Schicht 5) dient der Einrichtung, Aufrechterhaltung und Synchronisation einer Interaktion zwischen zwei kommunizierenden Netzwerkteilnehmern. Die *Darstellungsschicht* (Schicht 6) beschäftigt sich mit der Syntax und Semantik der Informationen, die zwischen zwei Netzwerkteilnehmern ausgetauscht werden. Dazu zählen

Datenkonvertierung, Verschlüsselung und Kompression. Die *Anwendungsschicht* (Schicht 7) dient der Übertragung beliebiger Daten einer Anwendung wie z.B. Email.

Damit jede Netzwerkschicht ihren Aufgaben nachkommen kann, müssen die erforderlichen Steuerinformationen in den Datenframe hinzugefügt werden. Die Steuerinformationen einer Netzwerkschicht werden als sogenannte Header den Daten der jeweils übergeordneten Schicht vorangestellt.

2.2. Übertragungsmedien

Im folgenden Abschnitt wird auf die häufig genutzten Übertragungsmedien eingegangen. Diese sind Ethernet, Wireless Local Area Network und HomePlug.

2.2.1. Ethernet

Das ursprüngliche Ethernet wurde im Jahr 1976 im Xerox Palo Alto Research Center entwickelt [7]. Seitdem wird die Entwicklung von Ethernet durch die Arbeitsgruppe 802.3 im Institute of Electrical and Electronics Engineers (IEEE 802.3) ständig weitergeführt. Das Ethernet ist auf den Netzwerkschichten 1 und 2 des ISO/OSI-Referenzmodells angesiedelt. In der ursprünglichen Form waren alle Ethernet-Geräte an ein einzelnes Kabel (Koaxialkabel) über einen sogenannten „Vampire Tap“ angeschlossen und bildeten eine Bustopologie. Wenn zwei oder mehrere Stationen gleichzeitig kommunizieren, würden die Pakete auf der Leitung eine Kollision verursachen und Paketinhalte zerstört werden. Damit dennoch mehrere Geräte über dasselbe Medium kommunizieren können, wurde das Verfahren Carrier Sense Multiple Access (CSMA) / Collision Detection (CD) eingeführt [8]. Das CSMA/CD ermöglicht, die Kollisionen auf der Leitung zu erkennen, woraufhin ein Gerät gegebenenfalls eine Neuübertragung initiieren kann.

Die aktuell verwendeten Ethernet-Standards sind Standard Ethernet (10 Mbps), Fast Ethernet (100 Mbps), Gigabit-Ethernet (1 Gbps) und Ten-Gigabit Ethernet (10 Gbps) [7]. Zur Datenübertragung werden Twisted-Pair-Kabel oder Glasfaser verwendet. Die gegenwärtig in der Praxis vorkommende Form von Ethernet ist Switched Ethernet [9]. Hierbei werden spezielle Geräte, sogenannte Switches, verwendet, um mehrere Netzwerkteilnehmer miteinander zu verbinden. Damit bilden Ethernet-Geräte eine Sterntopologie. Dadurch, dass die Kommunikation zwischen zwei Netzwerkteilnehmern immer über einen Switch abläuft, befindet sich jedes Gerät in einer eigenen Kollisionsdomäne. Hiermit spielt heutzutage das CSMA/CD-Verfahren für das Ethernet eine untergeordnete Rolle [6].

Ein Ethernet-Frame ist in Abbildung 3 dargestellt. Die Ziel- und Quelladressen (*Destination* und *Source*) werden auch als Hardware-Adressen oder Media Access Control (MAC)-Adressen bezeichnet. Die MAC-Adressen werden von Herstellern zugewiesen und sind weltweit eindeutig. Die Zieladresse ist eine eindeutige Kennzeichnung des Empfängers und

die Quelladresse ist eine eindeutige Kennzeichnung des Senders. Handelt es sich beim Empfänger um einen einzelnen Host, wird die Übertragung als Unicast bezeichnet. Mittels spezieller Zieladressen können Gruppen von Empfängern gebildet werden. In diesem Fall handelt es sich um Multicast oder Gruppenadressierung. Des Weiteren können alle Geräte im Netzwerk durch eine spezielle Adresse angesprochen werden. Diese Form der Adressierung wird als Broadcast bezeichnet. Virtual Local Area Network (*VLAN*) wird zur Kennzeichnung von virtuellen Netzwerken verwendet. Dieses Feld ist jedoch optional. Das Typenfeld (*Type*) gibt das Protokoll der nächsthöheren Schicht an, welches als Payload übertragen wird. Die Payload kann zwischen 46 und 1500 Byte lang sein. Wenn die tatsächlich zu übertragenden Daten kleiner 46 Byte sind, werden diese mit Padding-Bytes ergänzt. Der Frame wird mit einer Prüfsumme (*Checksum*) zur Fehlerkontrolle abgeschlossen.

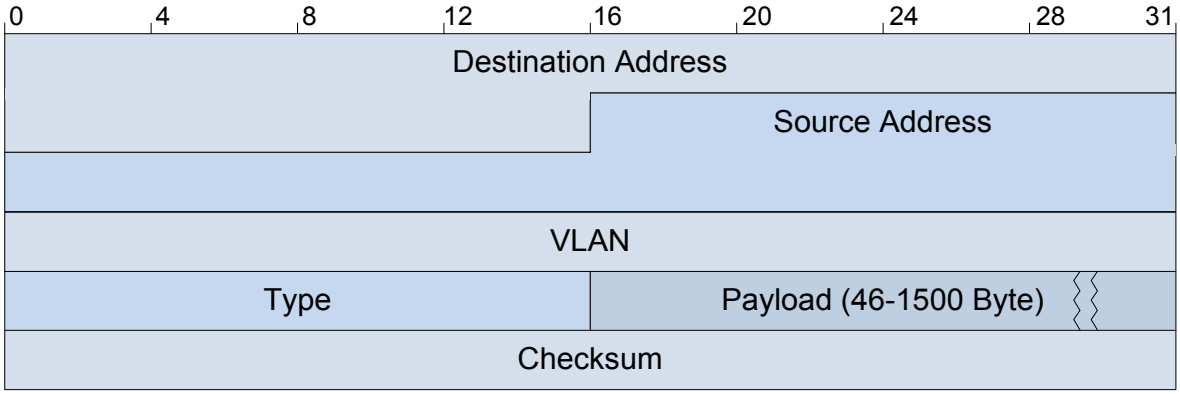

Abbildung 3: Ethernet-Frame

2.2.2. Wireless Local Area Network (WLAN)

Wireless Local Area Network (WLAN) ist eine drahtlose Technologie zur Datenübertragung. WLAN ist ein internationaler Standard, der als IEEE 802.11 entwickelt wird. Ähnlich dem Ethernet spezifiziert WLAN die unteren zwei Netzwerkschichten des ISO/OSI-Referenzmodells. Ein WLAN-Netzwerk wird auch als Basic Service Set (BSS) bezeichnet. WLAN unterstützt zwei Modi: Ad-hoc und infrastrukturbasiert. Ein infrastrukturbasiertes Netzwerk besitzt eine zentrale Station, auch Access Point (AP) genannt. Der AP übernimmt den Netzwerkaufbau und das Management wie z.B. Festlegung der BSS ID. Diese wird als eine eindeutige Kennzeichnung eines Netzwerks verwendet. Im Ad-hoc-Modus können die Stationen ein Netzwerk ohne AP aufbauen [7]. In diesem Fall müssen alle Management-Funktionen von den Stationen selbst übernommen werden.

Als Trägerfrequenz werden die Industrial, Scientific and Medical (ISM) Bänder 2,4 GHz und 5 GHz benutzt. Durch eine ständige Weiterentwicklung des WLAN-Standards sind unterschiedliche Bitübertragungsschichten entstanden, die auch unterschiedliche Datenraten

unterstützen. Der 802.11a war der erste WLAN-Standard mit einer Datenrate von 2 Mbps. Die aktuellste zum Zeitpunkt des Schreibens eingesetzte WLAN-Version ist 802.11ac, welche Datenraten bis zu 1 Gbps unterstützt.

Damit mehrere Teilnehmer im selben WLAN-Netzwerk kommunizieren können, wird das CSMA / Collision Avoidance (CA)-Verfahren eingesetzt. Im Gegensatz zu CSMA/CD, das die Kollisionen nur erkennt, wird bei CSMA/CA angestrebt die Kollisionen zu vermeiden. Diese können dennoch weiterhin auftreten [6]. Für eine verbesserte Kollisionsvermeidung kann das optionale Request to Send / Clear to Send (RTS/CTS)-Verfahren eingesetzt werden.

Ein WLAN-Daten-Frame ist in Abbildung 4 dargestellt. Das *Frame Control*-Feld enthält 11 Subfelder, die Steuerinformationen tragen. Das nächste Feld *Duration* gibt die Dauer in µs an, wie lange der Frame und seine Bestätigung den Kanal belegen. Die drei Adressfelder hängen von den Steuerinformationen ab. Im einfachsten Fall – wenn der Frame zwischen zwei WLAN-Stationen eines BSS geschickt wird – enthält Adresse 1 die Zieladresse, Adresse 2 die Quelladresse und Adresse 3 die BSS ID [7]. Die Sequenznummer (*Sequence Control*) dient der eindeutigen Kennzeichnung eines Frames, um z.B. Duplikate zu erkennen. Als *Payload* werden die Daten der nächsthöheren Netzwerkschicht übertragen. Der Frame wird mit einer Prüfsumme (*Checksum*) abgeschlossen.

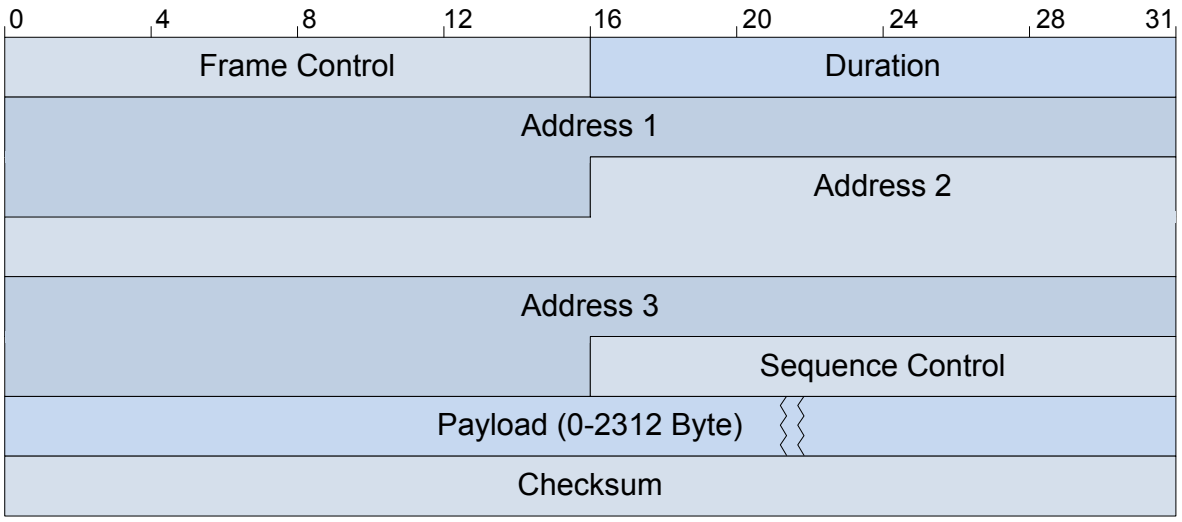

Abbildung 4: WLAN-Frame

WLAN verwendet dieselben MAC-Adressen wie Ethernet. Dadurch können die WLAN-Netzwerke ohne großen Aufwand in Ethernet-Netzwerke eingebunden werden. Hierfür muss lediglich eine Anpassung an die Ethernet-Frame-Struktur stattfinden.

2.2.3. HomePlug

HomePlug ist eine Kommunikationstechnologie, die Stromnetz (Powerline oder auch Powerline Communication, PLC) als Übertragungsmedium verwendet. Die Technologie wird

von der HomePlug Alliance entwickelt und gepflegt. Seit 2010 ist HomePlug eine IEEE 1901-Norm [10]. Die HomePlug-Spezifikation definiert die Netzwerkschichten 1 und 2 des ISO/OSI-Referenzmodells. Durch eine ständige Weiterentwicklung sind heute mehrere Versionen von HomePlug bekannt, die unterschiedliche Einsatzgebiete und Datenraten haben [11]. Für den Heimeinsatz ist HomePlug Audio/Video (AV) mit bis zu 200 Mbps Datenrate sowie eine Weiterentwicklung davon AV2 mit bis zu 1,5 Gbps vorgesehen [12]. Für den industriellen Einsatz ist HomePlug GREEN PHY mit niedriger Datenrate und geringer Leistungsaufnahme bestimmt. Diese drei HomePlug-Versionen sind darüber hinaus interoperabel.

Je nach HomePlug-Version können unterschiedliche Frequenzen für die Datenübertragung verwendet werden. Das 2-30 MHz Band wird jedoch von allen oben genannten HomePlug-Versionen unterstützt [11]. Durch die Ausnutzung des bestehenden Stromnetzes werden alle HomePlug-Geräte an ein gemeinsames Medium angeschlossen. Das HomePlug-Netzwerk besitzt damit eine Bustopologie. Dies bedeutet, dass jede Nachricht von jedem Netzwerkteilnehmer empfangen wird. Um mehreren Teilnehmern den Zugriff auf das Medium zu ermöglichen, wird standardmäßig das CSMA/CA-Verfahren eingesetzt [13]. Für die Anwendungen, die Quality of Service (QoS) erfordern, kann das optionale Time Division Multiple Access (TDMA)-Verfahren eingesetzt werden [14]. Für die Synchronisation der Netzwerkteilnehmer für das TDMA-Verfahren wird die Wechselstromfrequenz (50 Hz oder 60 Hz) benutzt.

Der Aufbau und das Management eines HomePlug-Netzwerks wird von einem Central Coordinator (CCo) übernommen. Der CCo kann grundsätzlich jedes HomePlug-Gerät sein. Die CCo-Funktionalität kann auch im laufenden Betrieb einer anderen Station übergeben werden [15]. Der CCo bestimmt unter anderem den Network Identifier (NID) und das Network Membership Key (NMK). Mittels NID können mehrere HomePlug-Netzwerke voneinander getrennt werden. Das NMK ermöglicht anderen Stationen, dem Netzwerk beizutreten [16].

Die Idee hinter der HomePlug-Technologie ist der Transport von Ethernet-Frames über das Stromnetz. In Abbildung 5 ist ein HomePlug-Frame dargestellt [16]. Der MAC Frame Header gibt den Typ und die Länge des Frames an. Das nächste Feld kann je nach Frame-Typ entweder den Zeitstempel (Arrival Time Stamp, *ATS*) oder eine Frame-Sequenznummer (*Confounder*) enthalten. Die Payload eines HomePlug-Daten-Frames enthält einen eingekapselten Ethernet-Frame. Der HomePlug-Frame wird mit einer Prüfsumme (*Checksum*) abgeschlossen. Die Geräteadressierung findet direkt über Ethernet-MAC-Adressen statt. Somit können Geräte im Ethernet-Netzwerk transparent mit den Geräten im HomePlug-Netzwerk kommunizieren.

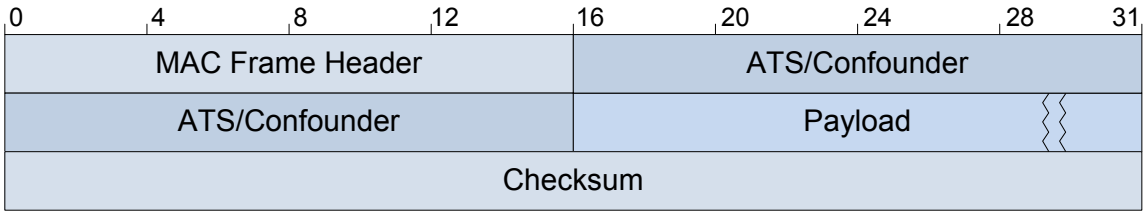

Abbildung 5: HomePlug Frame

2.3. Kommunikationsprotokolle

2.3.1. Internet Protocol (IP)

Internet Protocol (IP) ist ein Protokoll der Vermittlungsschicht des ISO/OSI-Referenzmodells. Das Protokoll stellt eine logische Adressierung bereit und ermöglicht das Routing der Pakete im Netzwerk [17]. Ein IP-Paket wird auch Datagramm genannt. IP ist ein verbindungsloses Protokoll. Zu den weiteren Aufgaben von IP gehören die Segmentierung und Reassemblierung von Daten, um Verbindungswege mit unterschiedlichen Maximum Transmission Units (MTUs) zu unterstützen. Die Struktur der IP-Adressen bildet die Grundlage für das Routing. Die IP-Adressen können unterteilt werden und damit IP-Adressen für Subnetze bilden. Eine IP-Adresse (Version 4) ist ein 32 Bit Wert, der eine Netzwerknummer und eine Host-Nummer darstellt. Die Netzwerknummern werden in der Regel von einer übergeordneten Institution wie z.B. durch Internet Service Provider (ISP) zugewiesen. Die Hostnummern werden lokal z.B. durch einen Administrator zugewiesen. Es existieren zwei Versionen des IP-Protokolls: IPv4 und IPv6. IPv6 verwendet u.a. 128 Bit für die IP-Adressen, die einen deutlich größeren Adressraum als IPv4 ermöglichen.

Die Struktur eines IPv4-Paketes ist in Abbildung 6 dargestellt. Die Felder *Identification*, *Flags* und *Fragment Offset* dienen den Fragmentierungs- und Reassemblierungszwecken. Das *Time to Live*-Feld ist ein Integer-Wert, der bei jedem Hop dekrementiert wird. Durch dieses Feld werden Routing-Loops vermieden. Die Prüfsumme (*Header Checksum*) wird ausschließlich für den Header selbst gebildet. Die *Source Address* gibt die IP-Adresse des Absenders und die *Destination Address* die IP-Adresse des Empfängers an. Die Beschreibungen anderer Felder können der IP-Spezifikation Request for Comments (RFC) 791 entnommen werden [18].

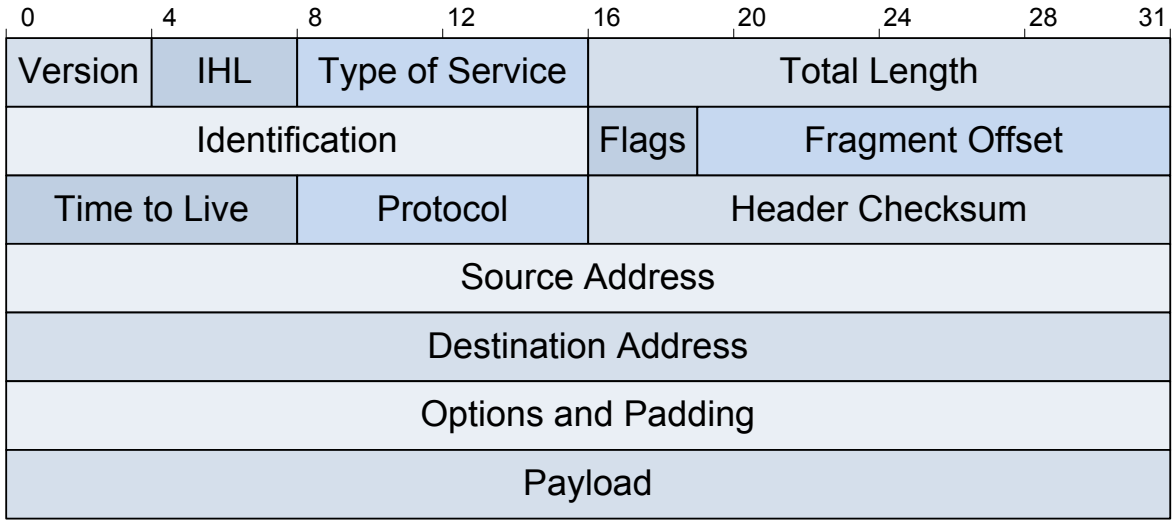

Abbildung 6: Struktur eines IP-Paketes

2.3.2. User Datagram Protocol (UDP)

User Datagram Protocol (UDP) ist ein verbindungsloses Protokoll der Transportschicht des ISO/OSI-Referenzmodells. UDP bietet einen einfachen, aber unzuverlässigen Service zum Nachrichtenaustausch [17]. UDP Ports werden benutzt, um verschiedene Programme auf einem Host zu adressieren. UDP fügt keine zusätzliche Zuverlässigkeit oder Flusskontrolle zu IP hinzu. Es wird lediglich eine Prüfsumme für die Erkennung von Übertragungsfehlern bereitgestellt. UDP wird vorwiegend für Anwendungen eingesetzt, die keine Zuverlässigkeit erfordern oder diese durch die höheren Schichten bereitgestellt wird. Ein weiterer Anwendungsfall für UDP ist der Einsatz von Multicast oder Broadcast, da in diesem Fall die Kommunikation verbindungslos abläuft.

Ein UDP-Paket ist in Abbildung 7 illustriert. Der Quell-Port (*Source Port*) identifiziert das Programm, welches das Paket erzeugt hat. Der Ziel-Port (*Destination Port*) adressiert das Programm auf dem Ziel-Host. Das Längenfeld (*Length*) gibt die Länge des UDP-Paketes in Bytes an. Der UDP-Header wird mit einer Prüfsumme (*Checksum*) abgeschlossen [19].

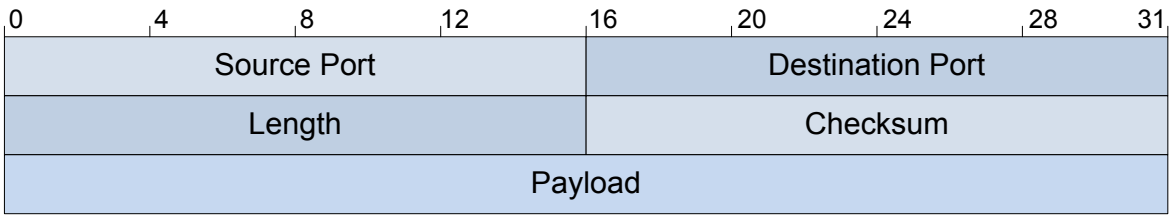

Abbildung 7: Struktur eines UDP-Paketes

2.3.3. Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) ist ebenfalls ein Protokoll der Transportschicht des ISO/OSI-Referenzmodells. TCP ist ein verbindungsorientiertes Protokoll, welches eine zuverlässige Zustellung der Pakete und eine Flusskontrolle bietet. Ähnlich UDP verwendet TCP Ports, um eine bestimmte Anwendung auf dem Host zu adressieren. Eine zuverlässige Verbindung kann nur zwischen zwei Stationen (Ende-zu-Ende) mittels TCP aufgebaut werden [17]. Die Verbindung kann dabei über mehrere Hops ablaufen. TCP ermöglicht eine Stream-Übertragung, sodass eine Anwendung keine Unterteilung der Daten auf Pakete vornehmen muss. Diese Aufgabe wird von TCP übernommen, indem die Bytes in Segmente gruppiert werden. Die Zustellung einzelner Segmente muss vom Empfänger quittiert werden. Beim Ausbleiben der Bestätigung werden die Segmente erneut übertragen. TCP kann mit verlorenen, duplizierten und fehlerhaften Paketen umgehen. Darüber hinaus bietet TCP eine Flusssteuerung. Beim Quittieren der empfangenen Daten teilt der Empfänger dem Sender mit, wie viele Daten dieser noch empfangen kann, ohne einen Pufferüberlauf hervorzurufen.

Ein TCP-Paket ist in Abbildung 8 dargestellt. Ähnlich UDP identifiziert der Quell-Port (*Source Port*) das Programm, welches das Paket erzeugt hat. Der Ziel-Port (*Destination Port*) adressiert das Programm auf dem Ziel-Host. Die Sequenznummer (*Sequence Number*) bildet gemeinsam mit der Bestätigungsnummer (*Acknowledgment Number*), der Prüfsumme (*Checksum*) und den *Flags* die Grundlage für eine zuverlässige Paketzustellung. Das Feld *Window* spiegelt die Puffergröße in Bytes wider, die für den Datenempfang noch verfügbar ist. Eine weitere Beschreibung der Felder kann der TCP-Spezifikation RFC 793 entnommen werden [20].

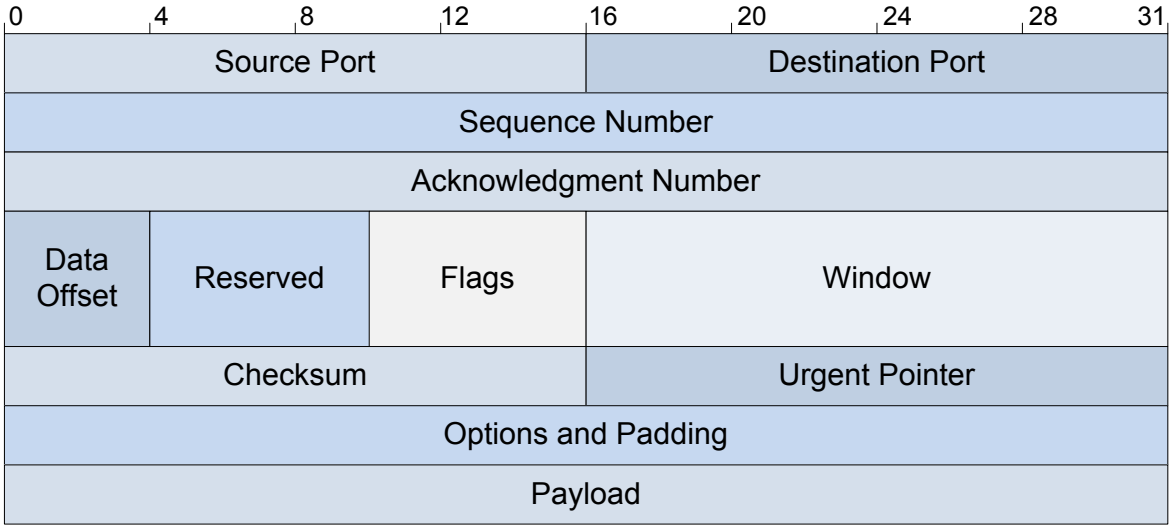

Abbildung 8: Struktur eines TCP-Paketes

2.4. Ausgewählte Netzwerk-Services

2.4.1. Address Resolution Protocol (ARP)

Address Resolution Protocol (ARP) ermöglich einen Service zur Auflösung von MAC-Adressen durch IP-Adressen. Für die Kommunikation zwischen Programmen auf zwei Hosts werden zur Adressierung die IP-Adressen und die Port-Nummern verwendet. Um einen Ethernet-Frame erstellen zu können, muss jedoch die MAC-Adresse des Empfängers bekannt sein [21]. Diese Aufgabe wird von ARP übernommen.

ARP ist auf der Sicherungsschicht des ISO/OSI-Referenzmodells angesiedelt. Ein ARP-Request mit der IP-Adresse des gesuchten Hosts wird mittels Broadcast an alle Netzwerkteilnehmer geschickt. Der Host, der die angefragte IP-Adresse besitzt, schickt ein ARP-Response mit seiner MAC-Adresse zurück an den Sender. Der Sender kann daraufhin ein Ethernet-Frame mit dem IP-Paket erstellen und die Kommunikation mit dem Ziel-Host starten. Ein Tupel aus der IP- und MAC-Adresse wird lokal in die ARP-Tabelle oder ARP-Cache beim Sender eingetragen. Die Einträge haben eine begrenzte Gültigkeit und müssen in regelmäßigen Abständen erneuert werden.

Ein ARP-Packet für IPv4 ist in Abbildung 9 dargestellt. Die *Operation* bestimmt, ob es sich um ein ARP-Request oder ein ARP-Response handelt. Im Falle von Request enthält das Feld *Target IP Address* die IP-Adresse des Ziel-Hosts. Da die Ziel-MAC-Adresse (*Target MAC Address)* nicht bekannt ist, wird das Feld mit Nullen gefüllt [7]. Die gesuchte MAC-Adresse wird im Feld *Source MAC Address* im ARP-Response übertragen. Die *Source IP Address* enthält in diesem Falle die IP-Adresse des angefragten Hosts. Eine ausführliche Beschreibung des ARP-Protokolls kann der ARP-Spezifikation RFC 826 entnommen werden [22].

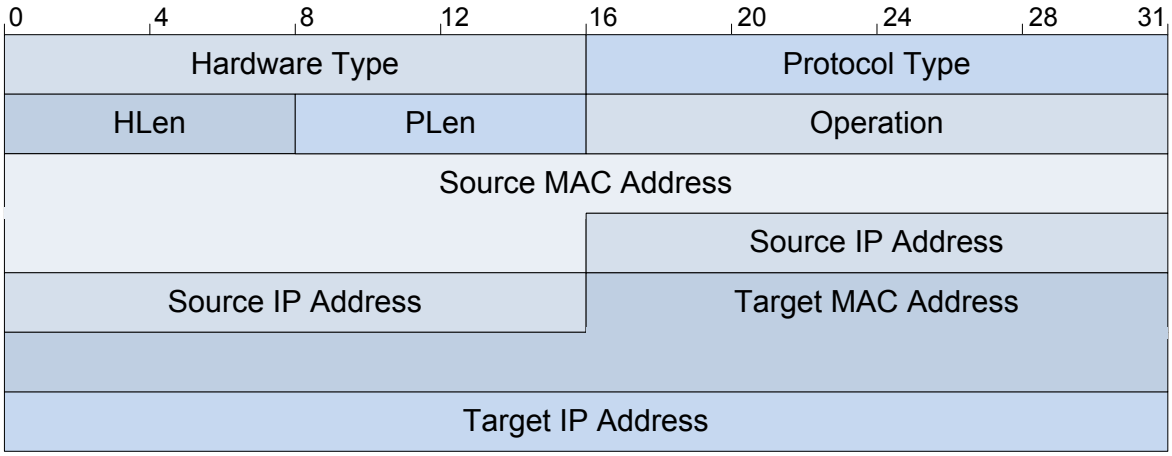

Abbildung 9: ARP-Packet für IPv4

2.4.2. Dynamic Host Configuration Protocol (DHCP)

Dynamic Host Configuration Protocol (DHCP) ermöglicht eine zentralisierte Zuweisung der IP-Adressen. DHCP nutzt das Leasing-Prinzip. Die IP-Adressen werden von den Hosts für eine vorgegebene Zeit geleast. Nach Ablauf der Leasing-Dauer müssen die IP-Adresse erneuert bzw. das Leasing verlängert werden. Die geleasten IP-Adressen und die zugehörigen MAC-Adressen werden auf dem DHCP-Server gespeichert. Der Server kann dabei eine statische und eine dynamische Tabelle führen. In der statischen Tabelle stehen permanente IP-Zuweisungen. Die dynamische Tabelle enthält dagegen temporäre IP-Zuweisungen [7].

DHCP ist ein Anwendungsschichtprotokoll des ISO/OSI-Referenzmodells. Als Transportprotokoll wird UDP verwendet. DHCP stellt eine Grundlage für Plug & Play-Netzwerke dar. Wenn sich ein neues Gerät im Netzwerk anmeldet, sucht es zunächst nach einem DHCP-Server mittels einer Broadcast-Nachricht (DHCPDISCOVER) [6]. Der DHCP-Server beantwortet diese Nachricht und macht sich dem neuen Gerät bekannt (DHCPOFFER). Anschließend bietet das Gerät um eine Zuweisung der IP-Adresse (DHCPREQUEST), welche dann vom DHCP-Server mitgeteilt wird (DHCPACK). Neben der automatischen IP-Zuweisung kann der DHCP-Server auch andere netzwerktechnische Informationen zur Verfügung stellen wie z.B. das Default Gateway oder einen Name Server [9]. Eine weitere Beschreibung von DHCP kann der Spezifikation RFC 2131 entnommen werden [23].

2.5. Software-Architektur-Stile

Durch eine konstant steigende Verbreitung der Netzwerktechnologien in nahezu allen Bereichen des Lebens sind im Laufe der Entwicklung viele neue Protokolle entstanden. Die Komplexität der Protokolle und die zu lösenden Aufgaben haben ebenfalls zugenommen. Um Lösungen für zukünftige netzwerktechnische Anwendungen schneller entwickeln zu können, war es notwendig, Technologien bereitzustellen, die die Konzipierung und Implementierung der Anwendungsprotokolle vereinfachen und automatisieren. Zwei Architektur-Stile wurden als Lösung für das Problem vorgeschlagen. Diese sind die Ressourcen-orientierte Architektur, auch unter der Bezeichnung Representational State Transfer (REST)-Architektur bekannt, und die Service-orientierte Architektur (SOA) [6].

Die SOA-basierte Herangehensweise zur Lösung des Problems bietet die Möglichkeit, Protokolle zu generieren, die auf jede beliebige Anwendung angepasst werden können. Die Kernelemente von SOA sind ein Framework für die Protokollspezifikation, Software-Toolkits für eine automatische Generierung einer Protokollimplementierung aus der Spezifikation sowie modulare Teilspezifikationen, die in Protokollen benutzt werden können.

Die REST zugrundeliegende Idee ist, alle Informationen und Funktionen als Ressourcen zu behandeln. Die Ressourcen werden durch Uniform Resource Identifier (URI) bzw. Uniform Resource Locator (URL) gekennzeichnet und über das Hypertext Transfer Protocol angesprochen. Die REST-Architektur spiegelt die Web-Architektur wider. Im Folgenden wird auf die einzelnen Architektur-Stile eingegangen.

2.5.1. Ressourcen-orientierte Architektur

Die REST-Architektur wurde zum ersten Mal in der Dissertation von Fielding beschrieben [24]. Die Architektur basiert auf dem Client-Server-Modell. Die Kommunikation zwischen dem Client und dem Server ist zustandslos (stateless). Das bedeutet, dass der Sitzungszustand komplett vom Client überwacht werden muss. Jede Anfrage vom Client an den Server muss alle Information enthalten, die für die Verarbeitung dieser Anfrage notwendig sind. Die Antworten vom Server können zwischengespeichert (gecached) werden. Solange die zwischengespeicherten Daten gültig sind, können sie als Antworten für erneute Anfragen benutzt werden, ohne eine neue Anfrage an den Server zu schicken. Die REST-Architektur benutzt ein uniformes Interface, welches von der Anwendung unabhängig ist. Jede beliebige Information wird als eine Ressource aufgefasst [25].

Jede beliebige Ressource kann mittels URI adressiert werden. Eine URI identifiziert vier Parameter: das Protokoll für den Zugriff auf die Ressource, den Host, den Port und den Pfad zur Ressource auf dem Host. Die URI wird im Klartext angegeben und besitzt das folgende Muster: *Protocol://Host:Port/Path* [7]. Es können auch weitere optionale Parameter wie Query und Fragment folgen. Eine vollständige Beschreibung des URI-Aufbaus kann der Spezifikation RFC 3986 entnommen werden [26].

Jede Ressource hat eine Repräsentation, die eine Abfolge von Bytes in einem bestimmten Format darstellt. Das uniforme Interface definiert vier Methoden, die auf eine Ressource angewendet werden können. Diese sind in Tabelle 1 aufgelistet [27].

Methode	Beschreibung
GET	Abruf der Repräsentation einer Ressource
POST	Versand von Informationen an eine Ressource
PUT	Neue Ressource erstellen
DELETE	Eine Ressource löschen

Tabelle 1: Methoden des uniformen Interfaces der REST-Architektur

In den folgenden Abschnitten wird auf die Vertreter der REST-Architektur eingegangen.

2.5.1.1. Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll der Anwendungsschicht des ISO/OSI-Referenzmodells. HTTP ist ein einfaches Request-Response-Protokoll. In der Transportschicht wird für eine sichere Übertragung TCP eingesetzt [9]. HTTP ist ein textbasiertes Protokoll. Die Request- und Response-Header werden im American Standard Code for Information Interchange (ASCII) codiert. HTTP wird vor allem für den Zugriff auf World Wide Web (WWW)-Inhalte eingesetzt. Darüber hinaus wird HTTP als ein Transportprotokoll für andere Anwendungen wie z.B. Machine-to-Machine (M2M)-Kommunikation eingesetzt und bildet eine Grundlage für den REST.

Eine HTTP-Request-Nachricht wird vom Client an den Server geschickt und besteht aus einer Request-Zeile, den Headern und einem optionalen Body (Nachrichten-Payload). Eine HTTP-Response-Nachricht wird vom Server an den Client geschickt und beinhaltet eine Statuszeile, die Header und einen optionalen Body. Die Request-Zeile beinhaltet eine Methode, die auf eine Ressource angewendet werden soll. Neben den Standardmethoden, die in Tabelle 1 aufgelistet sind, bietet HTTP vier weitere Methoden an: HEAD, TRACE, CONNECT und OPTION. Die Beschreibung der zusätzlichen Methoden kann der HTTP-Spezifikation RFC 2616 entnommen werden [28]. Der Methode folgt in der Request-Zeile ein URL, der die angefragte Ressource eindeutig identifiziert. Auf diese Ressource soll die angegebene Methode angewendet werden. Die Request-Zeile wird mit einer HTTP-Versionsnummer abgeschlossen [7]. Die Statuszeile enthält einen Statuscode, der angibt, ob das Request erfolgreich war oder nicht, und wenn nicht, warum. Zusätzlich zum Code wird der Status auch in Textform angegeben.

Nach der Request- bzw. Status-Zeile werden die Header übertragen. Diese können Request- oder Response-spezifisch oder für beide Nachrichtentypen vorgesehen sein. Die Header werden ebenfalls in Klartext übertragen. Die Aufgabe der Header ist, zusätzliche Informationen zwischen dem Client und dem Server auszutauschen. Alle Header werden nach demselben Muster aufgebaut: *HeaderName: HeaderValue*. Ein Header wird mit den aufeinanderfolgenden ASCII-Zeichen Carriage Return (CR) und Line Feed (LF) abgeschlossen, die einen Zeilenumbruch darstellen. Das Ende das Headerteils wird mit dem zweifachen CRLF (einer leeren Zeile) signalisiert.

Nach dem Headerteil folgt der Body. Dieser stellt die HTTP-Payload dar. Die Codierung und die Art des Body wird durch den Header *Content-Type* festgelegt. Der Body kann einer der vielen Multipurpose Internet Mail Extensions (MIME)-Typen sein.

2.5.1.2. Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) wurde für den Einsatz in eingebetteten Systemen mit geringem Energieverbrauch wie z.B. Sensornetzwerke konzipiert. Die Entwicklung von

CoAP wurde durch HTTP inspiriert, weist aber dennoch signifikante Unterschiede auf. CoAP ist ebenfalls ein Anwendungsschichtprotokoll. CoAP ermöglicht, die REST-Architektur auf Geräte mit eingeschränkten Ressourcen zu bringen und diese so in das „Internet of Things (IoT)“ einzubinden [29]. CoAP wurde im Juni 2014 als RFC 7252 standardisiert [30].

Im Gegensatz zu HTTP ist CoAP ein binäres Protokoll. Demzufolge sind die CoAP-Request- und CoAP-Response-Nachrichten deutlich kleiner als die von HTTP. Ein weiterer signifikanter Unterschied ist die Nutzung von UDP zur Übertragung der CoAP-Nachrichten. CoAP verwendet die zugrundeliegenden REST-Methoden wie in Tabelle 1 dargestellt. Zur Ressourcenadressierung finden ebenfalls URIs Verwendung, die jedoch nicht komplett in Klartext übertragen werden. Da die Nachrichtenübertragung mittels UDP keine Nachrichtenzustellung garantiert, bringt CoAP einen eigenen Mechanismus für die zuverlässige Übertragung mit. Die CoAP-Nachrichten können „Confirmable (CON)“ (bestätigt) oder „Non Confirmable (NON)“ (nicht bestätigt) sein. Die Confirmable-Nachrichten müssen von dem CoAP-Server bestätigt werden. Die Bestätigung kann sowohl zusammen mit einer Response-Nachricht als auch als eine separate Bestätigungsnachricht (ACK) geschickt werden. Das tatsächliche Response kann dann später verschickt werden. Diese sogenannte „Delayed Response“ unterscheidet CoAP ebenfalls von HTTP. Falls der Client keine Bestätigung vom Server bekommt, wird das Request wiederholt. Wenn eine Nachricht empfangen wurde, jedoch aus bestimmten Gründen nicht verarbeitet werden kann, wird sie dem Sender mit einer Reset-Nachricht (RST) signalisiert.

Ein weiterer Unterschied von CoAP zu HTTP ist die Unterstützung von Multicast. Da HTTP auf TCP basiert, ist per se kein Multicast möglich. Multicast stellt besonders in Automatisierungsszenarien eine wichtige Eigenschaft dar, da hierbei mehrere Netzwerkteilnehmer gleichzeitig informiert oder abgefragt werden können. Eine Anwendung, die darauf basiert, ist das Ressource-Discovery. Durch das Discovery kann ein Client mittels Multicast Ressourcen im Netzwerk finden, die bestimmten Kriterien entsprechen. CoAP unterstützt u.a. semantische Angaben wie Ressourcentyp und Interface-Typ. Das Discovery wird bei CoAP in eine separate Spezifikation namens Constrained RESTful Environments (CoRE) Link Format ausgelagert, die in RFC 6690 festgehalten wird [31].

Des Weiteren bietet CoAP einen Subscription/Notification-Mechanismus (CoAP Observe). Dieser ist ebenfalls für M2M-Anwendungen vorgesehen. Durch CoAP Observe kann der Client z.B. automatische Wertänderungen vom Server erhalten, ohne ein Request abschicken zu müssen. Voraussetzung dafür ist, dass der Client die Wertänderungen abonniert hat [32].

2.5.2. Service-orientierte Architektur (SOA)

SOA ist ein Designparadigma für die Interaktion zwischen heterogenen Komponenten unter den Gesichtspunkten der Selbstverwaltung und Interoperabilität. Jede Komponente kann

bestimmte Funktionen anbieten, an denen andere Komponenten interessiert sein können. Die selbstbeschreibenden Interfaces zu diesen Funktionen werden Services genannt [33] [34]. Die Services basieren auf offenen Standards, die für Interoperabilität sorgen. SOA ermöglicht Flexibilität, Wiederverwendbarkeit, Anpassbarkeit und Integration der Services in heterogene Umgebungen. Die Services sind lose gekoppelt und können dynamisch gefunden und genutzt werden. Die ursprüngliche SOA-Architektur definiert ein Service Registry, das ein zentrales Verzeichnis für alle verfügbaren Services in einer Umgebung darstellt. Eine Implementierung eines solchen Service Registry stellte z.B. Universal Description, Discovery and Integration (UDDI) dar [35]. Es wurden später jedoch weitere Mechanismen entwickelt, die auch ein Service Discovery ohne Service Registry ermöglichen.

Eine SOA definiert drei Rollen: Service Provider, Client und ein optionales Service Registry. Der Service Provider stellt einen Service zur Verfügung, der von einem Client in Anspruch genommen werden kann. Die Services können sich bei dem Service Registry anmelden und damit den Clients ermöglichen, danach im Service Registry zu suchen.

Jeder Service bietet eine eigene Beschreibung an, die die Eigenschaften und Funktionalitäten des Services beschreibt. Sie enthält alle Informationen, die für die Interaktion mit dem Service notwendig sind. Diese Beschreibung ist standardisiert und wird mit Extensible Markup Language (XML) dargestellt. Dadurch ist sie sowohl für Maschinen als auch für Menschen lesbar.

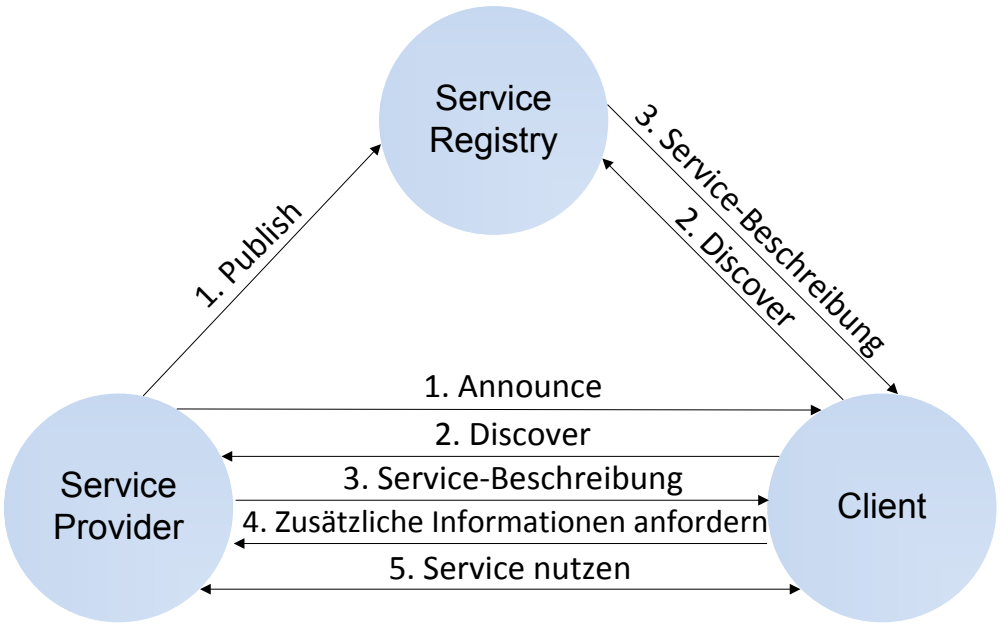

Abbildung 10: Rollen in SOA und ihre Interaktionen [34] [36]

Eine klassische SOA-Implementierung umfasst neben den drei Rollen fünf mögliche Interaktionen zwischen ihnen, die in Abbildung 10 illustriert sind. In Schritt 1 stellt sich der Service Provider im Netzwerk vor. Dies geschieht entweder durch die Veröffentlichung

(*Publish*) des Services im Service Registry oder durch eine Ankündigung (*Announcement*) im Netzwerk. In Schritt 2 schickt der Client eine Discovery-Anfrage an das Service Registry oder an das Netzwerk. Alternativ kann der Client auch auf Ankündigungen warten. In Schritt 3 erhält der Client eine Service-Beschreibung. Diese kann je nach durchgeführter Discovery-Art entweder vom Service Registry oder direkt vom Service Provider geschickt werden. In Schritt 4 fordert der Client eine detaillierte Beschreibung des Services an, die eine Interface-Beschreibung enthält. In Schritt 5 kann der Client den Service in Anspruch nehmen. Wenn der Service Provider das Netzwerk verlässt, muss er sich beim Service Registry und beim Client abmelden [34].

2.5.2.1. Web Services

Web Services (WS) stellen die SOA-Implementierung mit der höchsten Marktdurchdringung dar. WS stellen einen Satz aus modularen Protokollbausteinen zur Verfügung, die auf verschiedene Weise kombiniert werden können, um bestimmte Anforderungen einer Anwendung zu erfüllen. Die WS-Standards werden auch als WS-* bezeichnet. Kombinationen von WS-Standards werden Profile genannt [37]. WS stellen Standards zur Verfügung, die Service Discovery, Service-Beschreibung, Security, Policy und andere Funktionalitäten ermöglichen. WS definieren lediglich eine Schnittstelle zu einer Anwendung. Die Anwendung kann eine beliebige Implementierung besitzen und kann zur Laufzeit ausgetauscht werden. WS benutzen XML für die Daten-Codierung und Simple Object Access Protocol (SOAP) für den Datenaustausch. Diese wichtigen WS-Bestandteile werden in den folgenden Abschnitten erklärt. Anschließend wird ein spezielles WS-Profil für eingebettete Systeme namens Devices Profile for Web Services (DPWS) vorgestellt.

2.5.2.2. Extensible Markup Language (XML)

XML ist eine Sprache für die Beschreibung eines strukturierten Inhaltes [38]. Der Inhalt wird als Text repräsentiert. Eine vollständige XML-Beschreibung eines Inhaltes wird als XML-Dokument bezeichnet. Die sogenannten Tags (Markups) werden verwendet, um Informationen über den Inhalt bereitzustellen [6]. Die Inhalte werden zwischen den Start- und End-Tags eingeschlossen: *<tag>value</tag>*. Die XML-Syntax bietet eine verschachtelte Struktur aus Tag/Wert-Paaren, die eine Baumstruktur bilden. Solche Tag/Wert-Paare werden Elemente genannt. Zusätzliche Metainformationen zu einem bestimmten Wert können innerhalb eines Start-Tags als Schlüsselwort-Werte-Paare angegeben werden: *<tag key="keyvalue">value</tag>*. Solche Schlüsselwort-Werte-Paare werden als Attribute bezeichnet [39]. XML-Dokumente können durch die Textrepräsentation sowohl von Maschinen als auch von Menschen gelesen werden. Module, die XML-Dokumente interpretieren, werden XML-Parser genannt.

Die Struktur eines XML-Dokuments kann mit einem Schema beschrieben werden. Eine vollständige Beschreibung des Aufbaus eines XML-Dokuments wird als XML Schema oder XML Schema Definition (XSD) bezeichnet [40]. XSD selbst stellt ebenfalls ein XML-Dokument dar. Obwohl alle Werte in XML in Textform und damit als Strings dargestellt werden, bietet XML Schema Datentypen wie Integer, Float usw. an. Dadurch können atomare und komplexe Datentypen in XML abgebildet werden. XSD definiert damit nicht nur die Syntax, sondern auch ein abstraktes Datenmodell. Ein XML-Dokument, das der XSD entspricht, stellt eine Datensammlung dar, die mit dem entsprechenden abstrakten Datenmodell konform ist [6].

Durch den modularen Aufbau kann ein bestimmtes Schema ein Teil eines anderen Schemas sein. Es kann dadurch zu Namenskollisionen kommen, die für die Tags (Elementnamen) verwendet werden. Um dieses Problem zu lösen, werden Namespaces eingesetzt. Jeder Namespace wird durch einen eindeutigen URI identifiziert. Die Elemente werden dann einem Namespace zugeordnet. Gleiche Elementnamen sind dadurch erlaubt, solange diese unterschiedlichen Namespaces gehören. Eine Kombination aus dem Elementnamen und dem Namespace wird als Qualified Name (QName) bezeichnet. Damit ein URI nicht ständig einem Elementnamen vorangestellt wird, werden Namespace-Präfixe verwendet. Diese stellen eine kurze Schreibform für einen vollständigen URI dar und erlauben eine kompaktere Schreibweise. Weitere Informationen zu XML und XSD können den jeweiligen Spezifikationen [38] und [40] entnommen werden.

2.5.2.3. Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) ist ein leichtgewichtiges XML-basiertes Protokoll, das für den Datenaustausch von Web Services benutzt wird [41] [42]. SOAP ist ein Protokoll der Anwendungsschicht. SOAP selbst kann nach definierten Bindings über andere Protokolle der Anwendungsschicht wie z.B. HTTP [43] oder auch der Transportschicht wie z.B. UDP übertragen werden [44].

SOAP-Nachrichten werden mit XML codiert. Eine SOAP-Nachricht ist ein XML-Dokument, das aus einem obligatorischen SOAP Envelope, einem optionalen SOAP Header und einem obligatorischen SOAP Body besteht [17]. Die Struktur von SOAP kann wie folgt dargestellt werden:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
    </env:Header>
    <env:Body>
    </env:Body>
</env:Envelope>
```

Der SOAP Header soll Metadaten enthalten, die für die Zustellung und Verarbeitung des Nachrichteninhalts notwendig sind. Die Payload der Nachricht wird im SOAP Body übertragen. Der Inhalt des Headers und des Body ist von SOAP nicht spezifiziert. Dieser wird durch ein WS-Profil bestimmt [34].

2.5.2.4. Devices Profile for Web Services (DPWS)

Devices Profile for Web Services (DPWS) ist ein WS-Profil für eingebettete Systeme bzw. Geräte mit eingeschränkten Ressourcen. DPWS wurde im Jahr 2004 als ein möglicher Nachfolger für Universal Plug and Play (UPnP) vorgestellt [45]. DPWS 1.0 wurde im Jahr 2006 verabschiedet [46]. DPWS 1.1 ist im Jahr 2009 als OASIS[1]-Standard erschienen [47]. DPWS besteht aus Protokollen und WS-Standards, die notwendig sind, um eine sichere Kommunikation zwischen Geräten zu ermöglichen. Darüber hinaus kann DPWS mit anderen WS-Standards kombiniert und erweitert werden [48]. Der Aufbau von DPWS ist in Abbildung 11 dargestellt. Bevor auf die einzelnen DPWS-Bestandteile eingegangen wird, soll zunächst der Kommunikationsablauf erklärt werden.

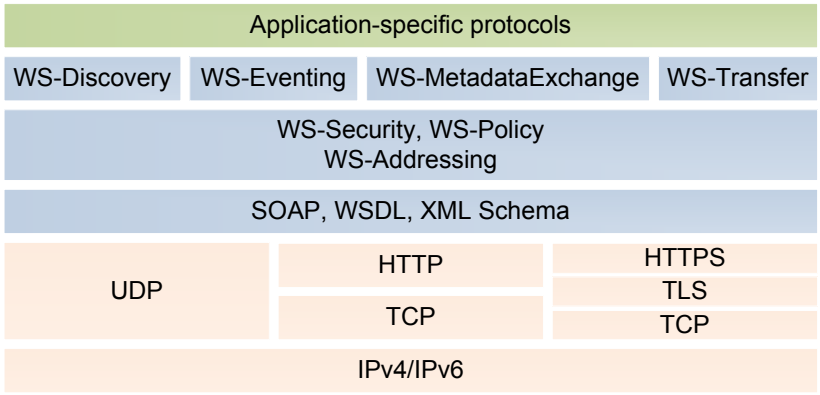

Abbildung 11: DPWS Stack [34]

Ein DPWS-Gerät (DPWS Device) wird als Hosting Service bezeichnet. Der Hosting Service enthält Informationen über das Gerät wie z.B. Gerätebezeichnung, Hersteller und andere. Darüber hinaus gibt er die vom Gerät angebotenen Services an, die als Hosted Services bezeichnet werden. Der Aufruf eines Services findet in mehreren Schritten statt.

[1] Organization for the Advancement of Structured Information Standards

Als erstes sucht der Client nach einem DPWS-Gerät (Discovery). Hierfür verschickt er eine *Probe*-Nachricht mit dem gewünschten Gerätenamen. Ein physikalisches Gerät kann unter Umständen mehrere logische Geräte beinhalten, z.B. Drucker und Scanner. Discovery mit einem leeren Gerätenamen führt eine Suche nach allen Geräten durch. Wenn es im Netzwerk kein zentrales Service Registry (Discovery Proxy) gibt, wird die Probe-Nachricht per Multicast mittels SOAP-over-UDP Binding verschickt. Alternativ kann der Client auf die Ankündigungen von Services durch die *Hello*-Nachrichten lauschen. DPWS-Geräte, die den Suchkriterien entsprechen, melden sich bei dem Client mit einer *ProbeMatch*-Nachricht. Diese Nachricht wird als Unicast per UDP geschickt. Die *ProbeMatch*-Nachricht enthält u.a. die sogenannte Endpoint Reference. Diese stellt Kontaktinformationen des DPWS-Gerätes dar. Falls eine *ProbeMatch*-Nachricht nur eine eindeutige ID des Gerätes – Universally Unique Identifier (UUID [49]) – und keine Informationen zum direkten Verbindungsaufbau enthält, werden diese per *Resolve*-Nachricht vom Client abgefragt. Die Resolve-Nachricht wird ebenfalls über Multicast mittels SOAP-over-UDP Binding verschickt. Das DPWS-Gerät mit der entsprechenden UUID antwortet in diesem Fall mit einer *ResolveMatch*-Nachricht, die die noch fehlenden Kontaktinformationen liefert. Diese Nachricht wird ähnlich *ProbeMatch*-Nachricht als Unicast per UDP geschickt. Alle weiteren Nachrichten basieren auf einer zuverlässigen Übertragung mittels SOAP-over-HTTP Binding.

Im nächsten Schritt kann der Client die Metadaten des Gerätes anfordern (*Get Metadata*). Diese liefern, wie oben erwähnt, die Beschreibung des Gerätes (Hosting Service) sowie Endpoint References der verfügbaren (Hosted) Services. Um die Beschreibung eines Services zu bekommen, kann der Client weiterhin eine *Get Metadata*-Nachricht an einen (Hosted) Service schicken. Die Antwort darauf enthält eine Service-Beschreibung, die mit Web Services Description Language (WSDL) beschrieben ist. Diese Beschreibung enthält alle Informationen, die notwendig sind, um mit dem Service zu interagieren. Nun kann der Client einen Service-Aufruf durchführen (Service Invocation). Als Antwort darauf erhält der Client eine Invocation Response. Falls alle Service-Informationen dem Client im Voraus bekannt sind, kann der Client alle vorherigen Schritte überspringen und direkt einen Service-Aufruf (Service Invocation) schicken.

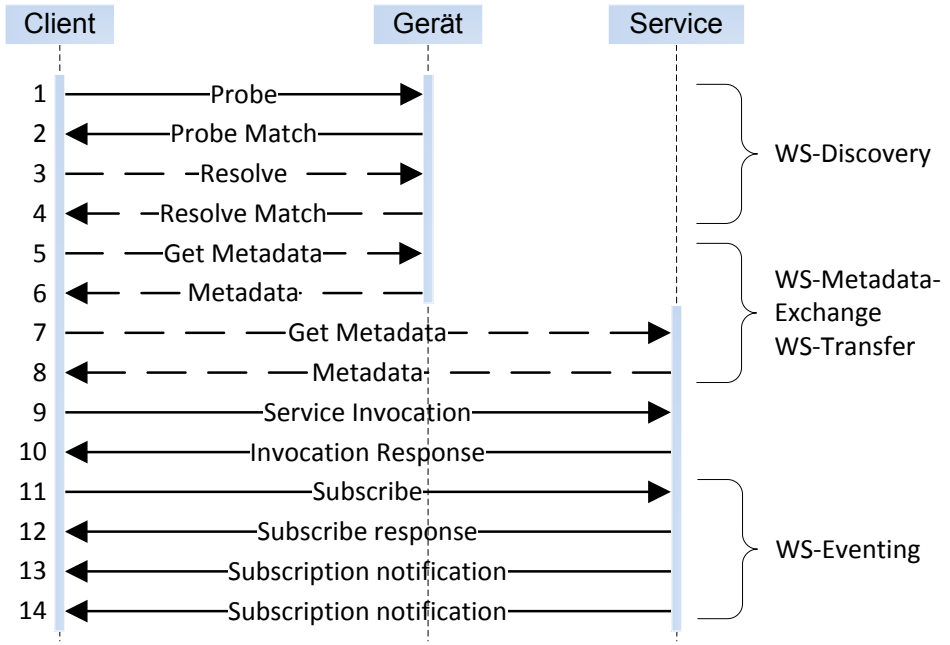

Abbildung 12: DPWS-Nachrichtenaustausch [45]

Um bestimmte Daten oder Werte aktuell zu halten, muss der Service in regelmäßigen Abständen abgefragt werden. Auch wenn keine neuen Daten vorliegen, muss der Service eine Antwort schicken. Um die Kommunikation bei solchen Anwendungsfällen effizienter zu gestalten, können Wertänderungen bei einem Service abonniert werden. Hierfür schickt der Client eine *Subscribe*-Nachricht an den Service. Der Service antwortet mit der *SubscribeResponse*-Nachricht und teilt dem Client mit, ob eine bestimmte Wertänderung erfolgreich abonniert wurde. Die Änderungen von abonnierten Werten werden mit einer *Notification*-Nachricht geliefert. Die beschriebenen Abläufe sind für einen besseren Überblick in Abbildung 12 illustriert.

Im Folgenden wird auf die einzelnen DPWS-Bestandteile eingegangen.

2.5.2.5. WS-Addressing

WS-Addressing spezifiziert Mechanismen, die es ermöglichen, Web Services auf eine transportprotokollunabhängige Weise zu adressieren [50]. Dadurch können SOAP-Nachrichten durch mehrere Hops und heterogene Transportprotokolle übertragen werden. WS-Addressing stellt das Konzept von Endpoint Reference vor. Für jedes Gerät existiert eine global eindeutige UUID, die ein DPWS-Gerät kennzeichnet. Diese wird im Adressenteil (*Address*) von Endpoint Reference eines DPWS-Gerätes angegeben. Die Services besitzen eine eindeutige HTTP-Transportadresse, die im Adressteil von ihren Endpoint Reference angegeben wird. Des Weiteren ermöglicht WS-Addressing einen zuverlässigen Nachrichtenaustausch mittels *Message ID*. Die *Message ID* erlaubt es, die Antworten den Anfragen zuzuordnen sowie Nachrichtenduplikate zu erkennen. Darüber hinaus kann mittels

WS-Addressing die Absicht einer Nachricht (*Action*) mitgeteilt werden. *Action* kann beispielsweise der Aufruf einer Funktion eines Services sein. WS-Addressing bietet zwar noch andere Funktionalitäten wie z.B. eine Umleitung von Antworten oder Fehlermeldungen an einen anderen Empfänger an, diese wurden jedoch von DPWS nicht übernommen.

2.5.2.6. WS-Policy

WS-Policy wird benutzt, um dynamische Bestandteile einer Kommunikation auszuhandeln [51]. WS-Policy stellt ein Framework zur Verfügung, mit deren Hilfe Einschränkungen, Anforderungen, Fähigkeiten und Charakteristiken der Kommunikationspartner zum Ausdruck gebracht werden können [34]. Eine Policy ist eine Auswahl an Alternativen. Der Kommunikationspartner kann die für ihn am besten geeignete Policy-Alternative aus einem Policy-Satz auswählen. Eine Policy-Alternative besteht aus einer Reihe von „Assertions“ (Behauptungen, Anforderungen), die vom Kommunikationspartner erfüllt werden müssen. Die Policy Assertions können auch optional sein. WS-Policy spezifiziert jedoch nicht, wie die Policy auf einen bestimmten WS-Standard angewandt werden muss. Diese Aufgabe wird von WS-PolicyAttachment übernommen [52].

2.5.2.7. WS-Security

Die Verschlüsselung, die Authentifizierung und Integrität können bei DPWS bereits durch Transport Layer Security (TLS) bzw. Hypertext Transfer Protocol Secure (HTTPS) übernommen werden. WS-Security ermöglicht ebenfalls die Verschlüsselung, die Authentifizierung und die Integrität auf WS-Ebene [53]. WS-Security ist ein Sicherheits-Framework, das einen großen zusätzlichen Overhead mitbringt. Da HTTPS bereits wichtige Sicherheitsmechanismen bietet, werden nur einige Aspekte von WS-Security verwendet. DPWS hat die Mechanismen von WS-Security nur für Discovery-Nachrichten übernommen, da diese HTTPS nicht nutzen können. Hierbei können nur Signaturen für die vom DPWS-Gerät geschickten Nachrichten während des Discovery-Prozesses verwendet werden, um die Echtheit des Gerätes zu verifizieren. Da DPWS auf Geräte mit eingeschränkten Ressourcen ausgelegt ist, sind die Sicherheits-Features optional. Dennoch kann DPWS bei Bedarf einen großen Sicherheitsumfang bieten.

2.5.2.8. WS-MetadataExchange

Web Services nutzen Metadaten, um dem Client notwendige Informationen mitzuteilen, die für die Interaktion mit dem Service notwendig sind. WS-MetadataExchange definiert, wie die Metadaten als eine Ressource dargestellt und mittels WS-Transfer angefordert werden können. Des Weiteren spezifiziert WS-MetadataExchange wie die Metadaten in eine Endpoint Reference eingebettet werden können und wie eine Anfrage und eine Antwort für die Übertragung der Metadaten aussehen soll [54].

WS-MetadataExchange definiert zwei mögliche Anfragearten für die Anforderung der Metadaten. Wenn die Endpoint Reference für die bestimmten Metadaten bekannt ist, z.B. durch das Discovery, kann der Client die Metadaten mittels WS-Transfer *Get*-Operation anfordern (vgl. Abbildung 12). Wenn die Endpoint Reference für die bestimmten Metadaten unbekannt ist, kann der Client eine *GetMetadata*-Anfrage an den WS Endpoint schicken, um alle Metadaten anzufordern.

Die Gerätebeschreibung enthält wichtige Informationen über das Device sowie die angebotenen Services. Die Metadaten werden in drei Abschnitte unterteilt: *ThisModel*, *ThisDevice* und *Relationship*. *ThisModel* stellt Metadaten über den Gerätetyp zur Verfügung wie z.B. Gerätehersteller, Modelname usw. *ThisDevice* enthält Informationen über ein konkretes Gerät wie z.B. Firmware-Version oder Seriennummer. *Relationship* gibt Auskunft über die Endpoint References der Services und deren Interface-Typen.

2.5.2.9. WS-Transfer

WS-Transfer wird für die Interaktion mit Ressourcen unter Benutzung von Web Services eingesetzt [55]. Die Interaktion erfolgt mittels Operationen *Get*, *Put*, *Delete* und *Create*, die ähnlich dem REST-Ansatz für die Standardoperationen wie Ressource lesen, aktualisieren, löschen und erstellen vorgesehen sind. DPWS verwenden WS-Transfer für den Zugriff auf Metadaten des Gerätes oder des Services, die in Form von Ressourcen dargestellt werden. Hierbei muss jedoch nur der *Get*-Operator unterstützt werden, mit dessen Hilfe die Metadaten angefordert werden können.

2.5.2.10. Web Services Description Language (WSDL)

WSDL ist ein XML-Dokument, das den Aufbau eines Services als eine Menge von *Endpoints* und deren Interaktionen (Operationen) mittels Nachrichten (Messages) beschreibt [56]. Operationen und Nachrichten werden zunächst abstrakt dargestellt. Anschließend werden sie einem konkreten Netzwerkprotokoll und Nachrichtenformat zugewiesen und bilden damit einen Endpoint. Die konkreten Endpoints werden zu einem abstrakten Endpoint zusammengefasst, der einen Service repräsentiert. WSDL ist erweiterbar, um eine Beschreibung von beliebigen Endpoints und deren Nachrichten unabhängig vom Netzwerkprotokoll und Nachrichtenformat zu ermöglichen.

Die Services, die durch eine Menge von Endpoints dargestellt werden, werden in WSDL als *Ports* bezeichnet. Eine abstrakte Darstellung von Endpoints und Nachrichten ermöglicht eine mehrfache Nutzung dieser Definitionen. Nachrichten sind eine abstrakte Darstellung der Daten, die übertragen werden, und *Port Types* sind eine abstrakte Menge von Operationen. Eine konkrete Spezifikation eines Protokolls bzw. eines Datenformats für einen bestimmten Port Type bildet ein Binding. Ein Port entsteht durch die Zuweisung einer Netzwerkadresse

zu einem Binding. Eine Menge von Ports bildet den Service. Folglich benutzt ein WSDL-Dokument folgende Elemente zur Definition eines Services, wie in Tabelle 2 gezeigt ist.

Element	**Beschreibung**
Types	Ein Container für die Definition von Datentypen
Message	Eine abstrakte Darstellung der zu übertragenden Daten
Operation	Eine abstrakte Darstellung einer Aktion, die vom Gerät unterstützt wird
Port Type	Ein abstrakter Satz an Operationen, die von einem oder mehreren Endpoints unterstützt werden
Binding	Eine konkrete Protokoll- und Datenformatspezifikation für einen bestimmten Port Type
Port	Ein Endpoint, definiert durch eine Kombination aus einem Binding und einer Netzwerkadresse
Service	Eine Sammlung von Endpoints

Tabelle 2: Elemente eines WSDL-Dokuments

Ein WSDL-Dokument kann in einen konkreten und einen abstrakten Teil aufgeteilt werden. Der Aufbau eines WSDL-Dokuments ist in Abbildung 13 dargestellt.

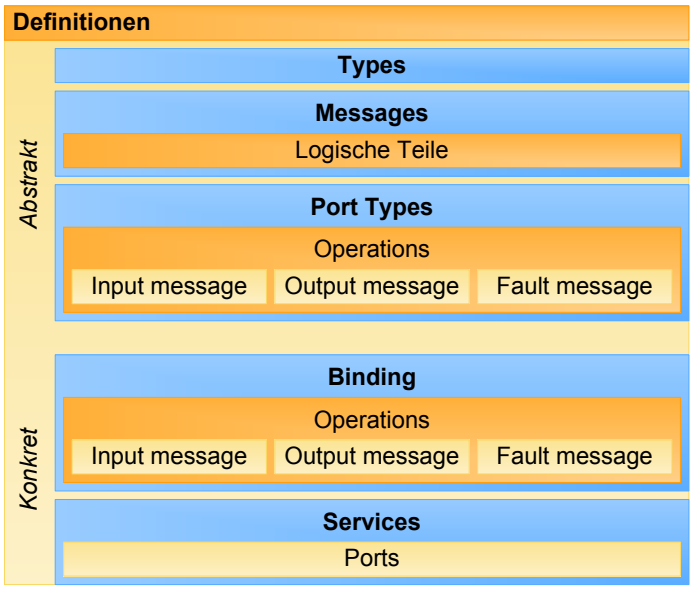

Abbildung 13: Aufbau eines WSDL-Dokuments [34] [36]

Abstrakter Teil: Der abstrakte Teil eines WSDL-Dokuments ermöglicht die Wiederverwendbarkeit durch die Beschreibung der Services unabhängig von technischen Details wie Transportprotokoll oder Nachrichtenformat. Zuerst müssen die Datentypen, die für Nachrichtenaustausch notwendig sind, definiert werden. Um die höchste Interoperabilität zu gewährleisten, nutzt WSDL XML-Schema für die Beschreibung der Datentypen. Messages

(Nachrichten) sind abstrakte Definitionen der Daten, die von und zu WS übermittelt werden. Die Messages bestehen aus einem oder mehreren logischen Teilen. Die Port Types können als abstrakte Interfaces zu beschriebenen Services angesehen werden. Port Type ist ein Satz an abstrakten Operationen. Eine Operation ist ein Satz an Messages, die einem bestimmten Datenaustauschmuster folgen. WSDL unterstützt die vier folgenden Datenaustauschmuster:

- One-way: Der Client schickt eine Nachricht an den Service
- Request-Response: Der Client schickt eine Nachricht an den Service und erwartet eine Antwort
- Solicit-Response: Der Service schickt eine Nachricht an den Client und erwartet eine Antwort
- Notification: Der Service schickt eine Nachricht an den Client

Das Datenaustauschmuster wird in der WSDL-Beschreibung implizit durch die Erscheinungsreihenfolge der Input- und Output-Messages angegeben. Request-Response und Solicit-Response-Operationen können zusätzlich eine Fault-Message für den Fall definieren, dass die Antwort nicht geschickt werden kann.

Konkreter Teil: Der konkrete Teil enthält die technischen Details zu WS. Ein Binding weist einen Port Type, seine Operationen und Messages einem bestimmten Protokoll und Datenformat zu. Mehrere Bindings können dabei mit einem Port Type erstellt werden. Alle definierten Port Types müssen sich im konkreten Teil widerspiegeln. Die konkreten Fault-Messages sind nicht definiert, da diese durch das Protokoll gegeben sind. Der Service stellt eine Sammlung von Ports dar. Ports sind die Netzwerkadressen für die Service Endpoints [34].

2.5.2.11. WS-Eventing

WS-Eventing stellt einen Publish/Subscribe-Mechanismus bereit, mit dessen Hilfe der Client automatisch über die Wertänderungen vom Service Provider benachrichtigt werden kann [57]. WS-Eventing definiert vier Rollen: den Subscriber, die Event-Quelle, die Event-Senke und den Subscription Manager. Der *Subscriber* kann bei einem Web Service Provider (*Event-Quelle*) ein Event (Wertänderung) abonnieren. Hierfür schickt dieser eine Subscription-Nachricht an die Event-Quelle und teilt ihr eine *Event-Senke* mit. Wenn das Event auftritt, benachrichtigt die Event-Quelle die Event-Senke über die Änderungen mittels einer Notification-Nachricht (vgl. Abbildung 12). In verteilten Publish/Subscribe-Systemen kann die Event-Quelle die Managementaufgaben an einen *Subscription Manager* übergeben. Die Subscriptions können zeitlich beschränkt sein, sodass der Subscriber diese bei Bedarf mit der Renew-Nachricht erneuern kann. Der Subscriber kann sich darüber hinaus über die GetStatus-Nachricht nach dem Status der Subscription erkundigen. Falls die Events nicht mehr an die

Event-Senke geschickt werden sollen, kann der Subscriber die Subscription vorzeitig mit einer Unsubscribe-Nachricht beenden.

2.5.2.12. WS-Discovery

WS-Discovery definiert ein Protokoll für die Suche nach den Services [58]. Der Client kann dabei nach einem oder mehreren Service Providern suchen. Die Suche erfolgt nach Service-Typen. Als Services werden bei WS-Discovery die Hosting Services (DPWS-Geräte) verstanden. Das Protokoll beschreibt zwei Operationsmodi, den Ad-hoc-Mode und den Managed Mode. Im Ad-hoc-Mode sendet der Client eine *Probe*-Nachricht an eine Multicast-Gruppe. Services, die die Suchanfrage erfüllen, senden eine Antwort (*ProbeMatch*-Nachricht) direkt an den Client zurück. Wenn die *ProbeMatch*-Nachricht keine Transport-Adresse enthält, schickt der Client eine *Resolve*-Nachricht an die Multicast-Gruppe mit der Endpoint Reference des gesuchten Services. Der Service mit der entsprechenden Endpoint Reference beantwortet diese Anfrage mit einer *ResolveMatch*-Nachricht und teilt dem Client seine Transportadresse mit (vgl. Abbildung 12). Der Service kann auch über mehrere Transportadressen verfügen, z.B. durch mehrere IP-Adressen.

Damit der Client das Netzwerk nicht regelmäßig nach den Services durchsucht, melden sich diese im Netzwerk automatisch mittels einer *Hello*-Nachricht an. Die *Hello*-Nachricht wird ebenfalls an die Multicast-Gruppe geschickt. Auf diese Weise kann der Client neue Services im Netzwerk ohne wiederholtes Suchen detektieren. Um einige Skalierbarkeitsaspekte zu verbessern sowie die Suche über die Grenzen eines Netzwerks zu ermöglichen bietet WS-Discovery den Managed Mode an. In diesem Modus senden die Services ihre Ankündigungen (*Hello*-Nachrichten) per Unicast an den Discovery Proxy und die Clients senden ihre Suchanfragen *Probe* und *Resolve* ebenfalls per Unicast an den Discovery Proxy. Discovery Proxy kündigt sich im Netzwerk mittels einer Multicast-Hello-Nachricht an. Die Adresse des Discovery Proxy kann auch im Client vorkonfiguriert werden. Der Discovery Proxy stellt allerdings einen Single Point of Failure (SPoF) dar. In einem Netzwerk, das auf Managed Mode ausgelegt ist, führt der Ausfall des Discovery Proxy zu der Nichtverfügbarkeit des Service Discovery.

Die Services, die das Netzwerk verlassen, melden sich mit einer *Bye*-Nachricht ab. Hierdurch werden Clients informiert, dass bestimmte Service Provider nicht mehr erreichbar sind.

2.6. Grundlagen der Gebäudeautomation

Gebäudeautomation ist ein wachsendes Gebiet, das sich mit der Vernetzung der Geräte im Gebäude beschäftigt. Der Bedarf an Vernetzung der Geräte im Gebäude entstand aus dem Wunsch nach mehr Komfort und Energieeffizienz. Durch die Vernetzung lassen sich die Funktionen der Gebäude sowie einzelne Geräte aus der Entfernung steuern. Ein

Gebäudeautomationssystem kann auch vollautomatisch auf bestimmte Ereignisse reagieren und Funktionen auslösen. Das einfachste Beispiel ist die automatische Beleuchtungsanpassung. Befindet sich ein Mensch im Raum, wird seine Präsenz durch spezielle Sensoren wie Präsenzmelder erfasst. Dann kann die Beleuchtung an die voreingestellte Helligkeit mithilfe von zusätzlichen Helligkeitssensoren angepasst werden. Ist der Raum dagegen leer, kann die Beleuchtung ausgeschaltet werden. Diese Funktion stellt gleichzeitig sowohl eine Komfort- als auch eine Energieeinsparfunktion dar.

Um Geräte zu vernetzen, ist ein geeignetes Kommunikationsprotokoll notwendig. Gegenwärtig ist Gebäudeautomation vor allem durch eine Vielzahl proprietärer Protokolle charakterisiert. Diese Protokolle sind in der Regel nicht interoperabel und erschweren damit die Integration verschiedener Hersteller in eine Installation. In diesem Kapitel wird auf ausgewählte Protokolle der Gebäudeautomation eingegangen, die derzeit am Markt vorzufinden sind.

Zur Einordnung der Technologien soll ein 3-Schichten-Modell für Automatisierungstechnik herangezogen werden [59]. Dieses ist in Abbildung 14 dargestellt.

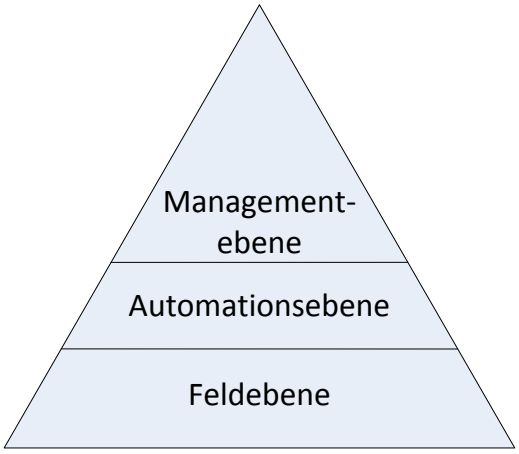

Abbildung 14: Ebenen der Gebäudeautomation

Die Feldebene ist die unterste Ebene in der Automatisierungshierarchie. In der Feldebene erfolgt eine direkte Kommunikation mit Sensoren und Aktoren. Die Kommunikation findet in der Regel über sogenannte Feldbusse statt. Die Automationsebene übernimmt die Steuerung und Regelung von Sensoren und Aktoren sowie das Sammeln der Messwerte. In der Managementebene erfolgen die Überwachung der Automatisierungsanlage sowie die Visualisierung der Daten. Im Folgenden werden ausgewählte Protokolle der klassischen Gebäudeautomation untersucht.

2.6.1. Klassische Gebäudeautomation

2.6.1.1. BACnet

BACnet steht für Building Automation and Control Networks. Das Protokoll wurde von der American Society of Heating, Refrigeration, and Air-Conditioning Engineers (ASHRAE) entwickelt und im Jahr 1995 als American National Standards Institute (ANSI)/ASHRAE-Norm 135 standardisiert [60]. Seit 2003 ist BACnet ebenfalls bei dem Deutschen Institut für Normung (DIN) als eine Europäische Norm (EN) und eine ISO Norm 16484-5 standardisiert [59]. Ziel von BACnet war, eine Interoperabilität zwischen verschiedenen Herstellern zu erreichen, da zu diesem Zeitpunkt jeder Hersteller ein eigenes proprietäres Protokoll eingesetzt hat.

BACnet kann über verschiedene Medien kommunizieren. RS-232 und RS-485 ermöglichen die serielle Kommunikation. Die Benutzung derselben unteren Kommunikationsschichten wie bei Local Operation Network (LonTalk) ist ebenfalls möglich. Weiterhin stellt Attached Resource Computer Network (ARCNET) eine echtzeitfähige Bustechnologie mit Token Passing dar und war als Konkurrenz zu Ethernet (IEEE 802.3) konzipiert. Mit einer raschen Entwicklung des Ethernets hat diese jedoch an Bedeutung verloren und wird nur vereinzelt in der Industrie eingesetzt [61]. Im Jahr 1999 wurde eine Erweiterung des Standards als BACnet/IP verabschiedet [62]. Dadurch wurde BACnet mit der Kommunikationsmöglichkeit über das IP-Protokoll ergänzt. Im Jahr 2011 wurden für BACnet verfügbare Medien mit der Funktechnologie ZigBee ergänzt [63].

Im Rahmen dieser Arbeit wird nur auf die Anwendungsschicht von BACnet eingegangen. Informationen über andere Schichten können der Norm DIN EN ISO 16484-5 entnommen werden [64]. Die Anwendungsschicht definiert ein Datenmodell für die im Gerät verfügbaren Informationen. Diese Informationen werden als Objekte repräsentiert. Die Properties (Eigenschaften) eines Objektes beschreiben bestimmte Aspekte der Software, der Hardware oder des Betriebs eines Gerätes. Alle Objekte eines Gerätes stellen das sichtbare Kommunikationsinterface eines BACnet-Gerätes. Der Zugriff auf die Objekte erfolgt mittels Funktionen, die Services genannt werden.

BACnet definiert 54 Objekttypen [65]. Darüber hinaus können die Hersteller eigene proprietäre Objekttypen definieren. Diese können jedoch in der Regel von anderen Herstellern nicht interpretiert werden. Die 18 Basistypen sind nachfolgend aufgelistet [66]:

- Analog Input
- Analog Output
- Analog Value
- Binary Input
- Binary Output
- Binary Value
- Multi-state Input
- Multi-state Output
- File
- Loop
- Device
- Event Enrollment
- Program
- Schedule
- Calendar
- Command
- Group
- Notification Class

Die mit *Binary* bezeichneten Objekttypen dienen der Steuerung oder der Zustandserfassung einer Komponente des Gerätes mit nur zwei möglichen Zuständen wie z.B. ein Schalter oder ein Relay. Die *Multi-state*-Objekttypen ermöglichen dagegen mehrere Zustände. Alle anderen Werte (Integer, Float) und Steuerungskomponenten, z.B. eines Sensors, können grundsätzlich mit den als *Analog* bezeichneten Objekten erfasst werden. Die *Calendar*- und *Schedule*-Objekte ermöglichen das Einstellen von zeitlichen Parametern. *Command*-Objekt ermöglicht die Ausführung eines Befehls oder einer Prozedur, wobei mehrere Werte in mehreren Objekten auf mehreren Geräten gleichzeitig oder nacheinander gesetzt werden können. Mit dem Objekt *Event Enrollment* können zusätzliche Events wie z.B. Alarmsignale definiert werden. Objekte wie Analog Input, Analog Output usw. verfügen bereits über die Eventing-Funktionalitäten und benötigen damit das *Event Enrollment-Objekt* nicht. Das *Group*-Objekt ermöglicht den Zugriff auf mehrere Objekte während einer Anfrage. Das Objekt *Loop* wird zur Abbildung der Regelalgorithmen verwendet. *Notification Class* steht in Zusammenhang mit *Event Enrollment* und ermöglicht das Management von Event-Subscriptions. Das *Program*-Objekt ermöglicht das Laden, die Ausführung, und das Beenden eines Programms im Gerät. Mit dem *File*-Objekt können Dateien gelesen und geschrieben werden. Eine besondere Rolle hat das *Device*-Objekt. Es enthält Metainformationen über das Gerät wie z.B. Hersteller, Firmware sowie eine Liste von Objekten und Services, die vom Gerät unterstützt werden. Das Device-Objekt steht damit in Zusammenhang mit dem Discovery-Mechanismus der BACnet-Geräte. Jedes Objekt kann in einem Gerät beliebig oft vorkommen, außer dem Device-Objekt.

Die Objekte bestehen aus Properties. Je nach Objekt können sich diese unterscheiden. Einige Properties sind für einen bestimmten Objekttyp obligatorisch, andere optional. BACnet ermöglicht weiterhin die proprietären Properties zu den Standardobjekten hinzuzufügen. Diese können jedoch in der Regel von anderen Herstellern nicht interpretiert werden. Nachfolgend sind beispielhaft die obligatorischen Properties eines Analog Input-Objektes dargestellt. Die Namen der Properties sollen hier selbsterklärend sein:

- Object_Identifier
- Status_Flags
- Object_Name
- Event_State
- Object_Type
- Out_Of_Service

- Present_Value
- Units

Der Zugriff auf Objekte bzw. Properties erfolgt, wie oben erwähnt, über Funktionen, die in BACnet-Kontext Services genannt werden. In Tabelle 3 sind einige ausgewählte Services aufgelistet. Die Services lassen sich in Service-Typen je nach Verwendung einteilen: Objektzugriff-Services, Alarm- und Ereignis-Services, Dateizugriff-Services, Device- und Netzwerkmanagement-Services sowie Virtual-Terminal-Services.

Service	**Erklärung**
ReadProperty/WriteProperty	Eine Property lesen/schreiben
ReadPropertyConditional	Auslesen mehrerer Properties, die bestimmte Bedingungen erfüllen
ReadPropertyMultiple/WritePropertyMultiple	Mehrere Properties lesen/schreiben
Who-Has	Discovery nach einem bestimmten Objekt
I-Have	Antwort auf Who-Has
Who-Is	Discovery nach einem bestimmten Gerät
I-Am	Antwort auf Who-Is
SubscribeCOV	Wertänderungen abonnieren (Change of Value - COV)
ConfirmedCOVNotification/ UnconfirmedCOVNotification	Bestätigte/unbestätigte Benachrichtigung über eine abonnierte Wertänderung

Tabelle 3: Ausgewählte BACnet-Services

Wie aus Tabelle 3 ersichtlich, bietet BACnet nicht nur einen Request-Response- sondern auch einen Notification-Mechanismus an. Der Client kann dabei automatisch über die Wertänderungen benachrichtigt werden, wenn er diese abonniert hat. Darüber hinaus bietet BACnet auch Discovery-Mechanismen an, wobei die Suche sowohl nach Geräten (Who-Is) als auch nach bestimmten Objekten (Who-Has) erfolgen kann. Die beiden Discovery-Nachrichten werden mittels Broadcast geschickt und können bei Bedarf von jedem Gerät beantwortet werden. Ein BACnet-Gerät muss die Discovery-Funktionalität nicht anbieten. Die BACnet-Services können als Nachrichten verstanden werden. Diese werden sowohl vom Service Provider als auch vom Client angeboten.

Für die Interoperabilität zwischen dem Service Provider und dem Client werden BACnet Interoperability Building Blocks (BIBB) eingesetzt. Diese beschreiben, welche Funktionalitäten der Service Provider und der Client unterstützen müssen, um eine bestimmte Operation ausführen zu können. Korrespondierende BIBBs bei dem Client und dem Service Provider sind Voraussetzung für eine funktionierende Kommunikation. Die BIBBs werden in 5 Interoperabilitätsbereiche (IOB) aufgeteilt [65]:

- Gemeinsame Datennutzung (Data Sharing, DS)

- Alarm- und Ereignis-Verarbeitung (Alarm and Event Management, AE)
- Zeitplan (Scheduling, SCHED)
- Trendaufzeichnung (Trending, T)
- Device- und Netzwerkmanagement (Device and Netzwork Management, DM)

Der BIBB-Name gibt an, zu welchem IOB er gehört, um welche Funktion es sich handelt und ob es sich um die Sicht des Clients (A) oder des Service Providers (B) handelt. Möchte beispielsweise ein Client die Operation ReadProperty an einem Sensor ausführen, muss dieser den BIBB DS-RP-A und der Sensor DS-RP-B entsprechend unterstützen. Eine vollständige Liste von BIBBs kann der BACnet-Spezifikation entnommen werden. Um die BACnet-Geräte nach ihrer Leistungsfähigkeit bewerten zu können, wurden Konformitätsklassen eingeführt. Jede Klasse beschreibt die jeweiligen Mindestanforderungen an ein Gerät. In Tabelle 4 sind die definierten Klassen aufgelistet. Die höheren Konformitätsklassen schließen die Eigenschaften der niederen Klassen bereits mit ein.

Um das Gerät einer Klasse zuzuordnen, werden von Herstellern Protocol Implementation Conformance Statement (PICS) verfasst. PICS enthält Informationen darüber, welche BIBBs unterstützt werden, welche Services als Client und Service Provider verfügbar sind, welche Standard-Objekttypen und proprietäre Objekttypen eingesetzt werden. Der oben beschriebene Aufbau der BACnet-Anwendungsschicht ist in Abbildung 15 dargestellt.

BACnet kann grundsätzlich auf allen Ebenen der Gebäudeautomation eingesetzt werden (vgl. Abbildung 14). Es findet in der Praxis jedoch vor allem Verwendung in der Automations- und Managementebene [59].

Kl.	**Bezeichnung**	**Erklärung**
1	Smart Sensor	Ein Gerät kann lediglich Objekt-Properties melden (ReadProperty)
2	Smart Actuator	Ein Gerät kann eine oder mehrere Objekt-Properties schreiben
3	Application Specific Controller	Ein Gerät kann mehrere Objekt-Properties lesen und schreiben, es kann sich selbstständig im Netzwerk anmelden, identifizieren und Auskunft über seine Objekte geben.
4	Advanced Application Controller	Das Gerät beinhaltet Client-Services, es kann ReadProperty und WriteProperty initiieren
5	Building Controller	Eine programmierbare Station, die für verschiedene Steuerungs- und Regelungsaufgaben eingesetzt werden kann
6	Operator Workstation	Benutzerschnittstelle zum BACnet-System

Tabelle 4: BACnet-Konformitätsklassen

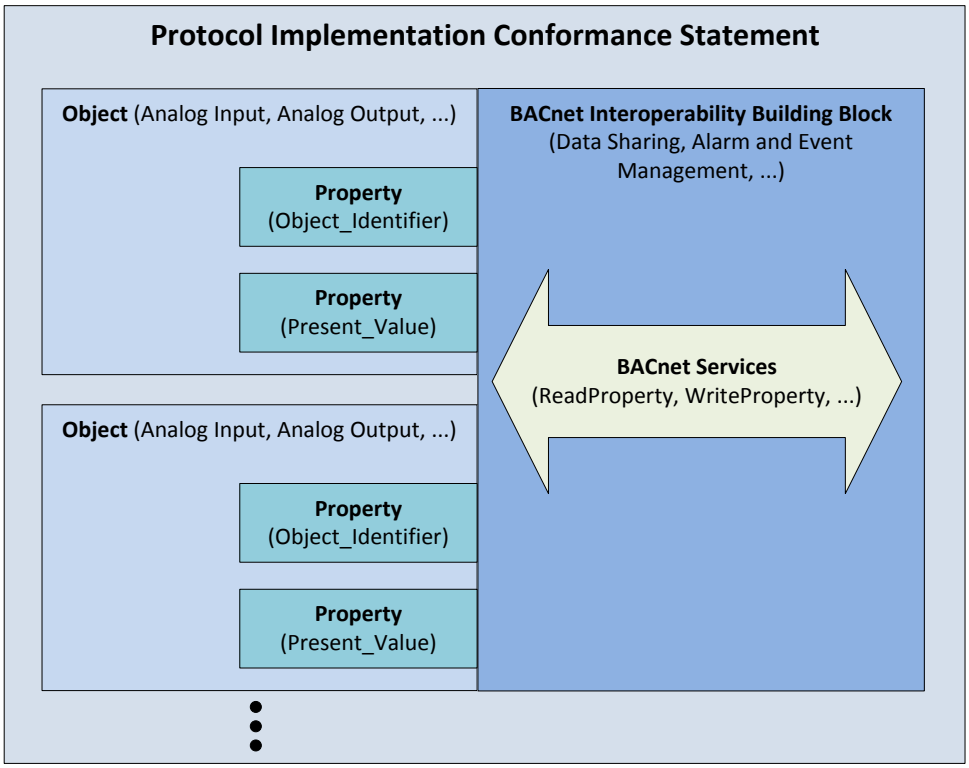

Abbildung 15: Struktur der BACnet-Anwendungsschicht

2.6.1.2. KNX

KNX ist ein Feldbus zur Vernetzung von Geräten in der Gebäudeautomation. KNX entstand als ein Zusammenschluss des Europäischen Installationsbusses (EIB), des BatiBUS und der European Home Systems (EHS). Die erste Spezifikation von KNX wurde von der KNX Association im Jahr 2002 veröffentlicht [67]. KNX ist dabei zum EIB rückwärtskompatibel. 2003 wurde KNX zum europäischen Standard EN 50090 [68], 2006 zum internationalen Standard ISO/IEC 14543-3.

KNX ist mehr als ein reines Kommunikationsprotokoll. Neben der Definition, wie die Daten transportiert werden, beschreibt der Standard ebenfalls, wie ein KNX-System verwaltet und wie die Geräte von verschiedenen Herstellern implementiert werden müssen, um ein ordnungsgemäßes Zusammenwirken zu ermöglichen [69].

Der KNX-Standard sieht für die Kommunikation der Geräte folgende Medien vor: KNX Twisted Pair (KNX TP), KNX Powerline (KNX PL), KNX Radio Frequency (KNX RF) sowie Ethernet (KNX IP) [70]. Die Planung einer KNX-Installation erfolgt in vier Schritten. Zuerst muss ein Übertragungsmedium gewählt werden. Danach müssen die erforderlichen Geräte wie Sensoren und Aktoren ausgewählt werden. Im nächsten Schritt sollen den Geräten eindeutige physikalische Adressen zugewiesen werden, wodurch auch die Topologie

eindeutig festgelegt wird. Die KNX-Topologie besteht aus Linien und Bereichen. Die Teilnehmer werden bestimmten Linien zugeordnet. Mehrere Linien werden an eine Hauptlinie angeschlossen und stellen einen Bereich dar. Die Bereiche können weiterhin über eine Bereichslinie miteinander verbunden werden. Die Geräteadressen werden in Form von *Bereich.Linie.Teilnehmer* angegeben. Neben den physikalischen Adressen erhalten die Geräte auch logische bzw. Gruppenadressen. Abschließend müssen die Geräte mit der speziellen Software „Engineering Tool Software (ETS)“ programmiert werden.

Die Kommunikation in KNX basiert auf dem Publish/Subscribe-Modell [71]. Die Sender benutzen dabei Gruppen-Adressen, um die Nachrichten mittels Multicast zuzustellen. Ein Gerät kann dabei Mitglied in mehreren Gruppen sein. Die Unicast-Nachrichten werden nur für Konfigurations- und Management-Zwecke genutzt.

Da KNX unterschiedliche Medien unterstützt, lässt sich der Protokoll-Stack in medienabhängige und medienunabhängige Schichten aufteilen. Im Rahmen dieser Arbeit wird nur die Anwendungsschicht näher erläutert. Informationen über andere Schichten können dem KNX-Standard EN 50090 entnommen werden.

Zum Datenaustausch verwendet KNX Kommunikationsobjekte (engl. Datapoints), die in Funktionsblöcke zusammengefasst werden. Diese sind das zentrale Konzept im KNX-Datenmodell. Die KNX-Kommunikationsobjekte können sich auf den Zustand eines Aktors, auf den Wert eines Sensors oder auch einen Parameter beziehen, der das Verhalten eines Gerätes steuert [69]. Die Kommunikationsobjekte können sowohl aus einem atomaren Datentyp als auch aus mehreren atomaren Datentypen (strukturierter Datentyp) bestehen [72]. Folgende typische atomare Datentypen werden von KNX unterstützt: Boolean, Character, 8 Bit Unsigned Integer, 8 Bit Signed Integer, 16 Bit Float usw. Ein Kommunikationsobjekt besitzt weiterhin eine Reihe von Attributen wie z.B. Nummer, Name, Funktion, Gruppenzugehörigkeit, Flags. Die Flags bestimmen, ob ein Kommunikationsobjekt gelesen oder geschrieben werden kann oder ob Wertänderungen gemeldet werden müssen. Die Metainformationen der Kommunikationsobjekte sind jedoch im regulären Betrieb nicht für andere Geräte sichtbar, sondern sie werden in der Konfigurationsphase gesetzt bzw. ausgelesen. Beispielhafte KNX-Kommunikationsobjekte sind ein Schalter (1 Bit Ein/Aus) oder ein relativer Dimmer (*Integer* Prozent). Jedes Kommunikationsobjekt wird einer Gruppe zugeordnet und ist über die Gruppenadresse adressierbar.

Die Kommunikationsobjekte werden zu Funktionsblöcken zusammengefasst und stellen in der Regel die Geräte selbst dar. Ein verbreiteter Funktionsblock ist ein elementarer Dimmer, der aus einem Schalter, einem relativen Dimmer und einem absoluten Dimmer besteht. Alle anderen Kommunikationsobjekte sind optional.

Das Verknüpfen von Kommunikationsobjekten wird mittels Bindings realisiert [69]. Der Zugriff auf die Kommunikationsobjekte erfolgt mittels Services. Die KNX-Services sind mit den Nachrichtentypen vergleichbar. Die Services lassen sich in zwei Klassen aufteilen. Die erste Klasse dient dem Datenaustausch zwischen den Geräten. Die wichtigsten und am meisten benutzten sind Services für die Gruppenkommunikation. Diese sind A_GroupValue_Read und A_GroupValue_Write. Die zweite Serviceklasse beinhaltet Services zur Konfiguration und Wartung wie z.B. das Setzen von Gruppenadressen, das Übertragen einer spezifischen Anwendung usw. Sowohl A_GroupValue_Read als A_GroupValue_Write Services werden als Multicast verschickt. A_GroupValue_Write Service wird benutzt, um neue Daten an eine Gruppe zu verteilen. A_GroupValue_Write ist ein unbestätigter Service und garantiert somit nicht die Zustellung der Nachricht. Jeder Knoten, der die angegebene Gruppenadresse besitzt, kann die Nachricht verarbeiten. Da keine andere Datenübertragungsmöglichkeit im laufenden Betrieb außer Multicast besteht, sind Sicherheitsmechanismen bei KNX schwer umsetzbar. Der A_GroupValue_Read Service repräsentiert dagegen eine Anfrage. Diese wird von allen Knoten, die der Gruppe angehören, mit A_GroupValue_Response beantwortet. Die Antwort wird ebenfalls per Multicast verschickt und ist äquivalent zur A_GroupValue_Write-Nachricht. Das bedeutet jedoch, dass die Knoten, die keine Anfrage gestellt haben, die Antwort trotzdem bekommen. Der oben beschriebene Aufbau der KNX-Anwendungsschicht ist in Abbildung 16 dargestellt.

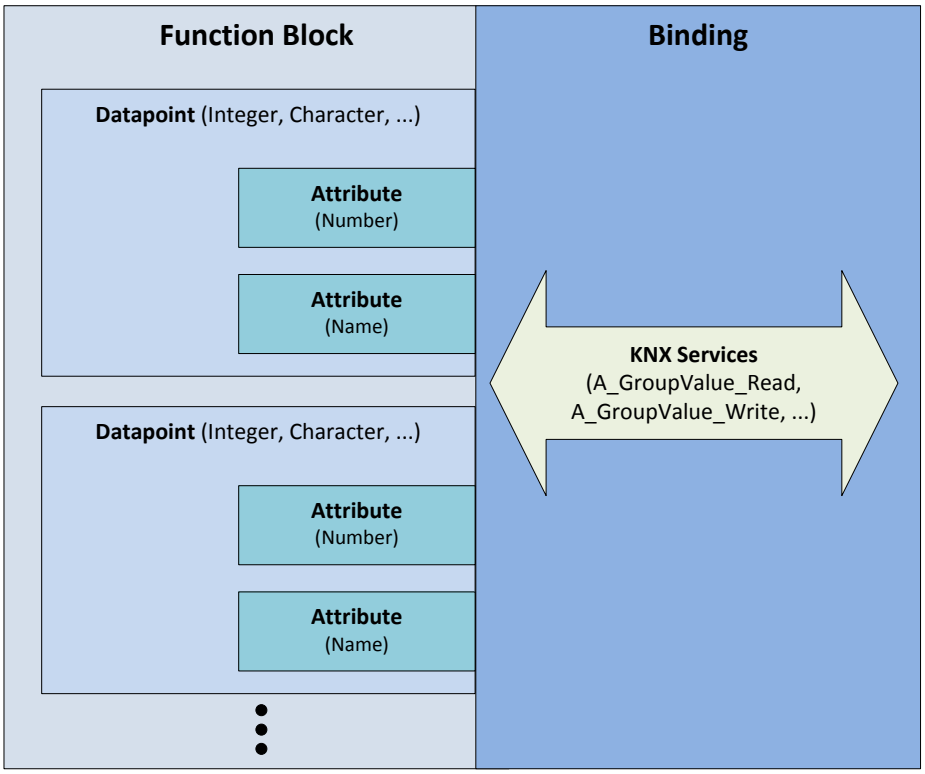

Abbildung 16: Struktur der KNX-Anwendungsschicht

Mit dem KNX-Standard können prinzipiell sowohl kleine als auch große Installationen umgesetzt werden. KNX-Produkte sind jedoch recht teuer. Eine Investition in die KNX-Installation lohnt sich im Allgemeinen nur, wenn mehrere Gewerke mit einander verknüpft werden sollen [59]. Die Installation von KNX-Geräten ist grundsätzlich nur durch Elektrohandwerksbetriebe oder Ingenieurbüros möglich.

KNX wird primär in der Feldebene der Gebäudeautomation eingesetzt (vgl. Abbildung 14). Einzig kann die IP-basierte Variante von KNX in der Automationsebene Verwendung finden [59].

2.6.1.3. LON

LON steht für Local Operation Network und ist ein im Jahr 1990 von der Firma Echelon entwickeltes Bussystem. LON oder auch LonWorks wurde zunächst im Jahr 1999 als ANSI / Electronic Industries Alliance (EIA)-709 und ANSI / Consumer Electronics Association (CEA)-709 standardisiert [73]. LON wurde erst 2005 zum europäischen Standard EN 14908 und 2008 zum internationalen Standard ISO / International Electrotechnical Commission (IEC) 14908 [74] [75]. LON steht in direkter Konkurrenz zu KNX.

LON besteht aus dem Kommunikationsprotokoll LonTalk, dem Neuron Chip, dem LonWorks Transceiver und den LonWorks Tools [59]. Der Neuron Chip stellt das Herzstück der LON-Technologie dar. Er beinhaltet 3 Prozessoren, Read Only Memory (ROM) und Random Access Memory (RAM). Jeder Prozessor ist für eine spezielle Aufgabe bestimmt. Der MAC-Prozessor steuert den Zugriff auf das Medium sowie den Empfang und Versand von Nachrichten. Der Netzwerkprozessor verarbeitet die oberen Schichten des LonTalk-Protokolls. Der Anwendungsprozessor führt eine spezifische Anwendung aus, die die Aufgaben eines Gerätes definiert [73]. Die Programmierung für den Neuron Chip erfolgt in einer speziellen Sprache Neuron C, die einen ANSI C Dialekt darstellt. Neben dem für LON üblichen Neuron Chip gibt es vereinzelte Lösungen, die auf einem Advanced „Reduced Instruction Set Computer (RISC)“ Machines (ARM)-Prozessor basieren, wie z.B. LC3020 Chip der Firma LOYTEC.

Für den Anschluss des Neuron Chips an das Übertragungsmedium wird ein separates Bauteil, der LonWorks Transceiver, verwendet. Für die Übertragung des LonTalk-Protokolls stehen Transceiver für folgende Medien zur Verfügung: TP, RS-485, PL, RF, Lichtwellenleiter (LWL), Infrarot (IR), Ethernet und andere [76]. Damit LON-Geräte im Netzwerk erkannt werden, haben sie eine weltweit eindeutige 48 Bit Identifikationsnummer, die Neuron ID genannt wird.

Für die Konfiguration der Geräte werden spezielle LonWorks Tools verwendet. Während der Konfiguration werden den Geräten logische Adressen zugewiesen. Diese bestehen aus einer

Domain, einem Subnetz und einer Node ID. Die Domain stellt das gesamte LON-Netzwerk dar, das mit einer Domain ID identifiziert wird. Das LON-Netzwerk besteht physikalisch aus mehreren Kanälen, die durch Router und Repeater verbunden sind. Logisch werden sie in Subnetze unterteilt. LON implementiert das ISO/OSI-Netzwerkmodell und besitzt sowohl medienabhängige als auch medienunabhängige Schichten. Im Rahmen dieser Arbeit wird jedoch hauptsächlich auf die Anwendungsschicht von LON eingegangen.

Die Netzwerkschicht von LON ermöglicht Unicast-, Multicast- und Broadcast-Nachrichten. Das LonTalk-Protokoll stellt für den Datenaustausch zwischen den Geräten im Betrieb vier Services zur Verfügung, die in Tabelle 5 dargestellt sind. Neben den Services für den Datenaustausch existiert noch eine Reihe von weiteren Services für das Netzwerkmanagement und die Diagnostik.

Service	**Erklärung**
Acknowledged	Die Nachricht wird an einen oder mehrere Knoten geschickt und eine Empfangsbestätigung wird erwartet.
Request/Response	Die Nachricht wird an einen oder mehrere Knoten geschickt und eine Antwort wird erwartet.
Repeated	Die Nachricht wird mehrmals an einen oder mehrere Knoten geschickt
Unacknowledged	Die Nachricht wird einmal an einen oder mehrere Knoten geschickt

Tabelle 5: LonTalk-Services für den Datenaustausch

Für den Austausch von Daten werden sogenannte Netzwerkvariablen verwendet [73]. Sie sind grundlegende Kommunikationsobjekte, die logische Datenpunkte der Anwendung darstellen. Die Netzwerkvariablen besitzen eine Reihe von Properties, die Metadaten eines Kommunikationsobjektes darstellen. Die Eingangsnetzwerkvariablen können Werte vom Netzwerk empfangen und die Ausgangsnetzwerkvariablen die Werte ins Netzwerk schicken. Die Verknüpfungen zwischen den Eingangs- und Ausgangsnetzwerkvariablen werden mittels Bindings realisiert, die während der Konfiguration auf den Geräten durch das Setzen spezieller Tabelleneinträge eingerichtet werden. Die Werte werden automatisch gemeldet, wenn sich der Wert geändert hat, oder auch in eingestellten zeitlichen Abständen. Wenn sich beispielsweise die Werte der Ausgangsnetzwerkvariablen ändern, werden die Tabelleneinträge benutzt, um die Nachrichten an die Knoten zu übermitteln, die die entsprechenden verknüpften Eingangsvariablen besitzen. Damit werden die Werte der Eingangsvariablen auf den verknüpften Knoten aktualisiert. Die Bindings können 1-zu-1, 1-zu-n und n-zu-1 sein. 1-zu-n-Bindings werden benutzt, wenn mehrere Eingangsnetzwerkvariablen den Wert einer Ausgangsnetzwerkvariable brauchen. Ein n-zu-1 Binding ist problematischer, da der Wert einer Eingangsnetzwerkvariable unter Umständen

von mehreren Quellen gleichzeitig überschrieben wird. Die Grundvoraussetzung für ein Binding ist die Kompatibilität der Ausgangsvariablen eines Sensors mit den Eingangsvariablen eines Aktors [59].

Um spezielle Automationsfunktionen und die entsprechenden Netzwerkvariablen zu unterteilen, werden sie ähnlich wie bei KNX in die Funktionsblöcke zusammengefasst. Ein Funktionsblock definiert das Interface eines Gerätes, das aus den Eingangs- und Ausgangsnetzwerkvariablen sowie Konfigurations-Properties besteht. Die Konfigurations-Properties können dabei nur von den LonWorks Tools gesetzt werden. Um die Interoperabilität zwischen verschiedenen Herstellern zu erreichen, werden standardisierte Templates für die Implementierung der Funktionsblöcke, sogenannte Standard Functional Profile Types (SFPT), von LonMark definiert. Diese definieren die obligatorischen und optionalen Netzwerkvariablen und Konfigurations-Properties. Für die Interoperabilität der Netzwerkvariablen und Konfigurations-Properties werden Standard Network Variable Types (SNVT) und Standard Configuration Property Types (SCPT) von LonMark spezifiziert [77]. Diese Variablen können von atomaren oder auch komplexen Datentypen sein. Zu den atomaren Datentypen gehören u.a. Short, Long und Quad Integer, Float, Enumeration usw. Die komplexen Datentypen sind Struct oder Union, gebildet aus atomaren Datentypen. Eine beispielhafte komplexe Netzwerkvariable ist SNVT_environment, die als ein Struct u.a. aus SNVT_environment.supplyVoltage, SNVT_environment.supplyCurrent und SNVT_environment.power besteht. Die Hersteller können weiterhin eigene proprietäre Netzwerkvariablen und Konfigurations-Properties verwenden, die jedoch von anderen Herstellern in der Regel nicht interpretiert werden können. Der oben beschriebene Aufbau der LON-Anwendungsschicht ist in Abbildung 17 dargestellt.

Ähnlich wie bei KNX ist die Installation von LON-Geräten grundsätzlich nur durch Elektrohandwerksbetriebe oder Ingenieurbüros möglich. Als eine klassische Bustechnologie wird LON in der Feldebene der Gebäudeautomation eingesetzt [59].

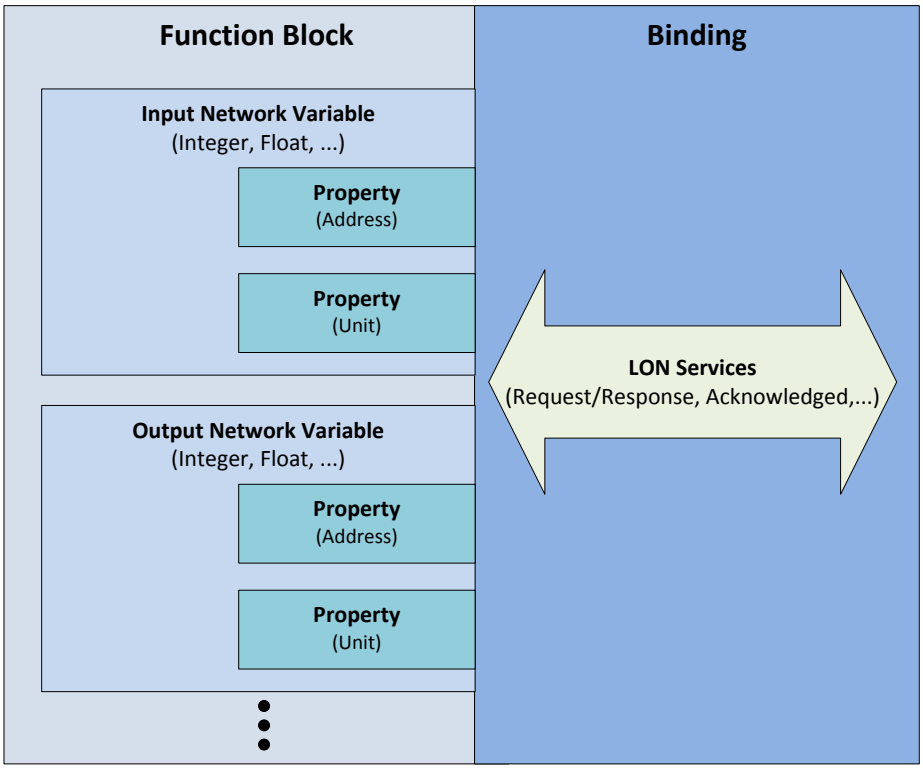

Abbildung 17: Struktur der LON-Anwendungsschicht

2.6.2. Neue Technologien der Gebäudeautomation und des Smart Home

2.6.2.1. Z-Wave

Z-Wave ist eine proprietäre Technologie für die Gerätekommunikation. Zum Zeitpunkt des Schreibens wurden nur die physikalische und Sicherungsschicht von Z-Wave von International Telecommunication Union Telecommunication Standardization Sector (ITU-T) im Jahr 2012 als G.9959 Norm standardisiert. Z-Wave wurde von der Firma Sigma Designs (ehem. Zensys) entwickelt. Diese ist gleichzeitig ein Hersteller der Z-Wave-Chips. Andere Hersteller, die die Technologie nutzen wollen, sind Mitglieder der Z-Wave Alliance. Ziel von Z-Wave ist, eine einfache und zuverlässige Methode zur Steuerung von Geräten zu Hause zu entwickeln [78]. Das Anwendungsgebiet von Z-Wave soll in erster Linie das Smart Home sein.

Z-Wave ist eine reine Funktechnologie. Für die Signalübertragung wird das Short Range Devices (SRD)-Band 868,42 MHz benutzt [79]. Die Z-Wave-Geräte bilden ein Mesh-Netzwerk, d.h., die Kommunikation läuft zwischen zwei Knoten über andere Netzwerkknoten als Vermittler ab. Die vermittelnden Knoten können dabei gewöhnliche Geräte sein, die ebenfalls Sensoren oder Aktoren sind. Eine Nachricht kann jedoch maximal vier Mal weitergeleitet werden [78].

In einem Z-Wave-Netzwerk gibt es zwei Gerätetypen: Controller und Slaves [80]. Die Controller können die Routen im Mesh-Netzwerk berechnen. Die Slaves sind Sensoren und Aktoren. Die berechneten Routen können an die Slaves weitergeleitet werden, um Verbindungswege zu erstellen. Das Netzwerk wird um einen Controller herum aufgebaut. Es können mehrere Controller im Netzwerk vorhanden sein, jedoch kann nur einer davon der Primary Controller sein. Dieser entscheidet u.a., welche Geräte dem Netzwerk beitreten dürfen. Die Controller können Kommandos initiieren und diese an andere Knoten schicken. Die Slaves enthalten grundsätzlich keine Routing-Tabelle. Die sogenannten Routing Slaves oder Enhanced Routing Slaves können vorkonfigurierte Routen vom Controller bekommen. Batteriebetriebene Geräte, die nicht permanent auf den Netzwerkverkehr lauschen, werden nicht als Vermittlerknoten berücksichtigt. Die Slaves können nur auf Kommandos antworten und diese ausführen [81]. Sie können nicht selbstständig Informationen an andere Slaves oder Controller senden, es sei denn, sie wurden dazu mit einem Kommando aufgefordert. Um Änderungen der Werte oder des Zustandes der Slaves festzustellen, muss der Controller die Slaves regelmäßig abfragen. Eine Ausnahme bilden die Routing Slaves, die Nachrichten wie z.B. Alarmnachrichten unaufgefordert schicken können.

Um einem Netzwerk beizutreten, benötig das Gerät mindestens einen „Initiator“. Ein Initiator kann ein physikalisch vorhandener Knopf, eine spezielle Betätigung des Knopfes (z.B. gedrückt halten für einige Sekunden), eine Kombination von Knöpfen (z.B. zwei Knöpfe gleichzeitig drücken) oder ein Punkt in einem Auswahlmenü sein. Die Slave-Initiatoren werden benutzt, um dem Netzwerk beizutreten bzw. es zu verlassen und die Knoten mit einander zu verknüpfen bzw. zu trennen. Die Controller verfügen darüber hinaus über Operations-Initiatoren, die für die Steuerung der verknüpften Geräte benutzt werden. Geräte, die über keine Knöpfe verfügen, können nur mittels Controller verknüpft werden.

Mehrere Z-Wave-Netzwerke werden voneinander mittels Home ID getrennt. Die Home ID ist ein 32 Bit Identifier und wird fest in die Controller einprogrammiert. Um einzelne Geräte zu adressieren, wird eine Node ID verwendet. Diese ist 8 Bit breit und wird vom Primary Controller zugewiesen. Ein Z-Wave-Netzwerk kann jedoch maximal 232 Knoten enthalten. Z-Wave unterstützt alle drei Standard-Adressierungsarten Unicast, Multicast und Broadcast. Die Unicast-Nachrichten können sowohl bestätigt als auch unbestätigt sein. Die Multicast-Nachrichten werden dabei nicht wie üblich an eine Gruppenadresse geschickt, sondern sie enthalten eine Liste von Zieladressen, die die Nachricht bekommen sollen.

Um gerätespezifische Informationen zu bekommen, werden Node Information Frames geschickt. Dieser Frame ist ein Teil des Z-Wave-Protokolls und beschreibt die Fähigkeiten eines Gerätes wie z.B. Gerätetyp (Controller oder Slave), welche Funktionen unterstützt werden, ob das Gerät Nachrichten weiterleiten kann und andere protokollspezifische Parameter [78].

Die Anwendungsschicht von Z-Wave besteht aus drei Teilen: Kommandoklasse, Kommando und Kommandoparameter. Kommandoklassen sind eine Gruppe von Kommandos und Antworten, die eine bestimmte Funktion eines Gerätes abbilden. Die Kommandoklassen werden in zwei Typen unterteilt: Z-Wave-Protokollkommandos und anwendungsspezifische Kommandos. Die Z-Wave-Protokollkommandos werden hauptsächlich für das Setzen der Home ID und Node ID auf den Geräten verwendet. Z-Wave definiert eine Menge von anwendungsspezifischen Kommandoklassen wie z.B. Multilevel_Sensor oder Thermostat_Setpoint. Für die Kommunikation müssen einige Kommandoklassen von den Geräten unterstützt werden, die für eine bestimmte Geräteklasse wie z.B. Thermostat vorgeschrieben sind. Welche Kommandoklassen unterstützt werden, wird von den Geräten während des Netzwerkbeitritts dem Controller mittels Node Information Frame gemeldet. Die meisten Kommandos beschränken sich auf SET-, GET- und REPORT-Methoden. Die SET-Methode ermöglich eine Modifikation eines Datums auf dem Gerät. GET ist eine Abfrage eines Datums und REPORT ist eine Antwort auf diese Abfrage [78]. Die Kommandoklasse SwitchMultilevel würde z.B. auf eine GET-Anfrage Get() den aktuellen Zustand des Schalters liefern. Mit der SET-Anfrage Set(level, duration) kann der neue Zustand des Schalters gesetzt und optional ein Zeitintervall für den neuen Zustand angegeben werden [82]. Die Parameter, die zusammen mit den Kommandos übertragen werden, haben in der Regel atomare Datentypen wie Integer, Float, String. Komplexe Datentypen werden bei einigen wenigen Kommandoklassen eingesetzt. Um Interoperabilität zu gewährleisten, müssen alle Kommandos, die zu einer Klasse gehören, auf dem Gerät implementiert werden. Der oben beschriebene Aufbau der Z-Wave-Anwendungsschicht ist in Abbildung 18 dargestellt.

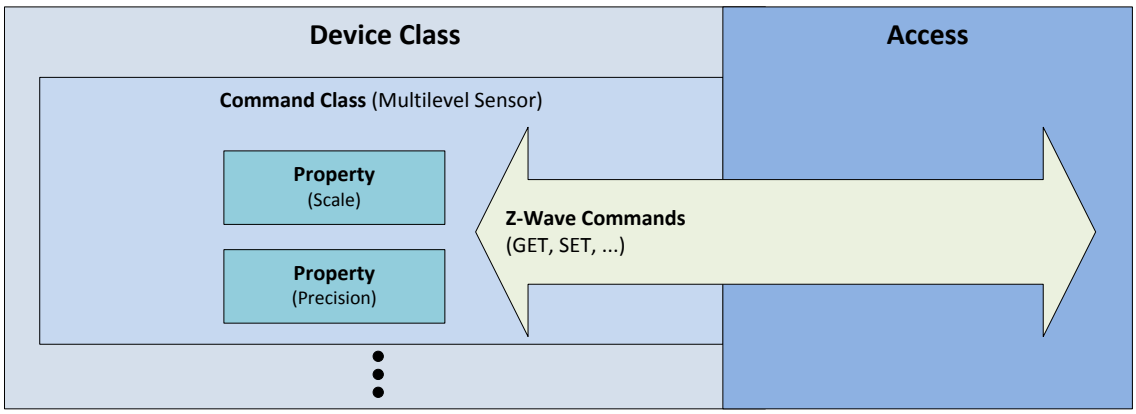

Abbildung 18: Struktur der Z-Wave-Anwendungsschicht

Der Vorteil der Z-Wave-Technologie ist eine leichte Konfigurierbarkeit der Geräte ohne ein tiefgreifendes Fachwissen. Der Primary Controller vom Z-Wave-Netzwerk stellt jedoch einen SPoF dar. Fällt dieser aus, wird das gesamte Netzwerk außer Betrieb gesetzt. Alle Geräte müssen dabei zurückgesetzt und mit dem neuen Primary Controller neu konfiguriert werden [80].

2.6.2.2. EnOcean

Die EnOcean-Technologie wurde von der Firma EnOcean GmbH entwickelt, die im Jahr 2001 als ein Spin-Off der Siemens AG entstand [83]. Das Hauptaugenmerk von EnOcean liegt auf den batterielosen energieautarken Geräten. Die Energie wird dabei aus der Umgebung mittels „Energy Harvesting“ geschöpft. Im Jahr 2012 wurde EnOcean zum internationalen Standard ISO/IEC 14543-3-10. Der Standard deckt dabei die OSI-Schichten 1-3 ab. Die Anwendungsschicht ist derzeit noch nicht standardisiert. EnOcean spezifiziert ein drahtloses Protokoll für Geräte mit niedrigem Energieverbrauch für das Wohnumfeld. Das Protokoll ist so entwickelt, dass der Energieverbrauch von Geräten extrem niedrig gehalten werden kann [84]. Andere Hersteller, die EnOcean-Produkte herstellen wollen, haben sich in der EnOcean Alliance zusammengeschlossen.

EnOcean ist eine reine Funktechnologie. Für die Signalübertragung wird in Europa das SRD-Band 868 MHz verwendet. Die Energiequellen können mechanischer (elektrodynamischer Energiewandler), solarer (Solarzellen) oder thermischer (Peltier-Elemente) Herkunft sein [83]. Für die Umgebungen oder Anwendungen, die kein Energy Harvesting ermöglichen, können ebenfalls batteriebetriebene Geräte eingesetzt werden. Im Gegensatz zu Z-Wave bildet EnOcean kein Mesh-Netzwerk, d.h., zwei kommunizierende Geräte müssen in unmittelbarer Reichweite sein. Falls das nicht möglich ist, können EnOcean Repeater Abhilfe schaffen. Diese können das Signal verstärken und damit längere Distanzen überbrücken [85].

Die Datenübertragung findet in der Regel unidirektional statt. Grund dafür vor allem, dass die Geräte nur für kurze Zeit eine Energieversorgung haben. Im Gegensatz zu anderen Funkprotokollen ist ein Mechanismus zur Kollisionsvermeidung (CSMA/CA) optional und wird von den energieautarken Geräten nicht unterstützt. Um die Zuverlässigkeit der Zustellung der Datenpakete zu erhöhen, werden diese bis zu drei Mal hintereinander gesendet [86]. Eine bidirektionale Kommunikation ist mittels Smart Acknowledge möglich [87]. Diese Funktion ist jedoch optional, da diese bei Sensoren einen zusätzlichen Energieverbrauch impliziert. Darüber hinaus ist ein zusätzlicher netzbetriebener Repeater oder Controller notwendig, der die Rolle eines Postmasters für den Sensor übernimmt.

Für die Interoperabilität zwischen verschiedenen Herstellern werden EnOcean Equipment Profiles (EEP) definiert. Ein EEP besteht aus drei Teilen: Radio-Telegramm-Typ, Basisfunktionalität und Gerätetyp. Der Typ des Telegramms beschreibt, welche Anwendungsdaten übertragen werden wie z.B. „Variable Length Data“, „Manufacturer Specific Communication“ oder „Remote Management“ [88]. Die Basisfunktionalität beschreibt eine Geräteklasse wie z.B. „Schaltwippe mit 2 Tasten“ oder „mechanische Klinke“. Der Gerätetyp beschreibt ein konkretes Gerät aus der Klasse wie z.B. Lichtsteuerung oder Fensterklinke respektive. Die übertragenen Nutzdaten sind stets von einem atomaren

Datentyp wie Integer oder Enumeration. Es können jedoch mehrere Werte in einem Telegramm geschickt werden wie z.B. Helligkeits- und Temperaturwerte.

Das Koppeln der EnOcean Geräte wird als „Teach-in“ (Lernprozess) bezeichnet. In der Teach-in-Prozedur wird festgelegt, welcher Empfänger (Controller) auf welchen Sender (Sensor/Schalter) hören soll. Für diesen Zweck besitzt jeder Sender eine einzigartige Sender-ID. Anhand der Sender-ID erkennt der Empfänger, ob das Gerät bereits bekannt (gelernt) oder unbekannt ist. Nachrichten von unbekannten Sendern werden verworfen. Der Lernprozess läuft grundsätzlich in zwei Schritten ab. Zuerst wird der Empfänger in den Lernmodus versetzt. Das kann durch die Betätigung einer speziellen Taste auf dem Gerät erfolgen. Der Sender soll nun sein Telegramm einmalig schicken. Das kann ebenfalls durch das Drücken einer Taste erfolgen. Im Lernmodus werden die empfangenen Telegramme vom Empfänger als autorisiert betrachtet und die Sender-ID wird gespeichert. Weitere Schritte können abhängig vom Gerätetyp folgen. Damit keine anderen Geräte fälschlicherweise gelernt werden, wird die Empfangsstärke des Empfängers reduziert. Die Geräte müssen dabei sehr dicht aneinander gebracht werden. Die Controller können alternativ mittels Remote Management-Nachrichten konfiguriert werden. Diese Art der Konfiguration muss jedoch von Controllern nicht zwingend unterstützt werden [88]. Die Nachrichten vom Sender werden immer als Broadcast geschickt. Der Sender hat somit keine Informationen, welche Empfänger die Nachricht erhalten sollen. Jeder Empfänger entscheidet beim Empfang eines Telegramms anhand der Sender-ID, ob er diese Nachricht auswerten soll oder nicht. Der oben beschriebene Aufbau der EnOcean-Anwendungsschicht ist in Abbildung 19 dargestellt.

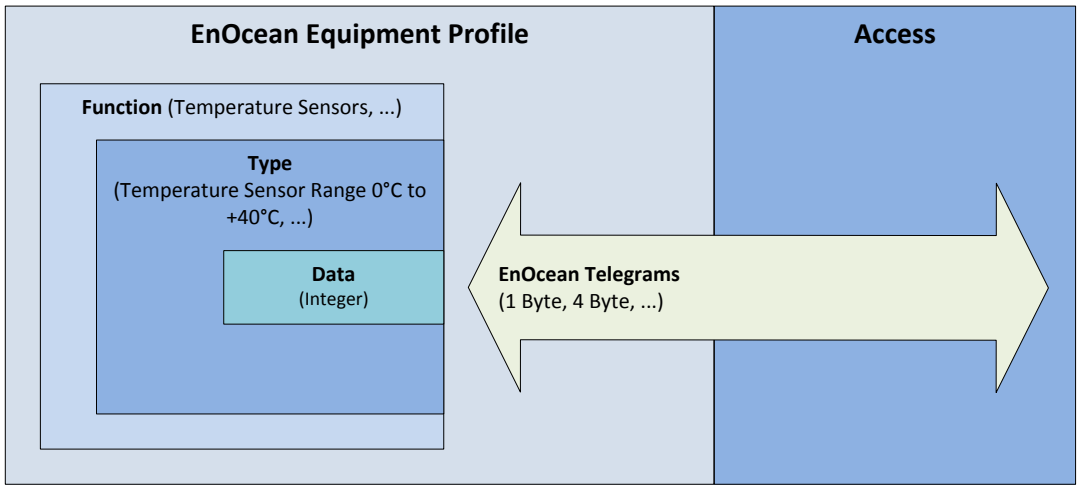

Abbildung 19: Struktur der EnOcean-Anwendungsschicht

EnOcean-Geräte stellen eine relativ einfache Funktionalität bereit. Für komplexere Installationen werden deswegen Gateways zur Anbindung der EnOcean-Geräte in die klassischen Protokolle der Gebäudeautomatisierung wie BACnet [89], KNX [90] und LON [91] angeboten. Die EnOcean-Technologie eignet sich prinzipiell sowohl für

Gebäudeautomation als auch Smart Home. Die Gerätepreise sind jedoch u.a. technologiebedingt relativ teuer.

2.6.2.3. ZigBee

ZigBee ist eine drahtlose Technologie für die Verbindung von kostengünstigen Geräten mit geringem Stromverbrauch. ZigBee basiert auf dem Standard IEEE 802.15.4, der die physikalische und Sicherungsschicht beschreibt. Die ZigBee-Technologie wird von ZigBee Alliance entwickelt, die 2002 gegründet wurde [92]. Im Gegensatz zu den meisten Protokollen der Gebäudeautomation ist die Spezifikation von ZigBee öffentlich verfügbar, was es zu einem offenen Standard macht. ZigBee wird als direkter Konkurrent zu Z-Wave angesehen.

ZigBee ist eine reine Funktechnologie. Für die Signalübertragung können sowohl 2,4 GHz als auch 868 MHz verwendet werden. Die physikalische Datenübertragung sowie der Medienzugriff werden in IEEE 802.15.4 standardisiert. Die ZigBee-Spezifikation beschreibt nur die oberen Netzwerkschichten (ab der Vermittlungsschicht). Es werden mehrere Netzwerktopologien wie Stern, Baum und Mesh unterstützt [93]. In der Sterntopologie wird das Netzwerk durch ein einziges Gerät, den sogenannten Coordinator, gesteuert. Alle anderen Geräte kommunizieren direkt mit dem Coordinator. In der Baum- und Mesh-Topologie wird das Netzwerk von einem Coordinator initialisiert und gestartet, es kann aber durch zusätzliche Router erweitert werden. In der Baumtopologie wird eine hierarchische Routing-Strategie zwischen den Routern eingesetzt. In der Mesh-Topologie kann das Routing über beliebige Geräte ablaufen. Jedes ZigBee-Netzwerk wird durch eine Personal Area Network (PAN) ID identifiziert, die vom Coordinator gewählt wird. Des Weiteren vergibt der Coordinator den beitretenden Geräten eindeutige Netzwerkadressen.

ZigBee unterscheidet drei logische Gerätearten: Coordinator, Router und Endgerät [94]. Der Coordinator ist in der Lage, ein Netzwerk aufzubauen. Der Router ist ein Gerät, das die Nachrichten an andere Geräte weiterleiten kann. Sowohl Router als auch Coordinator sind Full Function Devices (FFD). Die FFDs können dabei ebenfalls Sensoren und Aktoren sein. Das Endgerät oder auch Reduced Function Device (RFD) ist das einfachste ZigBee-Gerät ohne Vermittlungsfähigkeit.

Für die Interoperabilität zwischen verschiedenen Herstellern werden Anwendungsprofile (Application Profiles) wie z.B. „Home Automation“ oder „Smart Energy“ definiert. Die Profile können sowohl öffentlich als auch herstellerspezifisch sein. Ein Profil umfasst einen Satz von Netzwerkparametern, Gerätetypen sowie eine Reihe von Nachrichten und Attributen [95]. Jedem Profil wird eine Profile ID zugeordnet. Die Geräte werden als Anwendungsobjekte (Application Objects) modelliert. Jedes Anwendungsobjekt ist einem Anwendungsprofil zugeordnet.

Die Daten oder Funktionen, die von einem Gerät angeboten werden oder der Steuerung des Gerätes dienen, werden als Attribute bezeichnet. Jedes Attribut wird mit einer Attribute ID gekennzeichnet. Ein Attribut wird durch einen Attributdatentyp beschrieben. Die Attribute haben atomare Datentypen wie Integer, Float, String usw. Eine Gruppe von Attributen und zugehörigen Zugriffsbefehlen bildet einen Cluster. Der Cluster arbeitet nach dem Prinzip des Client/Server-Modells. Somit besteht ein Cluster aus der Client- und der Server-Seite [96]. Der Server-Cluster wird als Input Cluster und der Client-Cluster als Output Cluster bezeichnet. Jedes Anwendungsobjekt kann einen oder mehrere Cluster beinhalten. Ein Cluster stellt damit das Interface eines Gerätes oder einen Teil davon dar [97]. Jeder Cluster wird mit einer Cluster ID gekennzeichnet. ZigBee Alliance hat zu Interoperabilitätszwecken eine ZigBee Cluster Library (ZCL) entwickelt, die alle anerkannten Cluster wie z.B. „On/Off“ oder „Level Control“ bereitstellt [98].

Die Cluster-Befehle werden benutzt, um Attribute zu lesen (Read Attributes), zu schreiben (Write Attributes) und Wertänderungen zu melden (Configure Report und Report Attributes). Jeder Befehl wird mit der Command ID gekennzeichnet. Der Schreib- und Lesebefehl kann gleichzeitig auf mehrere Attribute angewendet werden.

Eine besondere Rolle hat das ZigBee Device Object (ZDO). Das ZDO dient der Steuerung und dem Management eines Gerätes wie die Bestimmung der Rolle des Gerätes im Netzwerk, Geräte- und Service-Discovery und andere [96]. Die Objekte auf einem Gerät werden mittels Endpoint IDs adressiert. Der Zugriff auf ein Attribut erfolgt unter Angabe der Netzwerkadresse, der Profile ID, der Endpoint ID, der Cluster ID und der Attribute ID.

Die logische Kopplung der Geräte miteinander wird als Binding bezeichnet. Dabei wird eine Beziehung zwischen zwei Endpoints hergestellt. Das Binding ist unidirektional. Es sind 1-zu-1, 1-zu-n und n-zu-1-Bindings erlaubt. ZigBee unterstützt somit Unicast-, Multicast- und Broadcast-Nachrichten. Eine Voraussetzung für das Binding ist die Unterstützung des entsprechenden Input bzw. Output Clusters. Um das Binding auszuführen, melden sich beide Geräte bei dem Coordinator mit dem Binding-Request. Stimmen die Binding-Parameter überein, verschickt der Coordinator ein Binding-Response mit entsprechenden Parametern des Gegenübers [96]. Diese werden in die lokale Binding-Tabelle des Senders aufgenommen. Das Binding wird darum auch als Source Binding bezeichnet. Anschließend können die Geräte direkt miteinander kommunizieren. Alternativ kann das Binding mit einem Inbetriebnahme-Tool realisiert werden, das es ermöglicht, direkt Einträge in der Binding-Tabelle des Senders zu erstellen [99]. Der oben beschriebene Aufbau der ZigBee-Anwendungsschicht ist in Abbildung 20 dargestellt.

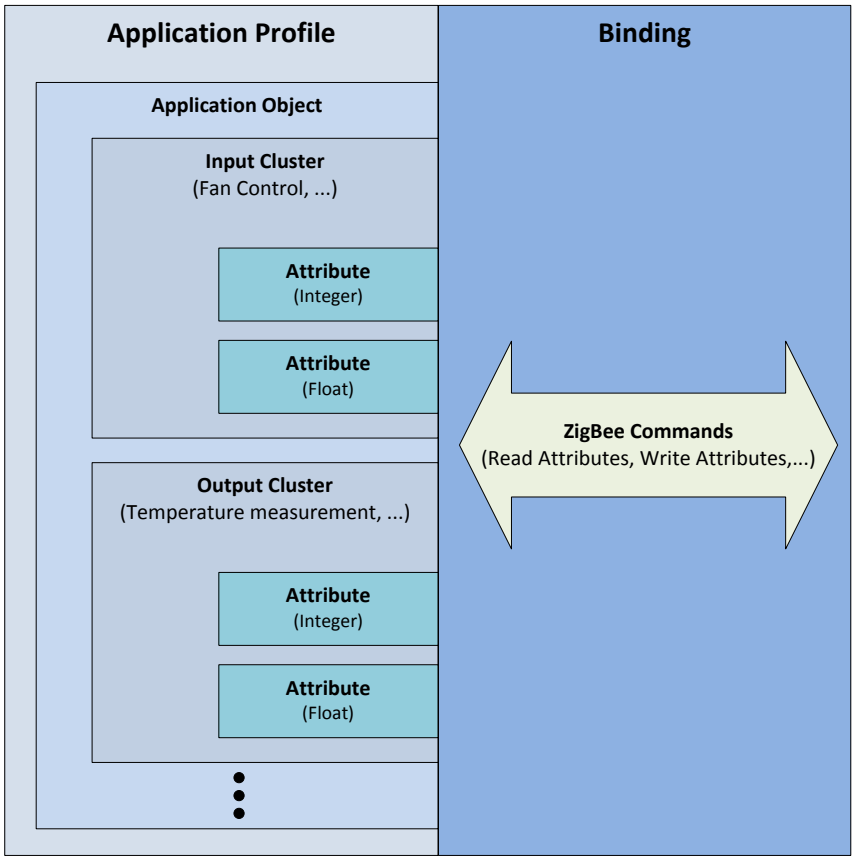

Abbildung 20: Struktur der ZigBee-Anwendungsschicht

Mit ZigBee IP wurde eine Stack-Erweiterung vorgestellt, die es ermöglicht, ZigBee-Nachrichten über IPv6 zu verschicken [100]. Hierfür wird der IPv6 over Low power Wireless Personal Area Network (6LoWPAN) Adaptation Layer verwendet, der ebenfalls auf IEEE 802.15.4 basiert. Damit eröffnen sich für ZigBee weitere Möglichkeiten; u.a. eine direkte Kommunikation mit anderen IP-fähigen Geräten in anderen Medien wie z.B. Ethernet.

Im Vergleich zu Z-Wave bietet ZigBee eine größere Palette an Möglichkeiten. Dadurch können komplexere Anwendungen umgesetzt werden. Den Herstellern von ZigBee-Produkten sind weniger Grenzen gesetzt. Allerdings sind in der Praxis viele ZigBee-Geräte dadurch nicht miteinander kompatibel. Darüber hinaus ist in der Praxis die Konfiguration der ZigBee-Geräte weniger benutzerfreundlich [101].

2.6.2.4. Weitere Technologien

Es existieren weitere proprietäre Technologien wie z.B. digitalSTROM von digitalSTROM AG [102] oder HomeMatic von eQ3 [103]. Geräte dieser Technologien sind zwar auf dem Markt erhältlich, werden jedoch von einer einzigen Firma hergestellt und sind nicht standardisiert. Der Nachteil solcher Technologien liegt darin, dass der Kunde sich an einen einzigen Hersteller bindet, der die Gerätepreise allein bestimmt. Verschwindet der Hersteller

vom Markt, muss im Worst Case die komplette Anlage ersetzt werden, was mit sehr hohen Kosten verbunden ist.

2.6.3. Smart Metering

2.6.3.1. M-Bus

Meter Bus (M-Bus) wurde für die Fernauslesung der Zähler für z.B. Wärme, Strom, Gas oder Wasser entwickelt. M-Bus entstand im Jahr 1992 unter der Leitung von Prof. Horst Ziegler von der Uni Paderborn in Zusammenarbeit mit Texas Instruments GmbH und Techem GmbH [104]. Das Protokoll wurde zunächst für die Zählwerterfassung der Wärmezähler im Jahr 1997 als Europäische Norm EN 1434 standardisiert [105]. Im Jahr 2002 ist M-Bus zu einem generischen Smart Metering Standard als EN 13757 geworden [106].

Für die Kommunikation werden die Übertragungsmedien Zweidrahtleitung, IR und RF eingesetzt. Für die drahtlose Übertragung wird das SRD-Band 868 MHz verwendet. Die Zweidrahtleitung ermöglicht darüber hinaus die Stromversorgung der Geräte. Die Spezifikation von M-Bus schreibt keine spezielle Topologie vor. In der Praxis wird das M-Bus-Netzwerk als Bus- oder Sterntopologie aufgebaut [107]. Die Spezifikation von M-Bus beschreibt die Bitübertragungs-, Vermittlungs-, Netzwerk- und Anwendungsschichten. Die Kommunikation zwischen den M-Bus-Geräten läuft nach Master/Slave-Prinzip ab. Es darf dabei nur einen Master im Netzwerk geben [108]. Die Slaves können nur mit dem Master und nicht miteinander kommunizieren. Die Kommunikation kann nur vom Master initiiert werden. Die Rolle der Slaves übernehmen die Zähler. Der Master ist eine Ausleseeinheit. Diese kann u.U. ein Personal Computer (PC) sein.

Da die Kommunikation im M-Bus-Netzwerk vom Master gesteuert wird, können zur Datenabfrage nur Request-Nachrichten (z.B. REQ_UD) vom Master und Response-Nachrichten (z.B. RSP_UD) von den Slaves geschickt werden [109]. Der Master kann dabei sowohl einen Slave (Unicast) als auch alle Slaves (Broadcast) adressieren [110]. Der Zugriff auf die Slaves kann sowohl lesend (REQ_UD) als auch schreibend (SND_UD) sein.

Für die Interoperabilität zwischen verschiedenen Herstellern werden Data Information Fields (DIFs) und Value Information Fields (VIFs) definiert. Ein DIF beschreibt die Länge, den Datentyp und die Codierung der Daten. Die Werte werden grundsätzlich als atomare Datentypen wie Integer, Float oder String dargestellt. Die Länge gibt die Anzahl der Bits für den Wert an wie z.B. Integer 8 Bit. Die Datencodierung kann beispielsweise ein Binary Coded Decimal (BCD) sein. Ein VIF repräsentiert den Wert, die Einheit und den Multiplikator. M-Bus definiert eine Reihe von VIFs wie z.B. Außentemperatur, Energie, Volumen und andere. Während der Konfigurationsphase wählt der Master die Messwerte mittels DIFs und VIFs auf dem Slave aus, die während des Betriebs übertragen werden sollen.

Im Betrieb schickt der Master ein Auslese-Request, auf das der Slave mit den vorkonfigurierten Werten antwortet [109]. Der oben beschriebene Aufbau der M-Bus-Anwendungsschicht ist in Abbildung 21 dargestellt.

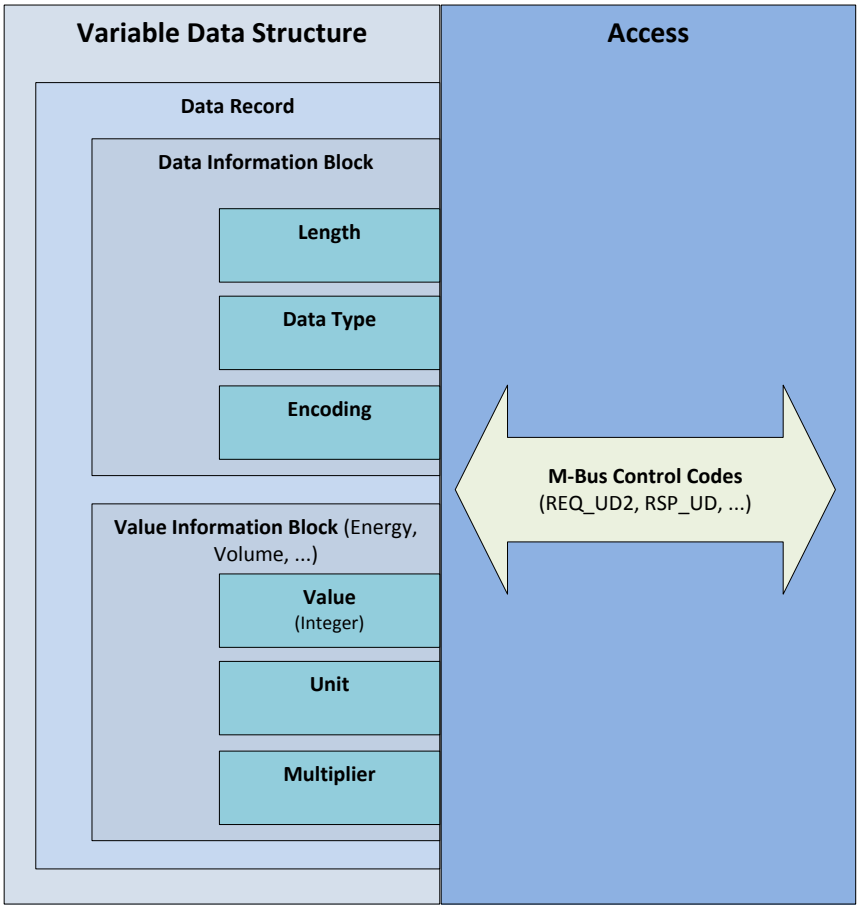

Abbildung 21: Struktur der M-Bus-Anwendungsschicht

Der Aufbau des M-Bus-Netzwerks und die komplette Steuerung durch einen Master ermöglichen es, die Slaves relativ einfach und kostengünstig umzusetzen. Die Slaves können darüber hinaus über den Bus selbst mit Strom versorgt werden und benötigen damit nur eine Leitung. Die maximale empfohlene Geschwindigkeit für die Datenübertragung beträgt 9600 Baud, was relativ langsam für z.B. Prozesssteuerung ist [110]. Das langsamste Gerät bestimmt dabei die Baudrate für das gesamte Netzwerk. Die Standardisierung auf Protokollebene ist lückenhaft, was für die Inkompatibilität zwischen verschiedenen Herstellern sorgen kann [107]. Ein Master stellt darüber hinaus einen SPoF dar. Im Falle eines Ausfalls ist keine Kommunikation mit den Slaves möglich.

2.6.3.2. DLMS/COSEM

Device Language Message Specification (DLMS) ist ein universelles und abstraktes Protokoll für Zählerkommunikation. Companion Specification for Energy Metering (COSEM) definiert die Transport- und Anwendungsschicht des DLMS-Protokolls [111]. DLMS/COSEM wurde

im Jahr 2002 zum internationalen Standard IEC 62056 und auch zum europäischen Standard EN 62056 [112] [113]. Die Entwicklung von DLMS/COSEM wird von DLMS User Association vorangetrieben.

DLMS unterstützt eine Reihe von Übertragungsmedien wie TP, IR, PL, Global System for Mobile Communications (GSM) und Ethernet, die in Form von Kommunikationsprofilen definiert werden [114]. Es wird u.a. ein TCP/UDP/IP basiertes Profil definiert, das prinzipiell die Übertragung über alle IP-unterstützenden Medien ermöglicht. Der Stack wird in medienabhängige und medienunabhängige Schichten aufgeteilt. Im Rahmen dieser Arbeit wird das Hauptaugenmerk auf die Anwendungsschicht von DLMS/COSEM gelegt. Der Aufbau anderer Schichten kann der Spezifikation EN 62056 entnommen werden.

Die Kommunikation zwischen den DLMS/COSEM-Geräten basiert auf dem Client/Server-Prinzip. Der Zähler nimmt dabei die Rolle eines Servers und eine Ausleseeinheit die Rolle des Clients ein. Die Werte können vom Client über ein Request vom Server angefordert werden (Request/Response), oder sie können vom Client abonniert werden, wobei sich der Server bei dem Client meldet, wenn Änderungen vorliegen, oder in regelmäßigen Zeitabständen. Ein physikalisches Messgerät wird als eine Menge logischer Geräte dargestellt. Ein logisches Gerät bietet eine Untermenge aller Funktionen von dem physikalischen Gerät. Für die Interoperabilität zwischen verschiedenen Herstellern werden diese Funktionen als COSEM-Interface-Objekte modelliert. Die COSEM-Interface-Objekte stellen Instanzen der Interface-Klassen dar. Eine Class ID bestimmt die Zugehörigkeit eines COSEM-Objektes zu einer COSEM-Klasse. Ein Objekt ist eine Ansammlung von Attributen und Methoden. Attribute spiegeln bestimmte Informationen eines Objektes wider. Die Attributwerte können ebenfalls das Verhalten der Objekte beeinflussen. Die Methoden eines Objektes werden dazu genutzt, die Attribute zu lesen oder zu schreiben [115].

Für die Adressierung der Attribute und der Methoden der COSEM-Objekte werden Short Names (SNs) oder Logical Names (LNs) verwendet. SNs sind für einfache Geräte gedacht und stellen eine direkte Adresse eines Attributes oder einer Methode dar. LNs erlauben eine Adressierung über die Class ID des COSEM-Objektes nach dem Muster class_ID->logical_name->attribute_index/method_index. Die Art der Adressierung der Attribute und Methoden wird beim Verbindungsaufbau ausgehandelt. Die Clients können auch mehrere Server gleichzeitig ansprechen. DLMS/COSEM bietet Unterstützung für Unicast, Multicast und Broadcast. DLMS/COSEM ermöglicht es außerdem, eine Liste der instanziierten Objekte aus dem Messgerät auszulesen. Die Messwerte werden durch Instanzen der Datenklassen bereitgestellt.

Die LNs werden durch Object Identification System (OBIS) ID repräsentiert [116]. OBIS ID besteht aus 6 Feldern je 8 Bit (A-F). Die ersten drei Felder geben das Medium, den Kanal und

die Messgröße an. Die letzten drei Felder sind variabel und werden durch die ersten drei bestimmt [117]. Diese wären z.B. für die elektrische Energie die Art der Messung, die Tarifstufe und der Vorwertzählerstand [118]. Die Messwerte werden je nach Datenklasse zusammen mit der Einheit, dem Skalierungsfaktor, dem Zeitstempel und dem Status übertragen [115]. Die Werte werden durch atomare Datentypen abgebildet. DLMS/COSEM definiert eine breite Palette an Datentypen wie Integer8, Integer16, Unsigned8, Float16 und andere.

Für den Zugriff auf Attribute und Methoden wurden spezielle Services definiert. Die Services werden nach LN/SN-Adressierung unterschieden. Für jede Adressierungsart werden 4 Servicetypen angeboten. Die LN-Adressierung definiert die Services GET, SET, ACTION und EventNotification. Der GET-Service dient dem Lesen eines oder mehrerer Attribute. Der SET-Service ermöglicht das Setzen eines oder mehrerer Attribute. Mit dem ACTION-Service können Methoden eines Objektes aufgerufen werden. Der EventNotification-Service wird benutzt, um den Client über die Änderungen eines Attributes zu informieren. In Analogie dazu werden für die SN-Adressierung die Services Read, Write, UnconfirmedWrite und InformationResponse definiert [114]. Der oben beschriebene Aufbau der DLMS/COSEM-Anwendungsschicht ist in Abbildung 22 dargestellt.

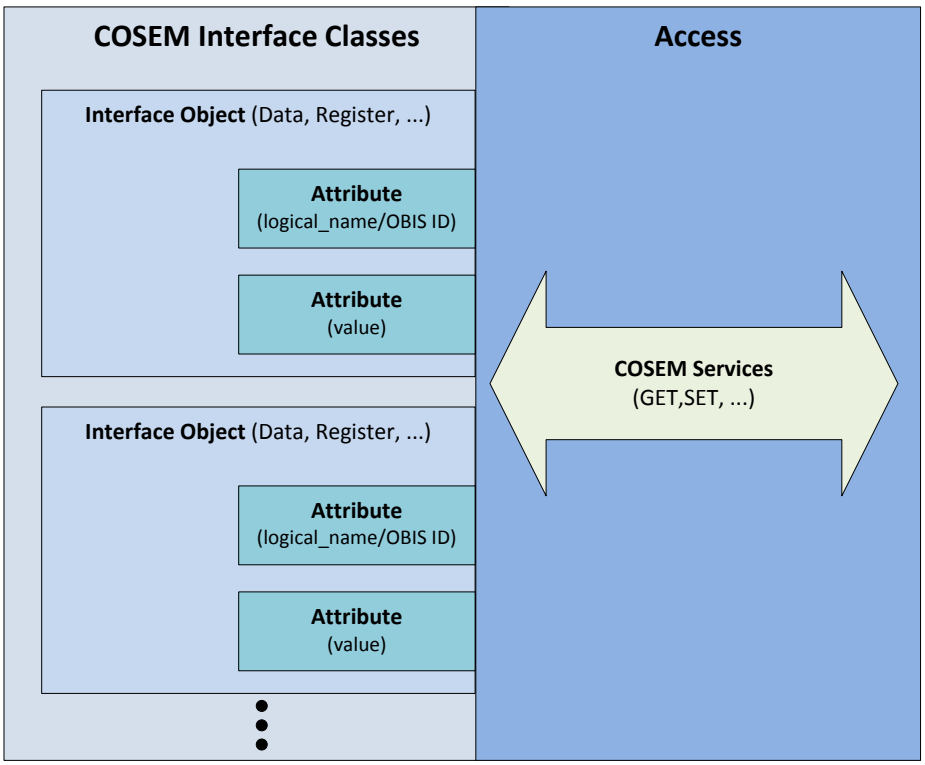

Abbildung 22: Struktur der DLMS/COSEM-Anwendungsschicht

DLMS/COSEM User Association besteht aus vielen Mitgliedern, die für eine Durchsetzung des Standards sorgen. Der Standard unterstützt viele Übertragungsmedien, die einen breiten

Einsatzbereich ermöglichen. Insbesondere die IP-Unterstützung erlaubt eine leichtere Integration in die Enterprise-Systeme. Der Standard ist jedoch nur auf die Zählersysteme beschränkt und für andere Anwendungsgebiete nicht einsetzbar.

2.6.3.3. SML

Smart Message Language (SML) ist ein spezielles Protokoll zum Auslesen und Steuern von Zählern. Das Protokoll ist im Rahmen des Projektes SyM^2 als eine Kooperation von Stromanbietern EnBW, E.ON und RWE und anderen Partnerunternehmen entstanden [119]. Die erste Spezifikation wurde im Jahr 2008 veröffentlicht [120]. Das SML-Protokoll wurde an die bereits bestehenden Normen angelehnt. Das Protokoll befindet sich in der Standardisierung zur europäischen Norm als prEN 62056-5-8. Die Zielsetzung bei der Entwicklung von SML war eine einfache Struktur zur Datenübertragung von Messwerten, die auf ressourcenbeschränkten eingebetteten Geräten eingesetzt werden kann.

Im Gegensatz zu anderen Protokollen ist SML grundsätzlich ein Anwendungsschichtprotokoll. SML unterstützt eine IP-basierte Kommunikation. Neben TCP und UDP können die Daten darüber hinaus über HTTP oder File Transfer Protocol (FTP) übertragen werden. Alternativ kann SML mittels eines speziellen Transportprotokolls über direkte serielle Punkt-zu-Punkt-Verbindungen wie RS-232 genutzt werden [121].

Die Kommunikation läuft nach dem Client/Server-Prinzip ab. Jeder Server wird durch eine eindeutige Server ID und jeder Client durch eine eindeutige Client ID gekennzeichnet. Ein Messgerät kann dabei mehrere virtuelle Server haben. Der Client kann sowohl einen (Unicast) als auch mehrere (Multicast) oder alle (Broadcast) Server ansprechen. Der Datenaustausch findet in Form von Dateien statt. Eine Datei besteht dabei aus mehreren SML-Nachrichten. Drei Dateitypen sind spezifiziert: SML-Auftragsdatei, SML-Antwortdatei und SML-Kombidatei. Die SML-Auftragsdatei enthält die Aufträge (Requests), die SML-Antwortdatei fasst die Antworten zusammen (Responses) und die SML-Kombidatei kann beides umfassen. SML definiert 20 Nachrichtentypen. Davon sind 10 Request-Typen (wie z.B. SML_GetProfileList.Req) und 10 entsprechende Response-Typen (wie z.B. SML_GetProfileList.Res). Die unterschiedlichen Request- und Response-Typen dienen vor allem der unterschiedlichen Datendarstellung. So können die Daten beispielsweise in einer gepackten oder in einer Listenform angefragt bzw. übertragen werden [121]. Eine Datendarstellung in Form von COSEM-Objekten ist ebenfalls möglich (vgl. Abschnitt 2.6.3.2).

Für die Interoperabilität zwischen verschiedenen Herstellern werden die Messdaten als Objekte modelliert. Zur Identifikation eines Datums setzt SML ähnlich wie DLMS/COSEM auf die OBIS-Kennzahlen. Ein SML-Objekt ist ein komplexer Datentyp, der aus mehreren atomaren Datentypen besteht. SML unterstützt eine Reihe von Standarddatentypen wie

Integer8, Integer16, Unsigned8 usw. Die Attribute eines Objektes sind z.B. der Messwert, die Einheit und der Skalierungsfaktor. Mit der Version 1.04 der SML-Spezifikation wurde zusätzlich zu einer binären Codierung der SML-Daten ebenfalls eine XML-basierte Codierung vorgestellt [121]. Der oben beschriebene Aufbau der SML-Anwendungsschicht ist in Abbildung 23 dargestellt.

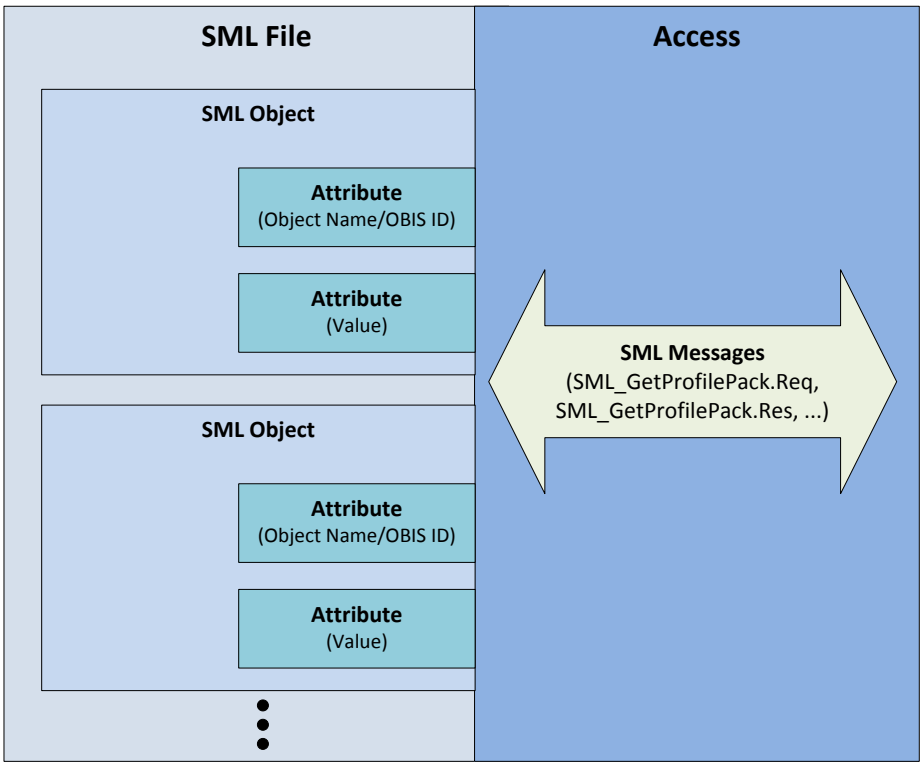

Abbildung 23: Struktur der SML-Anwendungsschicht

Der Vorteil von SML ist eine einfache Protokollhandhabung, da es ein Anwendungsschichtprotokoll ist und IP-Unterstützung bietet. Somit können alle Medien und Transportprotokolle zur Übertragung verwendet werden, die IP unterstützen bzw. auf IP aufbauen. Damit können die Geräte leicht in Enterprise-Systeme eingebunden werden. Zum Zeitpunkt des Schreibens stellte SML jedoch noch keinen Standard dar. Wie die bereits vorgestellten Protokolle für Smart Metering ist SML ebenfalls auf ein Anwendungsgebiet beschränkt und kann nur zur Steuerung und Fernauslesung der Zähler eingesetzt werden.

2.7. Peer-to-Peer-Netzwerke

Peer-to-Peer (P2P)-Netzwerke wurden als ein neues Netzwerkparadigma eingeführt, bei dem im Gegensatz zum üblichen Client-Server-Modell alle Kommunikationsteilnehmer gleichberechtigt sind. Diese als Peers bezeichneten Knoten weisen Eigenschaften sowohl von Servern als auch von Clients auf und werden deswegen auch als Servlents bezeichnet. Die P2P-Technologie ist im Jahr 1999 zum ersten Mal durch Napster berühmt geworden [122]. Das primäre Ziel bestand im Austausch von Dateien. Die Anwendungsfelder sind jedoch nicht

auf den Dateiaustausch begrenzt. Die dezentralisierten und selbstorganisierten Designprinzipien von P2P-Netzwerken machen sie für zukünftige Anwendungen, verteilte Systeme und Services im Hinblick auf hohe Skalierbarkeit und Ausfallsicherheit interessant. Die P2P-Mechanismen können benutzt werden, um auf Netzwerkressourcen oder -Services zuzugreifen. P2P-Netzwerke bilden ein logisches oder virtuelles Overlay auf der physikalischen Netzwerktopologie [123].

Im Gegensatz zu Client-Server-basierten Anwendungen speichern solche dezentralisierten Systeme die Daten auf mehreren, u.U. weit voneinander entfernen Peers. Für die Lösung dieser Aufgabe haben sich zwei grundsätzlich verschiedene Ansätze entwickelt: die unstrukturierten und die strukturierten P2P-Systeme. Die unstrukturierten P2P-Netzwerke lassen sich weiterhin in zentralisierte, dezentralisierte und hybride Systeme unterteilen. Die Struktur bezieht sich auf den Zusammenhang zwischen Daten und Peers. Die unstrukturierten P2P-Netzwerke haben demnach keinen Zusammenhang zwischen Daten und Peers. Das heißt, jedes Datum kann auf jedem Peer gespeichert werden. Die Struktur bringt dagegen Zusammenhänge zwischen Peers und Daten in ein P2P-System ein. Das heißt, gewisse Dateneigenschaften bestimmen, auf welchen Peers die Daten gespeichert werden können. Die Struktur wird mittels Distributed Hash Table (DHT) aufgebaut. Hierbei besitzen sowohl Peers als auch die Daten einen Hashwert. Ist der Daten-Hashwert dem Peer-Hashwert „ähnlich“, werden diese Daten auf dem Peer gespeichert. Die Bestimmung der Ähnlichkeit hängt von der DHT-Implementierung ab [123].

2.7.1. Unstrukturierte zentralisierte P2P-Netzwerke

Unstrukturierte zentralisierte P2P-Netzwerke besitzen einen zentralen Lookup-Server [124]. Der Lookup-Server kennt alle Daten und Peers, besitzt selbst jedoch nur ein Verzeichnis mit den Informationen, welche Daten auf welchen Peers gespeichert sind. Alle Suchanfragen werden an den Lookup-Server gerichtet (vgl. Abbildung 24). Dieser teilt dem suchenden Knoten die entsprechenden Peers mit, die über die Daten verfügen. Die Datenübertragung findet anschließend direkt zwischen dem suchenden Knoten und den Peers statt. Ein Beispiel dieses Ansatzes ist Napster [125]. Ein wesentlicher Nachteil dieses Ansatzes ist der SPoF-Charakter des Lookup-Servers. Für N Knoten verfügt dieser Ansatz über eine sehr gute Suchkomplexität $O(1)$, da nur der Lookup Server kontaktiert werden muss. Die Speicherkomplexität beträgt dagegen $O(N)$, da der zentrale Lookup-Server Informationen über alle Daten und Knoten verwalten muss.

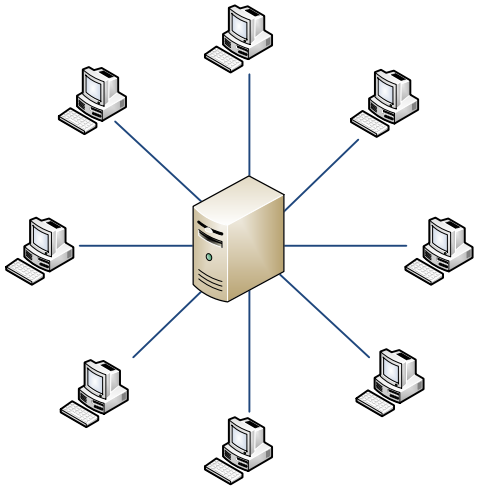

Abbildung 24: Unstrukturiertes zentralisiertes P2P-Netzwerk

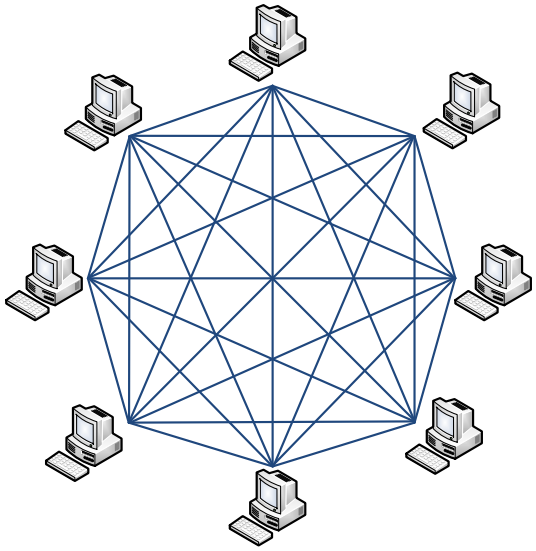

Abbildung 25: Unstrukturiertes dezentralisiertes P2P-Netzwerk

2.7.2. Unstrukturierte dezentralisierte P2P-Netzwerke

Unstrukturierte dezentralisierte P2P-Netzwerke weisen eine vollständig verteilte Architektur ohne eine zentrale Instanz auf. Die Peers verfügen über eine Liste mit den Nachbarknoten. Die Nachbarschaft wird anhand des logischen Overlays bestimmt. Die tatsächliche physikalische Entfernung der Peers wird dabei nicht berücksichtigt. Die Suchanfragen werden direkt an die Nachbarn geschickt. Diese leiten die Anfrage an ihre Nachbarn weiter. Das Netzwerk wird somit mit einer Anfrage geflutet (vgl. Abbildung 25). Die Nachfrage terminiert, wenn die gesuchte Datei gefunden wurde oder die maximal zulässige Anzahl der Weiterleitungen überschritten ist [124]. Eine Implementierung dieses Ansatzes stellt Gnutella

0.4 dar [126]. Die wesentlichen Nachteile dieses Ansatzes sind eine hohe Netzwerkauslastung während der Suche durch das Fluten des Netzes und ein nicht deterministisches Verhalten. Die Suche garantiert nicht, dass ein vorhandenes Datum gefunden wird. Für *N* Knoten verfügt dieser Ansatz über eine Suchkomplexität von mindestens *O(N)*, da ein Knoten alle anderen Knoten kontaktieren muss. Werden während des Flutens die bereits weitergeleiten Anfragen nicht abgefangen, kann die Komplexität deutlich ansteigen. Die Speicherkomplexität beträgt dagegen *O(1)*, da jeder Peer eine festgelegte Anzahl an Peers speichert, die von der Netzwerkgröße unabhängig ist.

2.7.3. Unstrukturierte hybride P2P-Netzwerke

Unstrukturierte hybride P2P-Netzwerke werden durch sogenannte Superpeers charakterisiert. Superpeers kennen eine Untermenge aller an das Netzwerk angeschlossenen Peers sowie andere Superpeers. Die Anfragen werden an die Superpeers gerichtet. Die Superpeers liefern als Antwort eine Liste von Peers, die über das gesuchte Datum verfügen. Wenn der angefragte Superpeer keine Informationen über das gesuchte Datum hat, wird das Netzwerk der Superpeers geflutet. Eine Datenübertragung findet direkt zwischen dem anfragenden Knoten und den Peers statt. Ein Beispiel dieses Ansatzes ist eDonkey2000 [124]. Alle Superpeers zusammen bilden prinzipiell einen verteilten zentralen Lookup-Server. Ein Peer ist jedoch nur einem Superpeer zugeordnet. Der Ausfall eines Superpeers führt im Gegensatz zum zentralisierten Ansatz nicht zum Totalausfall des Systems. Die an den Superpeer angeschlossenen Peers und deren Daten gehen jedoch verloren. Darüber hinaus soll die Anzahl der Superpeers proportional zu der Anzahl der Peers steigen, was die Skalierbarkeit des Ansatzes verringert. Die Suche bei diesem Verfahren kann ähnlich dem dezentralisierten Ansatz „False Negatives“ liefern, da das Fluten der Superpeers nach einer bestimmten Anzahl der Weiterleitungen abgebrochen wird. Für *N* Knoten verfügt dieser Ansatz über eine Suchkomplexität von mindestens *O(N)*, da ein Superpeer alle anderen Superpeers kontaktieren muss. Werden während des Flutens die bereits weitergeleiten Anfragen nicht abgefangen, kann die Komplexität höher sein. Die Speicherkomplexität beträgt dagegen *O(1)*, da jeder Superpeer eine maximal festgelegte Anzahl an Peers speichert (bis zu 100), die von der Netzwerkgröße unabhängig ist.

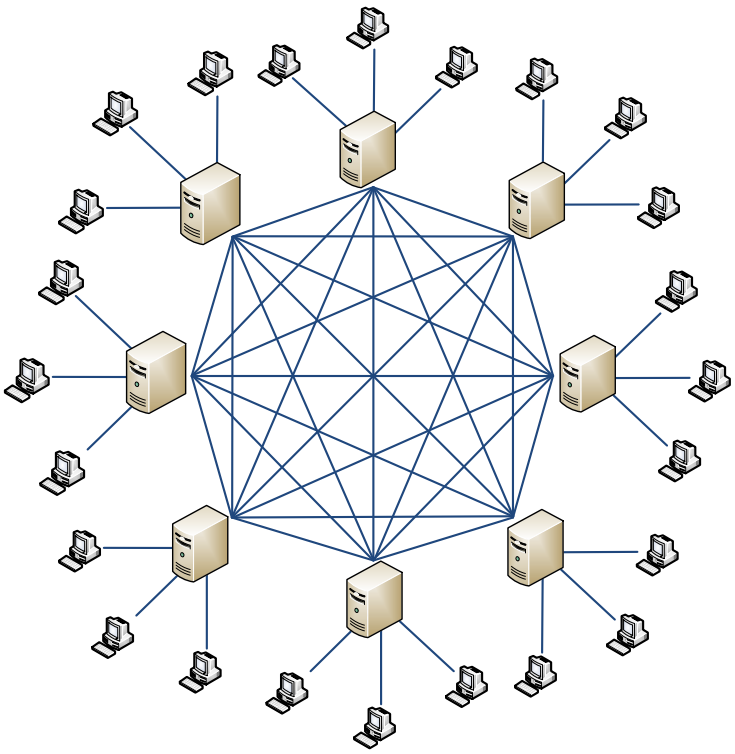

Abbildung 26: Unstrukturiertes hybrides P2P-Netzwerk

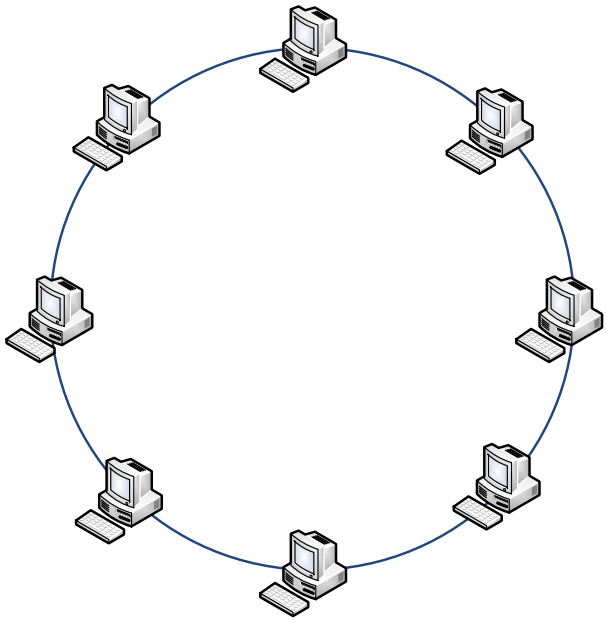

Abbildung 27: Strukturiertes P2P-Netzwerk

2.7.4. Strukturierte P2P-Netzwerke mit DHT

Strukturierte P2P-Netzwerke basieren auf der DHT, welche eine Struktur in das Netzwerk einbringt. Das Netzwerk besitzt keine zentrale Instanz. Jeder Knoten verfügt über eine kleine Anzahl der Kontakte. Durch die Abbildung von Hashwerten von Knoten und Daten auf einen

gemeinsamen Adressraum führt die Suche nach einem Datum zur Suche nach einem oder mehreren entsprechenden Peers. Die Suchanfragen werden an die bekannten Peers gestellt und können durch einige wenige Peers zum Zielknoten weitergeleitet werden. Die wenigen Weiterleitungen sind der DHT-Struktur geschuldet, die es ermöglicht, Anfragen an die Peers weiterzuleiten, die dem Zielhashwert am nächsten sind. Da keinem Knoten eine besondere Rolle zugeordnet ist, besitzt das P2P-Netzwerk mit DHT keine SPoFs. Da Knoten über mehrere Kontakte verfügen, existieren auch mehrere Wege zum Zielhashwert. Der Ausfall eines oder mehrerer Knoten beeinflusst die Funktionalität der DHT nicht. Durch die eindeutige Zuordnung der Daten-Hashwerte zu den Peer-Hashwerten ist die Suche in DHT deterministisch. Existiert ein Datum im Netzwerk, wird es immer gefunden [127]. Ein Beispiel des DHT-basierten Ansatzes ist das Kademlia-Protokoll [128] bzw. seine Implementierung Kad [129]. Für *N* Knoten verfügt dieser Ansatz über eine Suchkomplexität von *O(log(N))*, da Anfragen an einige wenige Knoten weitergeleitet werden, die dem Ziel am nächsten sind. Die Speicherkomplexität beträgt ebenfalls *O(log(N))*, damit ein Knoten eine ausreichende Abbildung des Netzes bereithält. In Abbildung 28 sind die Such- und Speicherkomplexitäten der genannten P2P-Technologien für einen besseren Überblick gegenübergestellt [130].

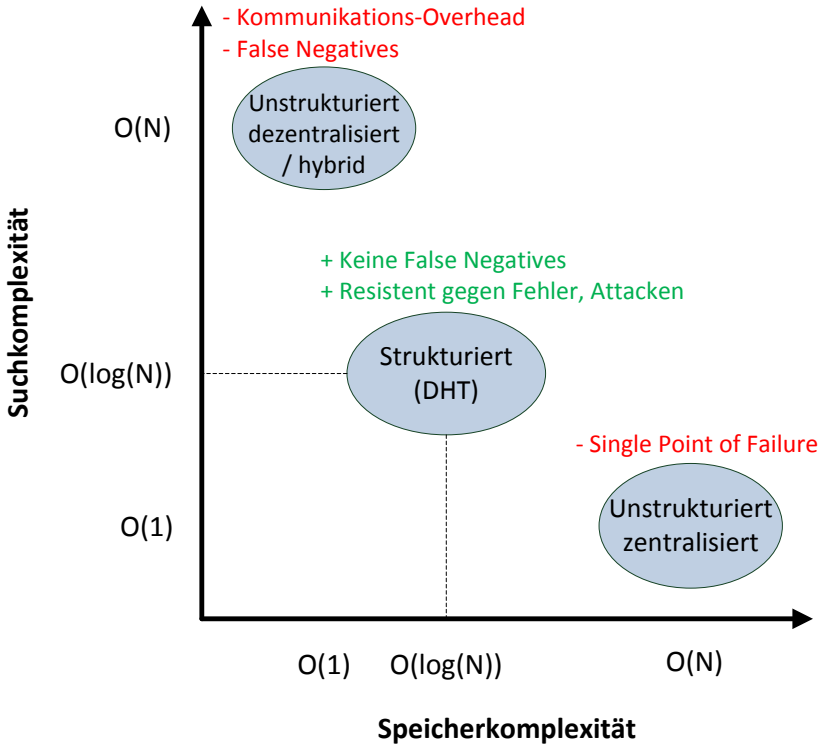

Abbildung 28: Speicher- und Suchkomplexität verschiedener P2P-Technologien

3. Web Services in der Gebäudeautomation

3.1. Motivation

Das schnelle Wachstum im Smart Home- und Smart Metering-Bereich hat zu einer Entwicklung vieler spezifischer Protokolle geführt. Die meisten dieser Protokolle sind weder miteinander noch mit bereits existierenden Standards kompatibel. Darüber hinaus sind die spezifischen Protokolle nur für ein enges Aufgabenspektrum geeignet und somit wenig zukunftssicher. Im Vergleich dazu durchdringen die WS-Technologien andere Tätigkeitsbereiche und beweisen damit ihre Zukunftsfähigkeit. Die dynamische Struktur und das modulare Designprinzip erlauben es, WS-Technologien auf nahezu alle Bedürfnisse anzupassen. Außerdem sind die WS-Spezifikationen öffentlich verfügbar.

Zukünftig sollen die Stromnetze „schlauer“ werden, um die Herausforderungen der Energiepolitik zu bewältigen. Neue Smart Metering-Protokolle wie SML, METERS&MORE [131] und andere zeigen keine Interoperabilität untereinander, obwohl sie demselben Zweck dienen. Die Gebäudeautomation ist ebenfalls durch eine große Anzahl nicht interoperabler Protokolle wie z.B. LON, KNX, BACnet gekennzeichnet. Darüber hinaus kann die Installation neuer Geräte nicht durch den Nutzer selbst, sondern nur durch einen Fachmann durchgeführt werden, was mit höheren Kosten verbunden ist. Diese Tatsache erschwert eine schnelle Ausbreitung der Automationsmechanismen aus dem Gebäude- in den Heimbereich (Smart Home).

Die mangelnde Interoperabilität kann durch die Nutzung von WS-Technologien gelöst werden, da diese über inhärente Plug&Play-Fähigkeiten verfügen und die Definition interoperabler Systeme erlauben. DPWS ermöglicht WS-Funktionalität auf kleinen Geräten mit begrenzten Ressourcen wie z.B. Smart Meter. Um den Einsatz von WS für spezielle Aufgaben zu ermöglichen, ist eine spezifische Profildefinition notwendig, die die besonderen Anforderungen des Anwendungsfalls berücksichtigt. Ein Profil besagt, welche Standards bei der jeweiligen Anwendung benutzt werden. Auf diese Weise kann ein Profil für z.B. Smart Metering definiert werden.

Ein Ziel dieser Arbeit ist, die Einsatzmöglichkeiten des Protokolls DPWS als übergreifende und harmonisierende Lösung in der Gebäudeautomation, dem Smart Metering und dem Smart Home zu untersuchen. Es soll dabei festgestellt werden, ob DPWS als Ersatz für andere Protokolle eingesetzt werden kann. Es wird dabei mit Smart Metering als einem Teilaspekt der Gebäudeautomation begonnen. Dabei werden die relevanten Smart Metering-Protokolle untersucht. Ein Vertreter der Smart Metering-Technologien wird für ein DPWS-Profil ausgewählt.

Zunächst wird ein Profil für Smart Metering, basierend auf SML, erarbeitet. Hierfür wird SML tiefgreifend analysiert. Einzelne SML-Bestandteile werden dabei identifiziert. Anschließend wird gezeigt, wie die Bestandteile von SML auf DPWS abgebildet werden können (vgl. Abbildung 29). Das Smart Metering-Profil wird dabei praktisch umgesetzt, getestet und evaluiert. Dabei werden die Mechanismen und Funktionsweise von DPWS sowie der Aufbau eines Profils, basierend auf einem Protokoll, erklärt. Hiermit wird erstmals die Möglichkeit der Abbildung eines Smart Metering-Protokolls auf DPWS bestätigt. Darüber hinaus wird festgestellt, dass das entwickelte Profil die gleiche oder sogar bessere Effizienz als ein spezifisches Protokoll aufweist. Anschließend soll gezeigt werden, dass das durchgeführte Abbildungskonzept auf andere Protokolle der Gebäudeautomation, des Smart Metering und des Smart Home ebenfalls anwendbar ist. Für die Darstellung der Allgemeingültigkeit des vorgeschlagenen Ansatzes wurde eine analytische Betrachtung herangezogen. Die analytische Betrachtung hat dabei eindeutig gezeigt, dass der vorgeschlagene Ansatz für jedes beliebige Protokoll angewendet werden kann. Dies unterstreicht die Richtigkeit der gewählten Vorgehensweise. Somit entfällt die Notwendigkeit, das Konzept an zahlreichen Protokollen der Gebäudeautomation, des Smart Metering und des Smart Home zu testen, da diese Vorgehensweise aufgrund des bereits analytisch bestätigten Konzeptes überflüssig wäre und keinen weiteren wissenschaftlichen Mehrwert bringen würde. Andere Protokolle aus unterschiedlichen Bereichen der Automatisierung, wie z.B. DLMS/COSEM aus dem Bereich Smart Metering, BACnet aus dem Bereich der klassischen Gebäudeautomatisierung oder ZigBee aus dem Bereich Smart Home, können somit nachweislich auf DPWS abgebildet werden.

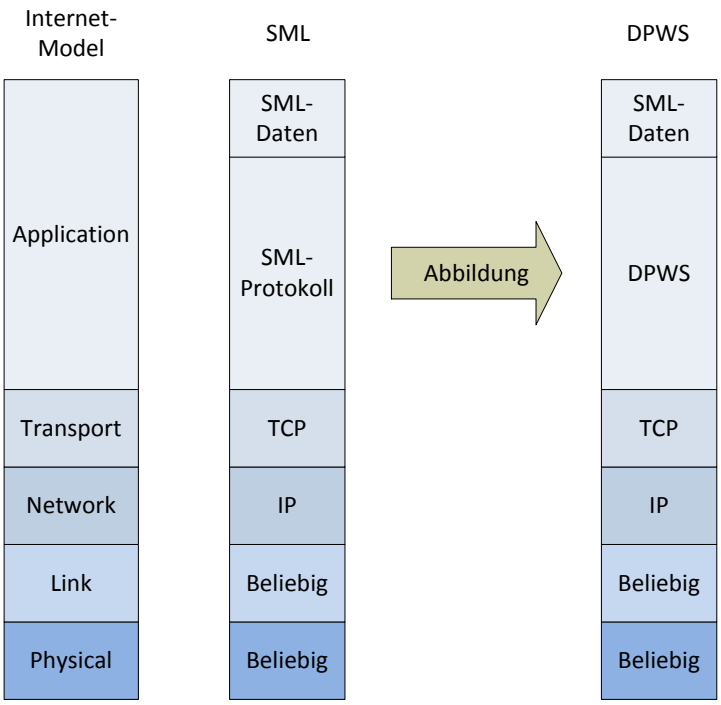

Abbildung 29: Abbildung von SML auf DPWS

Zusammenfassend ist der Hauptbeitrag dieses Kapitels wie folgt [1]:

- Ausarbeitung eines DPWS-Profils für Smart Metering
- Untersuchung der Kompressionsmethoden zu Overhead-Reduzierung
- Untersuchung der Abbildbarkeit anderer Protokolle auf DPWS
- Plug&Play-Kopplung der Geräte in der Gebäudeautomation

3.2. Stand der Technik

Einige Smart Metering-Pilotprojekte haben bereits in Italien, Schweden, Frankreich, den Niederlanden, Griechenland, Deutschland und anderen Ländern stattgefunden [132]. In den meisten Fällen wurden diese Projekte durch die jeweilige Regierung unterstützt. Das italienische Projekt Telegestore wurde bereits im Jahr 2001 vorgestellt [133]. Ziel des Projektes war die Einrichtung einer Smart Metering-Infrastruktur für ca. 30 Millionen Haushaltskunden, um die Nachfragespitzen bedienen zu können sowie die Abrechnung zu vereinfachen. Zusätzlich zu einer erweiterten Version des Lontalk-Protokolls wurde ein proprietäres High Level Data Link Control Protocol (HDLC), das SITRED genannt wurde, vorgestellt [134]. In Schweden wurde das Smart Metering durch die Gesetzgebung stark stimuliert [135]. Das entwickelte Smart Metering-System basiert auf dem proprietären Lontalk-Protokoll, das von der Echelon Corporation entwickelt wurde [136]. In Deutschland wurde das neue Protokoll Smart Message Language eingeführt. Die Entwicklung wurde von mehreren Energieanbietern wie EnBW, E.ON, RWE und anderen Unternehmen begleitet. Die Bereitstellung der Smart Meter wurde allerdings durch die mangelnde Kommunikationssicherheit ausgebremst. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) veröffentlichte Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen [137]. Demnach soll die Kommunikation mit dem Smart Meter über ein Gateway ablaufen (vgl. Abbildung 30). Das Gateway muss drei Smart Metering-Protokolle M-Bus, DLMS/COSEM und SML unterstützen. Die anderen Protokolle dürfen ebenfalls optional integriert werden, wenn sie die Sicherheitsanforderungen erfüllen. Laut der BSI-Sicherheitsrichtlinie müssen die Nachrichten zwischen dem Gateway und den Smart Metern mittels TLS abgesichert werden. Falls das Protokoll kein TLS unterstützt, müssen die Daten mit einem symmetrischen Verschlüsselungsverfahren und einer Signatur abgesichert werden.

Gegenwärtig ist IP das meist verbreitete Kommunikationsprotokoll in der Netzwerktechnik. Der wichtigste Vorteil von IP ist die Adressierung aller Geräte unabhängig vom darunterliegenden Medium. IP deckt sowohl drahtgebundene (Ethernet) als auch drahtlose (WLAN, HSPA, UMTS) bis zu „Low Power“-Kommunikationstechnologien (6LoWPAN) ab. Darüber hinaus wurde IP als ein Smart Grid-Standard vom National Institute of Standards and Technology (NIST) anerkannt [138]. SOA ist ein Standard in der Rechnerkommunikation.

Diese zeichnet sich durch eine hohe Anpassbarkeit und Interoperabilität aus. Darüber hinaus wurde SOA von der International Electrotechnical Commission (IEC) für die Nutzung in Smart Grids empfohlen [139]. DPWS ermöglicht eine skalierbare Architektur, volle Interoperabilität der messtechnischen Geräte, einen sicheren Datenaustausch unabhängig vom Kommunikationsmedium und kommt somit dem EU Smart Metering Mandat M/441 nach.

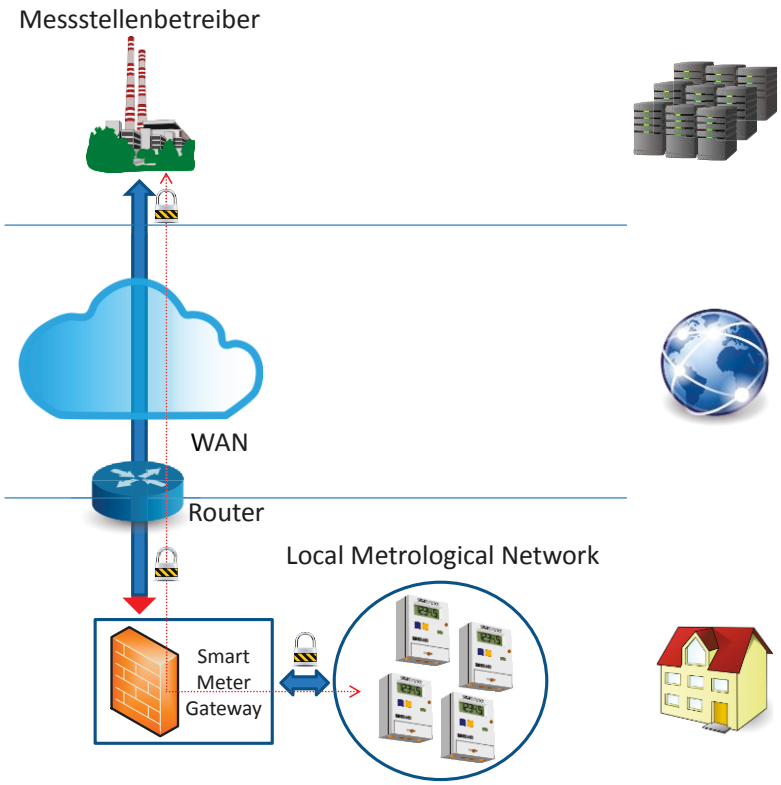

Abbildung 30: Infrastruktur des BSI-Schutzprofils für das Smart Metering

3.3. Auswahl eines Smart Metering-Protokolls als Basis für ein DPWS-Profil

Um ein Profil für DPWS zu entwickeln, wurde bei der Erfassung des Standes der Technik besonderer Wert auf die vom BSI priorisierten Protokolle M-Bus, DLMS/COSEM und SML gelegt. M-Bus ist ein Protokoll der Feldebene. Ein typisches Merkmal der Feldbusprotokolle (wie z.B. auch LON, KNX) sind sehr kleine Pakete (im Bereich von 10 Byte) mit wenig Kommunikationsoverhead. Die Feldbusprotokolle erfordern jedoch in der Regel einen sehr hohen Konfigurationsaufwand. M-Bus unterstützt kein IP-Protokoll und damit auch kein TLS. Für die Kommunikation mit dem Smart Meter Gateway wird ein symmetrisches Verschlüsselungsverfahren (Data Encryption Standard (DES) oder Advanced Encryption Standard (AES)) und eine Signatur verwendet. Im Vergleich zu TLS ist DES/AES weniger

sicher, da der symmetrische Schlüssel bei einem neuen Verbindungsaufbau nicht geändert wird.

Im Gegensatz zu M-Bus unterstützen DLMS/COSEM und SML das IP-Protokoll und TLS. DLMS/COSEM bietet dabei die TCP/UDP-Unterstützung mittels Bindings. SML ist ein reines Anwendungsschichtprotokoll und kann über beliebige Transportprotokolle wie TCP, HTTP, File Transfer Protocol (FTP) und andere übertragen werden. Sowohl bei DLMS/COSEM als auch bei SML liegt die Paketgröße bei der Nutzung des IP-Protokolls im Bereich von 100 Byte. Tabelle 6 gibt einen Überblick über die kumulativ übertragenen Datengrößen der einzelnen Protokolle auf der Anwendungsschicht (außer M-Bus). Es handelt sich hierbei nicht um allgemein gültige Vergleichswerte, sondern um ein ausgewähltes Beispiel, bei dem der aktuelle Zählerwert der Energie in kWh abgefragt wurde. Ein direkter Vergleich mit M-Bus ist nicht möglich, da dieser ein Bus-Protokoll ist und keine TCP/IP-Unterstützung bietet. DLMS/COSEM weist etwas kleinere Datenmengen auf, benötigt jedoch im Vergleich zu SML drei Request- und drei Response-Nachrichten (Association, Read, Release), was zu einem kumulativ höheren Overhead auf den unteren Schichten führen würde. Bei SML werden darüber hinaus zusätzliche Metadaten wie z.B. Zeitstempel immer übertragen.

Protokoll	**Request [Byte]**	**Response [Byte]**
DLMS/COSEM	81	86
SML	124	176
M-Bus	11	26

Tabelle 6: Datengrößen der beispielhaften Protokollframes

Bei DLMS/COSEM werden die Daten in Form von Objekten repräsentiert. Der Zugriff auf Objekte ähnelt durch die GET/SET-Methoden und die Objektpfade der REST-Architektur. Der Datenaustauch bei SML findet in Form einer Datei/eines Dokuments statt. Die Messwerte und Parameter werden als eine Sequenz von Elementen durch die entsprechenden Nachrichtentypen (Serviceaufrufe) abgefragt. Dadurch weist SML eine SOA-ähnliche Struktur auf. Einen Überblick über die Eigenschaften einzelner Protokolle liefert Tabelle 7. Die Eigenschaft *Architektur* bezieht sich dabei auf die Ähnlichkeit zu WS.

Aufgrund des Baukastenprinzips der WS wäre es möglich, das DPWS-Profil für Smart Metering auf Basis eines beliebigen Smart Metering-Protokolls zu realisieren. Auf Grund der oben besprochenen Überlegungen eignet sich jedoch SML besser als Grundlage für ein DPWS-Profil für Smart Metering und wurde daher verwendet.

	M-Bus	DLMS/COSEM	SML
Standard			
IP-Unterstützung			
Transport-Protokolle			
Open Source			
Sicherheit			
Architektur			
Discovery			

Tabelle 7: Features ausgewählter Smart-Metering-Protokolle

3.4. SML-Bausteine

SML ist ein Kommunikationsprotokoll für Smart Metering und wurde primär für Stromzähler entwickelt. Die SML-Spezifikation ist in die folgenden Teile untergliedert: Datenstrukturen für Payload-Übertragung, binäre SML-Codierung und das SML-Transportprotokoll. Das Letztere ist nur bei Punkt-zu-Punkt-Verbindungen anwendbar. Wie bereits erwähnt, ist SML ein Anwendungsschichtprotokoll. Die Daten werden in Form einer Datei/eines Dokuments übertragen. Die SML-Datei besteht aus einer Kette von Nachrichten. Es werden folgende 3 Dateitypen unterschieden: SML-Auftragsdatei, SML-Antwortdatei und SML-Kombidatei. SML-Auftragsdateien enthalten die Aufträge („Requests“), SML-Antwortdateien fassen die Antworten („Responses“) zu den Aufträgen zusammen, SML-Kombidateien enthalten beides. Jede SML-Datei beginnt mit einer OpenRequest- oder OpenResponse-Nachricht und endet entsprechend mit einer CloseRequest- oder CloseResponse-Nachricht. Dazwischen können beliebig viele Request- und Response-Nachrichten platziert werden. Die spezifizierten SML-Nachrichten in der Version 1.03 und deren Beschreibungen können Anhang A.1 entnommen werden.

Um die Funktionsweise von SML zu veranschaulichen, werden die GetProfileListRequest- und GetProfileListResponse-Nachrichten exemplarisch erklärt. Alle SML-Nachrichten folgen demselben Aufbau, der in Tabelle 8 dargestellt ist.

Die *transactionId* wird vom Auftraggeber erstellt. Diese wird später verwendet, um die Antworten den Aufträgen zuzuordnen. Mithilfe von *groupNo* können Nachrichtengruppen gebildet werden. Dadurch kann die Verarbeitungsreihenfolge der Nachrichten angegeben werden. *AbortOnError* zeigt das Verhalten der weiteren Nachrichtenverarbeitung in einem Fehlerfall an. Nachrichtenkörper befindet sich im *messageBody*-Feld. Um Übertragungsfehler zu erkennen, wird eine Prüfsumme im *crc16*-Feld gebildet. *EndOfSmlMsg* bezeichnet das Ende einer Nachricht.

Die GetProfileListRequest-Nachricht wird benutzt, um die Messwerte abzufragen. In der Antwort werden die Messwerte als eine einfache Liste übertragen. Der vollständige Aufbau der GetProfileListRequest-Nachricht kann Anhang A.1 entnommen werden. Nachfolgend wird nur auf die zwei wesentlichen Datenfelder eingegangen. *ServerId* bezeichnet die adressierte Datenquelle oder ein Softwaremodul und *ParameterTreePath* zeigt die angefragten Messwerte an.

Nachrichtenfeld	**Datentyp**
transactionId	String
groupNo	Unsigned8
abortOnError	Unsigned8
messageBody	SML Message Body
crc16	Unsigned16
endOfSmlMsg	EndOfSmlMsg

Tabelle 8: SML-Nachrichtenaufbau

Wenn die Anfrage erfolgreich verarbeitet wurde, werden die angefragten Werte in der GetProfileListResponse-Nachricht übertragen. Der vollständige Aufbau dieser Nachricht ist Anhang A.1 zu entnehmen. Nachfolgend wird nur auf einige ausgewählte Datenfelder eingegangen. Das Feld *actTime* bezeichnet die Zeit, wann der Server mit der Verarbeitung der Anfrage begonnen hat. Die Dauer der aktuellen Log-Periode wird in *regPeriod* präsentiert. Die angefragten Messwerte werden in Form einer Liste in *period_List* übertragen. Die Messwerte werden zusammen mit dem Namen, der Einheit, dem Skalierungsfaktor und einer optionalen Wertsignatur geschickt. Der Aufbau anderer Nachrichten kann in der SML-Spezifikation 1.03 nachgeschlagen werden [120].

3.5. DPWS-Profil für Smart Metering

In diesem Abschnitt wird die Möglichkeit einer Abbildung des SML-Protokolls auf DPWS gezeigt, anschließend wird die Performance beider Protokolle verglichen. Bei SML sind die Nachrichten ein Teil einer Datei/eines Dokuments. In der WS-Welt stellt eine Nachricht eine eigenständige Einheit dar. Folglich wird, um die Interoperabilität mit anderen WS-Standards zu behalten und die Vorteile der WS-Technologie nutzen zu können, eine SML-Datei aufgesplittet und ihre Teile werden auf WS-Standards abgebildet. Dabei repräsentiert eine DPWS-Nachricht eine Anfrage oder eine Antwort und entspricht einer SML-Datei mit einer Auftrags- oder einer Antwortnachricht. Diese Änderung hat nur einen Einfluss auf SML-Dateien mit mehreren Nachrichtentypen. Mehrere Messwerte können weiterhin durch eine Anfrage-/Antwortnachricht abgerufen werden. Der vorgeschlagene Ansatz wird anhand der

GetProfileListRequest-Nachricht gezeigt. Die Funktionalität der GetProfileListRequest-Nachricht kann durch HTTP, SOAP und WS-Addressing wiedergegeben werden. Zuerst wird HTTP betrachtet. *ServerId* entspricht dem Ressource-Pfad auf dem Server. In der WS-Welt ist dieser durch den URI repräsentiert. Ein URI hat diesen Aufbau: *serverHost:serverPort/serverId*. Er wird in HTTP wie folgt übertragen:

```
POST /serverId HTTP/1.1
Host: serverHost:serverPort
```

Die Felder *groupNo* und *abortOnError* können weggelassen werden, da sich nur eine Nachricht in der Anfrage oder Antwort befindet. Nachrichtentyp und *transactionId* werden in den SOAP-Header mithilfe von WS-Addressing platziert:

```
<soap:Header>
    <wsa:Action>messageType</wsa:Action>
    <wsa:MessageID>transactionId</wsa:MessageID>
</soap:Header>
```

Der *Action*-Tag gibt den Inhalt des SOAP-Body an, d.h. GetProfileListRequest-Operation. Um die Autorisierungsinformationen durch Benutzername und Password bereitzustellen, wird der WS-Security-Standard verwendet. Dieser Autorisierungsmechanismus ist nur bei einer verschlüsselten Kommunikation anwendbar. Ein Beispiel von WS-Security ist nachstehend dargestellt:

```
<soap:Header>
    <wsse:Security>
        <wsse:UsernameToken>
            <wsse:Username>username</wsse:Username>
            <wsse:Password>password</wsse:Password>
        </wsse:UsernameToken>
    </wsse:Security>
</soap:Header>
```

Die restlichen Parameter der GetProfileListRequest-Nachricht werden im SOAP-Body übertragen. Der Übersichtlichkeit halber werden nachstehend nur obligatorische Felder benutzt. Die optionalen Felder können analog hinzugefügt werden. Als Beispiel wird eine SML-Datei mit zwei Parameteranfragen betrachtet. Demnach kann das *parameterTreePath*-Feld eines Auftrags wie folgt durch den SOAP-Body widergespiegelt werden:

```
<soap:Body>
    <sml:GetProfileListReq>
        <sml:parameterTreePath>valueRequest1</sml:parameterTreePath>
        <sml:parameterTreePath>valueRequest2</sml:parameterTreePath>
    </sml:GetProfileListReq>
</soap:Body>
```

Nach dem Empfang eines Operationsaufrufs wird eine Antwort vom Server erzeugt. Die zum obenstehenden Aufruf entsprechende Antwort wird ebenfalls im SOAP-Body übertragen:

```
<soap:Body>
    <sml:GetProfileListRes>
        <sml:actTime>time</sml:actTime>
        <sml:regPeriod>period</sml:regPeriod>
        <sml:parameterTreePath>valueRequest1</sml:parameterTreePath>
        <sml:parameterTreePath>valueRequest2</sml:parameterTreePath>
        <sml:period>
            <sml:objName>name1</sml:objName>
            <sml:unit>unit1</sml:unit>
            <sml:scaler>scaler1</sml:scaler>
            <sml:value>value1</sml:value>
        </sml:period>
        <sml:period>
        ...
        </sml:period>
    </sml:GetProfileListRes>
</soap:Body>
```

Alle anderen Nachrichten und Parameter können auf dieselbe Weise zusammengesetzt werden. SML-Nachrichtenfelder *crc16* und *endOfSmlMsg* werden nicht durch Web Services widergespiegelt, da diese Felder bereits durch die unteren Schichten abgedeckt sind. Demnach kann die komplette SML-Kommunikation auf existierende WS-Standards abgebildet werden. Die vorgeschlagene Abbildung von SML auf DPWS ist in Tabelle 9 dargestellt.

SML-Attribut	**WS-Standard/Protokoll**
ServerId	HTTP
TransactionId	WS-Addressing
Message type	WS-Addressing
Username/Password	WS-Security
Signature	WS-Security
Andere Attribute	SOAP

Tabelle 9: Abbildung der SML-Attribute auf Web Services

Das SML-basierte Profil für WS kann darüber hinaus mit den Standard-DPWS-Features wie Geräte- und Service-Discovery erweitert werden. Dadurch können die verfügbaren Funktionen eines speziellen Gerätes gelernt werden.

3.5.1. Evaluierung des DPWS-Profils für Smart Metering

Für den Einsatz von DPWS auf Geräten mit beschränkten Ressourcen kann der an der Universität Rostock entwickelte, frei verfügbare uDPWS-Stack verwendet werden. Dieser benötigt nur 46 KB ROM und 7 KB RAM [140]. Das Hauptproblem beim Einsatz von Web Services in Umgebungen mit eingeschränkten Ressourcen ist weiterhin der hohe Kommunikationsoverhead. In Breitbandnetzen ist der Kommunikationsoverhead weitestgehend unproblematisch. Im Smart Metering-Umfeld kommen allerdings Technologien mit deutlich geringerer Bandbreite, wie Power Line Communication (PLC) (z.B. HomePlug Green PHY [16]) oder drahtlose Low Power-Technologien wie 6LoWPAN zum Einsatz [141]. Dort ist der Traffic-Overhead unerwünscht. Der größte Overhead tritt auf, wenn nur ein Wert, z.B. ein Integer, übertragen werden muss. Für Vergleichszwecke sind weitere Annahmen in Bezug auf die zu übertragenden Werte notwendig. Folgende Annahmen wurden gemacht: *Action* - 4 Byte, *serverId* - 6 Byte, *parameterTreePath* - 17 Byte. Die Länge des Strings *parameterTreePath* entspricht der hexadezimalen Schreibweise von OBIS (6 Gruppen mit Trennzeichen).

Angenommen, der Client (Gateway) fragt den aktuellen Energieverbrauch beim Server (Smart Meter) an. Bei SML wird diese Anfrage als eine Datei mit einer GetProfileListRequest-Nachricht geschickt. Bei DPWS stellt diese Anfrage einen Aufruf der GetProfileListReq-Operation dar. Der Server antwortet mit einer SML-Datei mit der GetProfileListResponse-Nachricht bzw. mit einem Rückgabewert des Operationsaufrufs. Für die Worst Case-Betrachtung werden alle optionalen Felder außer *serverId* weggelassen. Die Overheadrate kann mit Formel (1) berechnet werden.

$$Overheadrate = \frac{PaketInhalt - Payload}{PaketInhalt} \quad (1)$$

Da sowohl SML als auch DPWS dasselbe Transportschichtprotokoll (TCP) verwenden, wird hier der Paketinhalt als TCP-Payload betrachtet. Die Nutzlast einer Anfrage stellen *serverId* und *parameterTreePath* dar. Die Nutzlast einer Antwort sind alle obligatorischen Felder der GetProfileListResponse-Nachricht. Die resultierende Overhead-Berechnung ist in Abbildung 31 dargestellt.

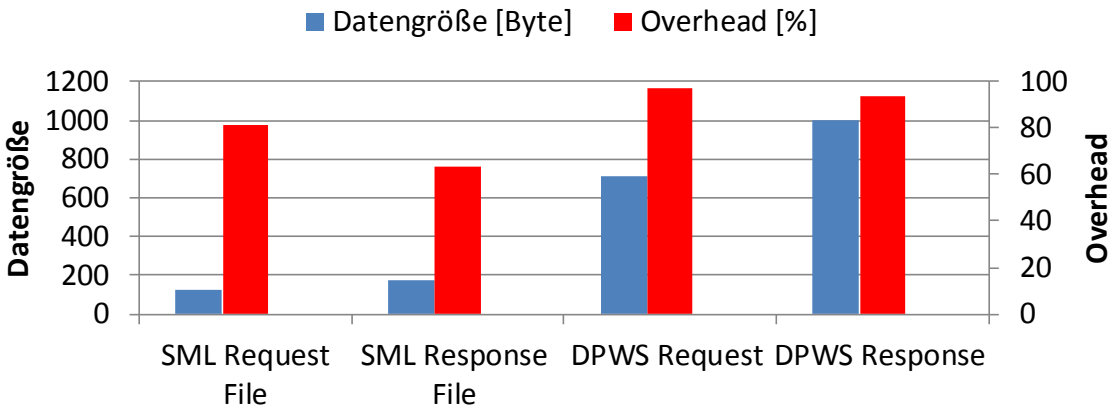

Abbildung 31: Vergleich des Kommunikations-Overheads von SML und DPWS

Wie erwartet, erzeugt DPWS einen sehr hohen Kommunikations-Overhead wegen des textbasierten Charakters von HTTP und der XML-Codierung von SOAP. Das erschwert den Einsatz von Web Services in Umgebungen mit eingeschränkten Ressourcen wie Smart Metering. Zur Traffic-Reduktion können Kompressionsmethoden angewendet werden. Standardkompressionsmethoden für HTTP wie gzip [142] oder Deflate [143] sind ungeeignet, da sie zu einer deutlich höheren CPU- und Speicherauslastung führen würden. Kompressionsmechanismen, die eine sehr hohe Kompressionsrate erreichen können, ohne zusätzliche CPU- und Speicherauslastung hervorzurufen, werden daher in dieser Arbeit verwendet und im Folgenden präsentiert.

3.5.2. Kompression

Um HTTP-Header zu komprimieren, wird das CoAP-Protokoll vorgeschlagen. CoAP wurde für den Datenaustausch in Sensornetzwerken entwickelt. Es hat einen HTTP-ähnlichen Aufbau, sodass HTTP sich leicht auf CoAP abbilden lässt. Im Gegensatz zu HTTP ist CoAP ein binärcodiertes Protokoll. Grundsätzlich unterstützt CoAP nur UDP. Allerdings wurde das TCP-Binding für zukünftige Spezifikationen offen gelassen. Um die Sicherheitsanforderungen des BSI erfüllen zu können, wird ein TCP-Binding vorgeschlagen. Damit kann TLS für die Gewährleistung der Kommunikationssicherheit verwendet werden.

Da TCP ein verbindungsorientiertes Protokoll ist und somit eine erfolgreiche Nachrichtenzustellung gewährleistet, wird CoAP-over-TCP im Non-Confirmable Mode benutzt. Die Länge einer Nachricht bei dem CoAP-over-UDP-Binding wird aus dem UDP-Header abgeleitet. Da der TCP-Header keine Angaben über die Datenlänge liefert, muss sich diese im CoAP-Header widerspiegeln. Die minimale Länge des CoAP-Headers ist 4 Byte. Dieser besteht aus folgenden Feldern: Version Number (*Ver*), Type (*T*), Option Count (*OC*), *Code* und *Message ID*. Als Grundlage wird hierbei CoAP Draft 12 herangezogen (siehe Abbildung 32). Das Type-Feld (*T*) spezifiziert den Typ einer Nachricht und das *Code*-Feld gibt an, ob es sich um eine Anfrage oder eine Antwort handelt. Die Anzahl der Optionen im

Header wird im *OC*-Feld angegeben. *Message ID* ist ein 2 Byte langes Feld, das nur in Confirmed Mode benutzt wird. Folglich kann das Feld für die Nachrichtenlänge im Non-Confirmable Mode verwendet werden. Die Funktionalität dieses Feldes würde dann dem „Content-Length"-Header von HTTP entsprechen. Um die DPWS-Funktionalität sicherstellen zu können, muss ein Mindestsatz an CoAP-Options (Headers) von Geräten unterstützt werden. Optionale HTTP-Header, die keinen Einfluss auf die DPWS-Funktionalität haben und von CoAP nicht unterstützt sind, werden bei der Kompression weggelassen. Der vorgeschlagene Mindestsatz an CoAP-Options ist in Tabelle 10 aufgeführt.

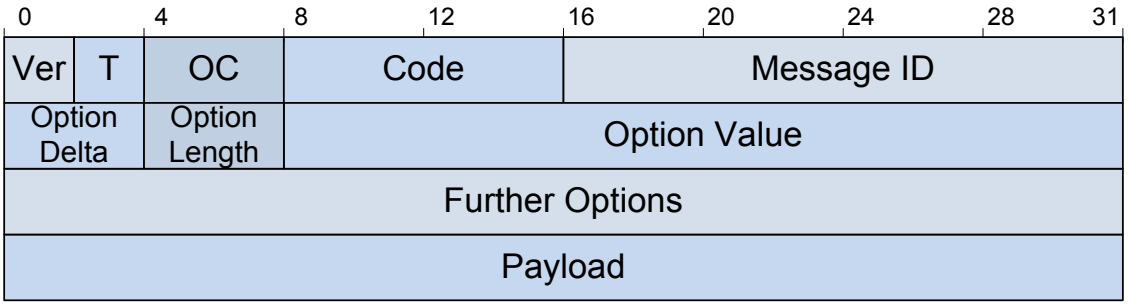

Abbildung 32: CoAP-Header nach CoAP Draft 12

Nummer	Name	Format
1	Content-Type	Uint
5	Uri-Host	String
7	Uri-Port	String
9	Uri-Path	String

Tabelle 10: Mindestsatz an CoAP-Options für DPWS

Content-Type-Option entspricht dem Content-Type-Header von HTTP. Uri-Host, Uri-Port und Uri-Path bilden zusammen die URI. Laut CoAP-Spezifikation haben Uri-Host- und Uri-Port-Options Standardwerte, wenn diese Options nicht in der Nachricht vorkommen. In diesem Fall werden Uri-Host und Uri-Port von der Ziel-IP-Adresse und der Zielportnummer abgeleitet, die im TCP-Header zu finden sind. Demnach wird es in den meisten Fällen ausreichend sein, nur die Content-Type- und die Uri-Path-Options anzugeben. Durch die Nutzung von CoAP als binäre HTTP-Codierung können hohe Kompressionsraten erreicht werden, wie in Abbildung 33 dargestellt ist. Diese Kompressionsmethode erzeugt keine zusätzliche CPU- und Speicherauslastung, da die CoAP-Options direkt (ohne vorherige Dekomprimierung) gelesen bzw. (ohne nachfolgende Kompression) geschrieben werden können (vgl. Abbildung 32). Ein Beschränken auf HTTP-Kompression würde jedoch nicht zu einer signifikanten Reduzierung des DPWS-Overheads führen. Um dieses Ziel zu erreichen,

ist eine weitere Kompression von XML und SOAP erforderlich. Für diesen Zweck wird Efficient XML Interchange (EXI) vorgeschlagen.

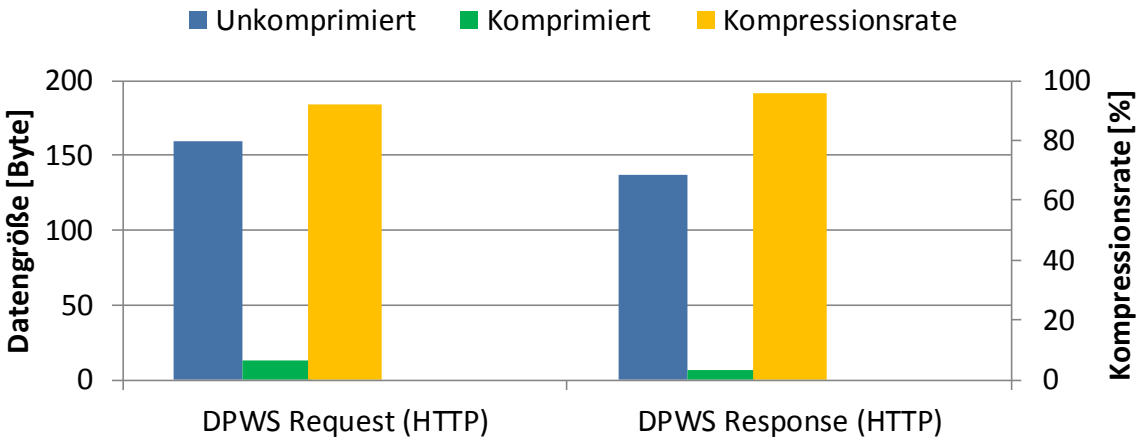

Abbildung 33: HTTP-Kompression

EXI stellt eine binäre XML-Codierung dar [144]. Die XML-Elemente werden durch Events repräsentiert. Ein EXI-Dokument besteht aus einer Folge der Events. Mithilfe des zusätzlichen Kontextes, der den Events zugeordnet werden kann, können XML-Attribute beschrieben werden. Im Gegensatz zu einer herkömmlichen Kompression ist der Zugriff auf einzelne Elemente direkt ohne vorherige Dekomprimierung möglich. Darüber hinaus werden XML-Elementwerte entsprechend dem Datentyp codiert. Ein Integer wird beispielsweise durch einen 4 Byte Wert anstelle eines 10 Byte Strings repräsentiert. EXI hat verschiedene Optionen, um den Kompromiss zwischen der Kompressionsrate und einer verlustfreien XML-Kompression genau festzulegen. Es können zum Beispiel XML-Namespace-Präfixe bei der Codierung erhalten und nicht durch IDs ersetzt werden. Das resultiert in einer präziseren XML-Transformation, aber auch in der höheren Datengröße.

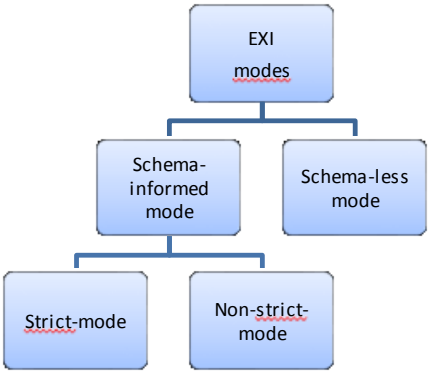

Abbildung 34: EXI Modes

Zur Codierung eines XML-Dokuments benutzt EXI Grammars (Grammatiken). Mithilfe von Grammars werden Event-Codes erzeugt. Events, die eine höhere Auftrittswahrscheinlichkeit haben, werden mit weniger Bits codiert. EXI unterscheidet zwei Modi: Schema-less und Schema-informed (vgl. Abbildung 34). Schema-less Mode wird benutzt, wenn keine Informationen über das XML-Schema vorliegen. In diesem Fall werden die integrierten Grammars benutzt. Diese Grammars werden durch einen lernenden Mechanismus ständig erweitert.

Daten, die einem Event zugeordnet sind, werden nur einmal codiert. Zum Beispiel wird der String eines schließenden Tags des XML-Dokuments nicht erneut codiert, da dieser bei dem zugehörigen öffnenden Tag bereits codiert wurde. Die besseren Kompressionsraten können im Schema-informed Mode erreicht werden. In diesem Fall sind die möglichen Events im Voraus bekannt und können effizienter durch weniger Bits codiert werden. Der Schema-informed Mode kann darüber hinaus in Strict und None-Strict unterteilt werden. Im Strict Mode muss das zu codierende Dokument exakt mit dem vorliegenden XML-Schema übereinstimmen. Jede Abweichung führt zu einem Codierungsfehler. Die Länge der Event-Codes kann dadurch jedoch reduziert werden. Im Non-Strict Mode sind Abweichungen erlaubt und werden mit den lernenden, integrierten Grammars codiert. Dieser Modus resultiert in längeren Event-Codes im Vergleich zu Strict Mode, es kann jedoch jedes Dokument codiert werden. Um die Funktionsweise des EXI zu veranschaulichen, ist in Abbildung 35 ein XML-Dokument dem entsprechenden EXI-Stream (Schema-informed Strict Mode) gegenübergestellt.

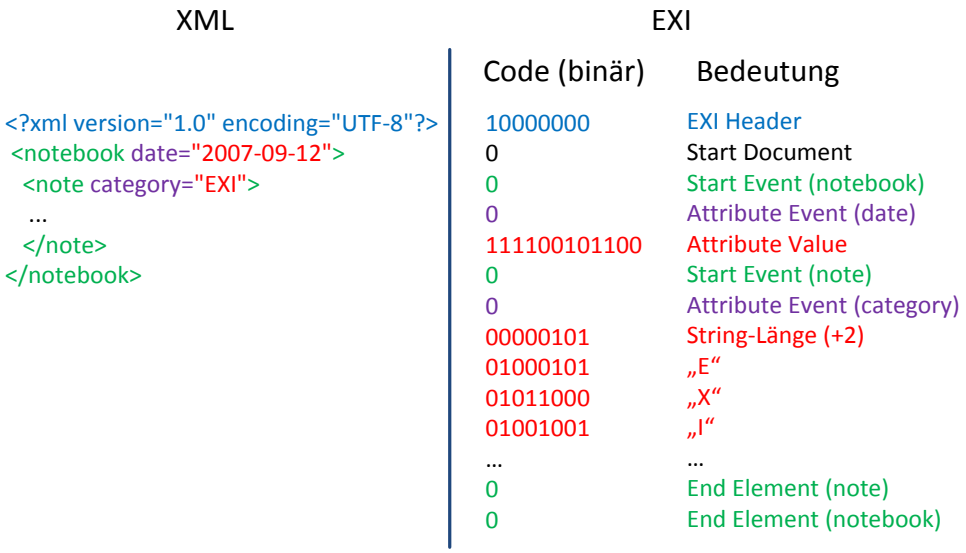

Abbildung 35: Gegenüberstellung von XML und EXI (Schema-informed Strict Mode)

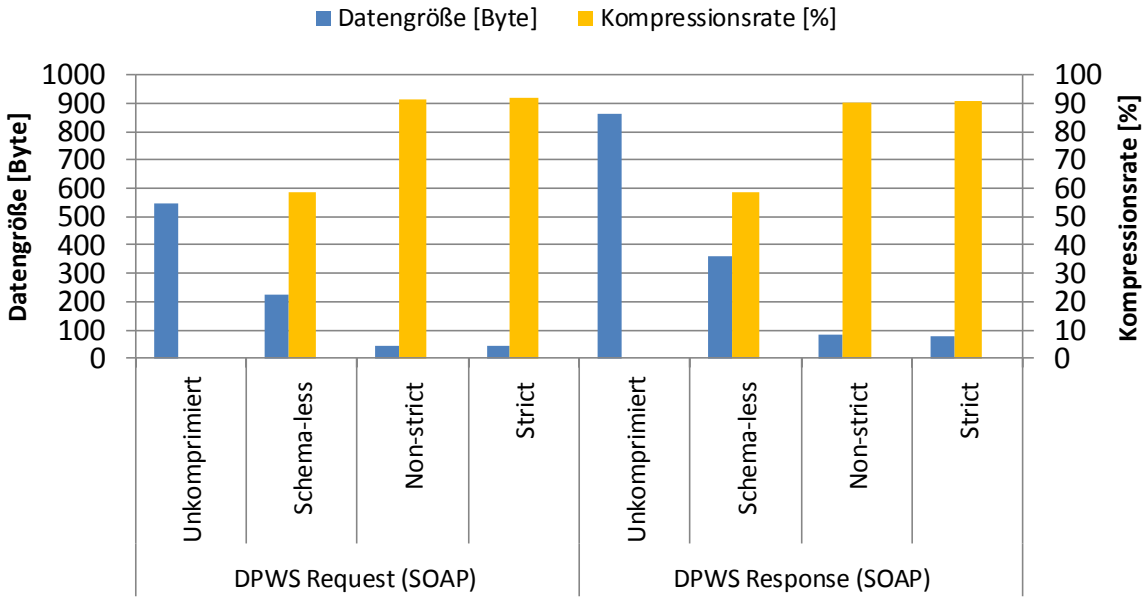

Abbildung 36: SOAP-Kompression

Für die weitere Overhead-Reduktion von DPWS wurde ein üblicher UUID, der für die Message ID in WS-Addressing genutzt wird, durch einen kürzeren 4 Byte Identifier ersetzt. Die mit EXI erzielten SOAP-Kompressionsergebnisse sind in Abbildung 36 dargestellt. Da das vorgeschlagene DPWS-Profil für eine spezielle Anwendung (Smart Metering) konzipiert ist, kann der Schema-informed Strict Mode benutzt werden. Einen Überblick über die benutzten Kompressionsmethoden in Bezug auf den Protokoll-Stack bietet Abbildung 37.

Die insgesamt erzielten DPWS-Kompressionsergebnisse zeigt Abbildung 38. Der vorgeschlagene Ansatz kann somit den DPWS-Kommunikations-Overhead um über 90 % senken. Die komprimierten DPWS-Nachrichten sind dementsprechend weniger als 100 Byte lang. Damit kann DPWS ohne Einschränkungen für Medien mit niedrigen Datenraten eingesetzt werden.

Ein erneuter Vergleich zwischen SML und komprimiertem DPWS ist in Abbildung 39 dargestellt. Wie in Abbildung 39 zu sehen, verursachen DPWS-Request- und DPWS-Response-Nachrichten weniger Overhead als SML, solange eine SML-Datei mit einer Auftrag- oder einer Antwort-Nachricht betrachtet wird. Mehrere Messwerte können weiterhin mit einer DPWS-Nachricht angefragt oder gesendet werden. SML wird nur dann weniger Kommunikations-Overhead verursachen, wenn mehrere unterschiedliche Nachrichtentypen übertragen werden müssen. Dies ist durch den sinkenden Overhead-Anteil der unteren Schichten (TCP/IP) bedingt. Es ist jedoch weniger üblich, unterschiedliche Auftrags- und Antwortmethoden mit einer spezifischen Messeinrichtung zu verwenden.

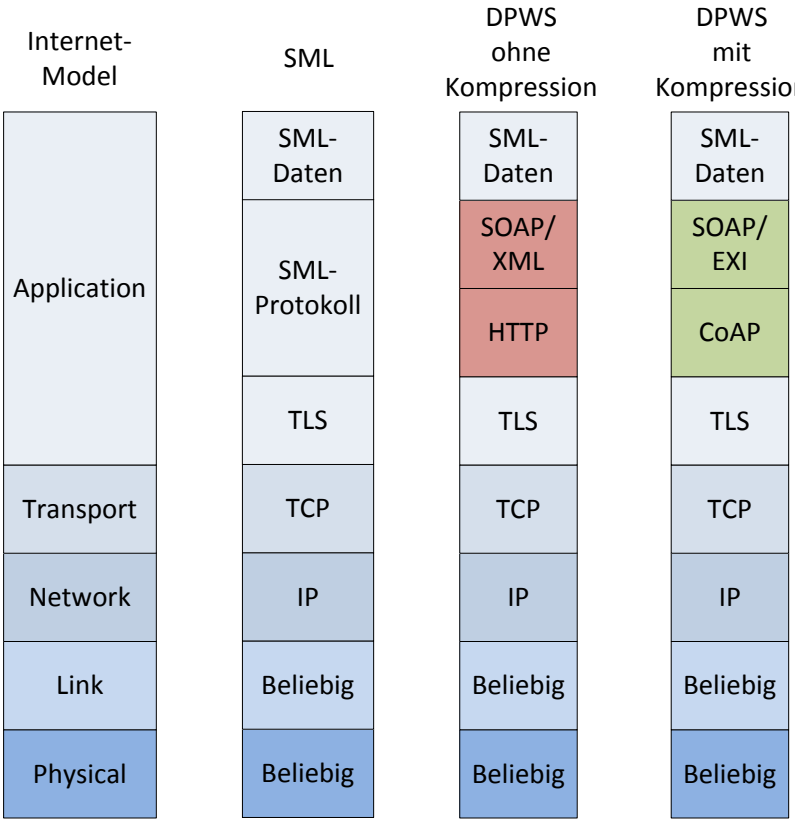

Abbildung 37: Überblick über die Kompressionsmethoden im Protokoll-Stack

Der vorgeschlagene Ansatz schränkt die WS-Funktionalität nicht ein und kann mit anderen WS-Standards erweitert werden. DPWS unterstützt standardmäßig TLS im vollen Umfang und erfüllt damit automatisch die BSI-Richtlinie.

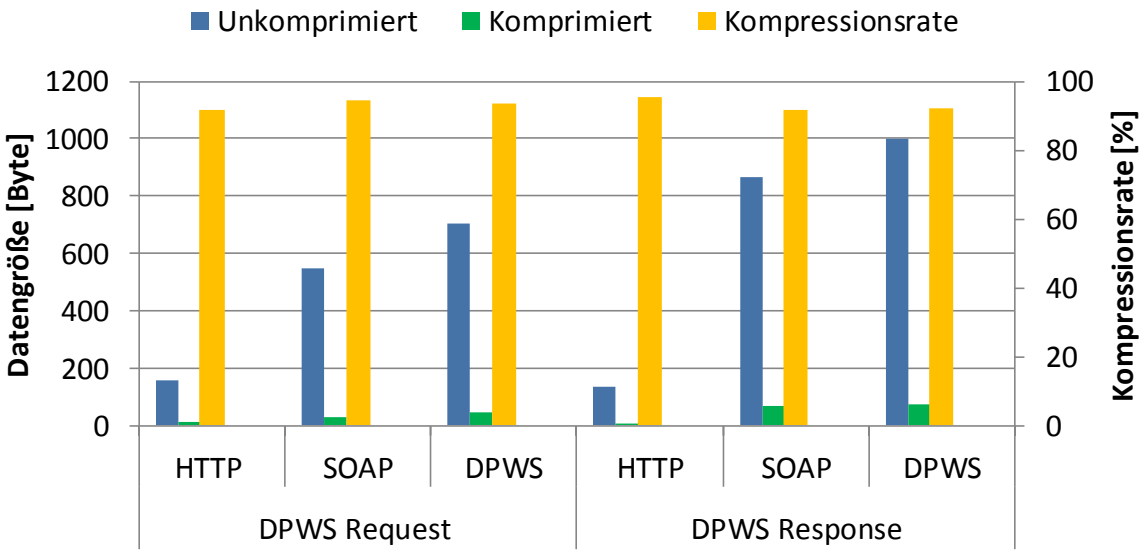

Abbildung 38: DPWS-Kompression

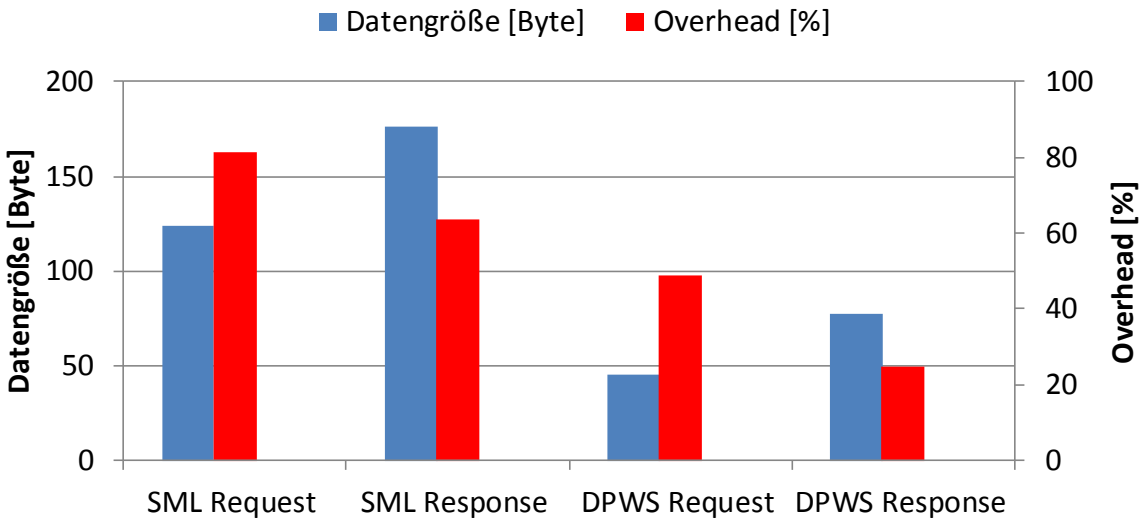

Abbildung 39: Vergleich des Kommunikations-Overheads von SML und komprimiertem DPWS

3.6. Analytische Betrachtung der Allgemeingültigkeit des vorgeschlagenen Ansatzes

Jedes Protokoll, unabhängig von Anwendung und Einsatzgebiet, lässt sich in folgende Bausteine unterteilen, die in Tabelle 11 aufgelistet sind.

D sei eine Menge von Datentypen, E sei eine Menge von Elementen und A sei eine Menge von Attributen, die von einem Protokoll unterstützt werden. Weiterhin sei M eine Menge von Nachrichten, die übertragen werden. T sei die Übertragungsart, die es ermöglicht, die Nachrichten zwischen Quelle und Senke zu übermitteln. Dann kann eine Menge von Protokollen P als ein kartesisches Produkt einzelner Bausteine wie folgt definiert werden:

$$P = D \times E \times A \times M \times T \tag{2}$$

Des Weiteren sei D' ein Menge von DPWS-Datentypen, E' eine Menge von DPWS-Elementen, A' eine Menge von DPWS-Attributen, M' eine Menge von DPWS-Nachrichten und T' die DPWS-Nachrichtenübertragungsart. Als P' wird dann eine Menge von DPWS-Profilen angenommen, die eine Abbildung anderer Protokolle auf DPWS darstellen. P' lässt sich dann als ein kartesisches Produkt der DPWS-Bestandteile definieren:

$$P' = D' \times E' \times A' \times M' \times T' \tag{3}$$

Bausteine	Erklärung
Datentypen	Die Repräsentation der Daten. Die atomaren Datentypen können weiterhin zur komplexen Datentypen zusammengefasst werden.
Elemente	Als Elemente werden Nutzdaten bezeichnet, die einen konkreten Wert eines Datentyps darstellen.
Attribute	Als Attribute werden zusätzliche Parameter bezeichnet, die Metainformationen zu den Elementen enthalten.
Nachrichten	Die Nachrichten sind ein Regelsatz, der beschreibt, welche Daten und ggf. in welcher Reihenfolge übertragen werden müssen.
Übertragung	Die Übertragung beschreibt, wie die Nachrichten zwischen Quelle und Senke übermittelt werden.

Tabelle 11: Generische Protokollbausteine

Wenn p' ein spezielles DPWS-Profil und ψ eine Abbildungsfunktion sind, für welche gilt

$$p \mapsto \psi(p) \in P',\ p \in P,\ \text{wobei} \qquad (4)$$
$$p' = \psi(p)$$

Dann ist es zu zeigen, dass

$$\text{wenn } \forall p \in P: D \to D', E \to E', A \to A', M \to M', T \to T', \text{ dann} \qquad (5)$$
$$\psi: P \to P'$$

Zunächst werden die Datentypen betrachtet. DPWS unterstützt alle bekannten atomaren Datentypen sowie alle komplexen Datentypen, die aus beliebigen atomaren Datentypen zusammengesetzt werden können (vgl. Abbildung 40). Das bedeutet:

$$\nexists d \in D: d \to d' \notin D' \qquad (6)$$

Somit gilt:

$$\forall d \in D: d \to d' \in D' \qquad (7)$$

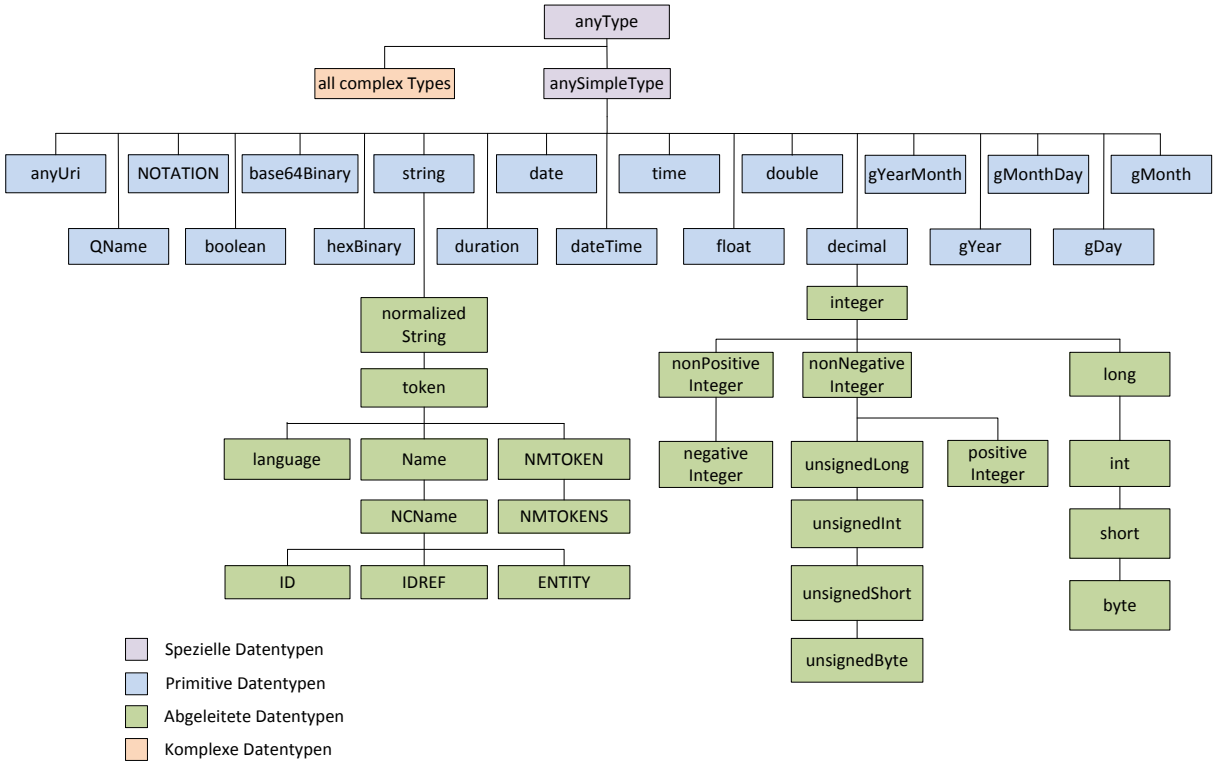

Abbildung 40: DPWS-Datentypen [145]

Als nächstes werden die Elemente betrachtet. Ein Element stellt dabei einen konkreten Wert eines bestimmten Datentyps dar. In DPWS werden die Werte in XML-Tags eingeschlossen. Es sei N_E die Bezeichnung eines Wertes und V_E der Wert selbst. Dann lässt sich für beliebige Bezeichnung-Wert-Paare folgende Darstellung realisieren:

$$<N_E> \mathrm{V_E} </N_E>$$

$$\text{wobei } E = (N_E, V_E)$$

Bei komplexen Datentypen kann der Wert in weitere Bezeichnung-Wert-Paare unterteilt werden:

$$<N_{E1}>$$
$$<N_{E2}> \mathrm{V_{E2}} </N_{E2}>$$
$$<N_{E3}> \mathrm{V_{E3}} </N_{E3}>$$
$$</N_{E1}>$$

Oder auch:

$$
\begin{aligned}
&<N_{E1}>\\
&\quad <N_{E2}>\\
&\qquad <N_{E3}>\ V_{E3}\ </N_{E3}>\\
&\quad </N_{E2}>\\
&</N_{E1}>
\end{aligned}
$$

Die Kaskadierung kann dabei beliebige Tiefe haben. Dadurch können beliebig komplexe Datentypen erstellt werden, deren Werte mit DPWS übertragen werden können. Beliebige Werte anderer Protokolle sind damit auf DPWS abbildbar. Somit gilt:

$$\forall e \in E: e \to e' \in E' \tag{8}$$

Die Attribute können zusätzliche Metainformationen zu den Elementen bereitstellen. Wie alle anderen Informationen in der Informatik müssen diese Informationen ebenfalls in Form von Datentypen repräsentiert werden. Eine Metainformation zu einem Messwert wäre z.B. die Kanalnummer. Die Kanalnummer kann wiederum durch einen Integer-Wert dargestellt werden. Deswegen besitzen Attribute selbst Datentypen. Diese Datentypen sind atomar, damit es keine Metainformationen zu Metainformationen gibt. Ein Element kann beliebig viele Metainformationen besitzen. Bei DPWS werden die Attribute innerhalb des Element-Tags platziert. Es sei N_A die Bezeichnung eines Attributes und V_A der Wert eines Attributes. Dann lässt sich für beliebige Bezeichnung-Wert-Paare folgende Darstellung realisieren:

$$<N_{E1}\ \ N_{A1}= V_{A1}\ \ N_{A2}= V_{A2}\ \ N_{A3}= V_{A3}\ \ldots> V_{E1} </N_{E1}>$$

$$\text{wobei } A = (N_A, V_A)$$

Somit gilt:

$$\forall a \in A: a \to a' \in A' \tag{9}$$

Nachrichten geben die Struktur an, mit der die übertragenen Daten und deren Reihenfolge festgelegt werden. Die Nachrichten transportieren Elemente und deren Metainformationen. Bei DPWS werden die Elemente in XML-Strukturen in deren Reihenfolge hintereinander angegeben. Dann kann jede beliebige Nachricht M als Abfolge von Elementen und deren Attributen wie folgt beschrieben werden:

$$
\begin{aligned}
&<N_{E1}\ N_{A11}= V_{A11}\ \ N_{A12}= V_{A12}\ \ldots > V_{E1} </N_{E1}>\\
&<N_{E2}\ N_{A21}= V_{A21}\ \ N_{A22}= V_{A22}\ \ldots > V_{E2} </N_{E2}>\\
&<N_{E3}\ N_{A31}= V_{A31}\ \ N_{A32}= V_{A32}\ \ldots > V_{E3} </N_{E3}>
\end{aligned}
$$

Die typischen Parameter, die in jeder Nachricht enthalten sein können, sind Adressierung, Nachrichtenkennzeichnung und Operation. Die Adressierung beschreibt, welche Quelle oder Modul angesprochen wird. Der Adressierungsparameter ist bereits ein Bestandteil von DPWS. Wenn *SourceAddress* die Adresse einer Quelle ist, dann kann diese nach dem oben gezeigten Verfahren wie folgt in DPWS integriert werden:

<wsa:Address> SourceAddress </wsa:Address>

Hierbei ist *SourceAddress* der Wert (V_E) und *wsa:Address* der Name des Wertes (N_E). Je nach Protokoll können die Nachrichten eine eindeutige Kennzeichnung tragen, um diese voneinander unterscheiden zu können. Dann sei *UUID* eine eindeutige Kennzeichnung einer Nachricht. Die Kennzeichnung der Nachrichten bei DPWS kann dann mittels eines XML-Tags *MessageID* erfolgen:

<wsa:MessageID> UUID </wsa:MessageID>

Ähnlich wie bei der Adressierung ist hier *UUID* der Wert (V_E) und *wsa:MessageID* der Name des Wertes (N_E). Die Operation gibt an, welche Informationen von einer Quelle angefragt wurden. Bei DPWS kann dafür ein bereits vordefinierter Parameter verwendet werden. Wenn *Operation* eine Operationskennung ist, kann diese wie folgt in DPWS integriert werden:

<wsa:Action> Operation </wsa:Action>

Hierbei ist der Wert V_E *wsa:Action* und der Name des Wertes N_E die *Operation*. Somit gilt:

$$\forall m \in M : m \to m' \in M' \tag{10}$$

Als nächstes wird die Übertragung betrachtet. Die Übertragung ist die Fähigkeit eines Protokolls, Nachrichten zwischen Quelle und Senke auszutauschen. DPWS unterstützt alle bekannten Nachrichtenaustauschmuster. Diese sind in Tabelle 12 dargestellt. Bei der Nachrichtenübertragung werden durch die Verwendung des IP ebenfalls alle Adressierungsarten Unicast, Multicast und Broadcast unterstützt.

DPWS unterstützt weiterhin unterschiedliche Bindings, die sowohl eine verbindungslose (z.B. UDP) als auch eine verbindungsorientierte (z.B. TCP) Kommunikation ermöglichen. Da es keine bekannte Nachrichtenübertragung gibt, die DPWS nicht unterstützt, gilt somit:

$$\forall t \in T : t \to t' \in T' \tag{11}$$

Art	Erklärung
One-Way	Es wird eine Anfrage abgeschickt, die auf Kommunikationsebene keine Antwort erfordert. Beispiel: Die Senke schickt der Quelle eine Nachricht.
Request-Response	Eine Anfrage wird mit einer Rückmeldung beantwortet. Beispiel: Die Senke schickt der Quelle eine Nachricht und bekommt eine Antwort zurück.
Solicit-Response	Diese Art ermöglicht eine Rückkopplung, wobei eine Request-Response-Kommunikation in die entgegengesetzte Richtung abläuft. Beispiel: Die Quelle schickt der Senke eine Nachricht und bekommt eine Antwort zurück.
Notification	Die Quelle informiert die Senke mit einer Nachricht. Beispiel: Die Quelle schickt der Senke eine Nachricht.

Tabelle 12: Übertragungsarten

Es wurde hiermit gezeigt (vgl. Formel (7) – (11)):

$$\forall p \in P: D \to D', E \to E', A \to A', M \to M', T \to T' \tag{12}$$

Damit gilt:

$$\forall p \in P: p \mapsto \psi(p) \in P' \text{ bzw. } \psi: P \to P', \text{ was zu zeigen war.} \tag{13}$$

Die analytische Betrachtung hat gezeigt, dass es unabhängig von der Protokollstruktur und spezifischen Protokolleigenschaften durch die flexiblen, anpassbaren und durchdachten Mechanismen von Web Services möglich ist, ein beliebiges Protokoll auf DPWS abzubilden (vgl. Abbildung 41). Konkret lassen sich also nachweisbar u.a. alle Protokolle im Smart Home, Smart Metering und Smart Building-Bereich auf DPWS abbilden (siehe auch Anhang A.2-A.4). Damit ist ein wesentliches Ziel dieser Arbeit nun auch analytisch bestätigt.

Besonders effiziente Lösungen erhält man sinnvollerweise, wenn man sich auf Protokolle aus der TCP/IP-Familie beschränkt. Dann sind Ergebnisse mit DPWS erzielbar, die bezüglich ihrer Codeeffizienz und des Kommunikationsbedarfs durch Einsatz der gezeigten Methoden mit den Originalprotokollen mithalten bzw. diese sogar deutlich übertreffen können.

Da beim Protokollentwurf zunehmend ausschließlich auf TCP/IP aufgesetzt wird, liegt hier also ein nicht nur theoretisch relevantes Ergebnis vor, sondern eines mit großer praktischer Tragweite für den Smart X – Bereich.

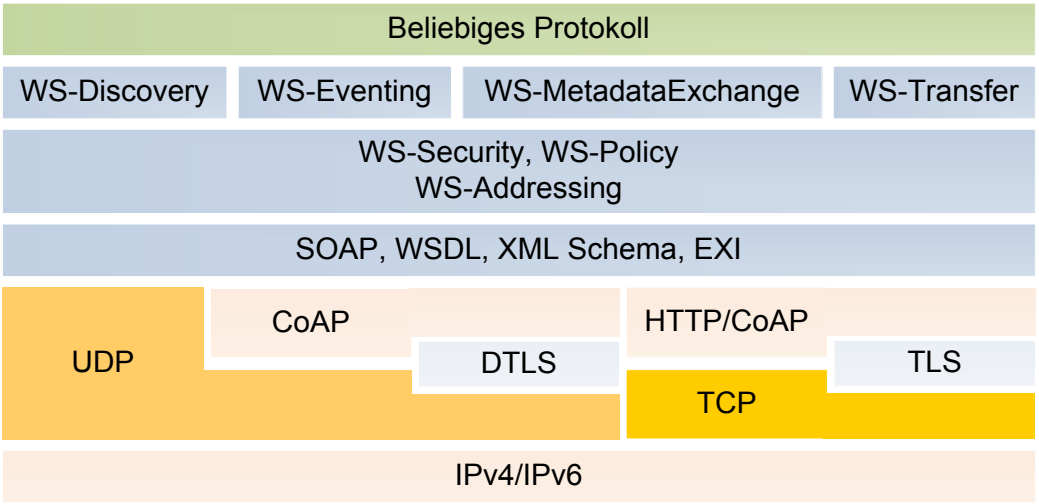

Abbildung 41: Abbildung eines beliebigen Protokolls auf DPWS

3.7. Plug&Play-Technologien in der Gebäudeautomation

Basierend auf dem Smart Metering-Profil ist der Einsatz von DPWS für die Gebäudeautomation möglich. Dabei können ebenfalls Medien mit niedrigen Datenraten wie PLC oder 6LoWPAN Verwendung finden. Im Gegensatz zum Smart Metering ist die Anzahl der Gerätetypen in der Gebäudeautomation unbegrenzt. Für eine dynamische und herstellerübergreifende Kommunikation bietet DPWS Plug&Play-Techniken an. Der wesentliche Vorteil besteht in dem minimalen Konfigurationsaufwand. Jedes Gerät kann auf Anfrage eine eigene Beschreibung in Form von WSDL schicken. Für die Aushandlung der dynamischen Kommunikationsparameter wie z.B. Übertragungsprotokoll, Verschlüsselung, Signaturart usw. wird WS-Policy eingesetzt. Die automatische Gerätesuche und das Pairing ermöglicht WS-Discovery. Mit diesen Mitteln können die Geräte ohne menschliches Eingreifen voll automatisiert interagieren. Damit ein Nutzer ohne technisches Wissen die Geräte konfigurieren kann, soll die Konfiguration möglichst einfach realisiert werden. Im folgenden Kapitel wird die Möglichkeit der Gerätekonfiguration mittels nur einer Taste vorgestellt.

3.7.1. Gerätekopplung

Geräte, die über eine Anzeige bzw. Eingabemöglichkeiten (z.B. Tastenfeld) verfügen, können zusätzlich mittels Interaktion mit dem Nutzer konfiguriert werden. Die einfacheren Geräte sollen zumindest über eine Taste oder eine ähnliche Eingabemöglichkeit verfügen. Beim Betätigen der Taste werden eine Probe- und eine Hello-Nachricht geschickt. Die Probe-Nachricht enthält alle gesuchten Services/Geräte. Die Hello-Nachricht gibt eigene angebotene Services bekannt. Angenommen, eine Klimaanlage (als Client) sucht einen Temperatursensor

und bietet gleichzeitig den Klimaanlagenservice (als Service Provider) an, dann wird die Probe-Nachricht mit dem folgenden Inhalt geschickt:

```
<soap:Body>
    <wsd:Probe>
        <wsd:Types>sh:TemperatureSensor</wsd:Types>
    </wsd:Probe>
</soap:Body>
```

Die Hello-Nachricht enthält dementsprechend den eigenen Klimaanlagenservice:

```
<soap:Body>
    <wsd:Hello>
        <wsd:Types>sh:AirConditioner</wsd:Types>
        …
```

WS-Discovery verwendet SOAP-over-UDP-Binding und wird an eine Multicast-Adresse geschickt. Jedes Gerät, das diese Anfrage bekommt, vergleicht die gesuchten Services mit den von ihm angebotenen Services. Stimmen diese überein, wird es vermerkt. Damit der Service Provider (Temperatursensor) eine Antwort schicken darf, muss ebenfalls eine Taste auf diesem Gerät betätigt werden. Ist das der Fall, verschickt es eine ProbeMatch-Nachricht an den Sender (Klimaanlage) zurück. Wird die Taste nicht gedrückt, muss der Service Provider (Temperatursensor) die Probe-Nachricht nach Ablauf des Timeouts verwerfen.

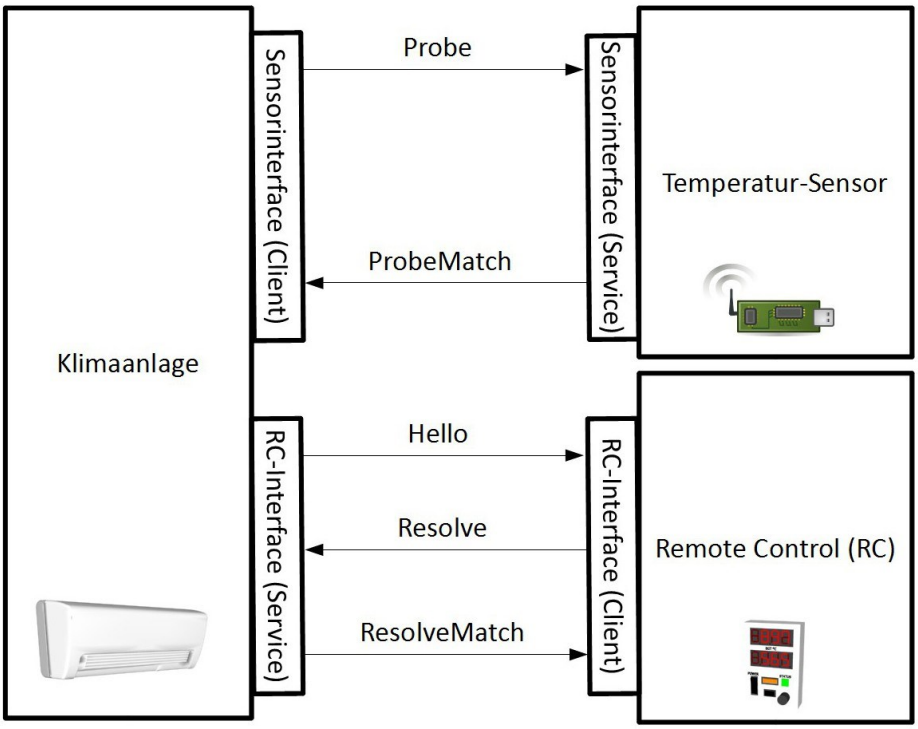

Abbildung 42: Geräte-Pairing mit DPWS

Der Klimaanlagenservice (als Service Provider) kann von einer Fernbedienung (als Client) benutzt werden. Ein ähnlicher Ablauf findet auf der Seite der Fernbedienung statt. Wird eine Hello-Nachricht empfangen, die einen gesuchten Service enthält, wird dieser ebenfalls vermerkt. Um die Geräte zu koppeln, muss auf der Fernbedienung ebenfalls eine Taste gedrückt werden. Dabei wird eine Resolve-Nachricht geschickt, die vom jeweiligen Service Provider (Klimaanlage) mit ResolveMatch-Nachricht beantwortet wird (vgl. Abbildung 42). Die ResolveMatch-Nachricht entspricht der ProbeMatch-Nachricht und trägt die Transportadresse des Gerätes (Xaddrs). Mithilfe der Transportadresse können direkte Nachrichten an den Service Provider geschickt werden. Im nächsten Schritt fordert der Client die Metadaten und WSDL vom Service Provider. Die Metadaten enthalten Informationen über den Hersteller sowie allgemeine Geräte- und Softwareinformationen. WSDL enthält die Service-Beschreibung, d.h. welche Funktionen zur Verfügung stehen und wie diese aufgerufen werden müssen (vgl. Abschnitt 2.5.2.10). Damit jedes Gerät mit den anderen WS-Systemen und -Implementierungen kompatibel ist, sollte für den Metadaten- und WSDL-Transport das standardmäßige SOAP-over-HTTP-Binding genutzt werden. Nach dem erfolgreichen Nachrichtenaustausch sind die Geräte gekoppelt. Ein Gerät kann für verschiedene Nachrichtentypen mehrere Bindings unterstützen. Angenommen, wenn der Temperatursensor das standardmäßige SOAP-over-HTTP-Binding und das vorgestellte SOAP-over-CoAP-Binding mit EXI-Codierung unterstützt, dann werden die beiden Bindings folgendermaßen durch das WSDL beschrieben:

```
<wsdl:binding name="SensorHTTPBinding"
    type="tns:TemperatureSensorInterface">
    <wsoap12:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="CurrentTemperature">
        <wsoap12:operation soapAction="" />
        <wsdl:input>
            <wsoap12:body use="literal" />
        </wsdl:input>
        …

<wsdl:binding name="SensorCoAPBinding"
    type="tns:TemperatureSensorInterface">
    <wsoap12:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/coap" />
    <wsdl:operation name="CurrentTemperature">
        <wsoap12:operation soapAction="" />
        <wsdl:input>
            <wsoap12:body use="literal"
                encodingStyle="http://www.w3.org/TR/exi/"/>
        </wsdl:input>
        …
```

Daraufhin kann die Klimaanlage das von ihr unterstützte Binding wählen und dieses bei allen Nachrichten an den Temperatursensor verwenden. Zusätzliche dynamische Parameter wie z.B. EXI Modes können mithilfe von WS-Policy angegeben werden.

Falls bestimmte Geräte sichere Services anbieten, die Verschlüsselung und Autorisierung erfordern, müssen Zertifikate vor dem Beginn der Kommunikation ausgetauscht werden. Der Zertifikataustausch muss aus Sicherheitsgründen „out of band“ stattfinden, d.h., es muss dafür ein anderer Übertragungskanal benutzt werden. Für diesen Zweck bietet sich die Near Field Communication (NFC) an. Dabei müssen die Geräte sehr dicht aneinander gehalten werden (wenige Zentimeter), sodass ein Abhören nahezu unmöglich ist.

Um die Kopplung von Geräten verschiedener Hersteller immer gewährleisten zu können, müssen Regeln für die Namensgebung der Geräte definiert werden. Hierbei wäre eine offene einheitliche Datenbank denkbar.

3.8. Zusammenfassung

Für die Bestätigung der Abbildbarkeit anderer Protokolle auf DPWS wurde zunächst ein DPWS-Profil für das Smart Metering entwickelt. Hierfür wurde der Stand der Technik untersucht. Ausgehend davon wurden die Anforderungen an das Profil abgeleitet. Es wurde ein Smart Metering-Profil, basierend auf dem SML-Protokoll unter Berücksichtigung der BSI-Richtlinie, entwickelt. Damit DPWS auf Medien mit niedrigen Datenraten eingesetzt werden kann, wurde eine geeignete Kompression mittels CoAP und EXI angewendet. Dadurch konnte der Kommunikationsoverhead der Web Services um 90 % reduziert werden. Im Rahmen dieser Arbeit wurde gezeigt, dass DPWS die Anforderungen an Smart Metering-Protokolle vollständig erfüllen kann. Die erreichte Performance ist mindestens so gut wie bei den spezialisierten Smart Metering-Protokollen bei gleichzeitig bleibendem dynamischen Charakter und Anpassbarkeit der Web Services. Des Weiteren wurde eine analytische Betrachtung herangezogen, die die Allgemeingültigkeit des vorgeschlagenen Ansatzes bestätigt. Damit wurde gezeigt, dass jedes beliebige Protokoll der Gebäudeautomation, des Smart Metering und des Smart Home auf DPWS abgebildet werden kann. Im Bereich der Gebäudeautomation werden darüber hinaus die Plug&Play-Techniken eingesetzt, um die Geräte voll automatisiert zu konfigurieren. Es wurde gezeigt, wie eine Konfiguration der Geräte mittels nur einer Taste ablaufen kann. Es wird dabei kein technisches Wissen vorausgesetzt, was eine Ausbreitung der Web Service-Technologie im Smart Home deutlich erleichtern kann.

4. Hardware-basierter Parser für Efficient XML Interchange

4.1. Motivation

DPWS oder Embedded Web Services nutzen XML für Datenrepräsentation und HTTP für Datentransport. Ein spezieller Parser ist notwendig, um die Nutzdaten aus einem XML-Dokument zu extrahieren. XML-Parser sind für nahezu alle Programmiersprachen verfügbar, da XML ein weitverbreitetes Datenaustauschformat ist. Das Parsen von XML-Dokumenten kann grundsätzlich auf String-Parsen zurückgeführt werden, was „softwarefreundlich“ ist.

Der größte Nachteil von XML ist die Dokumentgröße. Abhängig von der Dokumentstruktur kann der Overhead ein Vielfaches der Nutzdaten ausmachen. In bestimmten Automationsszenarien, in denen Echtzeitsysteme oder Deeply Embedded Devices eingesetzt werden, kann dieser Umstand kritisch werden. Um von Web Service-Technologien in solchen Systemen zu profitieren, wurde EXI entwickelt [144].

Im Gegensatz zu den allgemeinen Kompressionsmethoden nutzt EXI Kenntnisse über das XML-Format und ggf. über die XML-Struktur des jeweiligen Dokuments aus. Dies ermöglicht sehr hohe Kompressionsraten von bis zu 90 % (vgl. Abschnitt 3.5.2). EXI stellt XML-Tags als eine Reihenfolge von Events dar. Events können mit nur wenigen Bits (abhängig von EXI-Modus) kodiert werden. Eine weitere Eigenschaft, die EXI von anderen allgemeinen Kompressionsmethoden unterscheidet ist, dass die Daten direkt aus dem Dokument herausgelesen oder in das Dokument hineingeschrieben werden können ohne vorherige Dekomprimierung bzw. spätere Komprimierung. Folglich werden keine zusätzlichen CPU- und Speicherressourcen für die Berechnung verbraucht.

Um die Daten dennoch aus einem EXI-Dokument zu extrahieren, ist ein geeigneter Parser unentbehrlich. Mehrere Software-Parser wurden bereits vorgestellt. Im Rahmen dieses Kapitels wird jedoch der Fokus auf sehr schnelle Datenverarbeitung (für z.B. Echtzeitsysteme) gelegt. In solchen Szenarien sind Verarbeitungsgeschwindigkeiten von einigen Mikrosekunden erforderlich [146]. In diesem Kapitel wird die Möglichkeit der Implementierung eines EXI-Parsers in Hardware für sehr hohe Verarbeitungsgeschwindigkeiten untersucht. Der vorgeschlagene Ansatz kann darüber hinaus als Digital Signal Processor (DSP) bzw. Coprozessor in weiteren Szenarien verwendet werden, um die Haupt-CPU zu entlasten. Der Fokus dieser Arbeit liegt weiterhin auf einer dynamischen Charakteristik des EXI-Parsers, damit dieser leicht an Anwendungen angepasst werden kann, sowie auf der Verwendung von standardisierten Interface-Bussen, um leichte Integration in andere System-on-Chip (SoC)-Lösungen zu ermöglichen.

Zusammenfassend ist der Hauptbeitrag dieses Kapitels wie folgt [2]:

- Untersuchung der Anforderungen für die Realisierung des Hardware EXI-Parsers in Hardware
- Konzept und Implementierung des Hardware EXI-Parsers
- Evaluierung des Hardware/Software Co-Designs
- Vergleich mit existierenden Lösungen

4.2. Stand der Technik

Die aktuelle finale Version von EXI 1.0 wurde im Jahr 2011 vom World Wide Web Consortium (W3C) verabschiedet [144]. Ein Vergleich der Kompressionsraten zwischen EXI und GNU zip (gzip [142]) ist in [147] aufgeführt. Die Auswertung hat gezeigt, dass EXI viel kleinere Streams als gzip oder auch andere Kompressionsmethoden erzeugt [148]. Der Vorteil von EXI in Zusammenarbeit mit DPWS und CoAP wurde bereits in Abschnitt 3.5.2 gezeigt. Da EXI eine relativ neue Kompressionsmethode ist, sind nur einige Umsetzungen bekannt. Die funktionsreichste EXI-Implementierung ist EXIficient [149]. Diese ist in Java programmiert und kann EXI-Dokumente aus XML erstellen und umgekehrt. Viele EXI-Optionen werden dabei unterstützt. Eine weitere Implementierung ist OpenEXI [150]. Diese beiden Implementierungen erfordern eine leistungsstarke Hardware und ein Betriebssystem, das Java Virtual Machine unterstützt. Die zwei anderen Umsetzungen uEXI (entwickelt am Institut MD der Universität Rostock) [151] und EXIP [152] sind in der Programmiersprache C geschrieben und für den Einsatz in eingebetteten Systemen gedacht. Die Verarbeitungsgeschwindigkeit hängt dabei von der jeweiligen Hardware ab und kann stark variieren. Um die in dieser Arbeit vorgeschlagene hardwarebasierte Lösung zu evaluieren, wird die Performance mit den softwarebasierten Lösungen unter Verwendung eines Zynq Z-7020 Chips verglichen, der aus einem Dual Core ARM Cortex-A9 und einem Artix 7 Field Programmable Gate Array (FPGA) besteht. Darüber hinaus wird ein Hardware/Software-Prototyp ebenfalls evaluiert.

4.3. Realisierung des EXI-Parsers in Hardware

Ziel des EXI-Parsers ist, die Nutzdaten aus dem EXI-Stream zu extrahieren. Einerseits soll der Parser generisch sein, damit er für jede beliebige Anwendung und XML-Schema benutzt werden kann. Andererseits führt ein zur Laufzeit generischer Parser zu einem sehr hohen Hardware-Ressourcenverbrauch. Um die effizienteste Implementierung zu erreichen sowie die Hardware-Ressourcen zu sparen, wird somit die Entwicklung eines zur Synthesezeit generischen Parsers vorgeschlagen. Um die Hardware zu synthetisieren, wird Very High Speed Integrated Circuit Hardware Description Language (VHDL) verwendet. Die VHDL-Beschreibung des EXI-Parsers muss in Abhängigkeit von XML-Schema dynamisch erstellt

werden. XML-Schema kann vom Gerätehersteller zur Verfügung gestellt oder in einer Spezifikation für einen bestimmten Service-Typ definiert werden.

Für die Code-Generierung und das Parsen von XML-Schema wird die Open Source Software EXIficient verwendet. EXIficient ist eine Java-Implementierung der W3C EXI-Spezifikation. EXIficient Schema Parser und Grammar Builder werden für die Vereinfachung der Code-Generierung benutzt. EXIficient wurde mit zusätzlichen Klassen erweitert, die die interne Darstellung der EXI-Struktur verwenden, um den VHDL-Code zu erzeugen. Der erstellte VHDL-Code wird anschließend für die Hardware-Synthese benutzt. Die Schritte für die Erzeugung des EXI-Parsers sind in Abbildung 43 dargestellt.

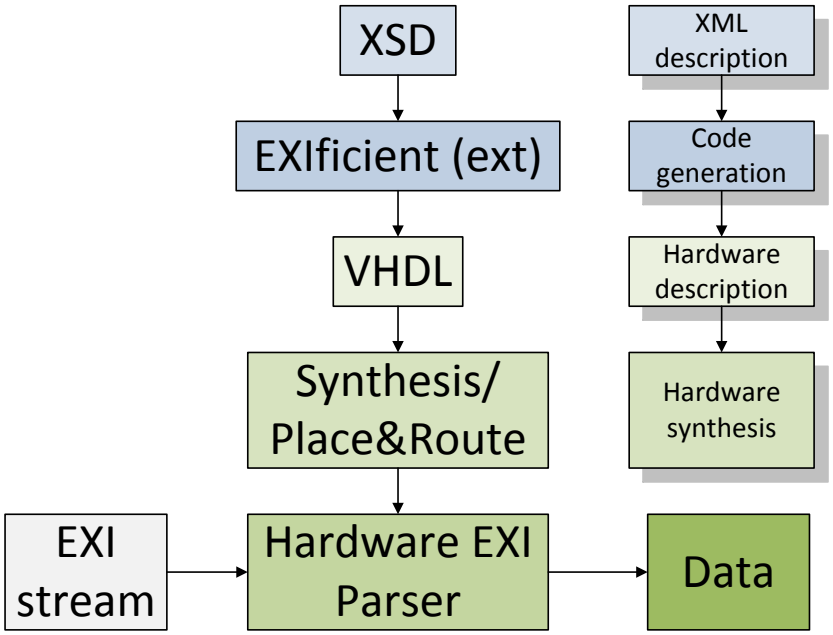

Abbildung 43: Generierung von Hardware EXI-Parser

4.3.1. Hardware-Beschreibung

Für ein besseres Verständnis der VHDL-Code-Generierung wird zunächst die angestrebte finale VHDL-Struktur unmittelbar vor der Synthese erläutert. Wie bereits erwähnt, nutzt EXI Events für die Abbildung der XML-Struktur. Die Reihenfolge der Events kann durch einen endlichen Automaten (Zustandsmaschine) beschrieben werden. Die Zustandsmaschinen können sehr leicht in Hardware implementiert werden, was EXI hardwaregeeigneter als XML macht. Abhängig vom aktuellen Event-Code muss die Zustandsmaschine in den entsprechenden Zustand wechseln. Für ein besseres Verständnis wird das Vorgehen anhand eines vereinfachten Beispiels erklärt (siehe Abbildung 44). Ein XML-Code-Ausschnitt enthält ein Oberelement und 4 Unterelemente. Nachdem das Oberelement erreicht wurde, können vier verschiede Events (Elemente) auftreten. Nur 2 Bits sind in diesem Fall notwendig, um die Events für die Unterelemente zu kodieren. Element 0 kann hierbei mit „00“ und Element 3 mit „11“ kodiert werden.

Abhängig von der Bitabfolge im EXI-Stream, wird die Zustandsmaschine in den jeweiligen Zustand springen. Auf diese Weise kann der EXI-Parser das Dokument verarbeiten. Die Zustände werden entsprechend dem XML-Schema erzeugt und dienen dem Parsen der Struktur. Zusätzlich zum Parsen der Struktur ist das Parsen der Daten ein weiterer wichtiger Bestandteil des EXI-Parsers. Wie oben erwähnt, werden die Daten mit EXI entsprechend dem Datentyp kodiert. Ein Integer-Wert wird beispielsweise als eine Zusammensetzung aus einem Boolean- und einem Unsigned Integer-Wert dargestellt. Der Boolean-Teil wird mit nur einem Bit kodiert und bildet das Vorzeichen des Integer-Wertes ab. Der Unsigned Interwert-Wert wird als eine Sequenz der Integer-Bytes kodiert und kann beliebig lang sein. Das Most Significant Bit (MSB) jedes Integer-Bytes zeigt die Präsenz eines weiteren Bytes an. Auf diese Weise kann die Sequenz terminiert werden. Um den gesamten Integer-Wert wiederherzustellen, müssen die 7 Least Significant Bits (LSBs) der Integer-Bytes in Little Endian-Reihenfolge zusammengesetzt werden.

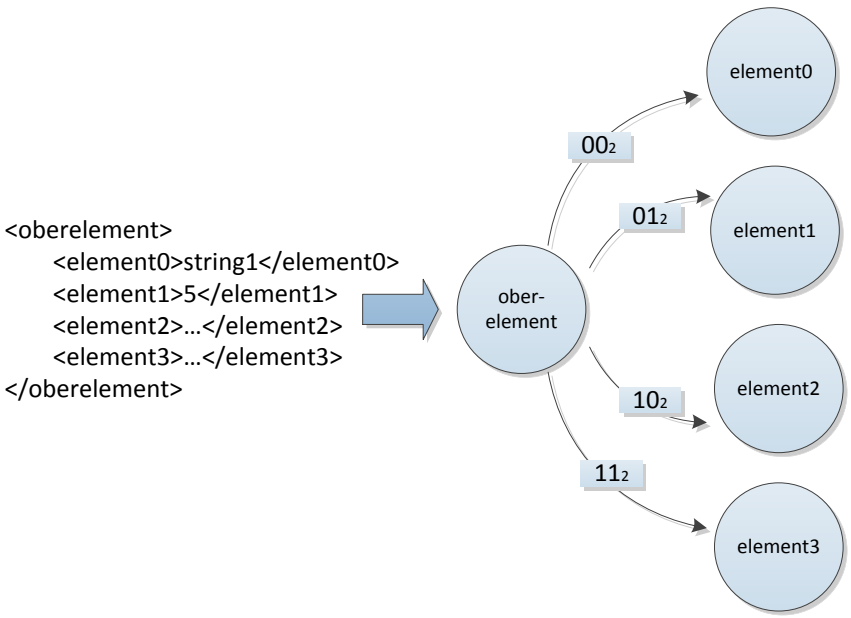

Abbildung 44: Parsen der EXI-Struktur

Wie erwähnt, wird jeder String durch EXI nur einmal kodiert. Die Strings werden den QNames zugeordnet. Ein QName besteht aus einem Uniform Resource Identifier (URI) und einem LocalName. Die QNames stellen XML-Tags dar. Um String-Parsen realisieren zu können, muss der Parser mehrere String-Tabellen führen. Es werden eine globale String-Tabelle und eine lokale String-Tabelle pro QName benötigt. Ein sogenannter Local Hit bedeutet, dass der gleiche String bereits für den jeweiligen QName kodiert wurde. Sein Index muss dann in der lokalen Tabelle nachgeschlagen werden. Ein Global Hit bedeutet, dass der gleiche String bereits für einen anderen QName kodiert wurde und dementsprechend soll der Index in der globalen String-Tabelle nachgeschlagen werden. Wenn ein String weder Local noch Global Hit liefert, wird er als ein neuer String betrachtet und in die beiden Tabellen eingetragen. Die maximalen Indizes der beiden Tabellen werden dann inkrementiert.

Die Anzahl der Bits für die Darstellung des Index ist nicht statisch, sondern wird abhängig von der Anzahl der Einträge in der Tabelle variiert.

Um Hardware-Ressourcen zu sparen, wird das Datenparsen nicht fest in jeden Pfad der Zustandsmaschine einprogrammiert, sondern in die speziellen Zustände für das Parsen der Daten ausgelagert. Auf diese Weise existieren die Hardware-Ressourcen für jeden Datentyp nur einmal. Der Logikaufwand für die Zustandsablaufsteuerung steigt jedoch. Der nächste Zustand muss in diesem Fall gemerkt und nach dem Datenparsen wiederhergestellt werden. Für jeden Datentyp wird eine eigene Parsen-Routine erzeugt.

Die Parsen-Routine ist in Abbildung 45 dargestellt. SE bezeichnet dabei das *Start Element* und EE das *End Element* Event. Die Elementwerte werden durch CH (*Characters*) beschrieben, da alle Werte bei XML durch Strings dargestellt werden. Der entwickelte EXI-Parser kann ebenfalls mit den Attributen und deren Werten umgehen. Diese werden aus Gründen der Übersichtlichkeit jedoch nicht im Beispiel aufgeführt.

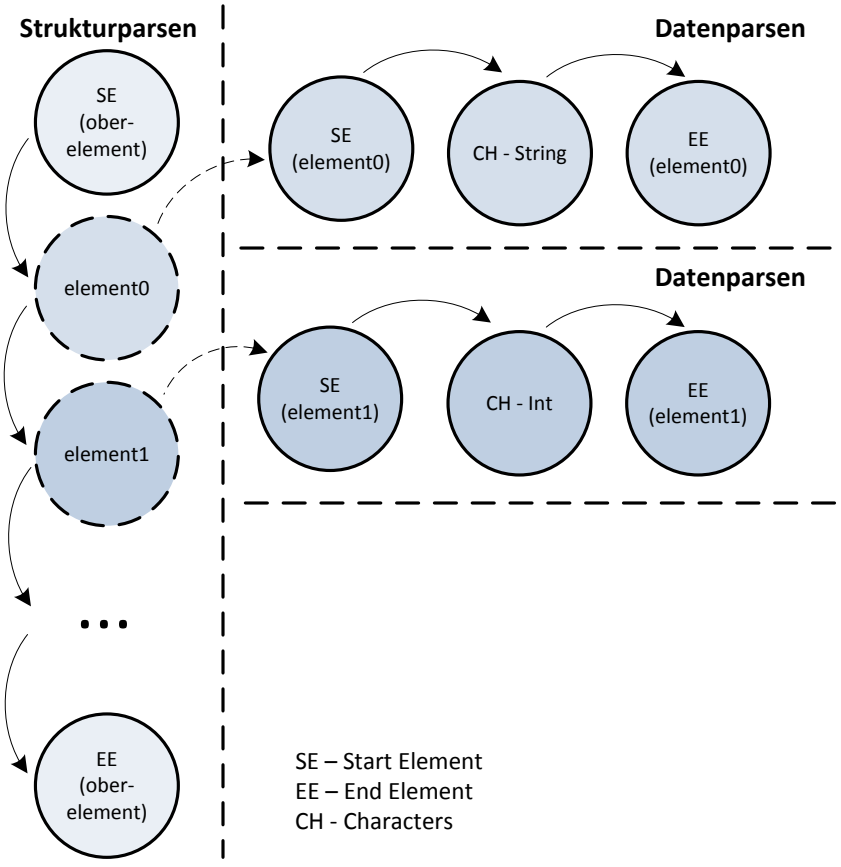

Abbildung 45: Parsen der Daten

Um mit den anderen Modulen kommunizieren zu können, benötigt der EXI-Parser Interfaces. Die Interfaces des Parser-Moduls sind in Abbildung 46 dargestellt. Die Erklärung der einzelnen Ein- und Ausgänge kann der Tabelle 13 entnommen werden. Neben den Clock- (clk) und Reset-Pins (rst) verfügt der EXI-Parser über einen 32 Bit Input-Bus (input), Input Valid (input_valid) und Read Out (read_out) Signale. Auf Input-Bus soll der EXI-Stream

geteilt in 32 Bit-Worte angelegt werden. Das erste 32 Bit-Wort muss die Länge des Streams angeben.

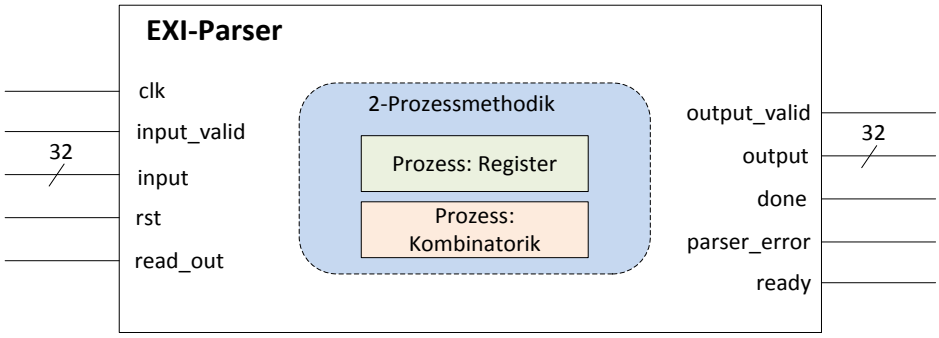

Abbildung 46: Interfaces des Hardware EXI Parsers

Als Ausgangssignale dienen ein 32 Bit Output-Bus (output), Output Valid (output_valid), Ready (ready), Done (done) und Error (parser_error) Flags. Wenn das Modul als Ready gekennzeichnet ist, können Daten auf den Input-Bus gelegt und durch das Setzen des Input-Valid-Flags für das EXI-Modul freigegeben werden. Das Done-Flag signalisiert die erfolgreich abgeschlossene Verarbeitung eines EXI-Streams. Das Error-Flag wird dagegen bei einem Verarbeitungsfehler gesetzt. Verarbeitungsfehler treten bei EXI-Formatfehlern auf. Um die extrahierten Nutzdaten herauszulesen, muss das Read Out Flag gesetzt werden. Die Nutzdaten werden über den Output-Bus ausgegeben. Jedem Datum werden dabei zusätzliche Metadaten vorangestellt. Die Daten auf dem Output-Bus sind gültig, solange das Output Valid Flag gesetzt ist. Die Nutzdaten werden durch IDs gekennzeichnet, sodass andere Module diese zuordnen können.

Signale	Bitbreite	Beschreibung
clk	1	Clock-Eingang
rst	1	Reset-Eingang
input	32	Eingangssignal für das Einlesen des EXI-Streams
input_valid	1	Signalisiert neue Daten am Input
read_out	1	Fordert die Daten an
ready	1	Signalisiert die Bereitschaft des Moduls
output_valid	1	Signalisiert neue Daten am Output
output	32	Ausgangssignal für das Ausgeben der verarbeiteten Daten
done	1	Signalisiert abgeschlossene Verarbeitung
parser_error	1	Signalisiert einen Verarbeitungsfehler

Tabelle 13: Beschreibung der Interfaces des Hardware-EXI-Parsers

Zusätzlich zu den Nutzdaten werden Metadaten gespeichert. Diese sollen die weitere Verarbeitung der Nutzdaten durch andere Module erleichtern. Die Metadaten werden durch ein 32 Bit Wort dargestellt. Sie besitzen den in Abbildung 47 gezeigten Aufbau. Die

Datentyp-ID gibt Aufschluss über den aktuellen Datentyp. Die zwei weiteren Felder: LocalName-ID und URI-ID bilden den QName und beschreiben damit den XML-Tag, zu dem das Datum gehört. Weitere Informationen sind datenspezifisch und können z.B. bei einem Integer-Wert dessen Länge angeben. Alle IDs werden während der Generierung des EXI-Parsers erzeugt.

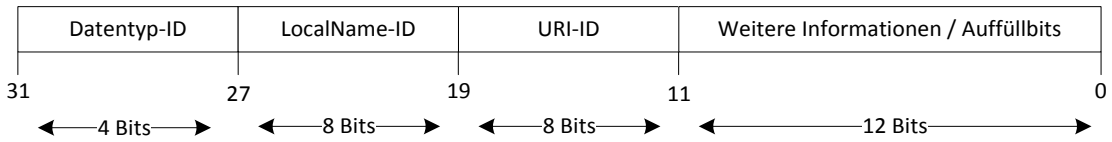

Abbildung 47: Metadaten der Nutzdaten

Der gesamte Hardware-EXI-Parser wird in 2-Prozessmethodik erzeugt, was das Verständnis des Quellcodes für den Anwender sowie das Debugging erleichtert. In der prototypischen Implementierung werden die eingehenden Daten in einem First In First Out (FIFO)-Speicher gespeichert, welcher aus Registern aufgebaut wird. Nach dem Reset befindet sich das System in einem Idle-Zustand. Dieser Zustand wird intern ebenfalls für das Zurücksetzen der verwendeten Register und das Löschen des Speichers nach einem abgearbeiteten EXI-Stream verwendet. Die Zustandsmaschine geht dann in den Initialisierungszustand über und wartet auf eingehende Daten. Wenn das Input Valid Flag gesetzt wird, schreibt das System den eingehenden EXI-Stream zunächst in den obengenannten FIFO-Speicher. Nachdem der EXI-Stream komplett übernommen wurde, wechselt die Zustandsmaschine in den Zustand für die Verarbeitung des EXI-Headers. Hier werden EXI-Unterscheidungsbits (distinguishing bits DB=„10“) bzw. EXI-Options-Flag (presence bit for EXI options (PBEO)) geprüft. EXI-Unterscheidungsbits ermöglichen, einen EXI-Stream eindeutig von einem XML-Stream zu unterscheiden, und sind ein fester Bestandteil des EXI-Headers. EXI-Options-Flag deutet auf das Vorhandsein der EXI-Optionen hin. Da die prototypische Implementierung keine EXI-Optionen unterstützt, muss das Bit auf 0 gesetzt sein. Wenn die Prüfung des EXI-Headers erfolgreich war, wechselt die Zustandsmaschine zur Verarbeitung des EXI-Bodys (vgl. Abbildung 44, Abbildung 45).

Die Verarbeitung des EXI-Streams läuft wie oben beschrieben ab. Wenn ein Fehler im EXI-Header oder -Body festgestellt wird, wird die weitere Verarbeitung abgebrochen. Ein Verarbeitungsfehler wird in diesem Fall gemeldet und die Zustandsmaschine wechselt wieder in den Initialisierungszustand. Wenn der Endzustand erreicht wurde, war die Verarbeitung erfolgreich und das Done-Flag wird gesetzt. Für einen besseren Überblick wird der beschriebene Ablauf in Abbildung 48 dargestellt.

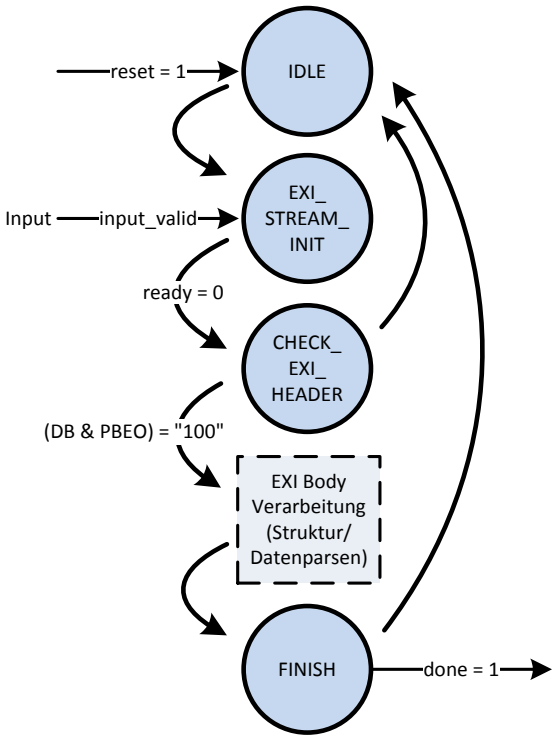

Abbildung 48: Zustandsmaschine des Hardware-EXI-Parsers

Um den nächsten Zustand während des Datenparsens zu speichern, wird ein Stack-Speicher verwendet. Der Stack-Speicher wird mithilfe von Registern aufgebaut. Bevor die Zustandsmaschine in einen Zustand für das Datenparsen wechselt, wird der nächste Strukturzustand auf den Stack gelegt. Danach wird der aktuelle Datentyp verarbeitet. Nachdem die Verarbeitung abgeschlossen ist, wird der nächste Zustand wieder vom Stack geladen und die Zustandsmaschine wechselt in diesen Zustand. Die gewonnenen Nutzdaten müssen innerhalb des Parsers zwischengespeichert werden. Erst nach einer vollständigen Abarbeitung des EXI-Streams können die Nutzdaten aus dem EXI-Parser ausgelesen werden. Für die Zwischenspeicherung wird ein FIFO-Speicher für jeden Datentyp erstellt. Neben den Nutzdaten werden zusätzliche Metadaten gespeichert. Diese Metadaten können später von anderen Modulen verwendet werden, um die Nutzdaten den jeweiligen Elementen zuzuordnen.

Das Auslesen der Daten kann von einem an den EXI-Parser angeschlossenen Modul durch das Setzen des Read Out-Flags ausgelöst werden. Im ersten Zustand der Datenausgabe werden die Metainformationen ausgewertet. Abhängig vom Datentyp können einige Konvertierungen vorgenommen werden. Im zweiten Zustand werden die Metadaten als 32 Bit Wort an den Ausgang gelegt. Dabei wird das Output Valid-Flag gesetzt. Im nächsten Zustand werden die Daten aus dem FIFO-Speicher ausgelesen und an den Ausgang weitergeleitet. Diese Prozedur wird solange wiederholt, bis der FIFO-Speicher leer ist. Anschließend wird es mit dem nächsten Datentyp fortgesetzt. Alle Ausgabezustände beginnen den Datentransport mit Metadaten bestehend aus einem 32 Bit Wort. Die Metadaten beinhalten den Datentyp, die LocalName ID und die URI ID (vgl. Abbildung 47). Mithilfe dieser Informationen können andere Module die Nutzdaten den XML-Elementen korrekt zuweisen. Die Form der

Datendarstellung nach den Metadaten hängt vom jeweiligen Datentyp ab. Die Ausgabe eines Datentyps kann mehrere Lesezyklen benötigen, z.B. im Falle von Strings.

4.3.2. Code-Generierung

Wie oben erwähnt, wird der Hardware-EXI-Parser zur Synthesezeit erzeugt. Die Generierung des VHDL-Codes kann in zwei Teile unterteilt werden: die statische und dynamische Hardware-Beschreibung. Die statische Hardware-Beschreibung beinhaltet Entity-, Signal-, Komponenten-, Prozess- und Variablen-Deklarationen und ist von XSD unabhängig. Darüber hinaus bietet die statische Beschreibung Zustände für Datentypenverarbeitung. Der dynamische Teil der Beschreibung setzt sich vor allem aus den Zuständen für Strukturparsen zusammen. Das Ergebnis der Code-Generierung ist eine VHDL-Datei *exi_parser.vhd*.

Der vorgeschlagene Code-Generator nutzt EXIficient-Klassen für die Erzeugung von EXI-Grammar. EXIficient unterteilt die Events in Fragmente von Elementgruppen. Ein Fragment stellt die Beschreibung eines Elementes dar. Die Elemente werden dabei alphabetisch nach LocalNames geordnet, wie in Abbildung 49 dargestellt. Diese Struktur wird vom entwickelten Code-Generator in ein Array der Klasse *State* umgewandelt, das für die Generierung der Strukturzustände verwendet wird. Diese Klasse besteht aus folgenden Feldern: Event-Name, Event-Typ, Attributname, Datentyp, Namespace URI, URI ID, Anzahl der Verzweigungen und Array der Verzweigungen. Alle Events, die zu einem Fragment gehören werden in das *State* Array eingetragen. Mithilfe eines Arrays aus Fragmenten kann die Struktur eines XML-Dokuments vollständig abgebildet werden. Jedes Element beginnt mit dem Event-Typ *Start Element* (SE). Alle zu diesem Event zugehörigen Informationen werden in das erste Feld des *States* Arrays eingetragen. Nach dem SE können beliebig viele weitere SE, *Attribute* (AT) oder *Characters* (CH) folgen. Das *States* Array wird solange aufgefüllt, bis alle Verzweigungen eines Fragments durchlaufen sind. Bevor ein neuer Eintrag erstellt wird, prüft der Generator, ob derselbe Eintrag bereits eingetragen wurde. In diesem Fall wird kein neuer Eintrag erstellt, sondern es wird der Index des existierenden Eintrages in das Array der Verzweigungen des letzten Events eingetragen. Die Verarbeitung eines Fragments ist abgeschlossen, wenn der Event-Typ *End Element* (EE) erreicht wurde.

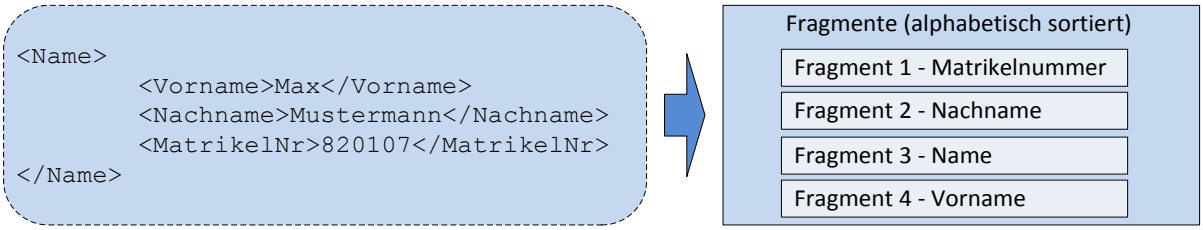

Abbildung 49: EXI-Fragment

Nachdem alle Events eines Fragments verarbeitet wurden, wird eine Funktion zur Code-Generierung aufgerufen. Die Funktion erzeugt Zustandsnamen und Zustandsübergänge für die Zustandsmaschine. Für diesen Zweck werden alle Einträge des *State* Array durchlaufen. Für den ersten Eintrag im *State* Array wird ein neuer Zustand generiert. Alle anderen Einträge stellen Zustandswechsel dar. Bevor der nächste Zustand erstellt wird, prüft der Code-Generator den Event-Typ. Wenn es sich um einen Event vom Typ CH oder AT mit einem

beliebigen Datentyp handelt, wird der VHDL-Code für den Wechsel in den entsprechenden Zustand für die Verarbeitung des Datentyps generiert. Darüber hinaus muss der nächste Zustand abgespeichert werden, um zum Strukturparsen zurückkehren zu können. Zusätzlich werden dazugehörige URI ID und LocalName ID für die Datenklassifizierung gespeichert. Wenn es sich beim nächsten Zustand nicht um CH oder AT handelt, wird ein Zustandswechsel hinzugefügt. Wenn der Event-Typ EE erreicht wurde, ist die Code-Generierung für das aktuelle Fragment abgeschlossen. Anschließend beginnt die Verarbeitung des nächsten Fragments. Die beschriebene Vorgehensweise wird in Tabelle 14 vereinfacht dargestellt.

State Array Index	Fragment 1	Fragment 2	Fragment 3
0	SE – element0	SE – element1	SE – oberelement
1	CH (String)	CH (Integer)	SE – element0
2	EE – element0	EE – element1	SE – element1
3	-	-	SE – element2
4	-	-	SE – element3
5	-	-	EE – oberelement

Tabelle 14: Erzeugung der Zustände anhand der Elementfragmente

Für die Generierung der Zustände für globale Elemente ist deren Anzahl von Bedeutung. Sie bestimmt die Anzahl von Bits vom EXI-Stream, die im ersten Schritt beim Parsen verarbeitet werden müssen. In jedem weiteren Schritt überprüft der Code-Generator die Anzahl von Verzweigungen. Hierfür wird der entsprechende Wert der Klasse *State* ausgewertet. Wenn der Wert größer als 1 ist, können unterschiedliche Zustandsübergänge folgen. Um den nächsten Zustand zu bestimmen, muss eine richtige Anzahl von Bits vom EXI-Stream eingelesen werden. Dies hängt von der Anzahl der Verzweigungen für das bestimmte Fragment ab und kann aus dem gleichnamigen Wert der *State*-Klasse ermittelt werden.

4.4. Hardware/Software Co-Design

In eingebetteten Systemen übernimmt die Hardware oft zeitkritische oder rechenintensive Aufgaben. Die verarbeiteten Daten werden dann zur Software-Komponente weitergeleitet. Das Ziel dieser Arbeit ist ebenfalls die Evaluierung einer solchen Möglichkeit, wobei der entwickelte Hardware-EXI-Parser als Teil eines Hardware/Software Co-Designs untersucht wird.

Damit die Verwendung der Software-Komponente leicht zu handhaben ist, wird diese ebenfalls durch den beschriebenen Code-Generator erzeugt. In der prototypischen Implementierung wird für die Software-Komponente die Programmiersprache C++ verwendet. Die Software-Komponente besteht aus den zwei Dateien ExiStreamProcessing.h und ExiStreamProcessing.cpp, wie es bei C++ üblich ist. Diese beiden Dateien werden zusammen mit dem VHDL-Code generiert. Die Benutzung dieser ist jedoch optional und für das Hardware/Software Co-Design bestimmt. Ähnlich wie der Hardware-Teil besitzt der

Software-Teil statisch und dynamisch erzeugte Instruktionen. Bibliotheken-Import oder globale Definitionen sind Beispiele für statisch erzeugte Code-Teile. Für eine eindeutige Zuweisung der Nutzdaten sowie eine leichte Nutzung der Software werden Variablennamen mit den entsprechenden LocalNames der XML-Elemente generiert. Nachdem der EXI-Stream durch die Hardware verarbeitet und an die Software weitergeleitet wurde, können diese Variablen für den Zugriff auf die Nutzdaten genutzt werden.

Eine wichtige Funktion stellt die Evaluierung der Daten dar, die von Hardware geschickt werden. Die Daten werden anhand der LocalName ID und URI ID den Variablen zugewiesen. Diese Funktionalität wird mithilfe einer Lookup-Tabelle realisiert, in der die IDs den entsprechenden Namespaces und LocalNames zugeordnet sind. Darüber hinaus werden Funktionen hinzugefügt, die es ermöglichen, Nutzdaten zu evaluieren und in Variablen des entsprechenden Datentyps zu speichern. Für die Interpretation der Daten werden spezielle Bitmasken definiert. Mit ihrer Hilfe können die Parameter schnell erkannt und die richtigen Variablen zugewiesen werden. Die Funktionen für die Verarbeitung der Datentypen lassen sich statisch erzeugen. Um auf die XML-Nutzdaten zuzugreifen, werden C++-Strukturen benutzt. Der Zugriff auf ein bestimmtes Element erfolgt in folgendem Format: *NamespaceName.LocalName*. Diese beiden Parameter können in XML-Dokument leicht gefunden werden. Die softwareseitige Darstellung der Nutzdaten ist in Abbildung 50 dargestellt.

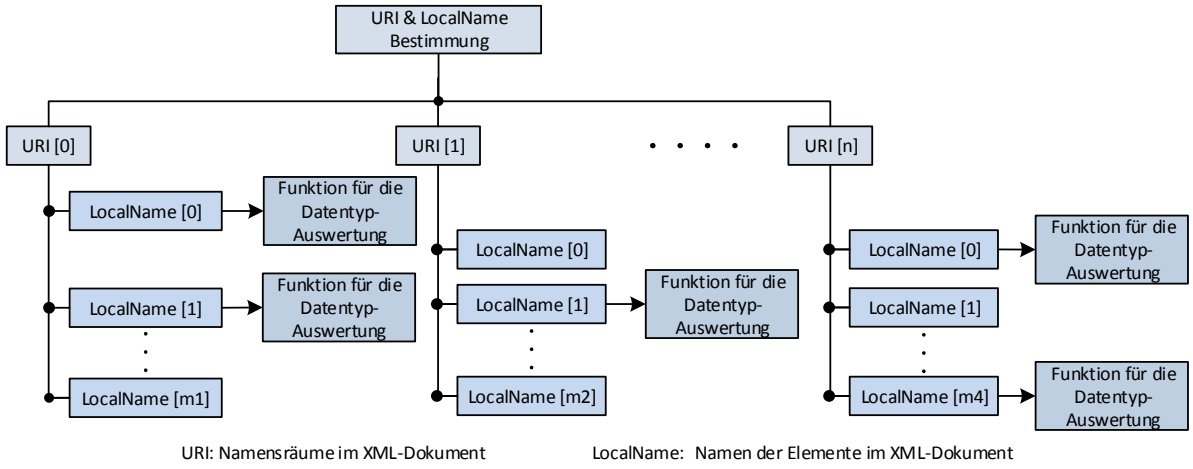

Abbildung 50: URI- und LocalName Abbildung im EXI-Software-Modul

In dem entwickelten Prototyp liest die Software nicht nur die Daten aus dem EXI-Parser, sondern versorgt ihn auch mit dem EXI-Stream. Für eine erleichterte Kommunikation wird Advanced Micro Controller Bus Architecture Advanced eXtensible Interface (AXI) verwendet. Im Rahmen dieser Arbeit wurde AXI4-Lite benutzt. AXI4-Lite ist ein Untertyp der AXI-Schnittstelle, der für die einfache Kommunikation gedacht ist. AXI4-Lite ist ein 32 Bit Bus. Drei zusätzliche 32 Bit Register sind für die Multiple Input Multiple Output (MIMO)-Schnittstelle notwendig. Das Hardware/Software Co-Design ist für ein besseres Verständnis in Abbildung 51 dargestellt. Register 0 und 1 werden für den Input und Output entsprechend verwendet. Register 2 wird für Steuerinformationen wie Output Valid, Ready, Done und Error Flags eingesetzt. Die restlichen Bits sind reserviert. Die Bus-

Zugriffssteuerung wird automatisch durch die AXI-Schnittstelle geregelt, sodass die Module, die an den Bus angeschlossen sind, nur die User Interface-Logik bereitstellen sollen, wie in Abbildung 51 gezeigt. Register der MIMO-Schnittstelle bedienen sich des Prinzips Memory Mapped Input/Output (MMIO). Hierbei können periphere Komponenten über die gewöhnlichen Speicheradressen bedient werden. Die Steuerregister solcher Komponenten werden dann einer bestimmten Speicheradresse zugewiesen. Folglich soll die Software zur Steuerung und Kommunikation mit dem Hardware-EXI-Parser auf dessen Register über die gewöhnlichen Speicheradressen zugreifen.

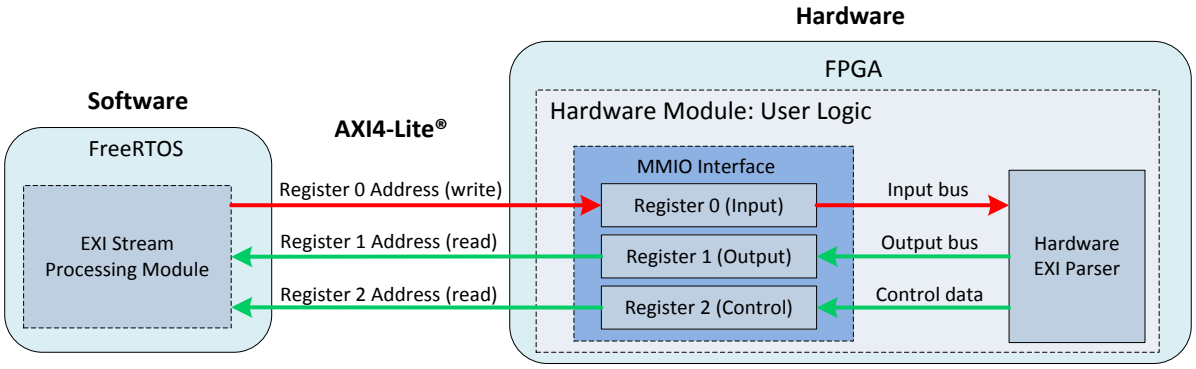

Abbildung 51: Hardware/Software Co-Design

4.5. Evaluierung

Um die vorgeschlagene Hardware-basierte EXI-Lösung zu evaluieren, kam die Evaluationsplattform ZedBoard zum Einsatz [153]. ZedBoard basiert auf einem Zynq Z-7020 Chip, der aus einem Field Programmable Gate Array (FPGA) und einem ARM Cortex A9 CPU mit 667 MHz besteht. Für eine genaue Zeitanalyse sowie zum Vergleich mit anderen Umsetzungen wurde das Echtzeitbetriebssystem FreeRTOS verwendet. Den Ablauf der Tests zeigt Abbildung 52.

Zum Testen des Systems wurden einige ausgewählte SOAP-Nachrichten erstellt. Aus der XML-Datei (A) wird mithilfe eines XSD-Generators (B) die dazugehörige XSD-Datei (C) erstellt. Die XML-Datei (A) und die entsprechende XSD-Datei (C) werden für den nächsten Arbeitsschritt an EXIficient (D) übergeben. Mithilfe von EXIficient (D) werden die Software- (F) und Hardwarekomponente (E) sowie der EXI-Stream (I) generiert, anschließend wird die Softwarekomponente im SDK (G) in das Echtzeitbetriebssystem FreeRTOS eingebunden und die Hardwarekomponente (E) mit dem EDK (H) erzeugt. Nach der erfolgreichen Erstellung werden die Software- und Hardwarekomponente auf das ZedBoard (J) übertragen und ausgeführt. Um die Verarbeitung der Daten durch das Gesamtsystem zu testen, wird der von EXIficient erzeugte EXI-Stream (I) verwendet.

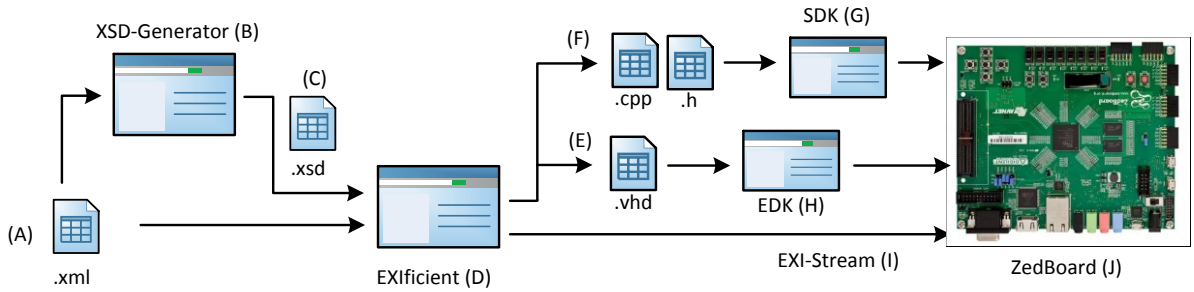

Abbildung 52: Testablauf

4.5.1. Messergebnisse der Hardware-Lösung

Als erstes wird die Verarbeitungsgeschwindigkeit des Hardware-EXI-Parsers als Einzelmodul bestimmt. Hierfür werden Standard Web Service-Nachrichten Hello, Bye, Probe, ProbeMatch, Service Invocation und Invocation Response verwendet. Die XSD-Datei wurde so erstellt, dass alle diese Nachrichten mit einem Parser geparst werden können. Die Messungen wurden mit einer Systemfrequenz von 50 MHz durchgeführt. Die Zeitmessung erfolgt im Simulator ModelSim. Sie beginnt mit der Überprüfung des EXI-Headers und endet mit dem Erreichen des Finish-Zustandes. In Tabelle 15 sind die Verarbeitungszeiten für die EXI-Streams aufgelistet.

EXI-Stream	**EXI-Größe in Byte**	***t* in µs**
request.exi	68	9,68
response.exi	88	10,6
hello.exi	284	19,62
bye.exi	259	18,3
probe.exi	197	15,5
probeMatch.exi	371	23,48

Tabelle 15: Verarbeitungszeiten der EXI-Streams in Hardware

Wie in Abbildung 53 dargestellt, hängt die Verarbeitungszeit eines EXI-Streams linear von dessen Größe ab. Zur Bestimmung der Hardwareauslastung wurde das Design mit der Entwicklungsumgebung Xilinx ISE Design Suite in der Version 14.4 synthetisiert. Der Ressourcenverbrauch nach Place&Route für das Hardware-Design liegt bei 12012 Registern (12 % der Gesamtanzahl) und 12800 Lookup Tables (LUTs) (24 % der Gesamtanzahl). Die maximal mögliche Frequenz liegt bei 57 MHz. Der Ressourcenverbrauch hängt dabei von der Anzahl der möglichen EXI-Events ab. In dem durchgeführten Test kann ein komplettes Web Service-basiertes System auf die Hardware-Lösung abgebildet werden. Alle wichtigen Nachrichten-Typen werden unterstützt und passen auf ein relativ kleines FPGA.

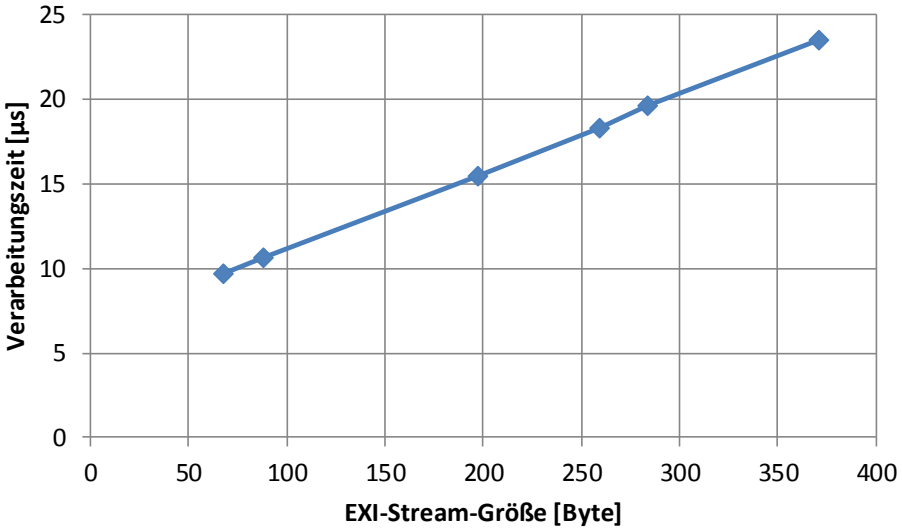

Abbildung 53: Verarbeitungszeit eines EXI-Streams in Hardware abhängig von der Stream-Größe

4.5.2. Messergebnisse für Hardware/Software Co-Design

Als nächstes wird das Hardware/Software Co-Design evaluiert. Für diesen Zweck werden dieselben Web Service-Nachrichten verwendet. Die Zeitmessung erfolgt in Software mithilfe des Xilinx SDK mit einer Zeitauflösung von 1 µs. Mithilfe der Zeitstempel wird die Start- und Stoppzeit einer Funktion ermittelt. Anschließend wird durch Subtraktion der Startzeit von der Stoppzeit die Verarbeitungszeit berechnet, die eine Funktion für die Ausführung benötigt. Die Zeitmessung wird für die einzelnen Funktionen durchgeführt und anschließend die Gesamtzeit ermittelt. Die Gesamtzeit setzt sich aus der Speicherreservierung, der Datenübergabe an den AXI4-Bus, der EXI-Verarbeitung bestehend aus dem EXI-Parsen und der Auswertung der empfangenen Daten von der Hardware sowie der Speicherfreigabe zusammen. Für jede Funktion werden drei Messungen durchgeführt und im Anschluss der Mittelwert berechnet. In Tabelle 16 sind die durchschnittlichen Verarbeitungszeiten der Funktionen sowie die Gesamtzeit für die einzelnen EXI-Nachrichten aufgelistet.

EXI-Stream	**EXI-Größe in Byte**	***Ø t* in µs**
request.exi	68	171
response.exi	88	180
hello.exi	284	260
bye.exi	259	247
probe.exi	197	218
probeMatch.exi	371	290

Tabelle 16: Verarbeitungszeiten der EXI-Streams durch das Hardware/Software Co-Design

Wie in Abbildung 54 dargestellt, hängt die Verarbeitungszeit eines EXI-Streams durch das Hardware/Software Co-Design ebenfalls linear von der EXI-Stream-Größe ab. Die Verarbeitung ist jedoch deutlich langsamer durch die limitierende Software-Komponente.

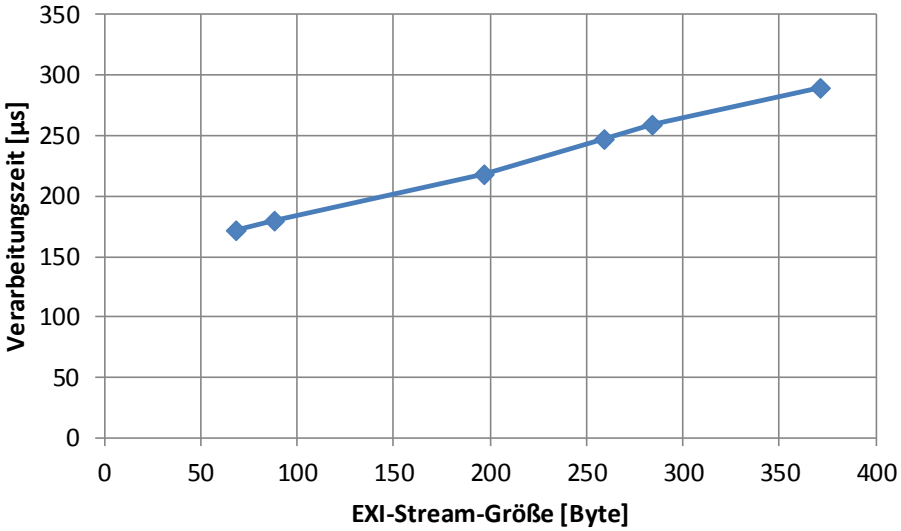

Abbildung 54: Verarbeitungszeiten der EXI-Streams durch das Hardware/Software Co-Design abhängig von der Stream-Größe

Um die Limitierung durch die Software besser zu untersuchen, wird die Verarbeitung der Request-Nachricht analysiert. Abbildung 55 zeigt die Zeiten für die einzelnen Verarbeitungsroutinen. Es ist zu erkennen, dass die Speicherreservierung, die Datenübergabe an den AXI4-Bus und die Speicherfreigabe einen großen Anteil (~ 50 %) der Verarbeitung in Anspruch nehmen. Darüber hinaus benötigt die Software für das Auslesen und die Auswertung der vom Hardwaremodul verarbeiteten Daten inklusive EXI-Parsen durch die Hardware 87 µs (ebenfalls rund 50 %) der Gesamtzeit. Hierbei muss beachtet werden, dass das Auslesen der Daten aus dem Hardware-Modul ebenfalls über den AXI4-Bus erfolgt. Durch die Implementierung der AXI4-Schnittstelle für die Kommunikation zwischen der Software- und Hardwarekomponente ergeben sich nun andere Ressourcenauslastungen der Hardware. Das Hardware/Software Co-Design benötigt somit 12999 Register (12 % der Gesamtanzahl) und 17868 LUTs (33 % der Gesamtanzahl). Wie erwartet, kommt es durch die Integration der AXI4-Schnittstelle zu einer höheren Hardwareauslastung. Die maximal erreichbare Frequenz des Gesamtsystems hat sich leicht verringert und liegt bei rund 53 MHz. Trotz des etwas gestiegenen Ressourcenverbrauchs passt das komplette Design für die Web Service-Kommunikation in das relativ kleine FPGA.

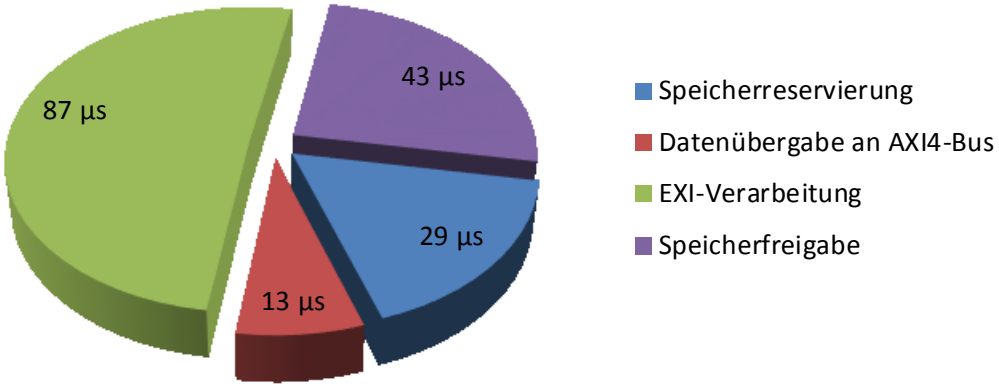

Abbildung 55: Einzelne Verarbeitungszeiten für die Request-Nachricht für das Hardware/Software Co-Design

4.5.3. Messergebnisse für reine Software-Lösung

Um die Leistungsfähigkeit der vorgeschlagenen Lösungen bewerten zu können, soll eine reine Software-Implementierung mit gleicher Funktionalität herangezogen werden. Hierfür wurde der Software-EXI-Parser µEXI [154] auf die Zielplattform portiert. Der Software-Parser wird für dieselben Web Service-Nachrichten aus derselben XSD generiert. Ähnlich wie Abschnitt 4.5.2 wurde die Messung in Software mittels Xilinx SDK durchgeführt. Es wurden Verarbeitungszeiten für einzelne Verarbeitungsabschnitte aufgenommen und daraus die Gesamtzeit gebildet. Demnach besteht die Gesamtverarbeitungszeit aus der Speicherreservierung, EXI-Stream-Verarbeitung und Speicherfreigabe. In Tabelle 17 sind die durchschnittlichen Verarbeitungszeiten für die einzelnen EXI-Nachrichten aufgelistet.

EXI-Stream	Größe in Byte	$\emptyset\ t$ in µs
request.exi	68	209
response.exi	88	218
hello.exi	284	352
bye.exi	259	323
probe.exi	197	289
probeMatch.exi	371	395

Tabelle 17: Verarbeitungszeiten der EXI-Streams in Software

Der Abbildung 56 kann entnommen werden, dass die Verarbeitungszeit mit zunehmender Größe ansteigt. Auch hier kann der lineare Zusammenhang zwischen der EXI-Stream-Größe und der Verarbeitungszeit beobachtet werden. Die Verarbeitungszeiten steigen jedoch weiter im Vergleich zu Hardware/Software Co-Design.

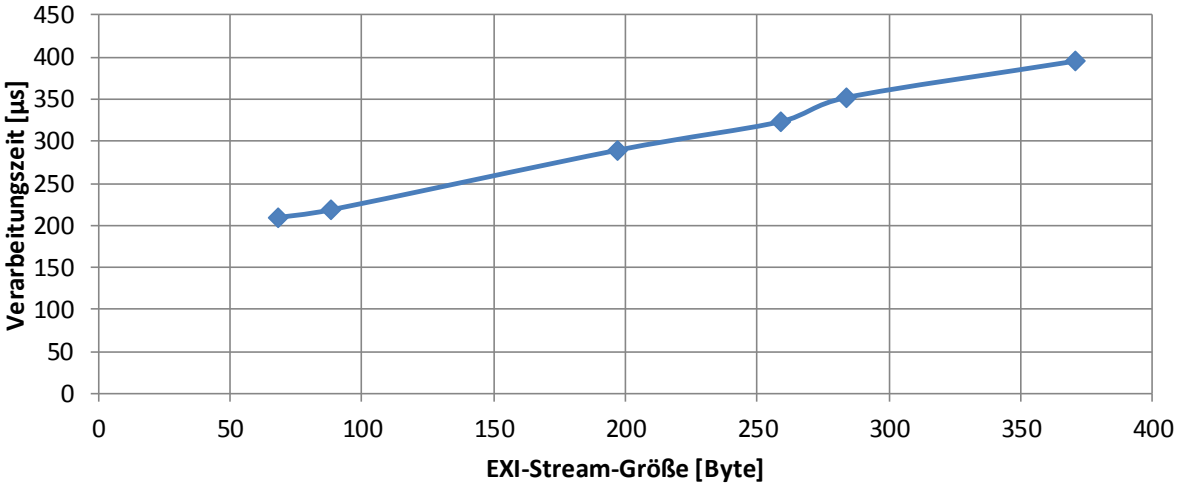

Abbildung 56: Verarbeitungszeiten der EXI-Streams durch das Hardware/Software Co-Design abhängig von der Stream-Größe

Um einen besseren Vergleich mit dem Hardware/Software Co-Design zu ermöglichen, werden Verarbeitungszeiten einzelner Funktion am Beispiel der Request-Nachricht untersucht. In Abbildung 57 sind die einzelnen Verarbeitungszeiten dargestellt. An diesem Beispiel ist zu erkennen, dass die EXI-Stream-Verarbeitung mit 139 µs rund 66 % der Gesamtzeit ausmacht. Die anderen 34 % werden für die Speicherreservierung und Speicherfreigabe benötigt. Während Speicherreservierung und Speicherfreigabe keine große Abweichung gegenüber dem Hardware/Software Co-Design darstellen, zeigt sich eine Erhöhung der reinen Verarbeitung des EXI-Streams (inkl. AXI4-Übertragung) um ca. 40 %. Die gesamte Verarbeitungszeit für das Request-Beispiel kann durch das Hardware/Software Co-Design um 18 % gesenkt werden. Die Beschleunigung der Verarbeitung ist nicht konstant und hängt ebenfalls von der Größe des EXI-Streams ab. In Abbildung 58 sind alle drei untersuchten Ansätze vergleichend dargestellt. Die reine Hardware-Umsetzung hat den kleinsten Anstieg der Verarbeitungszeit mit der steigenden EXI-Stream-Größe. Obwohl das Hardware/Software Co-Design bei dem verwendeten Beispiel eine Verbesserung der Verarbeitungszeit um 18 % mit sich bringt, steigt dieser Wert mit der steigenden EXI-Stream-Größe. Die Verarbeitungszeiten aller gemessenen Nachrichten können Abbildung 59 entnommen werden. Hierbei lassen sich Beschleunigungsraten von über 26 % beim Vergleich zwischen Software und Hardware/Software Co-Design bei Hello- und ProbeMatch-Nachrichten beobachten. Ein Vergleich zwischen reinen Software- und Hardware-Lösungen zeigt, dass der Hardware-EXI-Parser lediglich 5 % der Zeit eines Software-EXI-Parsers für den gleichen EXI-Stream benötigt. Diese sehr hohe Beschleunigung kann besonders in Umgebungen mit harten Echtzeitanforderungen eine entscheidende Rolle spielen, um die

Reaktionszeiten einzuhalten bzw. anderen Komponenten mehr Verarbeitungszeit zu ermöglichen.

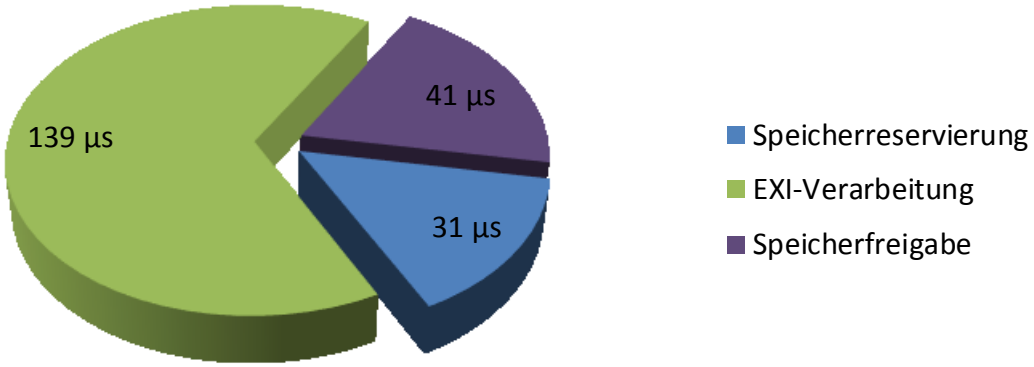

Abbildung 57: Einzelne Verarbeitungszeiten für die Request-Nachricht für die Software-Umsetzung

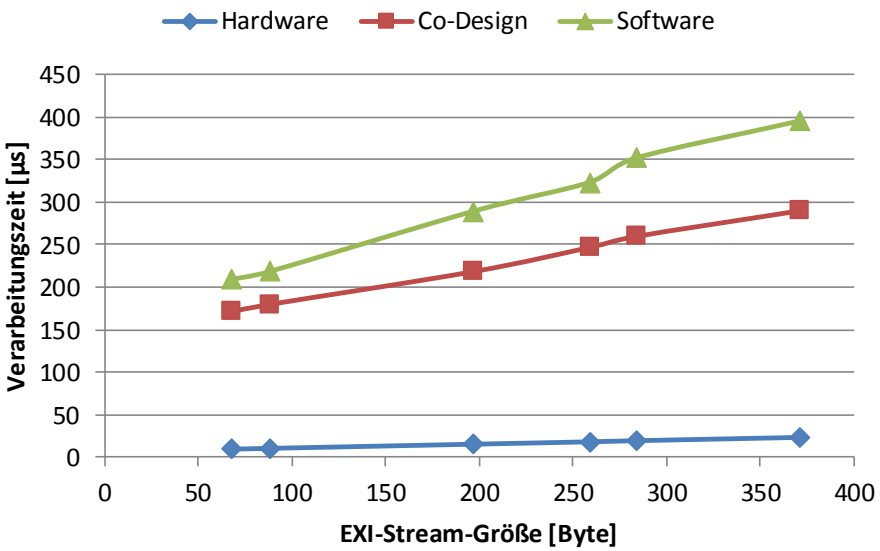

Abbildung 58: Vergleich der Verarbeitungszeiten verschiedener EXI-Parser-Implementierungen in Abhängigkeit der EXI-Stream-Größe

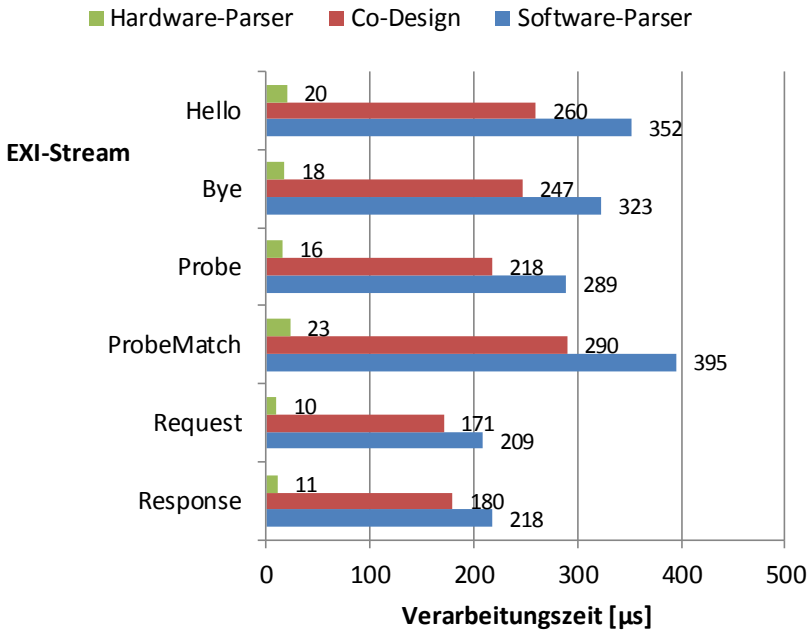

Abbildung 59: Vergleich der Verarbeitungszeiten verschiedener EXI-Parser-Implementierungen für spezifische Web Service-Nachrichten

4.6. Zusammenfassung

Im Rahmen dieser Arbeit wurde das Konzept und die Implementierung eines EXI-Parsers in Hardware vorgestellt. Der Parser kann dynamisch zur Synthesezeit erstellt und an ein beliebiges XML-Schema angepasst werden. Damit ist der Einsatz der Web Service-Technologie in „Deeply Embedded“ und Echtzeitumgebungen möglich. Dank der EXI-Kompression können XML-Dokumente in Hardware verarbeitet werden, was zu einer Performancesteigerung gegenüber einer reinen Softwarelösung führt. Der entwickelte Hardware-EXI-Parser kann die Verarbeitungszeit durchschnittlich um mehr als 20 % als Hardware/Software Co-Design und um ca. 95 % als eine reine Hardware-Lösung im Vergleich zu Software-basierten Ansatz reduzieren. Der Geschwindigkeitsvorteil kommt insbesondere bei größeren EXI-Nachrichten zur Geltung. Der Einsatz des Hardware-EXI-Parsers kann besonders in Umgebungen mit harten Echtzeitanforderungen und sehr kurzen Antwortzeiten wie z.B. Motion Control vorteilhaft sein. Hierbei werden die dynamische Struktur und die Interoperabilität der Web Services nicht durch die Anwendung eingeschränkt. Darüber hinaus kann der entwickelte EXI-Parser als ein Co-Prozessor für die EXI-Verarbeitung eingesetzt werden und dabei den Hauptprozessor entlasten. Durch die Verwendung von Standardkomponenten für die Parser-Generierung kann dieser leicht in existierende FPGA-basierte Systeme integriert werden. Die ebenfalls entwickelte Software-

Komponente kann die Daten vom Hardware-EXI-Parser empfangen und diese anderen Instanzen zur Verfügung stellen.

Hiermit wurde gezeigt, dass XML-Dokumente ebenfalls in Hardware-basierten Systemen mithilfe von EXI eingesetzt werden können. Das eröffnet neue Einsatzgebiete für die Web Service-Technologie.

5. Optimierung von WS-Discovery für große Netzwerke

5.1. Motivation

Geräte- und Service-Discovery ist eine wichtige Funktion für eine automatisierte Netzwerkkonfiguration. Ein automatisches Geräte-Discovery wird bereits von DPWS angeboten. Es ermöglicht, dass sich alle Geräte wie z.B. Smart Meter, Smart Meter Gateways oder Sensoren und Aktoren automatisch im Netzwerk identifizieren. Diese Funktionalität ist im Standard WS-Discovery spezifiziert. WS-Discovery stellt eine Grundlage für die Plug&Play-Kopplung von Geräten dar. Bei dem Discovery werden zwei Betriebsarten unterschieden: *Managed Mode* und *Ad-hoc-Mode*.

Bei dem Managed Mode handelt es sich um ein infrastrukturbasiertes Discovery mit einem zentralen Knoten wie dem *Discovery Proxy*. Discovery Proxys gehen auf UDDI zurück. Die Geräte melden die angebotenen Services beim Discovery Proxy an. Wird von einem Gerät eine Discovery-Anfrage gestellt, wird diese nicht vom Gerät selbst, sondern vom Proxy beantwortet (vgl. Abschnitt 2.5.2). Eine Anfrage kann auch direkt an einen Proxy gestellt werden, wenn seine IP-Adresse bekannt ist. Die Proxys können auch benutzt werden, um entfernte Netze nach Geräten zu durchsuchen. In diesem Fall ist das implizite Wissen über die IP-Adresse des Proxys unentbehrlich. Die Proxys erlauben somit einen einfachen Discovery-Mechanismus mit einer zusätzlichen Kontrolle über das Netz. Die zentralisierte Struktur bringt auch für solche Systeme typische Nachteile mit sich. Der Discovery Proxy stellt einen SPoF dar. Fällt der Proxy aus, ist in dem Netz kein Discovery mehr möglich. Auch andere Fehlfunktionen an diesem Knoten können sich im gesamten Netzwerk widerspiegeln. Aus diesem Grund wurde mit WS-Discovery eine neue Methode vorgestellt, die als Ad-hoc-Discovery bezeichnet wird.

Ad-hoc-Discovery verzichtet auf eine zentrale Instanz. Alle Discovery-Anfragen müssen dabei von Geräten selbst beantwortet werden. Damit jedes Gerät die Anfrage bekommen kann, wird sie von einem suchenden Gerät (Client) mittels Multicast geschickt. Das heißt, der Client trägt nicht eine konkrete IP-Adresse ein, sondern eine Multicast-Adresse (Gruppenadresse). Solche Anfragen werden von allen Geräten empfangen und verarbeitet. Wurde ein spezieller Service-Typ angefragt, müssen nur die Geräte antworten, die diesen Service anbieten. Enthält die Anfrage keine Service-Typen, müssen alle Geräte antworten. In diesem Fall kann der Client einen gewünschten Service-Typ selbst aussuchen. Der Vorteil dieser Methode ist der vollständig dezentralisierte Ansatz. Damit hat ein Ausfall eines oder mehrerer Geräte keinen Einfluss auf das gesamte Netzwerk. Die Standardspezifikation von WS-Discovery hat jedoch auch Nachteile in Bezug auf Skalierbarkeit. Nach einer Discovery-Anfrage müssen die Geräte innerhalb eines fest vorgegebenen Zeitfensters eine Antwort schicken. Mit einer steigenden Anzahl der Geräte steigt auch die Auslastung des Zeitfensters,

was zu einer steigenden Datenrate am Client und im Netzwerk führt. Verfügt der Client nicht über die notwendige Rechenleistung, können Pakete verworfen werden und damit auch die Discovery-Antworten. Im schlimmsten Fall kann es zu einem Buffer Overflow und damit auch zum Ausfall des Gerätes führen. Um eine bessere Skalierbarkeit zu erzielen, wurde das Verhalten von WS-Discovery im Rahmen dieser Arbeit in großen Netzwerken mittels Simulation untersucht und in einem Feldversuch bestätigt.

Für die Untersuchung wurde der *Network Simulator 3* (ns-3) verwendet [155]. Dieser bietet eine diskrete ereignisorientierte Netzwerksimulation. Der Vorteil von ns-3 ist, dass eine Simulation alle ISO/OSI-Schichten inklusive der physikalischen Schicht umfasst. Damit lassen sich komplexe Netzwerkabläufe und Verfahren testen. Als physikalisches Medium wurde für die Simulation *Switched Ethernet* als am meisten verbreitete Kommunikationstechnologie ausgewählt.

Zusammenfassend ist der Hauptbeitrag dieses Kapitels wie folgt [3] [5]:

- Untersuchung der Skalierbarkeit der Service Discovery für große Netzwerke
- Ausarbeitung der Optimierungen für den Discovery-Mechanismus
- Evaluierung der Simulationsergebnisse
- Durchführung eines Feldversuchs zur Bestätigung der Simulationsergebnisse

5.2. Stand der Technik

Service-Discovery ist ein essentieller Bestandteil der Plug&Play-Gerätekopplung. Zwei Arten des Discovery können betrachtet werden: das Infrastruktur-basierte und das Ad-hoc-Discovery. Für das Infrastruktur-basierte Discovery ist eine zentrale Instanz in Form eines Service-Brokers oder auch UDDI unentbehrlich. In [156] wurde eine DHT-basierte Struktur von UDDI vorgestellt. Die Struktur ist robust gegen Fehler im Service-Verzeichnis wegen des verteilten Charakters von DHT. Die Geräte sind jedoch nicht Teil von DHT, sondern nur die Service-Proxys. Wenn der lokale Service-Proxy ausfällt, haben die Geräte keinen Zugriff mehr auf das Service-Verzeichnis. Dadurch bleibt ein lokaler Service-Proxy weiterhin ein SPoF. In [157] schlagen die Autoren eine gemischte Architektur, bestehend aus Service-Proxys und Ad-hoc-Discovery, vor. Das ganze Netzwerk wird dabei in mehrere Subnetze unterteilt. Um ein Gerät im Subnetz zu finden, wird Ad-hoc-Discovery eingesetzt. Um über das Subnetz hinaus zu suchen, wird ein Service-Proxy verwendet. Die IP-Adresse des Service-Proxy wird dabei über DHCP als herstellerspezifische Option übermittelt. Diese Lösung wirkt dem SPoF-Charakter des Service-Proxy entgegen. Wenn dieser Proxy jedoch ausfällt, ist die Suche nach Services im betroffenen Subnetz nicht mehr möglich. Darüber hinaus kann das suchende Gerät nach wie vor durch die Antworten der Service Provider abhängig von der Subnetzgröße überlastet werden. Die Skalierbarkeit mit einer steigenden Anzahl von Subnetzen wurde von den Autoren nicht untersucht. Ein hybrider Discovery-

Ansatz wurde in [158] vorgestellt. Dieser Ansatz ist jedoch auf drahtlose Sensornetzwerke beschränkt. Die in dieser Arbeit vorgeschlagene Lösung ist dagegen unabhängig von der physikalischen Übertragungstechnologie. In [109] wurde die Suche nach Geräten als Netzwerk-Scan implementiert. Einzelne Netzwerkadressen, Adressbereiche oder Wildcards können für das Discovery genutzt werden. Dieser Discovery-Ansatz skaliert durch die Limitierung des Adressraums. Die Antwortzeit des Systems kann jedoch erheblich ansteigen. Der Adressraum kann ebenfalls durch die Verwendung von Semantikregeln reduziert werden [159]. Dennoch kann die Anzahl der Antworten und die entstehende Traffic-Auslastung nicht ohne weiteres vorhergesagt werden.

Die in dieser Arbeit vorgeschlagene Lösung besitzt keinen SPoF. Der Discovery-Mechanismus nutzt die Vorteile des Ad-hoc-Discovery aus. Die vorgeschlagenen Optimierungen ermöglichen Skalierbarkeit und Zuverlässigkeit von WS-Discovery für Netzwerke beliebiger Größe, ohne einen negativen Einfluss auf die Antwortzeit des Systems zu erzeugen. Der optimierte Discovery-Ansatz ist darüber hinaus zum WS-Discovery-Standard rückwärtskompatibel, sodass Geräte, die die Optimierungen nicht interpretieren können, weiterhin am Service Discovery-Prozess teilnehmen können.

5.3. Optimierungen von WS-Discovery

In dieser Arbeit wird ausschließlich das Ad-hoc-Discovery berücksichtigt, da es einen verteilten Prozess ohne SPoF darstellt. Der Nachteil von WS-Discovery ist eine eingeschränkte Skalierbarkeit durch statische Zeitvorgaben. WS-Discovery nutzt SOAP-over-UDP Binding. Diese Spezifikation schreibt vor, ein zufälliges Delay zwischen 0 und 500 ms abzuwarten, bevor eine Antwort (ProbeMatch) auf eine Discovery-Anfrage (Probe) verschickt wird [58]. Das zufällige Delay sorgt dafür, dass nicht alle Geräte gleichzeitig antworten. Darüber hinaus soll jede UDP-Nachricht ein zweites Mal nach 50-250 ms verschickt werden, um die Zuverlässigkeit zu erhöhen. Falls der Service Provider keine Transportadresse mit der ProbeMatch-Nachricht mitgeschickt hat, kann diese mittels einer Resolve-Nachricht abgefragt werden. Die Antwort darauf (ResolveMatch) stellt dabei keine kritische Funktion dar, da die ResolveMatch-Nachricht von einem einzelnen Gerät verschickt wird. Der beschriebene Nachrichtenaustausch ist in Abbildung 60 illustriert.

Mit wachsender Netzwerkgröße müssen sich mehrere Teilnehmer dasselbe Zeitintervall teilen, was zu einer steigenden Datenrate führt. Demnach würde ein dynamisch einstellbares Delay helfen, das Zeitintervall auf die Netzwerkgröße anzupassen. Mithilfe der Faustregel, dargestellt in Formel (14), ist es möglich, eine Schätzung über die mittlere Datenrate für den Client zu gewinnen.

$$D = S * \frac{G}{R} \tag{14}$$

Dabei bezeichnet D das Delay, S die Nachrichtengröße, G die Anzahl der Geräte und R die Datenrate. Das am meisten kritische Verhalten tritt jedoch während der Peak-Datenraten auf. Zu diesen Zeiten muss der Client ein höheres Datenvolumen u.U. in kurzer Zeit verarbeiten. Dadurch werden die Ressourcen des Clients höher beansprucht. Stehen zu diesem Zeitpunkt nicht genug freie Ressourcen zur Verfügung, führt das zum Buffer Overflow oder Paketverwürfen. In den nächsten Abschnitten werden die Optimierungen zur Verbesserung der Skalierbarkeit von WS-Discovery vorgestellt.

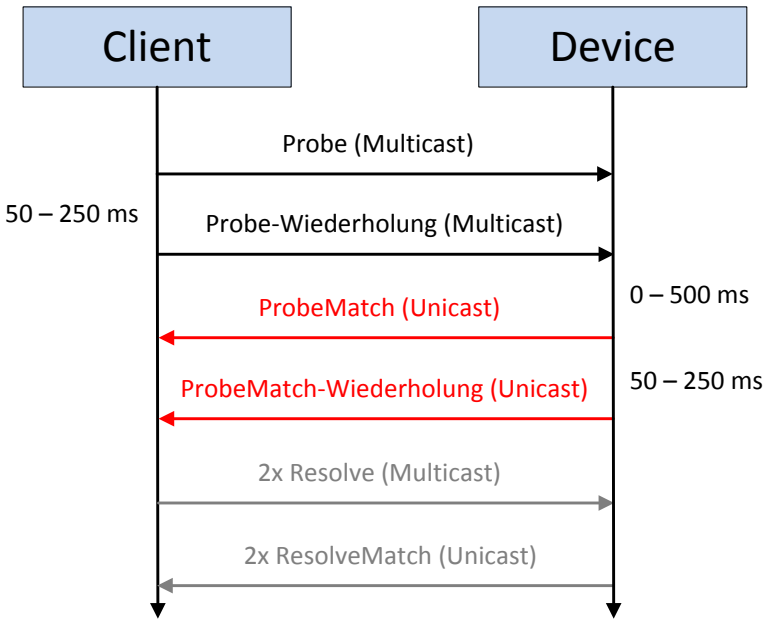

Abbildung 60: Nachrichtenaustausch während des Discovery-Prozesses

5.4. Einfügen der Hardware-Adresse

Für Device Discovery werden Probe-Nachrichten verwendet. Im Feld *Type* kann das gewünschte Gerät (Hosting Service) angegeben werden. Es kann auch leer gelassen werden, um nach allen Geräten zu suchen. Nachdem ein Service Provider eine Anfrage bekommen hat, wartet er das Delay ab und versucht, die Antwort zu schicken. In Ethernet-Netzwerken muss die MAC-Adresse bekannt sein, um ein Paket zu erzeugen. Im Falle einer Anfrage von einem unbekannten Gerät (Client) muss seine MAC-Adresse erfragt werden. Für diesen Zweck wird das ARP-Protokoll verwendet (vgl. Abschnitt 2.4.1). ARP-Anfragen werden ebenfalls als Multicast verschickt. Der Client antwortet mit seiner eigenen MAC-Adresse dem Service Provider. Anschließend kann die WS-Discovery-Antwort erstellt und an den Client verschickt werden. Dieser Ablauf ist in Abbildung 61 dargestellt.

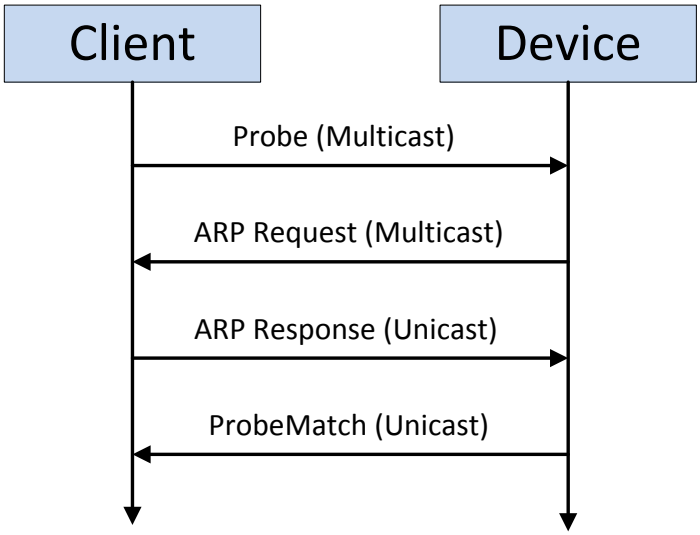

Abbildung 61: MAC-Adressauflösung während des Discovery-Prozesses

Wie von SOAP-over-UDP Binding verlangt, muss die Antwort nach 50-250 ms wiederholt werden, um eine höhere Zuverlässigkeit zu erzielen. Folglich verursacht eine WS-Discovery-Anfrage innerhalb kurzer Zeit 4 Mal so viele Nachrichten, wie es Geräte im Netzwerk gibt (ARP-Anfrage, ARP-Antwort, und 2 Mal WS-Discovery-Antwort). Der Client muss dementsprechend 3 Mal so viele Nachrichten verarbeiten wie Geräte im Netzwerk existieren (ARP-Anfrage und 2 Mal WS-Discovery-Antwort). Um die Skalierbarkeit zu erhöhen und sowohl das Netzwerk als auch den Client zu entlasten, wird das Einfügen einer Hardware-Adresse (MAC-Adresse) in die Discovery-Anfrage vorgeschlagen. Die Geräte können diese Adresse direkt für die Erzeugung der Ethernet-Pakete nutzen. Die Hardware-Adresse (HWAddr) kann wie folgt in die Probe-Nachricht eingefügt werden:

```
<s12:Body>
    <wsd:Probe>
        <wsd:Types />
        <wsd:HWAddr>01:01:01:01:01:01</wsd:HWAddr>
        …
```

Wenn die alten Geräte dieses neue Feld nicht interpretieren können, kann es ohne Auswirkungen ignoriert werden, da es die Funktionalität von WS-Discovery nicht beeinträchtigt. Die Übertragung der Hardware-Adresse resultiert in einer Halbierung der Anzahl der übertragenen Nachrichten. Durch das Vermeiden des ARP-Protokolls muss der Client nur noch die WS-Discovery-Antworten verarbeiten.

5.5. Dynamisches Delay

Mithilfe von ns-3 wurde das Netzwerkverhalten untersucht. Für das Simulationsszenario wurde 100 Mbps Ethernet (100BASE-T) – als am meisten genutzter Ethernet-Standard in der Automatisierung – exemplarisch ausgewählt. Die Größe der WS-Discovery-Antworten betrug in der Simulation 1176 Byte (keine Kompression verwendet), was einer Antwort mit zwei Service-Typen entspricht. Um das durch WS-Discovery verursachte Datenaufkommen zu bestimmen, wurde die Datenrate am Client abhängig von der Geräteanzahl im Netzwerk gemessen. In der Messung wurde zunächst auf eine Wiederholung der Pakete verzichtet, damit das Ergebnis mit der Formel (14) vergleichbar ist. Der Versuch wurde mit 200 Knoten ausgeführt. Wie in Abbildung 62 gezeigt, weicht die jeweilige Datenrate stark von der durchschnittlichen Datenrate ab. Die Peak-Datenrate kann doppelt so hoch wie die durchschnittliche Datenrate sein. Dementsprechend ist die Berücksichtigung der Peak-Datenrate ein wichtiger Aspekt, um den Client im „Worst Case“ zu schützen.

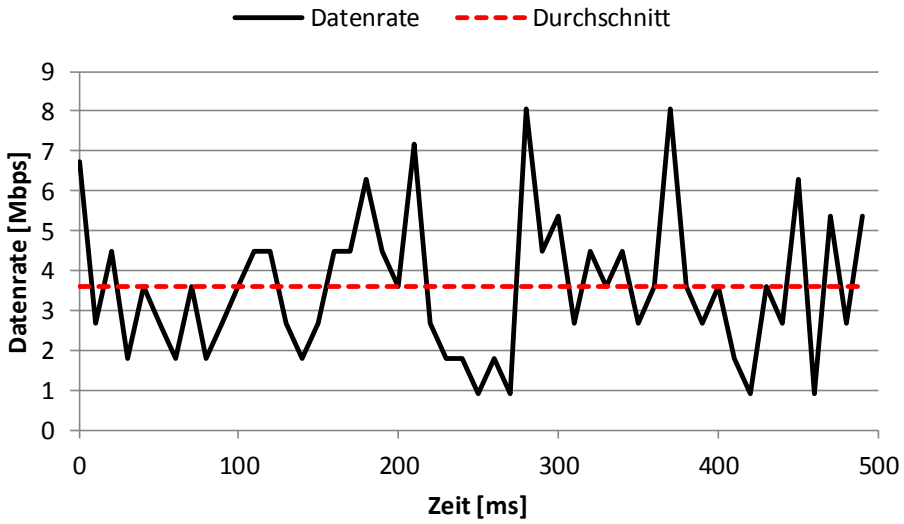

Abbildung 62: Vergleich der momentanen und der durchschnittlichen Datenraten für 200 Knoten

Um das Verhalten von WS-Discovery abzuschätzen, wurden die Anzahl der Knoten und das Delay variiert und die Datenrate am Client gemessen. Für eine höhere Genauigkeit wurden ca. 100 Simulationsläufe mit verschiedenen Werten durchgeführt. Die Ergebnisse sind in Abbildung 63 dargestellt.

Die Peak-Datenrate ist linear von der Knotenanzahl abhängig und stellt eine Potenzfunktion in Abhängigkeit vom Delay dar. Folglich muss das Delay so eingestellt werden, dass die gewünschte Peak-Datenrate nicht überschritten wird, um die Skalierbarkeit von WS-Discovery zu gewährleisten. Das Setzen des Delays auf einen sehr pessimistischen und damit hohen Wert würde zu einer vergrößerten Reaktionszeit des Gesamtsystems führen.

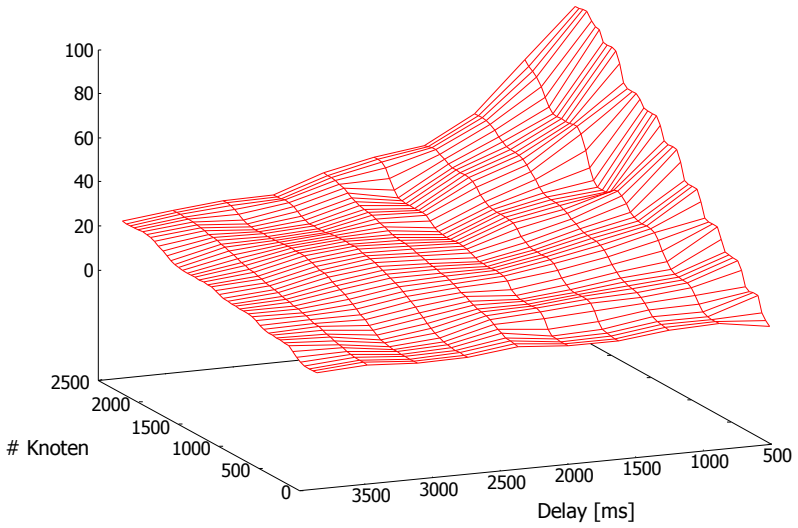

Abbildung 63: Peak-Datenrate in Abhängigkeit von Delay und Knotenanzahl

Die gesammelten Simulationsergebnisse wurden genutzt, um das Verhalten von WS-Discovery zu approximieren. Für die Approximation der Funktion wurde die Methode der kleinsten Quadrate verwendet. Das Ergebnis spiegelt sich in Formel (15) wider.

$$D = \left(\frac{5{,}25 * G + 407{,}94}{R}\right)^{1{,}3} \tag{15}$$

Dabei ist D das Delay in ms, G ist die Knotenanzahl und R ist die Datenrate in Mbps. Die Approximationsfunktion ist in Abbildung 64 dargestellt.

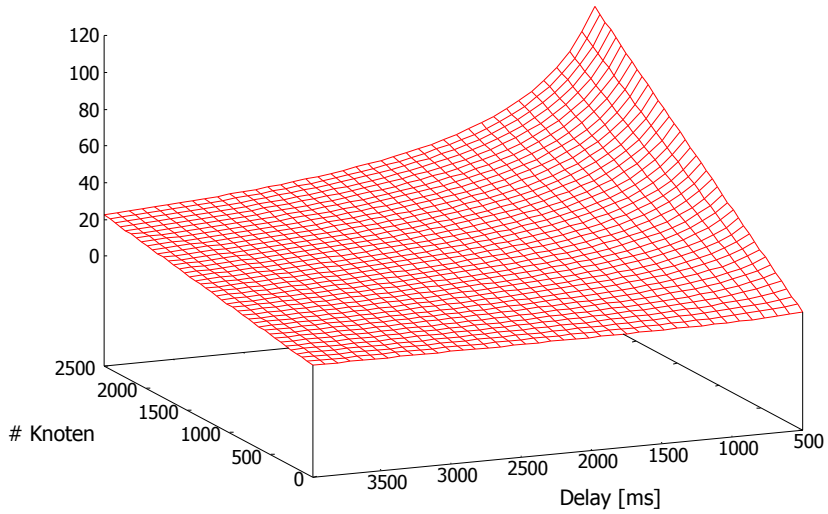

Abbildung 64: Approximation der Delay-Funktion

Formel (15) stellt einen Kompromiss zwischen Peak-Datenrate und Systemantwortzeit dar. Da diese Formel empirisch aus Schätzungen ermittelt wurde, sollten die Werte für Knotenanzahl und die Datenrate dennoch eher konservativ als optimistisch gewählt werden.

Die Berechnung des resultierenden Delays soll auf der Client-Seite stattfinden. Die gewünschte Datenrate kann beispielsweise vom Gerätehersteller vorgegeben werden. Die Anzahl der Knoten ist ein geschätzter Wert, der z.B. vom Netzwerkadministrator bestimmt werden kann. Wenn keine Informationen über das Netzwerk vorliegen, soll die maximale Netzwerkgröße gewählt werden. Die maximale Netzwerkgröße kann z.B. im Falle von IPv4 durch die Netzwerkmaske ermittelt werden. Für eine allgemeine Bestimmung der Netzwerkgröße wird im Abschnitt 5.8 ein Mechanismus vorgeschlagen.

Nach der Ermittlung des Delays soll dieses in die WS-Discovery-Anfrage eingefügt werden. Der Wert für das Delay in ms kann wie folgt übertragen werden:

```
<s12:Body>
    <wsd:Probe>
        <wsd:Types />
        <wsd:Delay>650</wsd:Delay>
        …
```

Alle Geräte, die eine Discovery-Anfrage bekommen, sollen den neuen Wert für das maximale Delay berücksichtigen und einen zufälligen Wert zwischen 0 und *Delay* ms bestimmen. Geräte, die das Feld nicht interpretieren können oder deren Implementierung keine neuen Delays zulässt, können den Standardwert nutzen.

Der vorgeschlagene Ansatz spreizt das Zeitintervall und reduziert dadurch die Peak-Datenrate. Die Abweichung der Peak-Datenrate von der mittleren Datenrate bleibt jedoch gleich. Um diese zu reduzieren, wird eine Knotengruppierung vorgeschlagen.

5.6. Knotengruppierung

Beim Betrachten der Funktion in Abbildung 62 fällt auf, das die Funktion viele Spitzen und Vertiefungen hat. Wenn es möglich wäre, die Knoten, die während der Spitzen senden, stattdessen während der Vertiefungen senden zu lassen, wäre die Datenrate ausgeglichen. Um dieses Verhalten zu erzielen, wird das gesamte Zeitintervall in Zeitschlitze eingeteilt. Jedem Knoten wird dabei ein Zeitschlitz zugeteilt. Es können sich jedoch mehrere Knoten einen Zeitschlitz teilen. Abhängig von der Zeitschlitzzuweisung können Zeitschlitze mit vielen Knoten oder auch ohne Knoten entstehen. Um eine effiziente Verteilung zu erzielen, wird ein IP-basierter Ansatz vorgeschlagen. In einem üblichen Ethernet-Netzwerk gibt es einen DHCP-Server (vgl. Abschnitt 2.4.2). Wenn ein neues Gerät dem Netzwerk beitritt, fordert es vom DHCP-Server eine IP-Adresse an. Üblicherweise werden die IP-Adressen iterativ von der kleinsten bis zur größtmöglichen zugewiesen. Dadurch ist es wahrscheinlich, dass sich die meisten Geräte im unteren Teil des Adressraums positionieren. Folglich wird eine sequentielle Zuweisung der Zeitschlitze zu vollen und leeren Zeitschlitzen führen. Eine effizientere Methode ist, die IP-Adressen aus den beiden Adressräumen (vollen und leeren)

einem Zeitschlitz zuzuteilen. In diesem Fall werden die Zeitschlitze nur halbvoll sein und damit steht jedem Knoten mehr Zeit für die Datenübertragung zu.

Die Knoten werden einem Zeitschlitz unter Verwendung der Modulo-Operation zugewiesen. Der Client muss entscheiden, wie viele Zeitschlitze für die aktuelle Konstellation angemessen sind. Um eine optimale Anzahl an Zeitschlitzen zu bestimmen, wurden Simulationen für 250 bzw. 1000 Knoten und einer variablen Anzahl an Zeitschlitzen durchgeführt. Darüber hinaus wird die kleinstmögliche Periode für einen Zeitschlitz in Höhe von 1 ms angenommen. Da Geräte mit verschiedenen Fähigkeiten dem Netzwerk beitreten, muss WS-Discovery garantieren, dass alle Geräte die zeitliche Auflösung von 1 ms unterstützen. Entsprechend der kleinstmöglichen Periode kann ein Zeitintervall von 500 ms maximal in 500 Zeitschlitze aufgeteilt werden.

Wie in Abbildung 65 gezeigt, werden mehrere lokale Maxima und Minima abhängig von der Anzahl der Zeitschlitze erzeugt. Es wird jedoch ein globales Minimum erzeugt, wenn die Knotenanzahl größer als eine maximal mögliche Anzahl der Zeitschlitze ist (1000 Knoten), d.h., wenn die kleinstmögliche Größe eines Zeitschlitzes erreicht ist. In dem anderen Fall (250 Knoten) wird ein globales Minimum erreicht, wenn die Anzahl der Zeitschlitze der Knotenanzahl entspricht. Das bedeutet, die Peak-Datenrate ist am kleinsten, wenn es pro Zeitschlitz nur einen Knoten gibt.

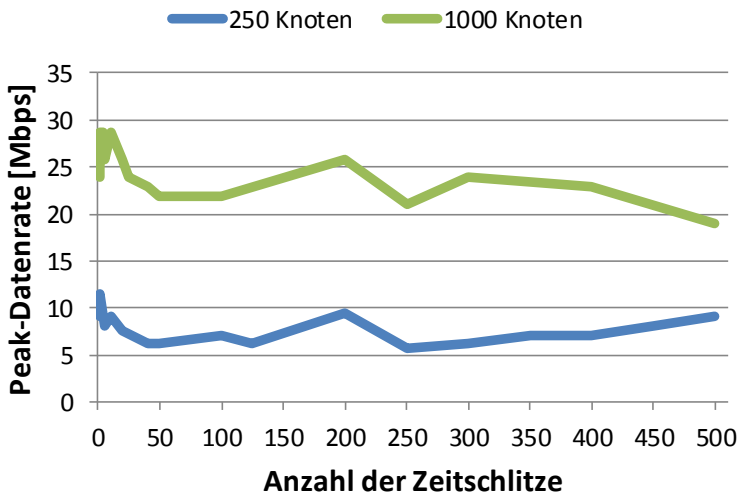

Abbildung 65: Peak-Datenrate in Abhängigkeit von der Anzahl der Zeitschlitze

In einem realen Netzwerk ist das jedoch weniger wahrscheinlich, da die IP-Adressen oft nicht exakt sequentiell verteilt sind. Wenn ein Gerät das Netzwerk verlässt, entsteht eine Lücke. Die freigegebenen IP-Adressen können jedoch neuen Geräten zugewiesen werden. Der vorgeschlagene Ansatz kann dennoch das gewünschte Verhalten approximieren. Ausgehend

von diesen Überlegungen wurde eine Formel für die Schätzung der Anzahl der Zeitschlitze hergeleitet:

$$S = S_{max,} \text{ für } G > S_{max} \text{ und } S = G, \text{für } G < S_{max} \tag{16}$$

Dabei ist S die Anzahl der Zeitschlitze, S_{max} ist die maximal mögliche Anzahl der Zeitschlitze und G ist die Knotenanzahl. Der Client kann die gewünschte Anzahl der Zeitschlitze in die WS-Discovery-Anfrage wie folgt einfügen:

```
<s12:Body>
    <wsd:Probe>
        <wsd:Types />
        <wsd:Slots>250</wsd:Slots>
        …
```

Jedes Gerät, das eine WS-Discovery-Antwort schicken will, muss seine Zeitschlitzzugehörigkeit bestimmen. Hierfür muss es die Modulo-Operation auf seine IP-Adresse entsprechend der Anzahl der Zeitschlitze anwenden. Anschließend muss ein zufälliges Delay innerhalb des zugehörigen Zeitschlitzes berechnet werden. Die zeitlichen Grenzen eines Zeitschlitzes können nach Formel (17) bestimmt werden. Hierbei sind S_{start} und S_{end} die Start- und Endzeitpunkte eines Zeitschlitzes, *IP* ist eine Integer-Darstellung einer IP-Adresse, *S* ist die Anzahl der Zeitschlitze und *D* ist das Delay.

$$S_{start} = \left\lceil (IP \bmod S) * \frac{D}{S} \right\rceil$$
$$S_{end} = \left\lceil \left((IP \bmod S) + 1 \right) * \frac{D}{S} \right\rceil \tag{17}$$

Abhängig von der gewählten Anzahl der Slots und des Delay können u.U. unterschiedlich lange Slots entstehen, die sich jedoch maximal um 1 ms unterscheiden. Es wird daher empfohlen, die Slot-Anzahl und das Delay nach Möglichkeit so zu wählen, dass eine gleichmäßige zeitliche Verteilung möglich ist.

5.7. Optimierung der Paketwiederholung

In allen vorherigen Betrachtungen wurden die Paketwiederholungen entsprechend dem SOAP-over-UDP Binding ausgelassen. Die Wiederholungspakete kollidieren offensichtlich mit den ersten verzögerten WS-Discovery-Antworten anderer Geräte. Wenn das erste Beispielszenario (vgl. Abbildung 62) mit der Paketwiederholung durchgeführt wird, treten noch höhere Datenraten auf (vgl. Abbildung 66). Die durchschnittliche Datenrate wurde dem gesamten Zeitintervall und der Anzahl der gesendeten Pakete entsprechend angepasst.

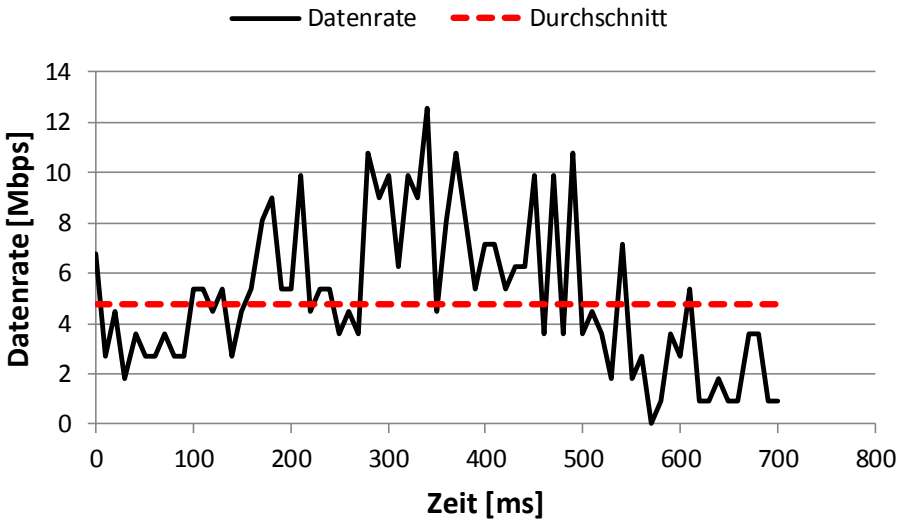

Abbildung 66: Datenrate von WS-Discovery mit Paketwiederholungen

Da SOAP-over-UDP Binding eine Paketwiederholung nach 50-250 ms nach dem Versenden des ersten Paketes vorschreibt, kann das Wiederholungspaket mit dem ersten Paket eines anderen Gerätes kollidieren, das um mehr als 50 ms durch einen Zufallswert verzögert wurde. Um zusätzliche Kollisionen mit Wiederholungspaketen zu vermeiden, wird eine Teilung des gesamten Zeitintervalls in das erste und zweite Antwortintervall vorgeschlagen. Das Wiederholungsintervall soll gleich im Anschluss an das Hauptintervall beginnen. Alle Optimierungen, die oben vorgeschlagen wurden, sollen ebenfalls auf das Wiederholungsintervall angewandt werden. Für diesen Zweck sollen die beiden Intervalle symmetrisch sein. Um die Zuverlässigkeit zu erhöhen und zufällige Netzwerkfehler zu vermeiden, soll das Wiederholungsintervall eine Spiegelung des Hauptintervalls darstellen. In diesem Fall wird die zeitliche Entfernung zwischen der Originalantwort und der Wiederholung maximal sein. Damit die Antwortzeit gleich bleibt, soll das Hauptintervall zugunsten des Wiederholungsintervalls gekürzt werden. Auf diese Weise werden die beiden Intervalle gleiche Peak-Datenraten erzeugen und eine ausbalancierte Verarbeitung der Pakete am Client ermöglichen. Die oben beschriebene Vorgehensweise illustriert Abbildung 67.

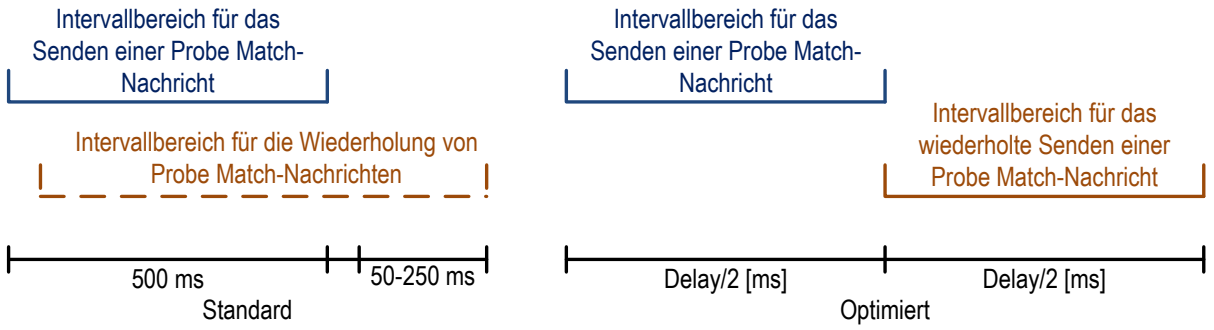

Abbildung 67: Optimierung der Paketwiederholung

Wenn alle Optimierungen auf das beschriebene Szenario (vgl. Abbildung 62) angewendet werden, kann eine wesentliche Reduzierung der Peak-Datenrate (über 50 %) auf der Client-Seite erreicht werden (siehe Abbildung 68). Wie bereits erwähnt, kann der theoretische Durchschnitt in der Realität nicht erreicht werden. Die vorgeschlagenen Optimierungen können jedoch die Approximation deutlich verbessern. Wenn das Gerät alle genannten Optimierungen unterstützt, kann die Faustregel, dargestellt in Formel (14), mit etwas pessimistischen Werten bereits den Anforderungen an die Peak-Datenrate genügen. Auf diese Weise kann die Formel (14) für ein optimiertes WS-Discovery für beliebige Anzahl der Geräte, Nachrichtengrößen und Datenraten genutzt werden, was ebenfalls in Simulationen nachgewiesen wurde. Die vorgeschlagenen Optimierungen werden als zusätzliche Optionen (Tags) übertragen. Falls ein Gerät diese nicht interpretieren kann, können diese ignoriert und Standardwerte benutzt werden. Daher ist das optimierte Discovery-Verfahren zu dem Standard WS-Discovery abwärtskompatibel.

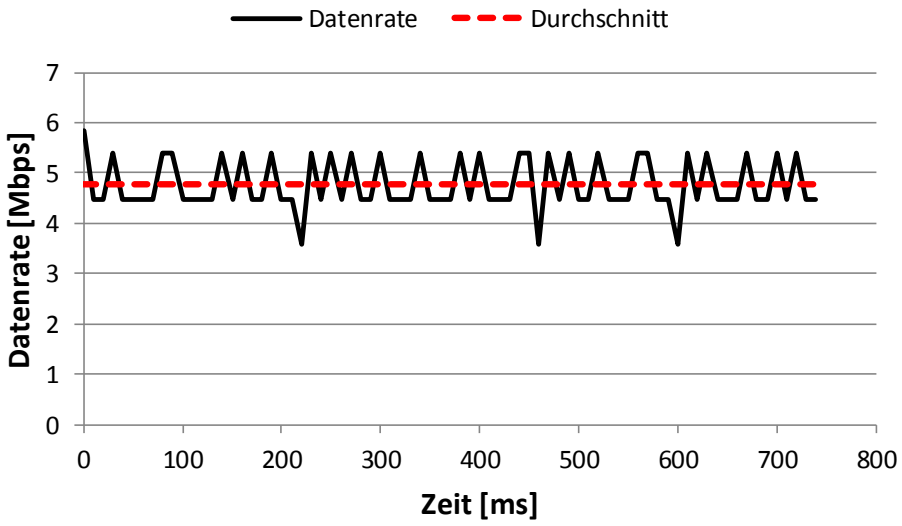

Abbildung 68: Datenrate von WS-Discovery mit angewandten Verbesserungen

5.8. Network Size Discovery

Der oben vorgeschlagene Ansatz stellt einen effizienten Discovery-Mechanismus für eine beliebige Knotenanzahl dar. Eine Bedingung muss jedoch erfüllt sein: Die Knotenanzahl muss im Voraus bekannt sein oder von einem Netzwerkadministrator vorgegeben werden. Dies ist allerdings in Automations- und Plug&Play-Netzwerken oft nicht gegeben, da der direkte Gerätezugriff nicht möglich ist oder nicht im Interesse des Betreibers liegt. Zur Lösung dieses Problems wird ein neuer Mechanismus namens *Network Size Discovery* vorgeschlagen. Dieses Feature wird von der Standardspezifikation von WS-Discovery nicht angeboten. Demnach sind zusätzliche Spezifikationen notwendig.

Wenn ein neues Gerät dem Netzwerk beitritt und sich mittels einer *Hello*-Nachricht ankündigt, wird es von anderen Geräten über die aktuelle Netzwerkgröße informiert. Alle anderen Geräte sollen zu diesem Zeitpunkt bereits Kenntnisse über die Netzwerkgröße besitzen. Da Automations- und Plug&Play-Netzwerke iterativ aufgebaut werden (einige Geräte werden zuerst angeschlossen), lernen sich die Geräte kennen, während das Netzwerk wächst. Folglich kann eine Bekanntmachung der Netzwerkgröße, die sogenannte NetSize-Nachricht, von jedem Gerät verschickt werden. Diese Nachricht wird als Multicast verschickt. Das hat zwei Vorteile: Erstens bleiben andere Geräte still, wenn einige Benachrichtigungen über die Netzwerkgröße detektiert wurden; zweitens können die Geräte ihren gespeicherten Wert zur Anzahl der Geräte überprüfen.

Wenn alle Geräte gleichzeitig eine Benachrichtigung über die Netzwerkgröße verschicken, wird das Netzwerk mit Multicast-Paketen überflutet. Dies kann zur Überlastung einzelner Netzwerkteilnehmer oder -komponenten führen. Zur Vermeidung hoher Netzwerklast muss dieses Verfahren geregelt werden. Um eine Benachrichtigung senden zu dürfen, muss ein Gerät eine Sendegenehmigung bekommen. Da es sich um ein dezentralisiertes Netzwerk handelt, muss jedes Gerät über die Sendegenehmigung selbst entscheiden. Diese wird durch eine Wahrscheinlichkeit bestimmt. Die Wahrscheinlichkeit, eine Benachrichtigung zu senden, hängt von der Geräteanzahl ab. Mit einer steigenden Geräteanzahl sinkt die Wahrscheinlichkeit. Darüber hinaus sollen Benachrichtigungen von mehreren Geräten in einem zeitlichen Abstand gesendet werden. Der Sendezeitpunkt wird, wie bei WS-Discovery üblich, über ein zufälliges Delay bestimmt. Wenn ein fehlerhaftes Gerät die Sendegenehmigung bekommen und einen falschen Wert senden würde, würde das zu Netzwerkfehlern (byzantinischen Fehlern) führen. Um einen byzantinischen Fehler zu detektieren und zu umgehen, sind mindestens 3 Benachrichtigungen notwendig, wenn das Netzwerk mehr als zwei Teilnehmer hat (Mehrheitsvotum). Falls alle 3 Werte unterschiedlich sind, werden weitere Benachrichtigungen benötigt.

Die Vorgehensweise wird zu einem besseren Verständnis anhand eines Beispiels mit 1000 Knoten erklärt. Alle angeschlossenen Geräte kennen bereits die Netzwerkgröße, die sie während des Netzwerkbeitritts gelernt haben. Ein neues Gerät, das an das Netzwerk angeschlossen wird, verschickt die Hello-Nachricht, um sich bekannt zu machen. Alle anderen Geräte inkrementieren automatisch den Wert für die Geräteanzahl, wenn sie die Hello-Nachricht empfangen. Im nächsten Schritt soll das neue Gerät mittels einer Multicast Network Size Discovery-Nachricht (NetSize-Nachricht) über die aktuelle Netzwerkgröße informiert werden. Wie bereits erwähnt, sind mindestens 3 Antworten nötig, um in großen Netzwerken einen byzantinischen Fehler zu vermeiden. Für diesen Zweck berechnen alle Geräte eine Sendewahrscheinlichkeit, damit 3 Antworten verschickt werden können. Für 1000 Geräte würde diese 0,3 % betragen. Ein einfacher Zufallszahlengenerator wird dafür

verwendet. In der ersten Phase soll die Zufallszeit zwischen 0 und 50 ms gewählt werden. Wenn ein Gerät die Sendegenehmigung bekommt, soll es diese Zeit abwarten, bevor es eine Network Size Discovery-Antwort schickt. In der ersten Phase werden 3 Nachrichten erwartet. Wenn ein Gerät bereits 3 Antworten von anderen Geräten detektiert, soll die eigene Nachricht verworfen werden. Trotz Sendewahrscheinlichkeit und Verwurf von Nachrichten können etwas mehr als 3 Antworten aufgrund der Netzwerkverzögerung verschickt werden. Dieser Ansatz garantiert jedoch, dass das Netzwerk nicht von Antworten überschwemmt wird. Im Fall einer Diskrepanz der gemeldeten Werte, oder wenn nicht genug Geräte eine Sendegenehmigung erhalten haben, wechseln alle Geräte in die zweite Phase. In der zweiten Phase werden die Sendewahrscheinlichkeit (Anzahl der Antworten) und das Zeitintervall verdoppelt. Diese Prozedur kann wiederholt werden, bis der richtige Wert ermittelt ist. Wenn die Anzahl der Geräte kleiner als 3 ist, wird nur die erste Phase benötigt. Obwohl das beschriebene Verfahren solange wiederholt werden kann, bis die Sendewahrscheinlichkeit 100 % erreicht, wird erwartungsgemäß die Network Size Discovery nach einigen wenigen Phasen abgeschlossen. In den meisten Fällen wird bereits die erste Phase ausreichend sein.

Wenn einige Geräte das Netzwerk verlassen, senden diese eine *Bye*-Nachricht. Alle anderen Geräte müssen daraufhin den Wert für die Netzwerkgröße verringern. Wenn ein Gerät abstürzt, nicht ordnungsgemäß ausgeschaltet oder vom Netzwerk getrennt wird, kann es die Bye-Nachricht nicht verschicken. Das stellt allerdings kein Problem dar, da in diesem Fall die tatsächliche Anzahl der Antworten während WS-Discovery etwas geringer als erwartet sein wird. Das hat keinen negativen Einfluss auf den Client.

Für die Network Size Discovery-Prozedur sollen zusammenfassend folgende Aspekte berücksichtigt werden:

- Jedes Gerät soll die Anzahl der Teilnehmer im Netzwerk speichern
- Ein neues Gerät wird von den im Netzwerk vorhandenen Geräten über die Netzwerkgröße mittels einer NetSize-Nachricht informiert
- Für eine erfolgreiche Ermittlung der Netzwerkgröße sind für ein Netzwerk mit mehr als 2 Geräten mindestens 3 NetSize-Nachrichten erforderlich
- Die Prozedur kann bei Bedarf mehrmals wiederholt werden (mehrere Phasen)
- Die NetSize-Nachrichten werden per Multicast verschickt
- Die Sendewahrscheinlichkeit für eine NetSize-Nachricht wird abhängig von der Netzwerkgröße berechnet und soll in 3 verschickten NetSize-Nachrichten resultieren (1. Phase)
- Hat ein sendewilliges Gerät bereits 3 NetSize-Nachrichten protokolliert, soll die eigene NetSize-Nachricht verworfen werden (1. Phase)
- Das anfängliche Zeitintervall für das Senden einer NetSize-Nachricht soll 50 ms betragen (1. Phase)

- Falls weniger als 3 NetSize-Nachrichten verschickt wurden oder bei Diskrepanz der Werte wechseln alle Geräte in die nächste Phase. In jeder nachfolgenden Phase werden die Sendewahrscheinlichkeit und das Zeitintervall verdoppelt.

Der beschriebene Algorithmus ist aus der Sicht eines bereits im Netzwerk vorhandenen Gerätes in Abbildung 69 veranschaulicht. WS-Discovery stellt keinen zuverlässigen Mechanismus zur Verfügung, die gleichzeitigen Hello-Nachrichten von einer großen Anzahl der Geräte zu regulieren. Das kann z.B. nach einem Stromausfall geschehen, wenn alle Geräte gleichzeitig eingeschaltet werden. Hierfür wird eine Optimierung im nächsten Abschnitt vorgeschlagen.

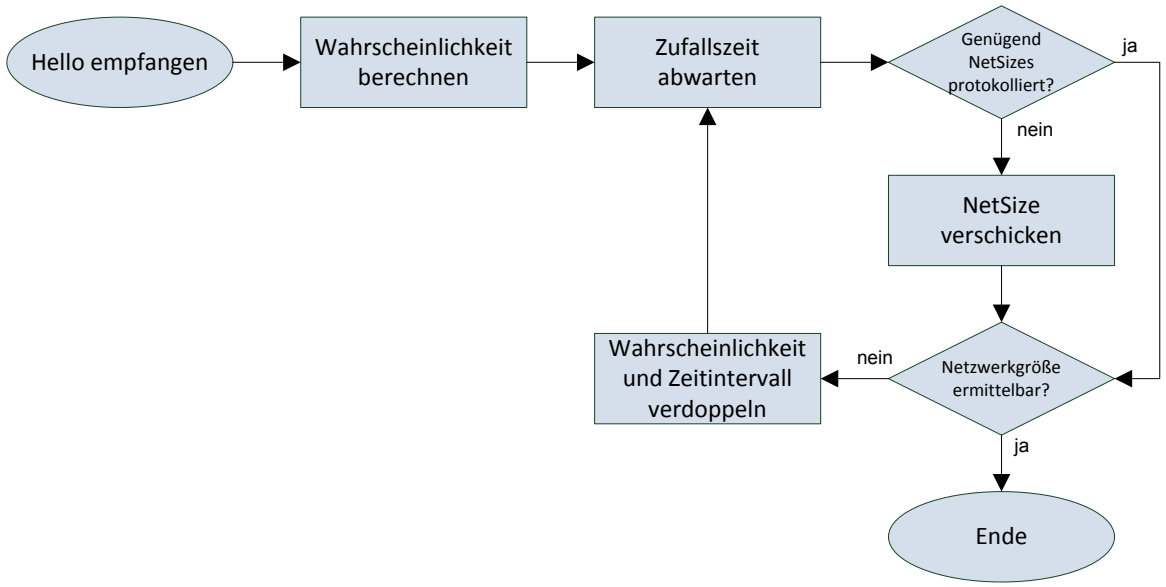

Abbildung 69: Network Size Discovery-Algorithmus aus der Sicht eines bereits im Netzwerk vorhandenen Gerätes

5.9. Optimierung des Startverhaltens

Die Hello-Nachrichten folgen nach der Standard WS-Discovery-Spezifikation demselben Verhalten wie die ProbeMatch-Nachrichten. Vor dem Versand einer Hello-Nachricht soll ebenfalls eine zufällige Verzögerung von standardmäßig 500 ms abgewartet werden. Die Hello-Nachricht soll anschließend nach standardmäßig 50-250 ms wiederholt werden (vgl. Abbildung 67 Standard). Bei einem gleichzeitigen Start mehrerer Geräte unterliegt das Hello-Verfahren denselben Einschränkungen bezüglich Skalierbarkeit wie das ProbeMatch-Verfahren. Dieser Fall ist nicht unwahrscheinlich, wenn beispielsweise ein großes Netzwerk in Betrieb genommen wird oder nach einem Stromausfall wieder hochfährt. Das in Abschnitt 5.8 beschriebene Verfahren zur Übermittlung der Netzwerkgröße würde hierbei eine zusätzliche signifikante Netzwerk- und Geräteauslastung bedeuten, da zusätzlich zu jeder Hello-Nachricht mindestens drei NetSize-Nachrichten kommen würden. Da diese Nachrichten per Multicast verschickt werden, müssen diese von allen Geräten erfasst und ausgewertet

werden, was zu einer zusätzlichen Auslastung aller Geräte führen würde. Darüber hinaus können hier ähnlich wie bei dem Discovery-Prozess hohe Peak-Datenraten entstehen, die die Geräte überlasten und in Worst Case bereits beim Start außer Betrieb setzen können. Darüber hinaus können sich parallel verschickte Hello- und NetSize-Nachrichten gegenseitig beeinflussen, wenn beispielsweise während der Verarbeitung einer NetSize-Nachricht eine neue Hello-Nachricht empfangen wird. Damit sollen die in dieser Phase verschickten Multicast-Nachrichten möglichst zeitversetzt bei den Geräten ankommen.

Es wird deswegen ein optimiertes Startverhalten empfohlen. Bevor sich ein Gerät im Netzwerk mittels einer Hello-Nachricht bekannt macht, soll dieses zunächst eine konstante Zeit abwarten und auf die anderen Hello- oder NetSize-Nachrichten lauschen. Damit können die Hello-Nachrichten zeitlich besser voneinander getrennt werden. Die erwähnte konstante Zeit soll die festgelegte Zeit für den Versand der NetSize-Nachrichten betragen. Wird keine Nachricht in dieser Zeit empfangen, soll wie im Standard definiert die Hello-Nachricht nach einer zufälligen Zeit von 0 bis 500 ms verschickt werden. Wird in dieser Zeit eine Hello- oder NetSize-Nachricht empfangen, soll die konstante Warteperiode erneut abgewartet werden. Die Netzwerkgröße, die mit der NetSize-Nachricht übertragen wurde, soll gespeichert werden. Alle weiteren Hello-Nachrichten sollen vom Gerät selbständig mitgezählt werden. Wenn ein Gerät eine eigene Hello-Nachricht abschicken darf und mindestens eine NetSize-Nachricht empfangen hat, kann auf die NetSize-Anforderung verzichtet werden. Damit können das Netzwerk und die Netzwerk-Teilnehmer entlastet werden. Somit sollen Geräte, die eine NetSize-Nachricht empfangen und noch keine eigene Hello-Nachricht verschickt haben, eine Hello-Nachricht mit einem zusätzlichen NoNetSize-Parameter verschicken. Damit werden alle anderen Geräte informiert, dass diese Hello-Nachricht nicht mit einer NetSize-Nachricht beantwortet werden muss. In einem großen Netzwerk werden dabei keine NetSize-Nachrichten auf jede Hello-Nachricht folgen, wenn das Netzwerk sich beispielsweise nach einem Ausfall wiederherstellt. Die Integration des zusätzlichen Parameters in eine Hello-Nachricht kann wie hier dargestellt erfolgen:

```
<s12:Body>
    <wsd:Hello>
        <wsd:NoNetSize>true</wsd:NoNetSize>
        …
```

Der Aufbau des Netzwerks kann sich durch das vorgeschlagene Verfahren verzögern. Es werden aber durch die vorgeschlagenen Mechanismen die Geräte vor einem weiteren Ausfall geschützt. Der Aufbau eines Netzwerks findet darüber hinaus nur einmalig statt und stellt daher keine zeitkritische Operation dar.

Ein weiterer Aspekt während der Initialisierung des Netzes ist die Bestimmung des ersten Gerätes. Falls ein Gerät eine Hello-Nachricht verschickt und darauf keine Antwort erhält,

nimmt dieses an, dass es das erste Gerät im Netzwerk ist. Damit wird der NetSize-Parameter initialisiert und das Gerät ist ein Initiator des Netzwerks. Den beschriebenen Ablauf verdeutlicht Abbildung 70.

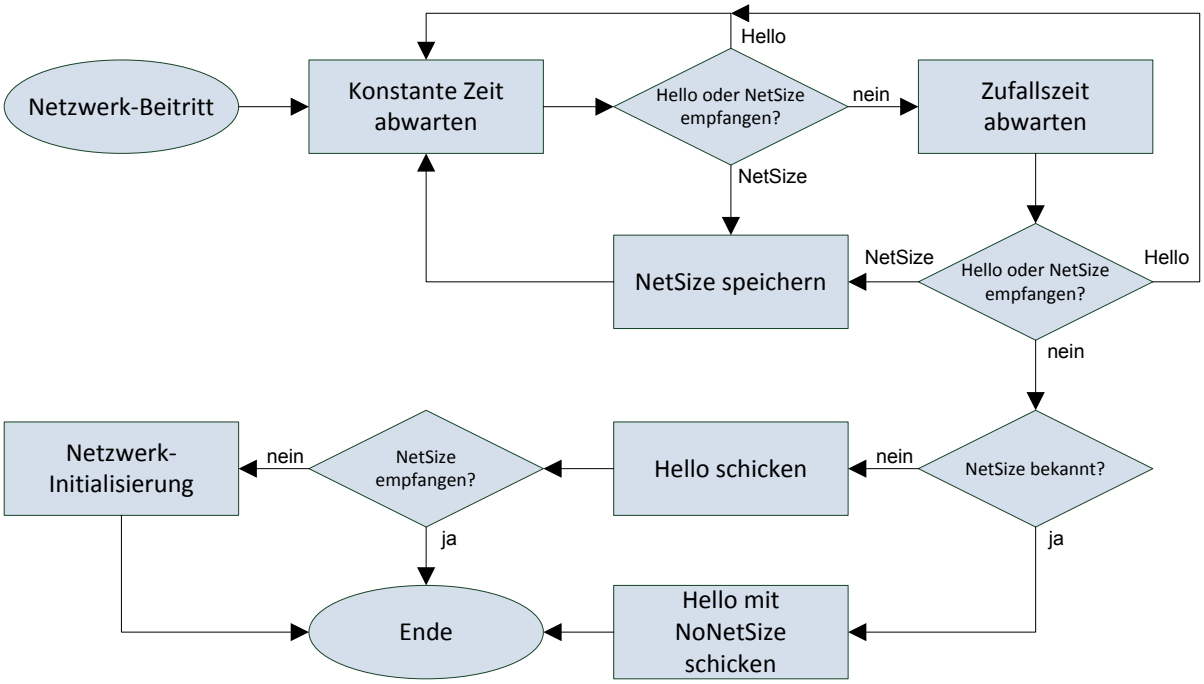

Abbildung 70: Optimierung des Startverhaltens

5.10. Zusammenfassung: 6-Schritte-Ansatz zur Reduzierung des Datenverkehrs beim Discovery

1. Einfügen der Hardware Adresse

Ziel: Reduzierung der Anzahl der Nachrichten durch das Einfügen von MAC-Adressen in die Discovery-Pakete. Dadurch wird die Versendung von ARP-Paketen vermieden.

Ergebnis: Halbierung der Gesamtanzahl verschickter Nachrichten.

2. Dynamisches Delay

Ziel: Vermeidung hoher Datenraten, die vom Client nicht verarbeitet werden und diesen u.U. außer Betrieb setzen können.

Ergebnis: Die Datenraten können dynamisch an die Leistungsfähigkeit des Clients angepasst werden.

3. **Knotengruppierung**

Ziel: Reduzierung der Peak-Datenraten und ein ausbalanciertes Verhalten von WS-Discovery.

Ergebnis: Die Peak-Datenraten werden auf ein Minimum reduziert und liegen nur geringfügig über dem Durchschnittswert.

4. **Optimierung der Paketwiederholung**

Ziel: Gewährleistung einer minimalen Peak-Datenrate trotz Paketwiederholung.

Ergebnis: Eine minimale Peak-Datenrate wird eingehalten, da das erste und zweite Antwortintervall voneinander getrennt werden, wodurch zusätzliche Kollisionen vermieden werden können.

5. **Network Size Discovery**

Ziel: Bestimmung der Anzahl der Geräte.

Ergebnis: Mit dem Wissen über die Anzahl der Geräte wird der Einsatz der beschriebenen Optimierungen ermöglicht.

6. **Optimierung des Startverhaltens**

Ziel: Vermeidung von hohem Datenverkehr bei einem gleichzeitigen Start vieler Geräte.

Ergebnis: Die Anmeldung von Geräten im Netzwerk wird verzögert hintereinander ausgeführt, wodurch hohe Datenraten vermieden werden.

5.11. Feldversuch

Um die entwickelten Mechanismen praktisch zu testen sowie die Simulationsergebnisse zu bestätigen, wurde im Rahmen dieser Arbeit ein Feldversuch mit realen Geräten durchgeführt. Für die Implementierung der DPWS-Geräte wurde der Java Multi Edition DPWS Stack (JMEDS) als Grundlage herangezogen [160]. Um Aussagen über das Verhalten von WS-Discovery zu treffen, werden alle Dienstanbieter auf der gleichen Hardware-Plattform ausgeführt. Um praxisnahe Ergebnisse zu erhalten, wird Hardware aus dem Bereich der eingebetteten Systeme eingesetzt. Somit wurden zur Durchführung des Feldversuchs Einplatinencomputer Raspberry Pi Model B eingesetzt. Die Spezifikationen des Raspberry Pi sind in Tabelle 18 aufgeführt [161]. Als Betriebssystem kam Raspbian Wheezy Linux zum Einsatz.

In der Simulation wurden die Optimierungen für WS-Discovery auf dem Übertragungsmedium Ethernet getestet. In dem Feldversuch sollen darüber hinaus andere

Medien zum Einsatz kommen, um einen allgemeingültigen Charakter und die Effizienz der vorgeschlagenen Optimierungen für WS-Discovery zu überprüfen. Als andere Übertragungsmedien wurden WLAN und HomePlug in Betracht gezogen.

Größe	85,60 mm × 53,98 mm × 17 mm
SoC	Broadcom BCM2835
CPU	ARM1176JZF-S (700 MHz)
GPU	Broadcom VideoCore IV
RAM	512 MB
Videoausgabe	FBAS, HDMI
Netzwerk	10/100-MBit-Ethernet-Controller
Schnittstellen	USB, 17 GPIO-Pins, SPI, I^2C, UART
ROM (Flash)	SD (SDHC und SDXC)
Leistungsaufnahme	max. 3,5W (700mA)
Stromversorgung	5 V

Tabelle 18: Raspberry Pi-Spezifikationen (Model B)

Das Netzwerk wird aus 20 Raspberry Pi aufgebaut. Die physikalische Vernetzung der Geräte hängt dabei von dem jeweiligen Übertragungsmedium ab. Als eine Testanwendung wurde ein Klimaanlagen-Service gewählt. Der Service wurde auf Basis von JMEDS implementiert. Die Service-Suche wird von einem Client aus gestartet, der auf einem PC mit Intel Pentium 4 CPU 3,2 GHz, 2 GB Arbeitsspeicher und Windows 7 Betriebssystem läuft. Auf diesem wird das Tool DPWS Explorer ausgeführt [162], das ebenfalls auf JMEDS basiert [160]. Das Tool ist grundsätzlich für Entwickler gedacht und ist in der Lage, nach DPWS-Geräten und Services zu suchen. Es kann Probe-Nachrichten verschicken und die Antworten von Geräten in einer grafischen Benutzerschnittstelle darstellen. Das Tool unterstützt ebenfalls alle anderen Nachrichtentypen wie Hello, Bye sowie Service Invocation.

Für die Versuchsanordnung werden mehrere Szenarien definiert. Dabei werden die Optimierungen für WS-Discovery in Bezug auf Skalierbarkeit untersucht. Um das Verhalten von WS-Discovery für eine unterschiedliche Anzahl an Netzwerkteilnehmern zu testen, sind mehrere logische Geräte pro physikalisches Gerät notwendig. Da die zu testenden Optimierungen nicht Teil des JMEDS-Frameworks sind, wurde der Test-Service als eine separate Java-Applikation umgesetzt und mit dem JMEDS-Framework die DPWS-Funktionalität validiert. Da im Rahmen des Versuchs nur WS-Discovery von Interesse ist, wurde nur diese Funktionalität umgesetzt. Die Testanwendung stellt somit eine vereinfachte Version von JMEDS dar, in der jedoch die zu testenden Parameter leicht geändert werden können. Es können auf einem physikalischen Gerät beliebig viele logische Geräte gestartet werden. Hierbei wird jedoch bei den Testversuchen auf die gleichmäßige Verteilung der logischen Geräte auf die physikalische Geräte geachtet, um die Auslastung der Geräteressourcen auszubalancieren. Die IP-Adressen der Geräte werden mittels eines DHCP-Servers zugewiesen. Da die Optimierungen Knotengruppierung sowie Network Size

Discovery unterschiedliche IP-Adressen der Geräte voraussetzen, werden fiktive IP-Adressen eingesetzt. Die fiktiven IP-Adressen werden jedem logischen Gerät zugewiesen. Nachfolgend werden die konzeptionellen Anordnungen für jedes Übertragungsmedium beschrieben.

In Abbildung 71 ist die Versuchsanordnung für die Vernetzung über Ethernet dargestellt. Hierbei wird die heute am meisten verbreitete Version Switched Ethernet verwendet. Die Geräte werden an die Switches angeschlossen und die Switches untereinander ebenfalls verbunden. Diese Versuchsanordnung hat eine Baumtopologie. Da jedes Gerät eine dedizierte Leitung zum Switch hat, sind Paketkollisionen ausgeschlossen. Falls mehrere Pakete gleichzeitig bei dem Switch ankommen, werden sie in die Empfangsqueues eingereiht. Je nach Switch-Implementierung werden die Pakete der Reihe nach aus den Empfangsqueues in die Sendequeues übernommen. Hierdurch können Verzögerungen entstehen bzw. die Paketreihenfolge kann vertauscht werden. Die Übertragungsdatenrate von Ethernet ist auf 100 Mbit/s begrenzt, da der Ethernet-Anschluss des Raspberry Pi maximal den 100BASE-T Ethernet Standard unterstützt.

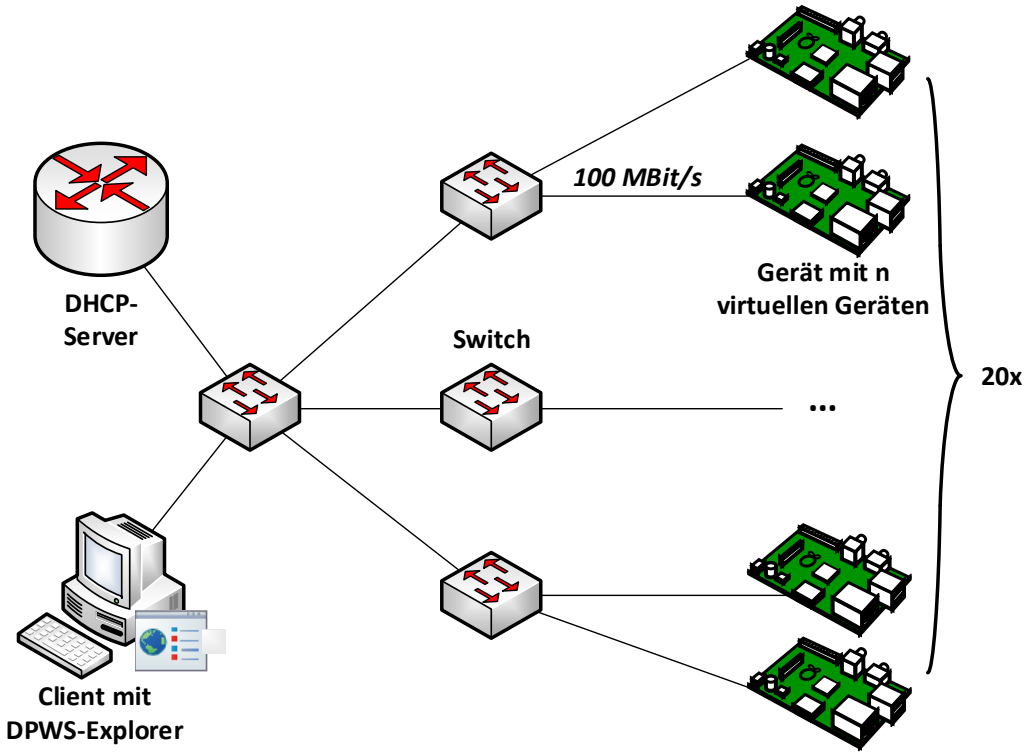

Abbildung 71: Versuchsanordnung für die Ethernet-Vernetzung

In Abbildung 72 ist die Versuchsanordnung für die Verbindung über WLAN dargestellt. Im Zentrum eines WLAN-Netzes befindet sich ein WLAN AP, der gleichzeitig die Rolle eines DHCP-Servers übernimmt. Zur Kommunikation findet der WLAN-Standard IEEE-802.11g mit einer Datenrate von 54 Mbit/s Verwendung. Im Gegensatz zu Ethernet teilen sich alle Netzwerkteilnehmer dasselbe Medium, sodass nur ein Gerät das Medium zum jeweiligen

Zeitpunkt beanspruchen kann. Wenn mehrere Geräte gleichzeitig Pakete schicken wollen, müssen sie in einen Wettbewerb um das Medium eintreten (Contention Window). Geräte, die den Wettbewerb nicht gewonnen haben, müssen auf das nächste Contention Window warten und erneut in den Wettbewerb eintreten. Der IEEE 802.11 Standard schreibt zwar ein CSMA/CA-Verfahren vor, jedoch sind Kollisionen nicht gänzlich ausgeschlossen (vgl. Abschnitt 2.2.2). Die Kollisionswahrscheinlichkeit kann durch das optionale RTS/CTS-Verfahren weiter reduziert werden. Es wird aber in der Regel von handelsüblichen WLAN APs standardmäßig nicht eingesetzt. Bei steigender Geräteanzahl erhöhen sich der Wettbewerb um das Medium und die Kollisionswahrscheinlichkeit.

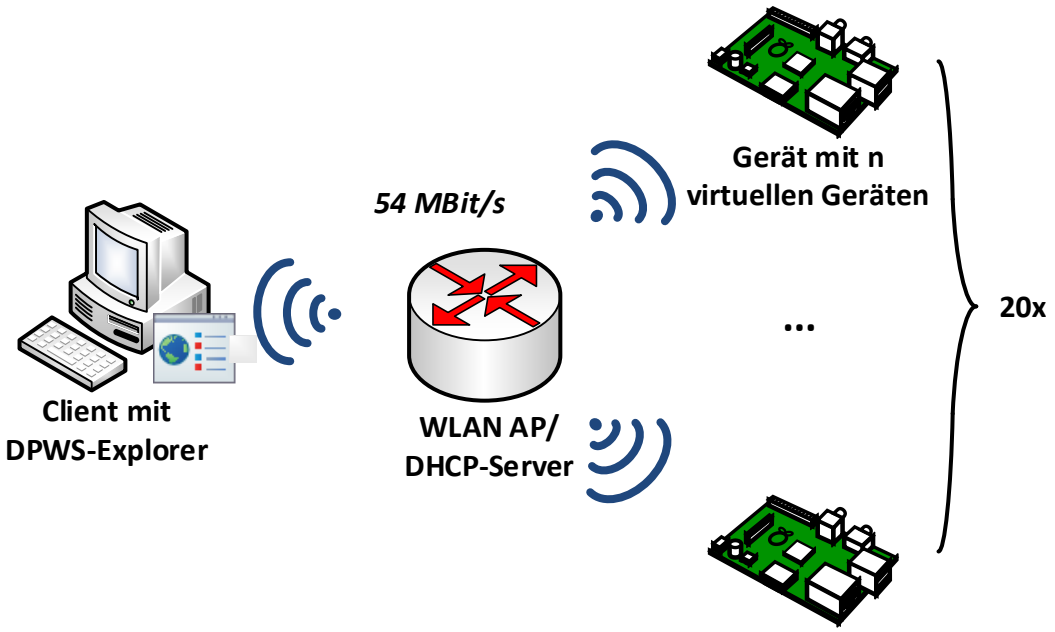

Abbildung 72: Versuchsanordnung für die WLAN-Vernetzung

In Abbildung 73 ist die Versuchsanordnung für HomePlug dargestellt. Die HomePlug-Schnittstelle befindet sich nicht unmittelbar in den Geräten selbst, sondern wird mittels Adapter realisiert. Die Kommunikation mit dem Adapter erfolgt dabei über Ethernet. Für die Powerline-Kommunikation wird die Technologie HomePlug AV2 eingesetzt (vgl. Abschnitt 2.2.3). Die eingesetzten Adapter ermöglichen Kommunikationsraten von bis zu 500 Mbit/s. Aus der Sicht des Gerätes ist die Kommunikation über das HomePlug transparent und unterscheidet sich nicht von der Kommunikation über Ethernet. Ähnlich wie bei WLAN kommunizieren die Geräte über ein gemeinsames Medium, sodass nur ein Gerät zu einem bestimmten Zeitpunkt auf das Medium zugreifen darf. Die Zugriffssteuerung wird dabei komplett vom Adapter übernommen. HomePlug AV2 verwendet ebenfalls das CSMA/CA-Verfahren wie WLAN. Die steigende Geräteanzahl führt hier ebenfalls zum steigenden Wettbewerb und einer höheren Auslastung des Mediums.

Für die Steuerung des Versuchsablaufs wurden spezielle Shell-Skripte entwickelt. Diese ermöglichen es, die eingestellten Versuchsparameter auf die Geräte zu übertragen sowie zeitnahe bzw. zeitgesteuerte Starts von virtuellen Geräten zu bewirken. Darüber hinaus können zur Laufzeit Ausgaben einzelner Geräte angefordert werden. Virtuelle Geräte können zur Laufzeit aufgefordert werden, ihre Teilnahme am WS-Discovery zu beenden.

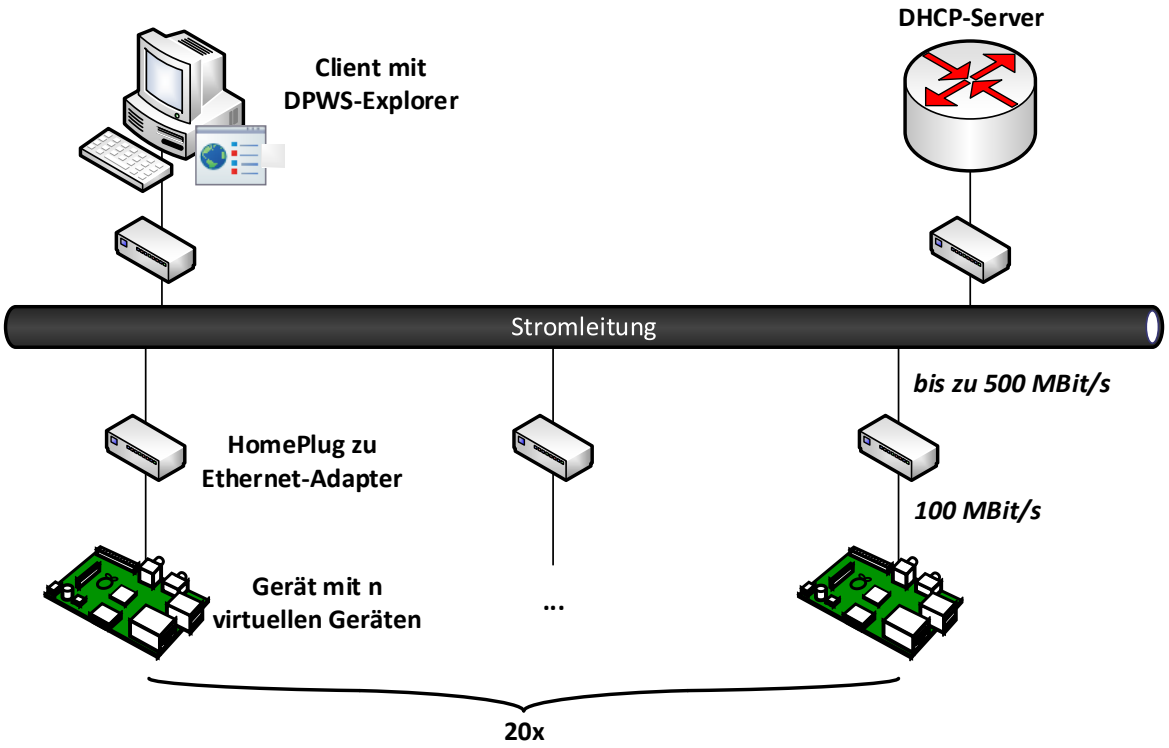

Abbildung 73: Versuchsanordnung für die Powerline-Vernetzung

Die oben beschriebenen Testszenarien wurden für Netzwerke mit 20, 40, 60, 80 und 100 Teilnehmern durchgeführt. Es wurden zusätzlich vereinzelte Untersuchungen mit bis zu 500 Teilnehmern durchgeführt.

5.11.1. Auswirkung von ARP

Wie in Abschnitt 5.4 beschrieben wurde, führt die Verwendung von ARP zu einer höheren Auslastung des Gerätes durch den Empfang von ARP-Requests und die Beantwortung dieser mit ARP-Responses. Obwohl die ARP-Nachrichten eine zusätzliche Belastung des Gerätes hervorrufen, tragen sie wenig zu der sichtbaren anfallenden Datenrate bei, da ARP-Pakete mit 60 Byte deutlich kleiner als Discovery-Pakete mit über 1 KB sind. Für die Messung der Auswirkung durch die ARP-Optimierung wird deswegen die anfallende Anzahl an Paketen und nicht die Datenrate ausgewertet. Die Messung wurde nur für Ethernet ausgeführt, da hierbei nur die Anzahl der Pakete und nicht die tatsächliche Datenrate von Interesse ist. In Abbildung 74 ist die Messung des ARP-Einflusses für 20 Geräte dargestellt. Es ist erkennbar, dass ARP-Pakete eine zusätzliche Beeinträchtigung der Rechenleistung bei Lastspitzen

verursachen. Die Übertragung der Hardware-Adressen, wie es in Abschnitt 5.4 beschrieben wurde, würde die Geräte entlasten und die Skalierbarkeit von WS-Discovery verbessern.

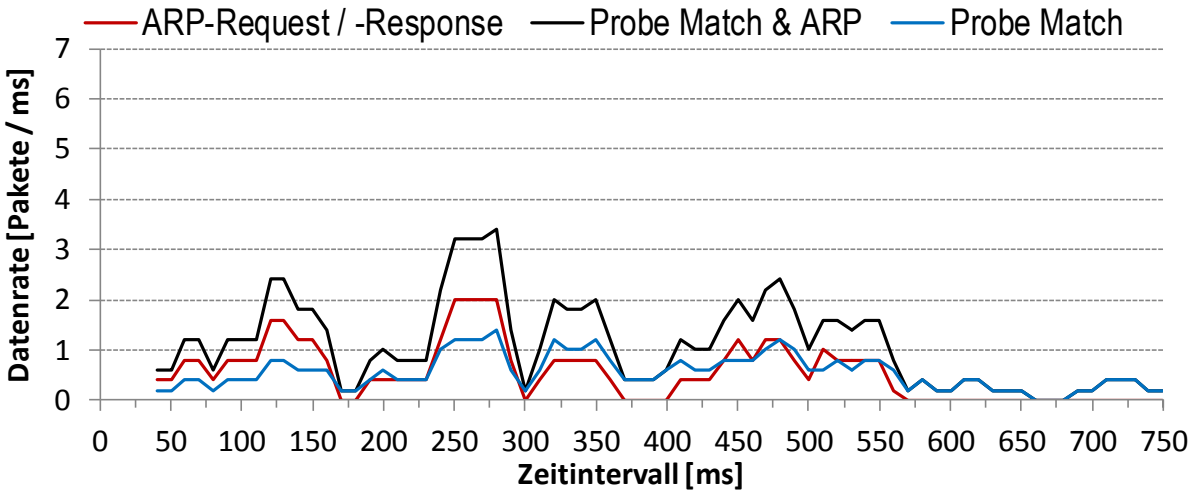

Abbildung 74: Einfluss von ARP-Nachrichten

5.11.2. Vergleich im identischen Zeitintervall

Zunächst werden die Optimierungen im gleichen Zeitintervall betrachtet. Hierbei wird der Einfluss des Mediums auf das Verhalten des WS-Discovery untersucht. Wie in Abbildung 68 gezeigt, reduzieren die vorgeschlagenen Optimierungen die Peak-Datenrate bei gleichbleibendem Intervall deutlich, sodass die resultierende Datenrate nahe dem Optimum liegt. Der Versuch wurde für 100 Teilnehmer durchgeführt. Als Grundlage wurde ebenfalls wie in der Simulation ein Standardzeitintervall von 750 ms verwendet, die Paketgröße betrug 1115 Byte. Die theoretische mittlere Datenrate liegt dabei bei ca. 2,268 Mbit/s. Bei allen nachfolgenden Messungen mit den Optimierungen wurde der negative Einfluss von ARP bereits durch die Übertragung der Hardware-Adresse beseitigt.

Ethernet

Zunächst wird der Vergleich für das Übertragungsmedium Ethernet durchgeführt. In Abbildung 75 sind die Messungen für das Standard WS-Discovery und ein optimiertes WS-Discovery dargestellt. Das gemessene Verhalten des WS-Discovery ist dem in der Simulation ermittelten Verhalten sehr ähnlich. Die höchste Datenrate entsteht beim Standard WS-Discovery wie erwartet in der Mitte des Intervalls, wo sich viele originale ProbeMatch-Nachrichten mit den Wiederholungsnachrichten überlagern. Zum Ende des Intervalls nimmt die Datenrate ab, da hier nur noch Wiederholungsnachrichten verschickt werden bzw. die Anzahl der Geräte, die noch keine Wiederholungsnachricht verschickt haben, stetig abnimmt. Es ist deutlich zu erkennen, dass die vorgeschlagenen Optimierungen die Datenrate nahezu

auf das gewünschte Niveau der mittleren Datenrate bringen. Wie ebenfalls bei der Simulation festgestellt wurde, erreicht die Peak-Datenrate bei dem nicht optimierten WS-Discovery etwa den doppelten Wert der theoretischen mittleren Datenrate. Für die durchgeführte Messung konnte die Peak-Datenrate um ca. 35 % reduziert werden. Die optimierte Lösung weist darüber hinaus nur einige wenige Lastspitzen im Gegensatz zur nicht optimierten Lösung auf. Somit wurde gezeigt, dass die vorgeschlagenen Optimierungen für das Übertragungsmedium Ethernet die erwarteten Verbesserungen bezüglich Datenrate bzw. Skalierbarkeit bringen.

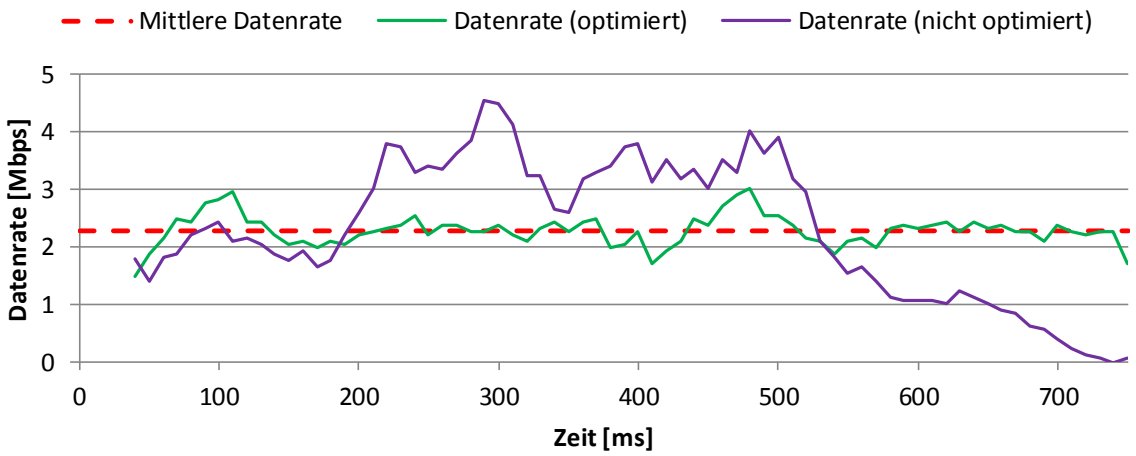

Abbildung 75: Gemessene Datenrate von WS-Discovery mit und ohne Optimierungen für 100 Teilnehmer im Ethernet-Netzwerk

WLAN

Der zweite Vergleich wird für das Übertragungsmedium WLAN durchgeführt. Wie bereits beschrieben, unterscheidet sich der Medienzugriff bei WLAN deutlich von dem des Ethernets. Während bei Ethernet CSMA/CD Verwendung findet, wird bei WLAN CSMA/CA eingesetzt. Die Messung erfolgte unter den gleichen Bedingungen wie für Ethernet. Die Messergebnisse für das nicht optimierte Verfahren und für das optimierte Verfahren sind in Abbildung 76 dargestellt.

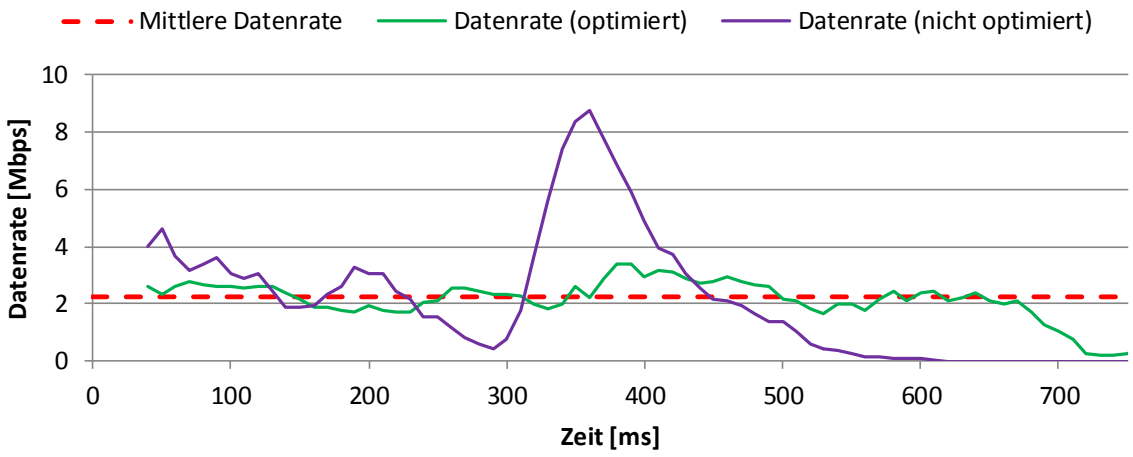

Abbildung 76: Gemessene Datenrate von WS-Discovery mit und ohne Optimierungen für 100 Teilnehmer im WLAN-Netzwerk

In Abbildung 76 ist klar erkennbar, dass die vorgeschlagenen Optimierungen das Verhalten von WS-Discovery in WLAN-Netzwerken deutlich verbessern. Im Gegensatz zu Ethernet können die Pakete nicht parallel verschickt werden. Die Stationen, die einen Sendewunsch haben, müssen darauf warten, den Zugriff auf das Medium zu bekommen. Die Lastspitzen führen zur höheren Auslastung des Mediums und noch höheren Lastspitzen. Daher entsteht bei WLAN eine hohe Lastspitze in der Mitte des Zeitintervalls. Diese ist um Faktor vier höher als die mittlere Datenrate. Erst wenn die Anzahl der sendewilligen Stationen abnimmt, sinkt auch die Datenrate. Im Gegensatz dazu konnte eine gleichmäßige Verteilung der Pakete durch die Optimierungen erreicht werden. Für das WLAN-Netzwerk ließ sich die Peak-Datenrate um ca. 60 % reduzieren.

HomePlug

Der nächste Vergleich der Datenraten erfolgt für das Übertragungsmedium Powerline mit dem Übertragungsstandard HomePlug AV2. HomePlug verwendet ähnlich wie WLAN ein CSMA/CA-Medienzugriffsverfahren. Aufgrund einer beschränkten Reichweite der HomePlug-Adapter und einer beschränkten Anzahl Steckdosen im Versuchslabor mussten einige Geräte mittels Ethernet Switches überbrückt werden. Die Messung stellt somit keine reine HomePlug-Messung dar, sondern eine gemischte Messung aus Ethernet und HomePlug. Die Messergebnisse für das nicht optimierte Verfahren und für das optimierte Verfahren zeigt Abbildung 77.

Die Vermischung unterschiedlicher Technologien spiegelt sich auch am Ergebnis wider. Dank des Ethernet-Anteils werden keine so hohen Spitzen wie bei WLAN erreicht. Mehrere Lastspitzen schmelzen allerdings dank Powerline zusammen. Die Messung bestätigt weiterhin die Effektivität der Optimierungen. In der Powerline-Versuchsanordnung konnte eine Reduzierung der Peak-Datenrate ebenfalls um ca. 30 % erreicht werden.

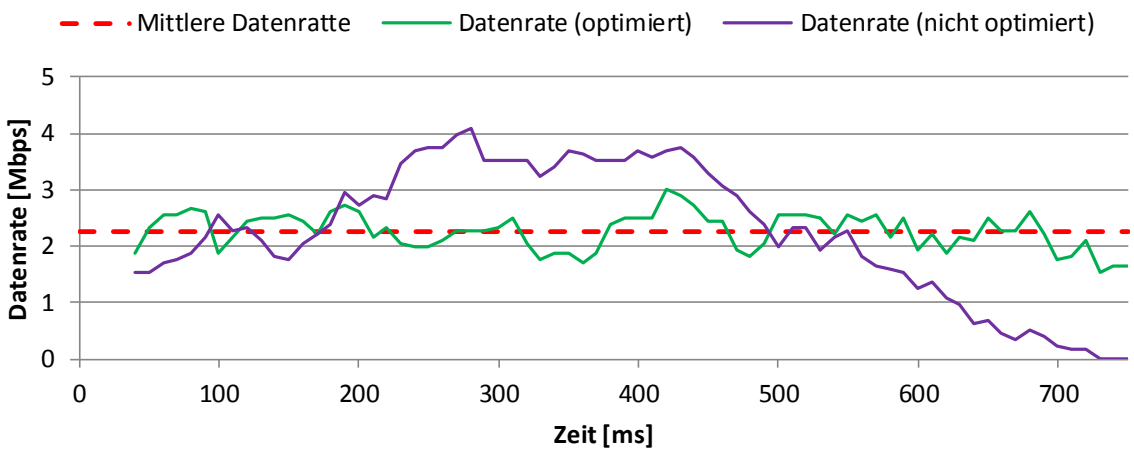

Abbildung 77: Gemessene Datenrate von WS-Discovery mit und ohne Optimierungen für 100 Teilnehmer im Powerline-Netzwerk

5.11.3. Dynamische Einstellung der Datenrate

In diesem Versuch wird die dynamische Einstellung der Datenrate getestet. Als Vorgabe wurde eine Datenrate von 1 Mbit/s gewählt. Das Netzwerk besteht weiterhin aus 100 Teilnehmern und die Paketgröße beträgt 1115 Byte. Das führt nach Formel (14) zu einem Delay von 1784 ms. Im Folgenden wird die beschriebene Versuchsanordnung für die Übertragungsmedien Ethernet, WLAN und HomePlug getestet.

Ethernet

In Abbildung 78 ist die Messung für das Ethernet dargestellt. Wie hier ersichtlich, ermöglichen die vorgeschlagenen Optimierungen, die Datenrate auf das gewünschte Niveau einzustellen. Die prozentual höheren Abweichungen vom Mittelwert hängen mit der geringeren Datenrate und der gleichbleibenden Paketgröße zusammen, da einzelne Pakete hierbei einen höheren Einfluss auf die Abweichung haben. Die praktischen Ergebnisse bestätigen dabei die Simulationsergebnisse für Ethernet und zeigen die Effektivität der vorgeschlagenen Optimierungen. Die Wirkung wurde ebenfalls in DPWS-Explorer sichtbar. So wurden bei dem nicht optimierten Verfahren zwischen 70 und 90 Service Provider gefunden. Bei der optimierten WS-Discovery wurden stets alle 100 Teilnehmer angezeigt.

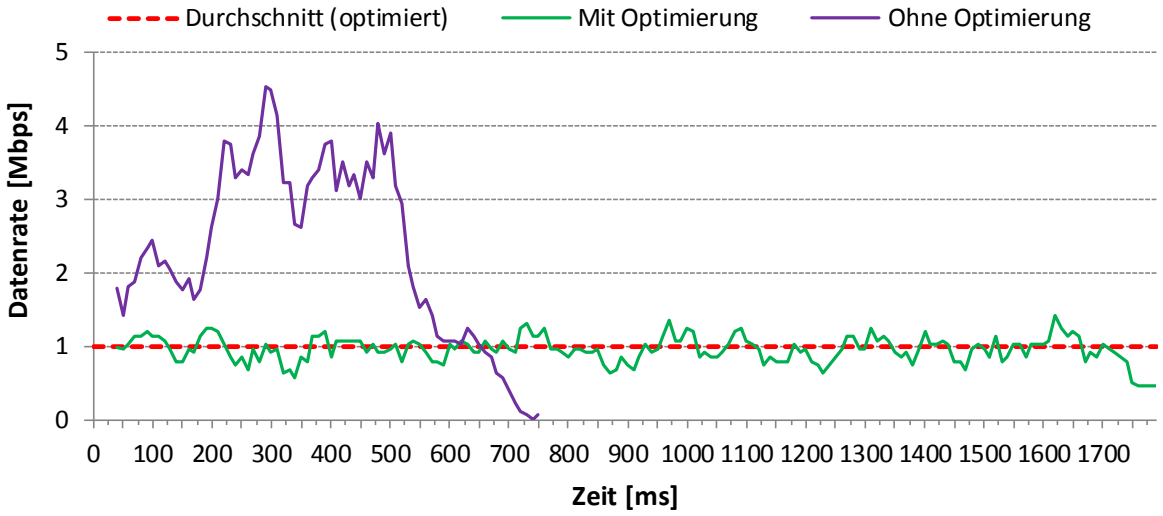

Abbildung 78: Dynamische Datenrate von WS-Discovery im Ethernet-Netzwerk

WLAN

In Abbildung 79 ist die Messung für WLAN dargestellt. Die Messung bestätigt die Effektivität der Optimierungen für das Übertragungsmedium WLAN. Die eingestellte Datenrate wird durch die Optimierungen angenähert. Die Wirkung der Optimierungen konnte in DPWS-Explorer ebenfalls beobachtet werden. Während bei dem Standard-WS-Discovery stets unter 60 Service Provider gefunden werden, wurden bei dem optimierten Verfahren stets alle Geräte gefunden. Hier wird ebenfalls deutlich, dass höhere Peak-Datenraten einen wesentlichen Einfluss auf den suchenden Client haben.

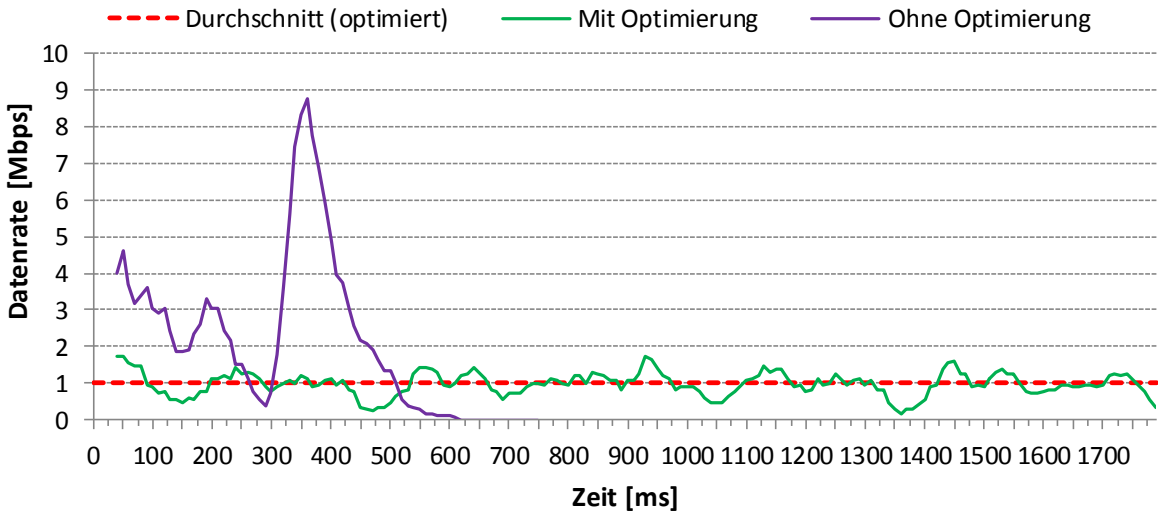

Abbildung 79: Dynamische Datenrate von WS-Discovery im WLAN-Netzwerk

HomePlug

Abschließend wird eine weitere Messung für HomePlug durchgeführt. Wie erwähnt, werden die Geräte bei dieser Versuchsanordnung nicht ausschließlich über Powerline verbunden,

sondern zum Teil auch über Ethernet. In Abbildung 80 ist die Messung ersichtlich. Für die HomePlug-Versuchsanordnung kann die Effektivität der Optimierungen auch für gemischte Netzwerke bestätigt werden. Bei der Erfassung der Service Provider mit dem DPWS-Explorer konnten beim Standard-WS-Discovery unter 90 Teilnehmer gefunden werden, während bei dem optimierten Verfahren erneut alle 100 Geräte gefunden wurden.

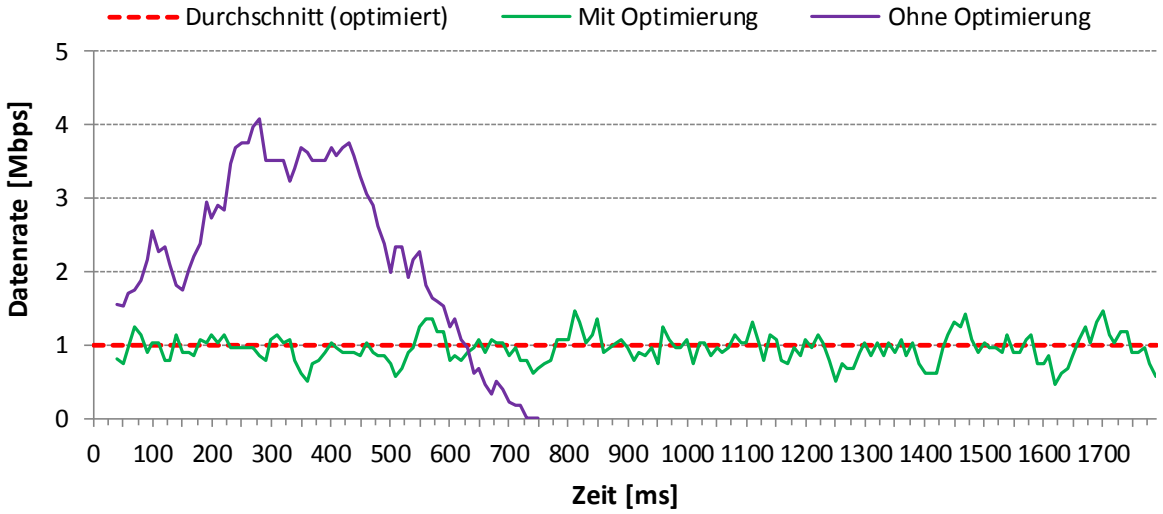

Abbildung 80: Dynamische Datenrate von WS-Discovery im Powerline-Netzwerk

5.11.4. Bestimmung der Anzahl der Teilnehmer

Wie in Abschnitt 5.8 beschrieben, wird die Anzahl der Teilnehmer in einem Netzwerk mittels der NetSize-Nachricht bestimmt. Wenn das Netzwerk mehr als zwei Teilnehmer hat, sind mindestens drei Nachrichten notwendig. Damit können byzantinische Fehler umgangen werden. Die NetSize-Nachricht wird als Antwort auf eine Hello-Nachricht verschickt. Damit nicht alle Teilnehmer antworten, wird der Versand von NetSize-Nachrichten durch eine Sendewahrscheinlichkeit begrenzt. Die Sendewahrscheinlichkeit hängt von der Anzahl der Teilnehmer im Netzwerk ab und kann deswegen variieren. Sie wird so angepasst, dass drei Nachrichten bei dem Empfänger zu erwarten sind. Damit die NetSize-Nachrichten nicht gleichzeitig verschickt werden, müssen die Geräte eine zufällige Zeit ähnlich wie bei der ProbeMatch-Nachricht abwarten. Die NetSize-Nachrichten werden darüber hinaus mittels Multicast verschickt. Wenn bereits ausreichend viele NetSize-Nachrichten verschickt wurden, sollen andere Stationen das Senden einstellen. Wenn weniger als drei Nachrichten verschickt wurden, wechseln alle Stationen in die nächste Phase, in der sowohl die Wahrscheinlichkeit als auch das Sendeintervall verdoppelt werden.

Der nachfolgende Versuch soll das Verhalten vom NetSize-Mechanismus in realen Netzwerken zeigen. Hierbei sind zwei Parameter von Bedeutung: die Wahrscheinlichkeit

eines Fehlschlags, wenn weniger als drei Nachrichten verschickt wurden, und die durchschnittliche Anzahl der verschickten Nachrichten während des NetSize Discovery-Prozesses. Der Versuch wurde mit 20 Geräten durchgeführt, dabei wurden 200 Durchläufe realisiert. Die Tabelle 19 zeigt die Wahrscheinlichkeit eines Fehlschlags für die jeweilige Phase. Im durchgeführten Versuch wurde das NetSize-Verfahren spätestens in der dritten Phase beendet.

Phase	Fehlschlagswahrscheinlichkeit
1	41,5 %
2	2,5 %
3	0,0 %

Tabelle 19: Fehlschlagswahrscheinlichkeit von NetSize-Discovery

Es wurde des Weiteren die durchschnittliche Anzahl der insgesamt erhaltenen NetSize-Nachrichten während des NetSize Discovery-Prozesses ermittelt. Sie beträgt für den durchgeführten Versuch 3,76 Nachrichten. Die erwartete Zeit für die Durchführung eines NetSize Discovery-Prozesses kann somit nach der Formel (18) ermittelt werden. Hierbei ist t die Gesamtlaufzeit eines NetSize Discovery-Prozesses, t_i ist das Zeitintervall für die jeweilige Phase und p_i die Wahrscheinlichkeit der Ausführung der jeweiligen Phase.

$$E(t) = \sum_{i=0}^{n} t_i * p_i \tag{18}$$

Das Initialintervall wurde mit 700 ms angenommen. Demnach beträgt die erwartete Gesamtlaufzeit des NetSize Discovery-Prozesses 1334 ms.

5.11.5. Zusammenfassung des Feldversuchs

Im Rahmen des Feldversuchs wurden die entwickelten Optimierungen in realen Netzwerken erprobt. Hierfür wurden mehrere Netzwerke auf Basis verschiedener Übertragungsmedien realisiert. Die Optimierungen wurden für Ethernet, WLAN und HomePlug getestet. Es wurde festgestellt, dass die Optimierungen unabhängig vom Medium effektiv die Peak-Datenrate reduzieren können. Die gewünschte Datenrate kann dabei mittels des dynamischen Delays eingestellt werden. Damit wurde gezeigt, dass die vorgeschlagenen Optimierungen sowohl in der Simulation als auch in realen Netzwerken zur gewünschten Verbesserung der Skalierbarkeit von WS-Discovery führen.

5.12. Zusammenfassung der Optimierung von WS-Discovery für große Netzwerke

In diesem Kapitel wurden anhand der Simulation Optimierungen für WS-Discovery vorgeschlagen, die die Skalierbarkeit wesentlich erhöhen. Der vorgeschlagene Ansatz fügt MAC-Adressen in WS-Discovery-Anfragen ein, wodurch ein hohes Datenaufkommen durch das ARP-Protokoll vermieden wird. Darüber hinaus wurde eine dynamische Anpassung des Delays vorgenommen, um die Peak-Datenraten am Client zu reduzieren. Eine zusätzliche Knotengruppierung sowie eine Optimierung der Paketwiederholung können die Datenraten weiterhin unabhängig vom Delay um über 50 % reduzieren. Um dynamische Parameter zur Laufzeit setzen zu können, sind Kenntnisse über die aktuelle Netzwerkgröße erforderlich. Für diesen Zweck wurde im Rahmen dieser Arbeit ein neues Verfahren namens Network Size Discovery entwickelt, mit dessen Hilfe die Netzwerkgröße ermittelt werden kann.

Bei der Inbetriebnahme eines großen Netzwerks können ähnliche Probleme wie bei WS-Discovery auftreten, wenn sich alle Geräte im Netzwerk bekanntmachen. Hierfür wurden ebenfalls Optimierungen vorgeschlagen, die das Startverhalten eines großen Netzwerks deutlich verbessern. Alle vorgeschlagenen Optimierungen entlasten wesentlich den Client und ermöglichen ihm eine gleichmäßige Paketverarbeitung. Sie können dank ihres dynamischen Charakters in Netzwerken beliebiger Größe verwendet werden und dabei die Skalierbarkeit des Ad-hoc-WS-Discovery wesentlich verbessern. Darüber hinaus sind die vorgeschlagenen Optimierungen zu der Standard-WS-Discovery-Spezifikation abwärtskompatibel.

Um die aus den Simulationen gewonnenen Erkenntnisse in der Praxis zu überprüfen, wurden mehrere Feldversuche durchgeführt. Hierfür wurde das optimierte WS-Discovery auf realen Geräten implementiert. Das Verhalten des WS-Discovery wurde dabei für verschiedene Medien wie Ethernet, WLAN und Powerline untersucht. Für die Durchführung der Versuche wurden viele physikalische Geräte verwendet. Die Experimente haben gezeigt, dass die vorgeschlagenen Optimierungen die Skalierbarkeit der WS-Discovery auch für reale Geräte wesentlich verbessern. Das gemessene Verhalten des optimierten WS-Discovery entspricht dem der Simulation. Hierdurch konnte die Wirksamkeit der Optimierungen auch für reale Netzwerke bestätigt werden.

6. Geräte- und Service-Discovery mittels Distributed Hash Table

6.1. Motivation

Obwohl im vorhergehenden Kapitel die vorgeschlagene Lösung die Skalierbarkeitsprobleme löst, basiert sie weiterhin auf Multicast und unterliegt dessen Einschränkungen. Demnach können nur Geräte im lokalen Netzwerk gefunden werden. Weiterhin verursacht das Fluten des Netzwerks einen höheren gemeinsamen Energieverbrauch, da die Multicast-Pakete von allen Weiterleitungsknoten wie Switches und Netzwerkteilnehmern verarbeitet und ausgewertet werden müssen.

In dieser Arbeit wird ein alternativer Ansatz vorgestellt, der nicht auf Multicast sondern auf Unicast basiert und sich des in P2P-Netzwerken etablierten Prinzips von DHT bedient. Die Vorteile dieses Ansatzes sind die hohe Skalierbarkeit, die Verwendung von Unicast-Nachrichten und die Datenflusskontrolle (vgl. Abschnitt 2.7.4). Die DHT-Umsetzung setzt auf Kademlia auf [128]. Im Gegensatz zu Multicast-Discovery werden mittels Unicast nur bekannte Geräte kontaktiert. Von diesen können weitere neue Kontakte angefordert werden. Um den gewünschten Service zu finden, werden Hash-Werte benutzt. Mithilfe der Hash-Werte kann die logische Entfernung zum Ziel-Service berechnet werden. Das suchende Gerät fragt nur diejenigen Kontakte an, die eine bessere Metrik in Bezug auf die Nähe zum Ziel-Hash-Wert liefern. Mit jedem Schritt wird die Entfernung zum Ziel-Service mindestens halbiert. Auf diese Weise erfolgt die Suche nach einem bestimmten Service mit logarithmischer Komplexität. Da der vorgeschlagene Ansatz auf Unicast-Nachrichten basiert, kann das Service-Discovery über mehrere Subnetze und sogar über das Internet ablaufen. Das suchende Gerät kann mehrere parallele Anfragen schicken, um die Suche zu beschleunigen. Das Gerät behält dabei die Kontrolle über den ankommenden Daten-Traffic und kann nicht durch die Antworten überlastet werden. Bei einer Verdoppelung der Netzwerkgröße wird nur ein zusätzlicher Schritt notwendig sein, um den Service zu finden, was in einer sehr hohen Skalierbarkeit sogar für große Netzwerke resultiert. Der Nachteil dieses Ansatzes im Gegensatz zu der in Kapitel 5 vorgestellten Lösung ist es, dass dieser Ansatz nicht mit dem WS-Discovery-Standard abwärtskompatibel ist. Das bedeutet, dass nur die Geräte bzw. Services gefunden werden können, die ein Teil der DHT sind.

Zusammenfassend ist der Hauptbeitrag dieses Kapitels zum Gebiet Geräte- und Service-Discovery wie folgt [4]:

- Ein DHT-basierter Ansatz zum Geräte- und Service-Discovery
- Ein Verfahren zum automatischen Bootstrapping in ein Netzwerk
- Eine prototypische Implementierung der vorgeschlagenen Lösung
- Messergebnisse für einen experimentellen Messaufbau

- Vergleich mit dem WS-Discovery-Standard

6.2. Stand der Technik

Da in diesem Kapitel ein alternativer Ansatz zu Service-Discovery vorgestellt wird, entspricht der Stand der Technik dem in Abschnitt 5.2 beschriebenen Sachverhalt. Im Gegensatz zu den in Abschnitt 5.2 genannten existierenden Lösungen verwendet der vorgeschlagene DHT-Ansatz eine vollständig dezentralisierte Struktur. Dabei sind alle Geräte Teil von DHT. Das suchende Gerät kann mithilfe der Unicast-Nachrichten Geräte bzw. Services in anderen Subnetzen finden und dabei eine vollständige Kontrolle über den Datenfluss behalten. Die in dieser Arbeit vorgeschlagene Lösung ist unabhängig von der physikalischen Übertragungstechnologie und ermöglicht ein deterministisches Verhalten von Service Discovery über mehrere Subnetze mit einer vollständigen Datenflusskontrolle unabhängig von der Anzahl der Geräte.

6.3. DHT-Funktionalität

P2P-Netzwerke wurden ursprünglich für den Datenaustausch entwickelt. Sie können auch als ein verteilter Dateispeicher betrachtet werden. Die wichtigsten Operationen sind Speichern, Suchen und Herunterladen von Dateien. Abhängig davon, wie die Dateien gespeichert und gefunden werden, lassen sich P2P-Netzwerke in unstrukturierte und strukturierte P2P-Netzwerke unterscheiden. In unstrukturierten Netzwerken können beliebige Daten auf beliebigen Peers gespeichert werden. Dagegen wird in strukturierten P2P-Netzwerken eine Beziehung zwischen Daten und Peers aufgebaut. Dabei bestimmt das Datum den Peer, auf dem es gespeichert werden kann (vgl. Abschnitt 2.7).

Um ein Datum in unstrukturierten dezentralisierten Netzwerken zu finden, müssen zunächst direkte logische Nachbarn gefragt werden. Diese leiten die Anfrage an ihre Nachbarn weiter usw. Auf diese Weise wird das Netzwerk mit Anfragen geflutet. Die Suche kann durch das Finden des Datums oder durch ein Terminierungskriterium abgebrochen werden. Das Terminierungskriterium schützt zwar das Netzwerk vor großflächigen Flutungen, kann aber auch dafür sorgen, dass ein existierendes Datum nicht gefunden wird. Das kann dann der Fall sein, wenn der suchende Peer und der Peer mit dem Datum logisch weit voneinander entfernt sind. Um dieses Problem zu umgehen, können eine zentrale Instanz in Form eines Services Brokers (zentralisiertes P2P-Netzwerk) oder einige Super-Peers mit den Kenntnissen über einen Teil des Netzwerks (hybrides P2P-Netzwerk) eingesetzt werden. Die beiden genannten Ansätze bringen jedoch einen SPoF in das P2P-Netzwerk ein. Der DHT-basierte Ansatz benötigt dagegen keine zentrale Instanz, da die Struktur durch die DHT vorgegeben ist. Darüber hinaus ermöglicht die DHT-Struktur eine logarithmische Komplexität im Worst Case, da nur die relevanten Peers während der Suche angefragt werden.

Die Beziehung zwischen Peers und Daten wird mithilfe von Hash-Werten hergestellt. Der Hash-Wert eines Datums wird aus charakteristischen Informationen berechnet, z.B. aus dem Dateinamen. Für die Berechnung des Hash-Wertes kommt z.B. das Hash-Wertverfahren Message-Digest Algorithm 5 (MD5) zum Einsatz [163]. Die Peer-Hash-Werte werden aus zufälligen Werten berechnet. Die Zuständigkeit eines Peers für ein Datum wird mithilfe einer Suchtoleranz ermittelt. Sie bestimmt ein Mindestmaß an Ähnlichkeit zwischen Peer- und Daten-Hash-Werten. Es gibt unterschiedliche Methoden zur Bestimmung der Ähnlichkeit. In dieser Arbeit wird die XOR-Metrik verwenden, wie sie von der Kademlia-Spezifikation definiert ist. Die Ähnlichkeit *S* kann, wie in Formel (19) gezeigt, berechnet werden.

$$S = ID_{peer}\, XOR\, ID_{data} \tag{19}$$

Der Peer ist dann für ein Datum zuständig, wenn *S* kleiner als der Suchtoleranzwert ist. Die XOR-Metrik wird auch als Prefix Matching bezeichnet, d.h. der Peer-Hash-Wert und der Datum-Hash-Wert sind umso ähnlicher, je mehr führende Bits übereinstimmen. Das gesamte Netzwerk kann als Baum, Zahlenstrahl oder Ring dargestellt werden, wie in Abbildung 81 gezeigt. Im Folgenden wird die Bezeichnung DHT-Ring verwendet.

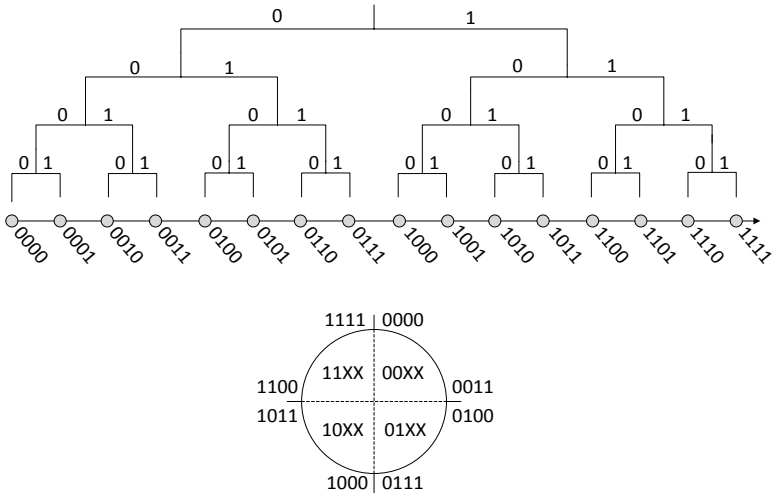

Abbildung 81: Darstellung eines strukturierten DHT-basierten Netzwerks für einen 4 Bit Adressraum

Für die Kommunikation mit anderen Knoten verwaltet jeder Peer eine Routing-Tabelle. In der Routing-Tabelle werden Kontaktinformationen über andere Peers wie z.B. IP-Adresse, Hash-Wert und XOR-Distanz gespeichert. Die Routing-Tabelle ist in Form eines binären Baumes organisiert. Die Kontakte werden in sogenannte Buckets eingeteilt und nach aufsteigender XOR-Distanz sortiert. In jedem Bucket kann die gleiche Anzahl von Kontakten abgelegt werden. Die maximale Anzahl der Kontakte pro Bucket kann von der Anwendung festgelegt werden. Die Buckets decken mit größerer Distanz vom eigenen Hashwert auch größere Bereiche des Distanzraumes ab. Somit kennt jeder Peer viele Kontakte in unmittelbarer Nähe

und nur wenige weit entfernte Kontakte. Zum besseren Verständnis ist eine Routing-Tabelle für einen 4 Bit Adressraum in Abbildung 82 dargestellt. Der Peer selbst befindet sich ganz links, da die Distanz zu sich selbst gleich 0 ist. Mehr Informationen über die Funktionsweise der Routing-Tabelle können [129] entnommen werden.

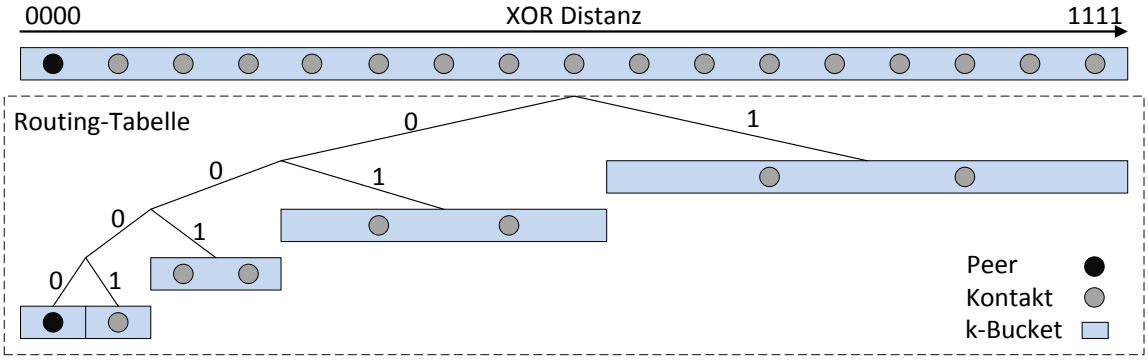

Abbildung 82: Routing-Tabelle eines Peers für 4 Bit Adressraum

Um dem Netzwerk beizutreten und einige Kontakte für die Routing-Tabelle zu erhalten, führt ein Knoten einen sogenannten Bootstrap durch. Für diesen Zweck muss ein Knoten bekannt sein, der bereits ein Teil des DHT-Netzwerks ist. Dieser Knoten wird als Bootstrap-Knoten verwendet und nach dem Netzwerkbeitritt gefragt. Der Bootstrap-Knoten trägt den Hash-Wert des neuen Peers in seine Routing-Tabelle ein, wenn dieser noch nicht bekannt ist und schickt darüber hinaus einige neue Kontakte aus dem DHT-Netzwerk zurück. Andere Kontakte können ebenfalls kontaktiert werden, um die Routing-Tabelle zu füllen. Die Routing-Tabelle wird regelmäßig aktualisiert, um die Kontaktinformationen aktuell zu halten und ausgeschiedene Kontakte zu entfernen.

Wenn ein Peer nach einem bestimmten Datum sucht, werden der Daten-Hash-Wert berechnet und einige Kontakte aus der Routing-Tabelle angefragt, die in der Suchtoleranz des Daten-Hash-Wertes liegen. Wenn solche Knoten in der Routing-Tabelle nicht verfügbar sind, werden die Knoten mit der höchsten Nähe zum Daten-Hash-Wert angefragt. Die angefragten Knoten liefern dabei neue Kontakte, die näher zum Ziel-Hash-Wert liegen. Die zuletzt erhaltenen Kontakte werden wieder angefragt. Auf diese Weise verringert der suchende Knoten die Entfernung zum Daten-Hash-Wert mit jedem Schritt, bis der gewünschte Knoten innerhalb der Suchtoleranz gefunden ist. Darüber hinaus können mehrere Kontakte parallel angefragt werden, um die Suche zu beschleunigen.

6.4. MultiKad

Der DHT-basierte Ansatz wurde primär zum Dateispeichern und –Lookup entwickelt. Die Anforderungen an das System sind dabei anders als beim Service Discovery. Um eine Datei zu finden, muss *mindestens ein* Peer innerhalb der Suchtoleranz gefunden werden. Wenn

mehrere Peers gefunden wurden, kann der Datei-Download beschleunigt werden. Im Fall von Service Discovery ist es wichtig, *alle* Service Provider zu finden, um anschließend den gewünschten auszuwählen. Daher genügt es in manchen Automatisierungsszenarien nicht, nur einige oder viele Geräte der gewünschten Kategorie zu finden (z.B. Drucker), aber nicht alle. Das zweite Problem betrifft die Zuordnung der Hash-Werte. In Kademlia kann jedes Datum theoretisch jedem Peer innerhalb der Suchtoleranz zugewiesen werden. In Automationsszenarien kann jedes Gerät nur seinen eigenen spezifischen Service zur Verfügung stellen. Das dritte Problem betrifft die Berechnung der Hash-Werte. Um den Hash-Wert für den Service zu berechnen, ist ein charakteristisches Attribut wie z.B. der Service-Name notwendig. Da aber viele Geräte denselben Service zur Verfügung stellen können, werden sie alle auf denselben Hash-Wert abgebildet. Dies wird zu einer Hash-Kollision führen. In der originalen Kademlia-Spezifikation ist jedoch keine Kollisionsauflösung der Hash-Werte möglich.

6.4.1. Architektur

Um eine skalierbare und effiziente DHT-basierte Lösung anbieten zu können, sollen die drei obengenannten Probleme gelöst werden:

- Auffindung aller Service Provider
- Zuweisung von Hash-Werten zu Geräten
- Auflösung von Hash-Kollisionen

Um alle Service Provider finden zu können, müssen sie unterschiedliche Hash-Werte besitzen, da nur ein Knoten pro Hash-Wert in die Routing-Tabelle eingetragen werden kann. Darüber hinaus können sich Peers mit dem gleichen Hash-Wert nicht im Netzwerk sehen. Das suchende Gerät kennt jedoch nur den gewünschten Service-Namen und kann folglich nur einen Hash-Wert berechnen. Wenn darüber hinaus ein neues Gerät dem Netzwerk beitritt, berechnet dieses den Hash-Wert aus dem eigenen Service-Typ. Bei der Verwendung der direkten Hash-Wert-Berechnung wird dieser mit den anderen Service Providern kollidieren, die denselben Service anbieten und bereits Teil des Netzwerks sind.

Um diese Probleme zu lösen, wurde eine zweischichtige DHT-Architektur namens MultiKad im Rahmen dieser Arbeit entwickelt. Der DHT-Ring der ersten Schicht wird Hauptring und der der zweiten Schicht – Subring oder Service-Ring – genannt. Jeder Ring hat eine eigene DHT. Die Hash-Werte des Hauptrings stellen Service-Typen dar. Die Hash-Werte des Service-Rings stellen Service Provider dar. Jedes Gerät besitzt zwei Hash-Werte – einen Haupt-Hash-Wert und einen Service-Hash-Wert – und ist Mitglied auf beiden Ringen. Der Haupt-Hash-Wert wird mittels MD5 aus dem Service-Namen berechnet. Der Service-Hash-Wert wird aus einem zufälligen Wert berechnet. Für jeden Hash-Wert wird eine Routing-Tabelle auf dem Gerät angelegt. Provider desselben Services werden auf demselben Hash-

Wert auf dem Hauptring platziert. Der Hauptring kann als eine verbindende Struktur zwischen den Service-Ringen angesehen werden. Die vorgeschlagene Architektur ist in Abbildung 83 dargestellt. Der Drucker-Service ist beispielhaft hervorgehoben.

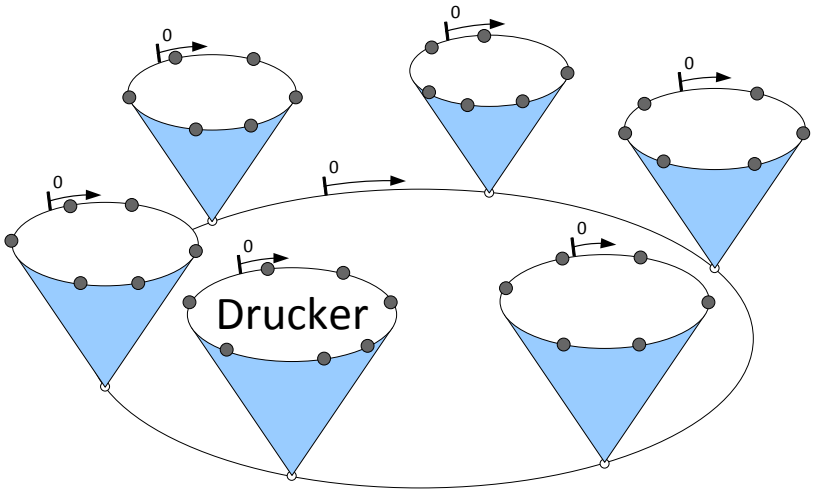

Abbildung 83: MultiKad-Architektur

Diese Architekturperspektive ist nur von außen sichtbar. Aus der Sicht des Gerätes ist der Service-Ring mit dem Hauptring über ein einziges Gerät verbunden, da nur ein Eintrag pro Hash-Wert in der Routing-Tabelle möglich ist. Demnach sieht ein Gerät andere Geräte des eigenen Service-Typs auf dem Service-Ring und ein Gerät pro Service-Typ auf dem Hauptring (siehe Abbildung 84).

Der Subring ist jedoch nicht über ein Gerät mit dem Hauptring verbunden, sondern jedes Gerät des Subrings ist mit verschiedenen Geräten anderer Subringe über den Hauptring verbunden. Die DHTs der beiden Ringe sind voneinander getrennt. Die Geräte können die Schicht durch die Benutzung des entsprechenden Hash-Wertes wechseln. Um die Anfragen auf verschiedenen Ringen zu unterscheiden, werden verschiedene Portnummern benutzt.

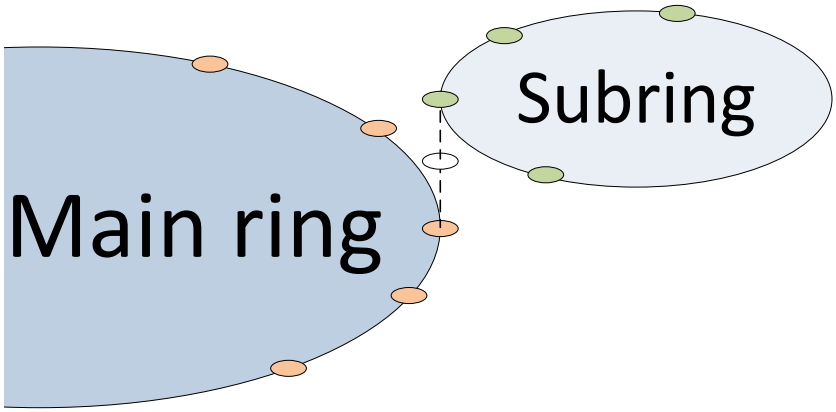

Abbildung 84: MultiKad-Architektur aus der Sicht eines Gerätes

Besonders in großen Netzwerken kennt jedes Gerät eine Teilmenge anderer Geräte; so entsteht ein komplexes Geflecht aus Verbindungen der Subringe über die Ebene des Hauptrings. Folglich wird ein Subring nicht vom Hauptring getrennt, wenn die Geräte das Netzwerk verlassen. Sogar im Worst Case, wenn aus beliebigen Gründen der Subring von dem Hauptring getrennt wird, wird dieser während der Wartungsphase wieder angebunden. Die Wartungsphase ist Teil von Kademlia. Mehr Informationen darüber können [129] entnommen werden.

6.4.2. Netzwerkbeitritt

Um dem Netzwerk beizutreten, muss ein Knoten den Bootstrapping-Prozess durchlaufen. Ein Kontakt (Bootstrap-Knoten) muss bekannt sein, um das Bootstrapping zu starten. Dieser Prozess benötigt in MultiKad drei Schritte.

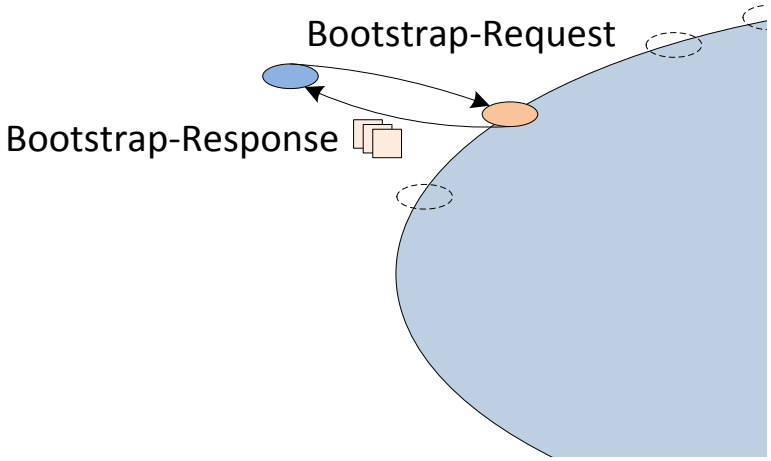

Abbildung 85: MultiKad-Bootstrap Schritt 1

Schritt 1 ist ein regulärer Kademlia Bootstrap. Der Bootstrap-Knoten wird mit dem Haupt-Hash-Wert angefragt. Der Haupt-Hash-Wert wird mittels MD5 vom eigenen Service-Typ berechnet. Die Antwort enthält einige Kontakte aus dem Haupt-Ring (vgl. Abbildung 85). Wenn der Hash-Wert des Knotens nicht in der Routing-Tabelle des Bootstrap-Knotens enthalten ist, wird dieser darin hinzugefügt. Anderenfalls behält der Bootstrap-Knoten den alten Eintrag und schickt dessen Kontaktinformationen dem beitretenden Knoten. In diesem Fall kann Schritt 2 übersprungen werden.

Schritt 2 ist die Suche nach dem eigenen Hash-Wert. Für diesen Zweck werden die Kontakte aus Schritt 1 angefragt. Die Suchtoleranz wird auf 1 gesetzt, d.h., die Hash-Werte müssen identisch sein (vgl. Abbildung 86). Es wird hierbei nach einem beliebigen Knoten mit dem gesuchten Hash-Wert gesucht (AnyNode-Suche). Die Suche ist nur dann erfolgreich, wenn ein Knoten des eigenen Service-Typs gefunden wurde. Wird kein anderer Knoten mit

demselben Hash-Wert gefunden, ist das Bootstrapping abgeschlossen. Der beitretende Knoten ist dann der erste Knoten mit diesem Service-Typ. Anderenfalls fährt der beitretende Knoten mit dem Schritt 3 fort.

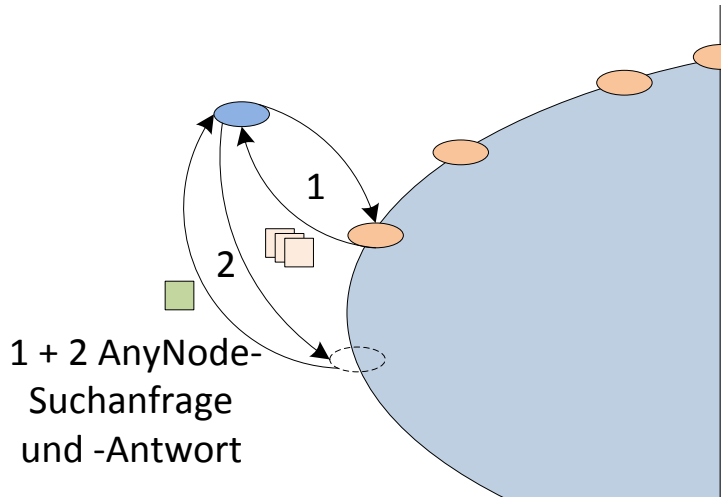

Abbildung 86: MultiKad-Bootstrap Schritt 2

Schritt 3 schließt einen Schichtwechsel ein. Der Knoten aus Schritt 2 ist nun der Bootstrapping-Knoten für den Service-Ring. Der beitretende Knoten schickt ihm eine Anfrage mit dem Service-Hash-Wert. Der Service-Hash-Wert wird aus einem zufälligen Wert berechnet. Als Antwort bekommt der beitretende Knoten einige Kontakte von Subring zurück (vgl. Abbildung 87). Diese neuen Kontakte werden ebenfalls angefragt, um weitere Kontakte zu bekommen. Auf diese Weise kann die Routing-Tabelle gefüllt und der eigene Service-Hash-Wert kann bei anderen Knoten registriert werden. Hiermit ist das Bootstrapping abgeschlossen und der beitretende Knoten ist nun ein Teil des Netzwerks. Dieser fährt dann mit der Wartung fort, um die Routing-Tabelle aktuell zu halten.

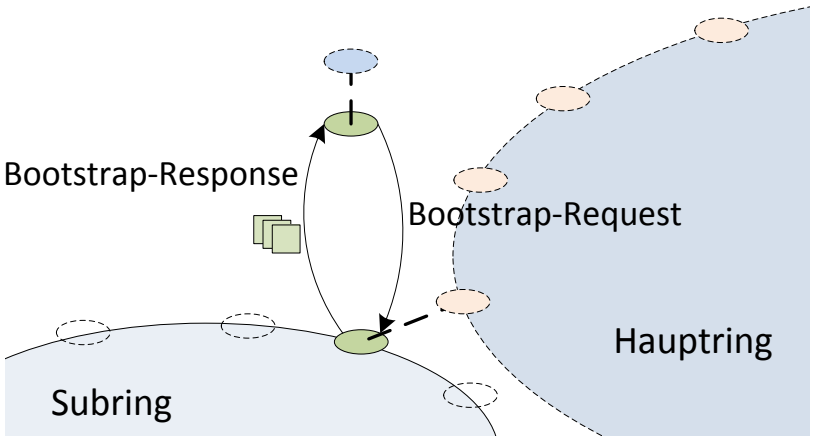

Abbildung 87: MultiKad-Bootstrap Schritt 3

6.4.3. DHT-basiertes Discovery

DHT-basiertes Discovery wird vom Client initiiert, welcher nach einem bestimmten Service-Typ suchen will. Der Client legt für diesen Zweck eine Routing-Tabelle und einen speziellen Client-Hash-Wert an. Der Client-Hash-Wert wird nicht in die Routing-Tabellen anderer Knoten übernommen und ist nicht Teil des Netzwerks. Ähnlich wie der Netzwerkbeitritt läuft das Service Discovery in 3 Schritten ab.

Schritt 1 beginnt mit einem bekannten Knoten aus dem Hauptring. Der Client schickt eine Bootstrap-Anfrage an diesen Knoten und bekommt die ersten Kontakte vom Hauptring. Der Client-Hash-Wert wird dabei jedoch nicht in die Routing-Tabelle des Bootstrap-Knotens eingetragen (siehe Abbildung 88, Punkt 1).

Schritt 2 ist die Suche nach dem gewünschten Service-Typ. Der Name des Service-Typs wird mittels MD5 berechnet und der resultierende Hash-Wert stellt das Suchkriterium dar. Die Suche wird mit der minimalen Suchtoleranz ausgeführt, d.h., die Hash-Werte müssen identisch sein. Der suchende Knoten schickt Anfragen an die Knoten, die dem Ziel-Hash-Wert am nächsten liegen. Die in den Antwortnachrichten enthaltenen Kontakte werden wieder angefragt, bis der gesuchte Hash-Wert gefunden wird (siehe Abbildung 88, Punkte 2, 3). Diese Suche hat eine logarithmische Komplexität, da die Entfernung zum Ziel-Hash-Wert pro Suchschritt mindestens halbiert wird. Wenn der gesuchte Hash-Wert nicht gefunden wird, ist der gesuchte Service-Typ nicht im Netzwerk verfügbar. Anderenfalls wird das Suchverfahren in Schritt 3 fortgesetzt.

Schritt 3 erfordert einen Schichtwechsel. Der im vorherigen Schritt gefundene Knoten wird auf der Ebene des Subrings aufgefordert, alle bekannten Kontakte vom Subring zu schicken. Im Gegensatz zu Schritt 2 setzt diese Anfrage die Suchtoleranz auf Maximum. Alle neuen Kontakte werden wieder angefragt. Dieser Vorgang wird solange wiederholt, bis keine neuen Kontakte empfangen wurden (siehe Abbildung 88, Punkte 4-6). Alle gesammelten Kontakte stellen die Service Provider dar. Nach dem Abschluss des Discovery-Prozesses verfügt der Client über eine vollständige Liste der Service Provider, kann einen oder mehrere davon auswählen und den Service wie gewöhnlich in Anspruch nehmen.

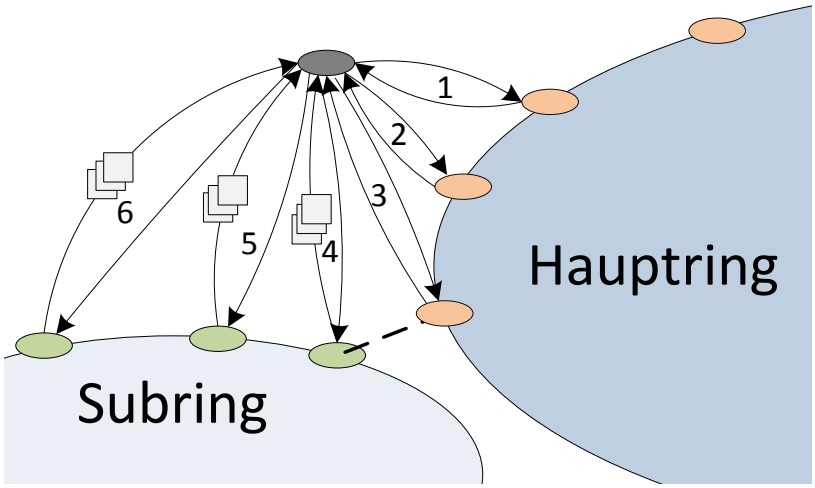

Abbildung 88: DHT-basiertes Discovery

Die Suche im Schritt 3 hat eine lineare Komplexität, da alle Knoten angefragt werden müssen, um 100 % aller Service Provider zu finden. Mehrere Knoten können jedoch parallel angefragt werden. Die Anzahl der parallelen Anfragen hängt von der Leistungsfähigkeit des suchenden Gerätes ab und kann dynamisch angepasst werden, um den Discovery-Prozess zu beschleunigen. Die zeitliche Komplexität für das Worst Case-Verhalten des vorgeschlagenen DHT-basierten Discovery-Ansatzes kann mit Formel (20) beschrieben werden.

$$O(m,n) = log_2(n) + \frac{m}{k} \tag{20}$$

Hierbei ist n die gesamte Anzahl der Service-Typen im Netzwerk, m die Anzahl der Service Provider des gesuchten Service-Typs und k die Anzahl der parallelen Anfragen auf dem Subring.

6.5. Automatisches Bootstrapping

Wie oben beschrieben, soll für den Netzwerkbeitritt und Service-Discovery ein Kontakt bekannt sein. Der Bootstrap-Knoten dient als ein Eintrittspunkt in das Netzwerk. Ein beliebiger Knoten, der bereits Teil des Netzes ist, kann für das Bootstrapping benutzt werden. Es ist damit ausreichend, einen beliebigen Kontakt aus dem Netzwerk einem neuen Gerät oder einem suchenden Client mitzuteilen. Kademlia stellt keinen Mechanismus zum automatischen Bootstrapping bereit. Folglich muss der erste Kontakt manuell auf dem neuen Gerät oder dem suchenden Client eingestellt werden. Während das kein Problem für einen Computer darstellt, kann dieser Umstand für Automationsnetzwerke ein Hindernis sein, besonders wenn Plug&Play-Funktionalität angestrebt wird. Wenn die manuelle Konfiguration nicht möglich ist oder vermieden werden soll, kann der erste Kontakt mittels Multicast kennengelernt werden. Wie oben erwähnt, ist der Einsatz von Multicast mit einigen

Einschränkungen und Nachteilen verbunden. In dem vom Autor vorgeschlagenen Ansatz wird Multicast jedoch zum Auffinden eines beliebigen Knotens und nicht für Service Discovery verwendet. Da Multicast auf lokale Netzwerke beschränkt ist, muss mindestens ein Knoten, der bereits Teil von DHT ist, in dem lokalen Netzwerk vorhanden sein.

Da die Größe des lokalen Netzwerks nicht bekannt ist, würde ein naiver Ansatz – alle Geräte zur Antwort aufzufordern – zur genannten Überlastung des Gerätes führen. Folglich muss ein Mechanismus gefunden werden, die Anzahl der Antworten zu beschränken.

In dieser Arbeit wird ein automatisches Bootstrap-Verfahren, basierend auf der inkrementellen Suchtoleranz, vorgestellt. Für ein automatisches Bootstrapping soll jeder Knoten einen Bootstrap-Hash-Wert erzeugen. Dieser Hash-Wert wird nur für automatisches Bootstrapping verwendet und ist für manuelles Bootstrapping nicht notwendig. Die Bootstrap-Hash-Werte werden aus zufälligen Werten berechnet.

Die Anfrage zum automatischen Bootstrapping wird mittels Multicast mit zwei signifikanten Parametern geschickt: Ziel-Hash-Wert und Suchtoleranz. Der Ziel-Hash-Wert wird ebenfalls aus einem zufälligen Wert berechnet. In der ersten Bootstrap-Anfrage wird die Suchtoleranz auf ein Minimum gesetzt. Nur ein Gerät, das den Ziel-Hash-Wert hat, soll antworten und seine Kontaktinformationen zum Standard-Bootstrapping an das anfragende Gerät schicken. Nachdem diese Anfrage abgeschickt wurde, startet der bootstrappende Knoten einen Timer. Alle DHT-Knoten im lokalen Netzwerk erhalten die Anfrage und vergleichen den Ziel-Hash-Wert mit dem eigenen Bootstrap-Hash-Wert unter Berücksichtigung der mitgesendeten Suchtoleranz. Wenn diese Parameter bei keinem Knoten eine Übereinstimmung verursachen, wird der Timer für die Bootstrap-Anfrage ablaufen. Der Timeout-Wert wird dabei von dem Client bestimmt. Anschließend wird die Suchtoleranz verdoppelt und die Anfrage wird erneut geschickt. Die Verdoppelung der Suchtoleranz bedeutet, dass ein Bit weniger beim Ziel-Hash-Wert mit dem Bootstrapping-Hash-Wert anderer Knoten übereinstimmen muss. Wenn die zweite Anfrage wieder nicht erfolgreich war, wird die Suchtoleranz weiter verdoppelt und eine neue Anfrage verschickt. Diese Prozedur wird solange wiederholt, bis mindestens ein Knoten mit seinen Kontaktinformationen antwortet. Nachdem ein oder mehrere Kontakte erhalten wurden, kann einer davon für Standard-Bootstrapping verwendet werden.

Um die Performance des vorgeschlagenen Bootstrapping-Mechanismus zu testen, wurde eine Simulation entwickelt. In der Simulation wurde eine unterschiedliche Anzahl an Hash-Werten erzeugt und ein Bootstrapping mit der inkrementellen Suchtoleranz durchgeführt. Hierbei wurde die Länge des Bootstrapping-Hash-Wertes zwischen 20 und 32 Bit variiert. Für jede Parametereinstellung wurden 100 Simulationsdurchläufe ausgeführt. Die Simulationsergebnisse sind in Abbildung 89 und Abbildung 90 dargestellt.

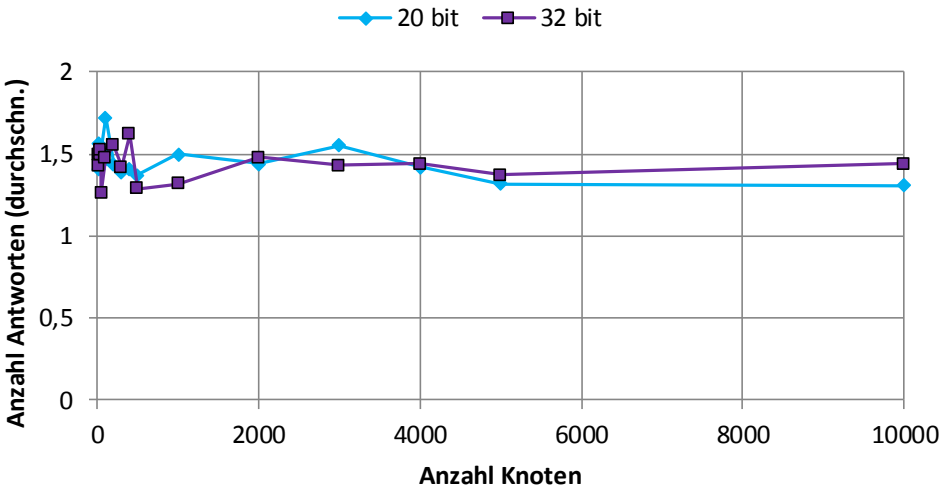

Abbildung 89: Durchschnittliche Anzahl der Antworten für 20 und 32 Bit Hash-Werte

Es ist offensichtlich, dass die Anzahl der Antworten und die Anzahl der führenden Bits nahezu unabhängig von der Länge des Bootstrapping-Hash-Wertes sind. 1,5 Antworten werden durchschnittlich während des automatischen Bootstrapping unabhängig von der Geräteanzahl empfangen. Im Worst Case konnten 11 Antwortet detektiert werden. Die Simulationsergebnisse zeigen die Skalierbarkeit der vorgeschlagenen Lösung. Das suchende Gerät kann somit nicht mit den Antworten überlastet werden. Dies garantiert ein zuverlässiges automatisches Bootstrapping in beliebig großen Netzwerken.

Die Anzahl der führenden Bits bezeichnet die Länge des Präfixes, wenn die erste Übereinstimmung stattgefunden hat (siehe Abbildung 90). Mit steigender Geräteanzahl steigt auch die Länge des Präfixes. Die Präfixlänge hängt jedoch logarithmisch von der Anzahl der Knoten ab. Die Länge des Hash-Wertes bestimmt die durchschnittliche Anzahl der Schritte während des automatischen Bootstrapping, die notwendig sind, um eine Antwort zu bekommen. Diese bestimmen die Zeit, die für das Bootstrapping notwendig ist. Ein Beispiel: Es seien der Timeout-Wert für eine Bootstrapping-Anfrage auf 50 ms, der Bootstrapping-Hash-Wert auf 20 Bit (ca. 1 Million Gerät möglich) gesetzt und 1000 Geräte in einem lokalen Netzwerk angenommen. Dann wird das neue Gerät 550 ms benötigen, um einen Bootstrap-Knoten zu finden. Da die Parameter für das automatische Bootstrapping vom Client gesetzt werden, kann ein kleinerer Timeout-Wert für die Beschleunigung des Verfahrens verwendet werden. Darüber hinaus können mehrere Bits pro Schritt besonders in den ersten Phasen übersprungen werden. Der Bootstrapping-Prozess stellt jedoch keine zeitkritische Operation dar, da dieser nur einmal anlässlich des Netzwerkbeitritts ausgeführt werden muss.

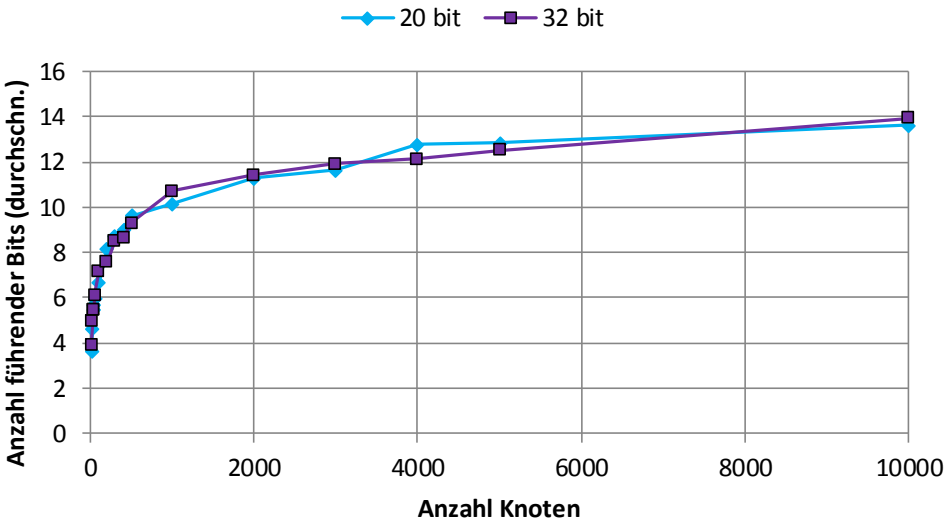

Abbildung 90: Durchschnittliche Anzahl der führenden Bits für 20 und 32 Bit Hash-Werte

6.6. Implementierung und Evaluierung

Um den entwickelten Ansatz zum DHT-basierten Discovery zu testen, wurde ein Prototyp implementiert. Als Zielplattform kam das Evaluation Board ZedBoard zum Einsatz (vgl. Kapitel 4.5). In der Versuchsanordnung wurde nur die ARM CPU in Anspruch genommen. Der Prototyp wurde in C++ geschrieben. Er läuft unter dem Echtzeitbetriebssystem FreeRTOS für eine genaue Zeitmessung. Die Testimplementierung besitzt eine vollständige Kademlia-Funktionalität und wurde mit dem vorgeschlagenen DHT-basierten Service Discovery-Algorithmus erweitert. Darüber hinaus wurden Standard-Kademlia-Nachrichten durch angepasste WS-Discovery-Nachrichten ersetzt. Für die Nachrichtenübertragung wurde 1 Gbps Ethernet-Netzwerk benutzt. Ein Netzwerk mit 10 Geräten, die über einen Switch verbunden sind, wurde aufgebaut. Es wurden durchschnittliche Anfrage-Antwort-Zeiten zwischen zwei Geräten während des Discovery-Prozesses gemessen. Eine geschätzte Worst Case-Performance des Service Discovery kann für beliebige Netzwerke mit Formel (20) berechnet werden. In dem verwendeten Testszenario benötigt ein Gerät 330 µs, um neue Kontakte von einem anderen Gerät zu bekommen. Die Standardspezifikation von WS-Discovery sieht insgesamt 750 ms für Service Discovery vor. In der gleichen Zeit würde das DHT-basierte Service Discovery mindestens einen Service Provider in einem Netzwerk mit ca. $1{,}4*10^{684}$ Knoten finden. Wenn ein Netzwerk mit einem Service-Typ angenommen wird, können 2271 Service Provider in der gleichen Zeit aufgefunden werden. In komplexeren Netzwerken setzt sich die Zeit für Service Discovery aus Service-Typ-Discovery und Service Provider Discovery zusammen. Wenn ein Netzwerk mit 10000 Geräten unter den konstant bleibenden Kommunikationsparametern angenommen wird, wird der erste Service Provider

nach ca. 4,6 ms im Worst Case gefunden. Es sei weiter angenommen, dass ein Gerät drei parallele Anfragen während des Service Discovery verschicken kann. In diesem Fall können in den verbleibenden 745,4 ms mehr als 6700 Service Provider gefunden werden. In realen Netzwerken können die Kommunikationsparameter sehr stark in Abhängigkeit von Netzwerkauslastung und Geräteleistung variieren. Das suchende Gerät kann jedoch die volle vorhandene Leistung beanspruchen, da das Antwortverhalten deterministisch ist. Neben der guten Performance bietet die vollständige Kontrolle durch den Client über den Discovery-Prozess einen entscheidenden Vorteil, weil damit Überlastungen bzw. Traffic Peaks vermieden werden können.

6.7. Zusammenfassung von DHT-basierten WS-Discovery

In dieser Arbeit wurde ein DHT-basierter Ansatz zum Geräte- und Service-Discovery vorgestellt. DHT wurde primär für hochskalierende verteilte P2P-Netzwerke zum Speichern und Herunterladen von Dateien entwickelt. In dieser Arbeit wurden die Vorteile von DHT wie die Skalierbarkeit, die Robustheit und das Vermeiden von SPoFs genutzt und auf das Service Discovery übertragen. Als DHT-Variante wurde Kademlia eingesetzt. Da DHT auf Hash-Werten basiert, werden die Service-Typen mit dem Hashing-Algorithmus MD5 gehasht. Der vorgeschlagene DHT-basierte Ansatz für Service Discovery benutzt eine zweischichtige Architektur. Die erste Schicht repräsentiert den Haupt-DHT-Ring. Auf dem Hauptring können Service-Typen gefunden werden. Hierbei kann ein Service Provider gefunden werden, der den gewünschten Service zur Verfügung stellt. Dieser Knoten wird als Bootstrap-Knoten für den Service-Ring verwendet. Auf dem Service-Ring können andere Service Provider desselben Service-Typs gefunden werden. Die Zeitkomplexität für den vorgeschlagenen Ansatz ist $O(m,n) = log_2(n) + \frac{m}{k}$, wobei n die Gesamtanzahl der Service-Typen im Netzwerk ist, m die Anzahl der Service Provider des gewünschten Service-Typs und k die Anzahl der Parallelanfragen auf dem Subring. Für die Kommunikation zwischen den Geräten während des Discovery-Prozesses werden nur Unicast-Nachrichten verwendet. Auf diese Weise kann das Discovery über mehrere Subnetze hinweg ausgeführt werden, im Gegensatz zu Standard WS-Discovery. Das suchende Gerät behält die vollständige Kontrolle über den Discovery-Prozess und kann den Datenfluss entsprechend seinen freien Ressourcen und seiner Verarbeitungsgeschwindigkeit anpassen. Folglich werden Überlastungen am Gerät in großen Netzwerken vermieden. Um DHT-Discovery in Plug&Play-Netzwerken einsetzen zu können, wurde ein Ansatz zum automatischen Bootstrapping entwickelt. Der Ansatz basiert auf Multicast und einer inkrementellen Suchtoleranz. Die Multicast-Nachrichten werden jedoch ausschließlich für das Bootstrapping verwendet, um einen beliebigen Knoten im lokalen Netzwerk zu finden. Die Simulationsergebnisse haben gezeigt, dass durchschnittlich 1,5 Antworten während des automatischen Bootstrapping unabhängig von der Netzwerkgröße erhalten werden können. Dementsprechend wird das bootstrappende Gerät nicht mit den

Antworten überwältigt. Alle entwickelten Mechanismen zeigen eine sehr gute Skalierbarkeit für beliebige Netzwerkgrößen und können für Service Discovery in großen Netzwerken eingesetzt werden.

7. Zusammenfassung und Ausblick

Die vorliegende Arbeit widmet sich dem Einsatz von Web Services in Automationsnetzwerken, untersucht die dabei auftretenden Probleme und schlägt neue Lösungen vor.

7.1. Einsatz von Web Services in der Gebäudeautomation

Im Rahmen dieser Arbeit wurde der Einsatz von Web Services in Automationsnetzwerken, insbesondere in der Gebäudeautomation, untersucht. Es wurde gezeigt, dass jedes beliebige Protokoll der Gebäudeautomation auf DPWS abgebildet werden kann. Darüber hinaus ermöglichen die vorgeschlagenen Kompressionsmethoden, den durch Web Services verursachten Kommunikations-Overhead um 90 % zu reduzieren. Die Protokoll-Abbildungen (DPWS-Profile) können hierbei dieselben und u.U. sogar niedrigere Datenmengen erzeugen als spezifische proprietäre binäre Protokolle. Die Plug&Play-Fähigkeiten von DPWS ermöglichen darüber hinaus eine automatische Suche und Anbindung der Geräte. Es wurde gezeigt, dass zwei DPWS-Geräte lediglich mittels einer Taste miteinander gekoppelt werden können. Da Web Services eine vollständig offene Technologie sind, können Hersteller die Gerätepreise reduzieren, da keine Lizenzkosten anfallen würden. Darüber hinaus können die selbstbeschreibenden Interfaces der DPWS-Geräte helfen, die Interoperabilitätsprobleme zwischen verschiedenen Herstellern zu beseitigen. Web Services können dadurch als eine übergreifende und harmonisierende Lösung in der Gebäudeautomation eingesetzt werden.

7.2. Einsatz von Web Services in Automationsnetzwerken mit Echtzeitanforderungen

Im Rahmen der Arbeit wurde die Einsatzfähigkeit von Web Services in Automationsnetzwerken mit Echtzeitanforderungen in Bezug auf XML-Übertragung bzw. Parsen untersucht. Damit die Web Services für beliebige Echtzeitanwendungen wie Motion Control eingesetzt werden können, bedarf es einer hohen Datenverarbeitungsgeschwindigkeit. Die Datenübertragung kann mit XML-Kompression durch EXI verbessert werden. Es wurde im Rahmen dieser Arbeit ein Hardware-basierter Parser für EXI entwickelt, der die gewünschte Datenverarbeitung mit sehr hohen Geschwindigkeiten ermöglicht. Dieser kann dynamisch zur Synthesezeit generiert und damit auf jede beliebige Anwendung angepasst werden. Der vorgeschlagene Hardware-EXI-Parser kann EXI-Streams in wenigen Mikrosekunden parsen. Darüber hinaus bietet er standardisierte Bus-Interfaces und kann in komplexere Systeme integriert werden oder auch als ein DSP den Hauptprozessor entlasten. Hiermit wurde gezeigt, dass Web Service-Nachrichten in harter Echtzeit geparst werden können. Die Verarbeitung der Nachrichten kann durch einen Hardware-Parser im Vergleich zu Software-Umsetzungen enorm beschleunigt werden (Reduktion der Verarbeitungszeit um

95 %), sodass Web Services zukünftig sogar für Anwendungen mit sehr kurzen Reaktionszeiten wie z.B. Motion Control eingesetzt werden können.

7.3. Untersuchung der Skalierbarkeit von Web Services in Automationsszenarien

Im Rahmen dieser Arbeit wurde die Skalierbarkeit von Web Services in großen Netzwerken untersucht. Als ein Skalierbarkeitsproblem wurde die Geräte- und Service-Discovery identifiziert. Diesem liegt der WS-Discovery Standard zugrunde. Es wurde festgestellt, dass Service Discovery nach der Standardspezifikation eine geringe Skalierbarkeit für steigende Netzgrößen zeigt. Aus diesem Grund wurden Optimierungen vorgeschlagen, die die Skalierbarkeit von WS-Discovery deutlich verbessern. Es wurden zunächst Optimierungen vorgeschlagen, die zum WS-Discovery Standard abwärtskompatibel sind. Diese ermöglichen dem Gerät, die Discovery-Parameter so zu setzen, dass die entstehende Geräte- und Netzwerkauslastung auf das gewünschte Niveau eingestellt werden kann. Der vorgeschlagene Ansatz ist von der Netzwerkgröße unabhängig. Die steigende Geräteanzahl bewirkt nur die Änderung der Discovery-Parameter. Dadurch kann WS-Discovery in Netzwerken beliebiger Größe eingesetzt werden. Die Einschränkung des vorgeschlagenen Ansatzes hängt mit der Verwendung von Multicast-Nachrichten zusammen, wodurch das Discovery nur in lokalen Netzwerken möglich ist. Darüber hinaus sollen alle Geräte die eingestellten Discovery-Parameter berücksichtigen, um den gewünschten Effekt zu erreichen.

Um die Einschränkungen von WS-Discovery zu umgehen, wurde im Rahmen dieser Arbeit ein weiterer alternativer Ansatz zu Service Discovery, basierend auf DHT, vorgeschlagen. Dieser Ansatz bedient sich einer Technologie, die in P2P-Netzwerken eingesetzt wird. Um DHT für Service Discovery einsetzen zu können, wurden notwendige Änderung an der DHT-Architektur vorgenommen. Der Vorteil dieses Ansatzes ist, dass der suchende Client während Service Discovery nur Unicast-Nachrichten verwendet. Dadurch kann die Suche nach den Services über das lokale Netzwerk hinweg stattfinden. Der Client kann darüber hinaus den Fluss des Discovery-Prozesses vollständig steuern und an seine Leistungsfähigkeit anpassen. Der DHT-Ansatz ist jedoch mit dem WS-Discovery Standard nicht abwärtskompatibel. Alle Geräte, die am Service Discovery teilnehmen möchten, müssen ein Teil der DHT sein. Dieser Ansatz zeigt dennoch, dass Service Discovery mit DHT eine sehr gute Skalierbarkeit aufweist und einige Nachteile der Standard Discovery verbessert. Die Ergebnisse dieser Arbeit können in zukünftige Spezifikationen einfließen.

7.4. Ausblick

Der aktuelle Technologietrend geht in Richtung der Übertragung von Internet-Technologien auf andere Einsatzgebiete. Die aktuellen Forschungen sagen eine Explosion der Anzahl der verbunden Geräte voraus [164]. Diese verbundenen Geräte werden als IoT bezeichnet. Damit

die Vision von IoT möglich ist, müssen die Interfaces der Geräte miteinander kompatibel sein. Die aktuell in der Gebäudeautomation eingesetzten Protokolle können das nicht gewährleisten. Web Services können in der Praxis eine zukünftige Technologie der Gebäudeautomation werden. Für eine Verbesserung der Interoperabilität zwischen verschiedenen Herstellern kann eine gemeinsame Datenbank geschaffen werden, in der standardisierte WS-Interfaces sowie Design Guidelines für Geräte festgehalten werden können. Es können darüber hinaus standardisierte Methoden zur entfernten Gerätekonfiguration durch z.B. mobile Geräte der Nutzer entwickelt werden. Diese würden die Inbetriebnahme der Geräte weiter vereinfachen.

A. Anhang

A.1. SML-Nachrichten

Die spezifizierten SML-Nachrichten in der Version 1.03 und deren Beschreibungen sind in Tabelle 20 dargestellt.

Nachrichtentyp	**Beschreibung**
OpenRequest/Response	Beginn einer SML-Datei
CloseRequest/Response	Ende einer SML-Datei
GetProfilePackRequest/Response	Anfrage von Messwerten, die in einer gepackten Form übertragen werden
GetProfileListRequest/Response	Anfrage von Messwerten, die in einer einfachen Listenform übertragen werden
GetListRequest/Response	Anfrage von Messwerten, die in einer vorparametrierten Listenform übertragen werden
GetProcParameterRequest/Response	Abfrage der Betriebsparameter
SetProcParameterRequest/Response	Setzen der Betriebsparameter
AttentionResponse	Übertragen von Quittungen, Fehlermeldungen, Warnungen oder andere Hinweise

Tabelle 20: SML-Nachrichtentypen (Version 1.03)

Die GetProfileListRequest-Nachricht wird benutzt, um die Messwerte abzufragen. Der Aufbau der *GetProfileListRequest*-Nachricht ist in Tabelle 21 dargestellt. *ServerId* bezeichnet die adressierte Datenquelle oder ein Softwaremodul. *WithRawdata* erlaubt das Senden zusätzlicher Rohdaten. *BeginTime* und *endTime* geben die Messperiode an. *ParameterTreePath* zeigt die angefragten Messwerte an. Die Messwerte können aus verschiedenen Kanälen ausgelesen werden, die in *object_List* spezifiziert sind. Mit *dasDetails* können zusätzliche Anfrageparameter berücksichtigt werden.

Nachrichtenfeld	Datentyp	Optional
serverId	String	Ja
username	String	Ja
password	String	Ja
withRawdata	Boolean	Ja
beginTime	Unsigned32	Ja
endTime	Unsigned32	Ja
parameterTreePath	String List	Nein
object_List	String List	Ja
dasDetails	SML Tree Type	Ja

Tabelle 21: Aufbau der GetProfileListRequest-Nachricht

Nachrichtenfeld	Datentyp	Optional
serverId	String	Nein
actTime	SML Time	Nein
regPeriod	Unsigned32	Nein
parameterTreePath	String List	Nein
valTime	SML Time	Nein
status	Unsigned64	Nein
period_List	SML Period List	Nein
rawData	String	Ja
periodSignature	String	Ja

Tabelle 22: Aufbau der GetProfileListResponse-Nachricht

Wenn die Anfrage erfolgreich verarbeitet wurde, werden die angefragten Werte in der GetProfileListResponse-Nachricht übertragen. Der Aufbau dieser Nachricht ist in Tabelle 22 dargestellt. Das Feld *actTime* bezeichnet die Zeit, wann der Server mit der Verarbeitung der Anfrage begonnen hat. Die Dauer der aktuellen Log-Periode wird in *regPeriod* präsentiert. Das Feld *valTime* stellt den Zeitstempel des Messwertes an. Zusätzliche Rohdaten können in Form eines Strings in *rawData* übertragen werden. Für die Integritätsüberprüfung kann eine

Nachrichtsignatur in *periodSignature* angegeben werden. Die angefragten Messwerte werden in Form einer Liste in *period_List* übertragen. Die Messwerte werden zusammen mit dem Namen, der Einheit, dem Skalierungsfaktor und einer optionalen Wertsignatur geschickt.

A.2. Abbildung von SML auf DPWS (Smart Metering)

Am folgenden Beispiel soll die Abbildbarkeit von SML auf DPWS verdeutlicht werden. Alle SML-Datentypen können ohne weiteres auf DPWS-Datentypen abgebildet werden:

Datentypen: Integer8, Integer16, Unsigned8, Boolean, String, …

Am Beispiel eines Übertragungsobjektes SML_ListEntry, das einen Messwert darstellt, wird die Abbildbarkeit auf DPWS gezeigt:

Elemente: **SML Entries:**

z.B.: `SML_ListEntry`

```
{
    objName (Octet String),
    status (Unsigned64),
    unit (Unsigned8),
    value (Integer32),
    …
}
```

Abbildung:

```
<SMLListEntry>
    <objName>Name</objName>
    <status>200</status>
    <unit>1</unit>
    <value>32</value>
</SMLListEntry>
```

Das Attribut eines Messwertes “attribute” kann wie folgt auf DPWS abgebildet werden:

Attribute: z.B. attribute (Octet String)

Abbildung:

```
<SMLListEntry attribute="Attribute"> … </SMLListEntry>
```

Anhand einer ausgewählten SML-Nachricht zur Abfrage der Messwerte SML_GetProcParameter.Req wird die Abbildbarkeit von SML-Nachrichten auf DPWS veranschaulicht:

Nachrichten: z.B. `SML_GetProcParameter.Req`

```
{
    serverId,
    username,
    password,
    parameterTreePath
}
```

Abbildung:

```
<soap:Envelope>
    <soap:Header>
        <wsa:Action>SMLGetProcParameterReq<wsa:Action>
        <wsa:To>serverId</wsa:To>
        <wsse:Security>
            <wsse:UsernameToken>
            <wsse:Username>username</wsse:Username>
            <wsse:Password>password</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <sml:SMLGetProcParameterReq>
            <sml:parameterTreePath>010203040506
            </sml:parameterTreePath>
        </sml:SMLGetProcParameterReq>
    </soap:Body>
</soap:Envelope>
```

Die von SML unterstützten Übertragungsarten sind:

Übertragung: Request-Response, Notification

A.3. Abbildung von BACnet auf DPWS (klassische Gebäudeautomation)

Am folgenden Beispiel soll die Abbildbarkeit von BACnet auf DPWS verdeutlicht werden. Alle BACnet-Datentypen können ohne weiteres auf DPWS-Datentypen abgebildet werden:

Datentypen: Unsigned, Boolean, Character String, …

Am Beispiel eines Übertragungsobjektes Analog Input, das einen analogen Eingangswert darstellt, wird die Abbildbarkeit auf DPWS gezeigt:

Elemente: **BACnet Objects:**

z.B. Analog Input

```
{
    Object_Name (Character String),
    Present_Value (Unsigned),
    Local_Time (Time),
    …
}
```

Abbildung:

```
<Object>
    <ObjectName>Name</ObjectName>
    <PresentValue>32</PresentValue>
    <LocalTime>06:00</LocalTime>
    …
</Object>
```

Das Attribut eines BACnet-Objektes "Object Type" kann wie folgt auf DPWS abgebildet werden:

Attribute: z.B. Object Type (Enumeration)

Abbildung:

```
<Object type="Analog Input"> … </Object>
```

Anhand einer ausgewählten BACnet-Nachricht zur Abfrage eines Parameters eines Objektes „ReadProperty Request" wird die Abbildbarkeit von BACnet-Nachrichten auf DPWS veranschaulicht:

Nachrichten: z.B. ReadProperty Request

```
{
    Object Identifier,
    Property Identifier,
    Property Array Index
}
```

Abbildung:

```
<soap:Envelope>
    <soap:Header>
        <wsa:Action>ReadPropertyRequest<wsa:Action>
        <wsa:To>ObjectIdentifier</wsa:To>
    </soap:Header>
    <soap:Body>
        <bac:ReadPropertyRequest>
```

```
                <bac:PropertyIdentifier>Identifier
                </bac:PropertyIdentifier>
                <bac:PropertyArrayIndex>10</bac:PropertyArrayIndex>
            </bac:ReadPropertyRequest>
        </soap:Body>
    </soap:Envelope>
```

Die von BACnet unterstützten Übertragungsarten sind:

Übertragung: One-Way, Request-Response, Notification

A.4. Abbildung von ZigBee auf DPWS (Smart Home)

Am folgenden Beispiel soll die Abbildbarkeit von ZigBee auf DPWS verdeutlicht werden. Alle ZigBee-Datentypen können ohne weiteres auf DPWS-Datentypen abgebildet werden:

Datentypen: Unsigned Integer 8 Bit, Boolean, Character String, …

Am Beispiel eines Übertragungsobjektes (Clusters) Thermostat, wird die Abbildbarkeit auf DPWS gezeigt:

Elemente: **ZigBee Clusters:**

```
Thermostat
{
    LocalTemperature (Signed 16 Bit Integer),
    OutdoorTemperature (Signed 16 Bit Integer),
    Occupancy (8 Bit Integer),
    …
}
```

Abbildung:

```
<Cluster>
    <LocalTemperature>24</LocalTemperature>
    <OutdoorTemperature>20</OutdoorTemperature>
    <Occupancy>1</Occupancy>
</Cluster>
```

Das Attribut eines ZigBee-Objektes “Cluster Type” kann wie folgt auf DPWS abgebildet werden:

Attribute: z.B. Cluster Type (Enumeration)

Abbildung:

```
<Cluster type="Thermostat"> … </Cluster>
```

Anhand einer ausgewählten ZigBee-Nachricht zur Abfrage eines Parameters eines Objektes (Clusters) „APSDE-DATA.request“ wird die Abbildbarkeit von ZigBee-Nachrichten auf DPWS veranschaulicht:

Nachrichten: z.B. `APSDE-DATA.request`

```
{
    DstAddress,
    ProfileId,
    ClusterId,
    …
}
```

Abbildung:

```
<soap:Envelope>
    <soap:Header>
        <wsa:Action>ApsdeDataRequest<wsa:Action>
        <wsa:To>DstAddress</wsa:To>
    </soap:Header>
    <soap:Body>
        <zbe:ApsdeDataRequest>
            <zbe:ProfileId>10</zbe:ProfileId>
            <zbe:ClusterId>5</zbe:ClusterId>
        </zbe:ApsdeDataRequest>
    </soap:Body>
</soap:Envelope>
```

Die von ZigBee unterstützten Übertragungsarten sind:

Übertragung: Request-Response, Notification

Literaturverzeichnis

[1] V. Altmann, J. Skodzik, F. Golatowski und D. Timmermann, „Investigation of the Use of Embedded Web Services in Smart Metering Applications," *38th Annual Conference of the IEEE Industrial Electronics Society,* pp. 6172-6177, 2012.

[2] V. Altmann, J. Skodzik, P. Danielis, N. Pham Van, F. Golatowski und D. Timmermann, „Real-Time Capable Hardware-based Parser for Efficient XML Interchange," *9th IEEE/IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP),* 2014.

[3] V. Altmann, P. Danielis, J. Skodzik, F. Golatowski und D. Timmermann, „Optimization of Ad Hoc Device and Service Discovery in Large Scale Networks," *18th IEEE Symposium on Computers and Communications (ISCC),* pp. 833-838, 2013.

[4] V. Altmann, J. Skodzik, P. Danielis, J. Müller, F. Golatowski und D. Timmermann, „A DHT-based Scalable Approach for Device and Service Discovery," *12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC),* 2014.

[5] V. Altmann, H. Bohn und F. Golatowski, „Web Services for Embedded Devices," in *The Industrial Communication Technology Handbook 2nd Edition*, CRC Press, 2014.

[6] L. L. Peterson und B. S. Davie, Computer Networks: a systems approach, Morgan Kaufmann Publishers, 2012.

[7] B. A. Forouzan, Data Communications and Networking, McGraw-Hill, 2007.

[8] F. Halsall, Computer Networking and the Internet, Addison-Wesley, 2005.

[9] A. S. Tanenbaum und D. J. Wetherall, Computer Networks, Prentice Hall, 2011.

[10] Institute of Electrical and Electronics Engineers, „IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications," *IEEE 1901 Standard,* 2010.

[11] HomePlug Powerline Alliance, „HomePlug AV2 Technology," 2012.

[12] H. A. Latchman, S. Katar, L. Yonge und S. Gavette, Homeplug AV and IEEE 1901: A Handbook for PLC Designers and Users, John Wiley & Sons, 2013.

[13] S. Katar, M. Krishnam, R. Newman und H. Latchman, „Harnessing the potential of powerline communications using the HomePlug AV standard," *RF Design,* pp. 16-26, 2006.

[14] S. Goldfisher und S. Tanabe, „IEEE 1901 Access System: An Overview of Its Uniqueness and Motivation," *IEEE Communications Magazine,* pp. 150-157, 2010.

[15] S. Gavette, „HomePlug AV Technology Overview," *Windows Hardware Engineering Conference ,* 2006.

[16] HomePlug Powerline Alliance, „HomePlug GREEN PHY Specification Release Version 1.00," 2010.

[17] J. Dong, Network Protocols Handbook, Javvin Press, 2007.

[18] J. Postel, Internet Protocol, Internet Standard RFC 791, 1981.

[19] J. Postel, User Datagram Protocol, Internet Standard RFC 768, 1980.

[20] J. Postel, Transmission Control Protocol, Internet Standard RFC 793, 1981.

[21] J. F. Kurose und K. W. Ross, Computer Networking - A Top-Down Approach, Pearson Education, 2012.

[22] D. C. Plummer, An Ethernet Address Resolution Protocol, Internet Standard RFC 826, 1982.

[23] R. Droms, Dynamic Host Configuration Protocol, Internet Standard RFC 2131, 1997.

[24] R. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.

[25] L. Richardson und S. Ruby, RESTful Web Services, O'Reilly Media, 2007.

[26] T. Berners-Lee, R. Fielding und L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, Internet Standard RFC 3986, 2005.

[27] T. Bayer und D. M. Sohn, REST Web Services, yeebase media GmbH, 2007.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee, Hypertext Transfer Protocol - HTTP/1.1, Internet Standard RFC 2616, 1999.

[29] L. Richardson, M. Amundsen und S. Ruby, RESTful Web APIs, O'Reilly, 2013.

[30] Z. Shelby, K. Hartke und C. Bormann, „The Constrained Application Protocol (CoAP)," Internet Standard RFC 7252, 2014.

[31] Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, Internet Standard RFC 6690, 2012.

[32] F. G. Guido Moritz, „IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) and Constrained Application Protocol (CoAP)," in *The Industrial Communication Technology Handbook 2nd Edition*, CRC Press, 2014, pp. 38.1-38.13.

[33] T. Erl, Service Oriented Architecture: Concepts, Technology, and Design, Pearson Education Inc., 2005.

[34] H. Bohn und F. Golatowski, „Web Services for Embedded Devices," in *Embedded Systems Handbook, Second Edition*, CRC Press, 2009, pp. 19.1-19.31.

[35] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. Dovey und andere, „UDDI Version 3.0.2," *UDDI Spec Technical Committee Draft,* 2004.

[36] I. Melzer, Service-Orientierte Architekturen mit Web Services, Spektrum Akademischer Verlag GmbH, 2010.

[37] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard, „Web Services Architecture," *W3C Working Group Note,* 2004.

[38] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler und F. Yergeau, „Extensible Markup Language (XML) 1.0 (Fifth Edition)," *W3C Recommendation,* 2008.

[39] H. Vonhoegen, Einstieg in XML, Galileo Press, 2011.

[40] D. C. Fallside und P. Walmsley, „XML Schema Part 0: Primer Second Edition," W3C Recommendation, 2004.

[41] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen und andere, „SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," *W3C Recommendation,* 2007.

[42] N. Mitra und Y. Lafon, „SOAP Version 1.2 Part 0: Primer (Second Edition)," *W3C Recommendation,* 2007.

[43] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau und H. F. Nielsen, „SOAP Version 1.2 Part 2: Adjuncts (Second Edition)," *W3C Recommendation,* 2007.

[44] R. Jeyaraman, „SOAP-over-UDP Version 1.1," *Public Review Draft,* 2009.

[45] J. Schlimmer, A Technical Introduction to the Devices Profile for Web Services, MSDN, 2004.

[46] S. Chan, D. Conti, C. Kaler, T. Kuehnel, A. Regnier und andere, Devices Profile for Web Services, Microsoft Corporation, 2006.

[47] T. Nixon, A. Regnier, D. Driscoll und A. Mensch, Devices Profile for Web Services Version 1.1, OASIS, 2009.

[48] G. Moritz, E. Zeeb, S. Prüter, F. Golatowski, D. Timmermann und R. Stoll, „Device Profile for Web Services and the REST," *The 8th IEEE International Conference on Industrial Informatics (INDIN),* pp. 584-591 , 2010.

[49] P. Leach, M. Mealling und R. Salz, „A Universally Unique IDentifier (UUID) URN Namespace," Internet Standard RFC 4122, 2005.

[50] M. Gudgin, M. Hadley und T. Rogers, „Web Services Addressing 1.0 - Core," *W3C*

Recommendation, 2006.

[51] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri und andere, „Web Services Policy 1.5 - Framework," *W3C Recommendation,* 2007.

[52] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri und andere, „Web Services Policy 1.5 - Attachment," *W3C Recommendation,* 2007.

[53] A. Nadalin, C. Kaler, R. Monzillo und P. Hallam-Baker, „Web Services Security: SOAP Message Security 1.1 (WS-Security)," *OASIS Standard Specification,* 2006.

[54] K. Ballinger, B. Bissett, D. Box, F. Curbera, D. Ferguson und andere, „Web Services Metadata Exchange 1.1 (WS-MetadataExchange)," *W3C Member Submission,* 2008.

[55] J. Alexander, D. Box, L. F. Cabrera, D. Chappell und G. D. andere, „Web Services Transfer (WS-Transfer)," *W3C Member Submission,* 2006.

[56] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana, „Web Services Description Language (WSDL) 1.1," *W3C Note,* 2001.

[57] D. Box, L. F. Cabrera, C. Critchley, F. Curbera, D. Ferguson und andere, „Web Services Eventing (WS-Eventing)," *W3C Member Submission,* 2006.

[58] V. Modi und D. Kemp, „Web Services Dynamic Discovery (WS-Discovery) Version 1.1," *OASIS Standard,* 2009.

[59] C. Hübner, H. Merz und T. Hansemann, Gebäudeautomation: Kommunikationssysteme mit EIB/KNX, LON und BACnet, Carl Hanser Verlag GmbH & Co. KG, 2009.

[60] S. T. Bushby, „BACnet - A standard communication infrastructure for intelligent buildings," *Automation in Construction,* pp. 529-540, 1997.

[61] „ARCNET - the deterministic, robust fieldbus," e.V., ARCNET User Group, [Online]. Available: http://www.arcnet.de/. [Zugriff am 14 März 2014].

[62] C. Müller, „BACnet als Standardbussystem in der Gebäudeautomation," Honeywell AG, 2002.

[63] N. Nardone und K. Schader, „BACnet Wireless Options Expand to Include ZigBee," *BACnet International,* 2011.

[64] International Organization for Standardization, Building automation and control systems - Part 5: Data communication protocol, DIN EN ISO 16484-5:2012 Norm, 2012.

[65] H. R. Kranz, BACnet Gebäudeautomation 1.12, Cci Dialog Gmbh, 2012.

[66] B. Swan, „The Language of BACnet-Objects, Properties and Services," Alerton Technologies, Inc., 1996.

[67] KNX Association, „Standardization," [Online]. Available: http://www.knx.org/knx-en/knx/technology/standardisation/index.php. [Zugriff am 17 März 2014].

[68] Europäisches Komitee für Normung, „Elektrische Systemtechnik für Heim und Gebäude (ESHG) - Teil 1: Aufbau der Norm," DIN EN 50090-1:2011 Norm, 2011.

[69] W. Kastner, F. Praus, G. Neugschwandtner und W. Granzer, „KNX," in *The Industrial Electronics Handbook: Industrial Communications Systems, Second Edition*, CRC Press, 2011, pp. 42.1-42.13.

[70] KNX Association, „Grundlagenwissen zum KNX Standard," 2013.

[71] F. Praus, „A versatile networked embedded platform for KNX/EIB," Dimplomarbeit, 2005.

[72] KNX-Association, „Interworking," Home and Building Management Systems.

[73] U. Ryssel, H. Dibowski, H. Frank und K. Kabitzsch, „LonWorks," in *The Industrial Electronics Handbook: Industrial Communications Systems*, CRC Press, 2011, pp. 41.1-41.13.

[74] „LonWorks Technology Achieves ISO/IEC Standardization," LonMark International, 2008. [Online]. Available: http://www.lonmark.org/news_events/press/2008/1208_iso_standard. [Zugriff am 18 März 2014].

[75] International Organization for Standardization, „Information technology - Control network protocol - Part 1: Protocol stack," ISO/IEC EN 14908-1:2012 Norm, 2012.

[76] C. Brönniman, „Technische Grundlagen zur LonWorks Technologie," LonMark Schweiz, 2010.

[77] Lonmark International, „LonMark Resource Files Version 14," 2013. [Online]. Available: http://types.lonmark.org. [Zugriff am 19 März 2014].

[78] M. Galeev, „Catching the Z-Wave," *Electronic Engineering Times India,* 2006.

[79] International Telecommunication Union, „Short range narrow-band digital radiocommunication transceivers – PHY and MAC layer specifications," in *Series G: Transmission Systems and Media, Digital Systems and Networks*, ITU-T Recommendation G.9959, 2012.

[80] J. Franck, „Z-Wave Node Type Overview and Network Installation Guide," Zensys A/S, 2008.

[81] N. T. Johansen, „Z-Wave Protocol Overview," Zensys A/S, 2006.

[82] Z-Wave.Me Team,, „Z-Way Developers Documentation," 2013.

[83] T. Köthke, „Self-powered Radio Technology for Building Automation Systems," *Hannover Messe,* 2011.

[84] International Organization for Standardization, „Wireless Short-Packet (WSP) protocol optimized for energy harvesting - Architecture and lower layer protocols," in *Information technology - Home Electronic Systems*, ISO/IEC 14543-3-10 Standard, 2012.

[85] A. Anders, „Reichweitenplannng für EnOcean Funksysteme," EnOcean GmbH, 2009.

[86] EnOcean GmbH, „EnOcean Radio Protocol 2," *Specification V1.0,* 2013.

[87] EnOcean GmbH, „Smart Acknowledge," *System Specification,* 2013.

[88] EnOcean Alliance, „EnOcean Equipment Profiles (EEP),“ *Technical Task Group Interoperability,* 2013.

[89] M. Dugré, F. Freyer und A. Anders, „BACnet and EnOcean enable Energy Efficient Buildings,“ 2009.

[90] T. Weinzierl und A. Anders, „KNX and EnOcean,“ 2009.

[91] LonMark International & EnOcean Alliance, „LonMark® International and EnOcean Alliance team-up for optimal network topologies in building automation,“ 2008.

[92] Z. Alliance, „Our goal is to provide standards to help you control your world,“ [Online]. Available: http://www.zigbee.org/About/AboutAlliance/TheAlliance.aspx. [Zugriff am 28 März 2014].

[93] S. Mahlknecht, T. Dang, M. Manic und S. A. Madani, „ZigBee,“ in *The Industrial Electronics Handbook - Industrial Communications Systems*, CRC Press, 2011, pp. 50.1-50.10.

[94] ZigBee Alliance, „ZigBee Specification,“ 2008.

[95] S. Ashton, „ZigBee Technology Overview,“ ZigBee Alliance, 2009.

[96] A. Elahi und A. Gschwender, ZigBee Wireless Sensor and Control Network, Prentice Hall, 2009.

[97] Dusan Stevanovic, „Zigbee / IEEE 802.15.4 Standard,“ ZigBee Alliance, 2007.

[98] ZigBee Alliance, „ZigBee Cluster Library Specification,“ 2012.

[99] Texas Instruments, „Z-Stack Developer’s Guide,“ 2011.

[100] ZigBee Alliance, „ZigBee IP Specification,“ 2013.

[101] J. Ploennigs, „Drahtlose und drahtgebundene Sensoren-Aktoren-Netzwerke,“ Technische Universität Dresden, 2012.

[102] digitalSTROM AG, „digitalSTROM,“ [Online]. Available: http://www.digitalstrom.com/. [Zugriff am 1 April 2014].

[103] eQ3, „HomeMatic,“ [Online]. Available: http://www.eq-3.de/homematic.html. [Zugriff am 1 April 2014].

[104] C. Bories, „Einrichtung einer intelligenten Ausleseeinheit für Verbrauchsmeßzähler,“ Dimplomarbeit, 1995.

[105] Europäisches Komitee für Normung, „Heat meters - Part 3: Data exchange and interfaces,“ EN 1434-3:1997 Standard, 1997.

[106] Europäisches Komitee für Normung, „Communication system for meters and remote reading of meters Data exchange,“ EN 13757-1:2002 Standard, 2002.

[107] Beckhoff Automation GmbH, „M-Bus-Anbindung für Energie- und Verbrauchszähler über TwinCAT,“ 2010.

[108] A. Papenheim, C. Bories und H. Ziegler, „The M-Bus: A Documentation,“ Universität Paderborn, 1997.

[109] Europäisches Komitee für Normung, „Communication systems for and remote reading of meters - Part 3: Dedicated application layer,“ EN 13757-3:2013 Standard, 2013.

[110] Europäisches Komitee für Normung, „Communication systems for and remote reading of meters - Part 2: Physical and link layer,“ EN 13757-2:2004 Standard, 2004.

[111] DLMS User Association, „What is DLMS/COSEM,“ [Online]. Available: http://www.dlms.com/information/whatisdlmscosem/index.html. [Zugriff am 03 April 2014].

[112] DLMS User Association, „We speak the same language,“ 2011.

[113] Europäisches Komitee für Normung, „Datenkommunikation der elektrischen Energiemessung - DLMS/COSEM - Teil 1-0: Normungsrahmen für die intelligente Messung,“ DIN EN 62056-1-0:2014 Norm, 2014.

[114] DLMS User Association, „DLMS/COSEM Architecture and Protocols,“ 2009.

[115] DLMS User Association, „COSEM interface classes and OBIS identification system,“ 2013.

[116] Europäisches Komitee für Normung, „Datenkommunikation der elektrischen Energiemessung - DLMS/COSEM - Teil 6-1: COSEM Objekt Identification System (OBIS),“ DIN EN 62056-6-1 Norm, 2013.

[117] M. Steffl, „Messwertstatus und OBIS-Kennzahlen Gas,“ *Datenmanagement in der Gasverordnung,* 2011.

[118] EDI@Energy, „OBIS-Kennzahlen-System,“ Bundesverband der Energie- und Wasserwirtschaft, 2013.

[119] VDE, „SyM2,“ [Online]. Available: http://www.vde.com/de/fnn/arbeitsgebiete/messwesen/Sym2/Seiten/default.aspx. [Zugriff am 04 April 2014].

[120] M. Wisy, „Smart Message Language Version 1.03,“ EMSYCON GmbH, 2008.

[121] Bundesamt für Sicherheit in der Informationstechnik, „Anlage IV: Feinspezifikation „Drahtgebundene LMN-Schnittstelle" Teil b: „SML – Smart Message Language“,“ in *Technische Richtlinie BSI TR-03109-1*, 2013.

[122] A. Fischedick, „Napster History,“ Napster, 2010.

[123] R. Steinmetz und K. Wehrle, Peer-to-Peer Systems and Applications, Springer-Verlag Berlin Heidelberg, 2005.

[124] J. Eberspächer und R. Schollmeier, „First and Second Generation of Peer-to-Peer Systems,“ in *Peer-to-Peer Systems and Applications*, Springer-Verlag Berlin Heidelberg, 2005.

[125] Dr. Scholl, „Napster protocol specification,“ 2000. [Online]. Available: http://opennap.sourceforge.net/napster.txt. [Zugriff am 07 Mai 2014].

[126] The Gnutella Developer Forum, „The Annotated Gnutella Protocol Specification v0.4," [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/. [Zugriff am 07 Mai 2014].

[127] K. Wehrle, S. Götz und S. Rieche, „Distributed Hash Tables," in *Peer-to-Peer Systems and Applications*, Springer-Verlag Berlin Heidelberg, 2005.

[128] P. Maymounkov und D. Mazieres, „Kademlia: A peer-to-peer information system based on the XOR metric," *International workshop on Peer-To-Peer Systems (IPTPS),* pp. 53-65, 2002.

[129] R. Brunner, A performance evaluation of the Kad-protocol, Institut Eurécom: Master Thesis, 2006.

[130] P. Danielis, Peer-to-Peer-Technologie in Teilnehmerzugangsnetzen, Dissertation, Universität Rostock, 2012.

[131] Enel S.p.A., „METERS AND MORE," prEN/TS 5ZZZZ Draft, 2011.

[132] A. Weidlich, S. Karnouskos und J. Ringelstein, „Technology trends for smarthouse/smartgrid," European Commision, 2010.

[133] S. Rogai, „Telegestore Project Progress & Results," *IEEE Internation Symposium on Power Line Communications and its Applications (ISPLC),* 2007.

[134] B. Botte, V. Cannatelli und S. Rogai, „The Telegestore project in Enel's metering system," *The 18th International Conference on Electricity Distribution,* pp. 1-4, 2005.

[135] J. O'Shaughnessy, M. Barash und C. Stanfield, „E.on Sweden selects Echelon's NES smart electricity metering system for 370,000 customers," Echelon Corporation, 2006. [Online]. Available: http://www.echelon.com/company/press/2006/eonse.htm. [Zugriff am 09 Mai 2014].

[136] J. Söderbom, „Smart Meter roll out experiences from Vattenfall," Vattenfall GmbH, 2012.

[137] Bundesamt für Sicherheit in der Informationstechnik , „Anforderungen an die

Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen," Technische Richtlinie BSI TR-03109, 2011.

[138] National Institute of Standards and Technology , „NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0," 2011.

[139] SMB Smart Grid Strategic Group (SG3), „IEC Smart Grid Standardization Roadmap," 2010.

[140] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski und D. Timmermann, „Implementing powerful Web services for highly resource-constrained devices," *Pervasive Computing and Communication Workshops (PerCom Workshops),* pp. 332-335, 2011.

[141] J. Montenegro, N. Kushalnagar und J. Hui, „Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *RFC 4944 Standard,* 2007.

[142] P. Deutsch, „GZIP file format specification version 4.3," Internet Standard RFC 1952, 1996.

[143] P. Deutsch, „DEFLATE Compressed Data Format Specification version 1.3," Internet Standard RFC 1951, 1996.

[144] J. Schneider und T. Kamiya, „Efficient XML Interchange (EXI) Format 1.0 (Second Edition)," W3C Recommendation, 2014.

[145] P. V. Biron und A. Malhotra, „XML Schema Part 2: Datatypes Second Edition," *W3C Recommendation,* 2004.

[146] P. Neumann, „Communication in industrial automation – what is going on?," *Control Engineering Practice,* pp. 1332-1347, 2007.

[147] C. Bournez, Efficient XML Interchange Evaluation, W3C Working Draft, 2009.

[148] G. Moritz, D. Timmermann, R. Stoll und F. Golatowski, „Encoding and Compression for the Devices Profile for Web Services," *IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA),* pp. 514-519,

2010.

[149] EXIficient team, „EXIficient,“ Siemens Research and Development, 2013. [Online]. Available: http://exificient.sourceforge.net/. [Zugriff am 14 Mai 2014].

[150] D. Dawson und T. Kamiya, „A Quick Introduction to OpenEXI,“ The OpenEXI Project, 2012.

[151] G. Moritz, „Web Services in stark ressourcenlimitierten Umgebungen,“ Universität Rostock, Dissertation, 2013.

[152] R. Kyusakov, „Embeddable EXI Processor in C,“ Embedded Internet Systems Laboratory (EISLAB), 2013. [Online]. Available: http://exip.sourceforge.net/. [Zugriff am 14 Mai 2014].

[153] Avnet Inc., „ZedBoard,“ 2014. [Online]. Available: http://www.zedboard.org. [Zugriff am 20 Mai 2014].

[154] M. Rethfeldt, Optimierte Datenkodierungsverfahren für ressourcenlimitierte Umgebungen, Universität Rostock: Bachelorarbeit, 2010.

[155] NS-3 Consortium, „Network Simulator 3 (ns-3),“ [Online]. Available: http://www.nsnam.org/. [Zugriff am 09 Mai 2014].

[156] S. Banerjee, S. Basu, S. Garg, S. Garg, S.-J. Lee, P. Mullan und P. Sharma, „Scalable grid service discovery based on UDDI,“ *The 3rd international workshop on Middleware for grid computing,* pp. 1-6, 2005.

[157] S. Poehlsen und C. Werner, „Robust Web Service Discovery in Large Networks,“ *IEEE International Conference on Services Computing,* pp. 521-524, 2008.

[158] R. Moreno-Vozmediano, „A hybrid mechanism for resource/service discovery in ad-hoc grids,“ *Future Generation Computer Systems,* p. 717–727, 2009.

[159] Z. ZhiHao, H. JiPing, D. Ting und W. Yu, „Semantic Web Service Similarity Ranking Proposal Based on Semantic Space Vector Model,“ *Second International Conference on Intelligent System Design and Engineering Application (ISDEA),* pp. 917-920, 2012.

[160] Web Services for Devices, „Java Multi Edition DPWS Stack (JMEDS),“ [Online]. Available: http://www.ws4d.org/jmeds/. [Zugriff am 09 Mai 2014].

[161] Embedded Linux Wiki, „RPi Hardware,“ [Online]. Available: http://www.elinux.org/RPi_Hardware. [Zugriff am 09 Mai 2014].

[162] Web Services for Devices, „DPWS Explorer,“ [Online]. Available: http://www.ws4d.org/dpws-explorer/. [Zugriff am 09 Mai 2014].

[163] R. Rivest, „The MD5 Message-Digest Algorithm,“ Internet Standard RFC 1321, 1992.

[164] A. Joshipura, „Infrastructure Innovation: Can the Challenge be Met?,“ Ericsson, 2010.

Liste der Veröffentlichungen

1. Jan Skodzik, Vlado Altmann, Peter Danielis, Moritz Koal, Dirk Timmermann: “An Optimized WS-Eventing for Large-Scale Networks“, *8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE)*, ISBN: 978-1-4799-4845-1, Barcelona, Spain, September 2014.
2. Peter Danielis, Jan Skodzik, Vlado Altmann, Eike Björn Schweissguth, Frank Golatowski, Dirk Timmermann, Jörg Schacht: “Survey on Real-Time Communication Via Ethernet in Industrial Automation Environments”, *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, ISBN: 978-1-4799-4845-1, Barcelona, Spain, September 2014.
3. Vlado Altmann, Jan Skodzik, Peter Danielis, Johannes Müller, Frank Golatowski, Dirk Timmermann: „A DHT-based Scalable Approach for Device and Service Discovery”, *12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 97-103, ISBN: 978-0-7695-5249-1, DOI: 10.1109/EUC.2014.23, Milan, Italy, August 2014.
4. Vlado Altmann, Hendrik Bohn, Frank Golatowski: “Web Services for Embedded Devices”, in *The Industrial Communication Technology Handbook 2nd Edition*, ISBN: 978-1-4822-0732-3, CRC Press, Boca Raton, Florida, USA, July 2014.
5. Vlado Altmann, Jan Skodzik, Peter Danielis, Nam Pham Van, Frank Golatowski, Dirk Timmermann: “Real-Time Capable Hardware-based Parser for Efficient XML Interchange”, *9th IEEE/IET International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pp. 415-420, ISBN: 978-1-4799-2581-0, Manchester, UK, July 2014.
6. Jan Skodzik, Vlado Altmann, Peter Danielis, Arne Wall, Dirk Timmermann: „A Kad Prototype for Time Synchronization in Real-Time Automation Scenarios”, *World Telecommunications Congress 2014*, pp. 1-6, ISBN: 978-3-8007-3602-7, Berlin, Germany, June 2014.
7. Jan Skodzik, Peter Danielis, Vlado Altmann, Dirk Timmermann: „HaRTKad: A Hard Real-Time Kademlia Approach”, *11th IEEE Consumer Communications & Networking Conference (CCNC)*, pp. 566-571, ISBN: 978-1-4799-2356-4, DOI: 10.1109/CCNC.2014.6866588, Las Vegas, USA, January 2014.
8. Jan Skodzik, Peter Danielis, Vlado Altmann, Dirk Timmermann: „Extensive Analysis of a Kad-based Distributed Storage System for Session Data”, *18th IEEE Symposium on Computers and Communications (ISCC)*, pp 489-494, ISBN: 978-1-4799-3755-4, DOI: 10.1109/ISCC.2013.6754994, Split, Croatia, July 2013.

9. Vlado Altmann, Peter Danielis, Jan Skodzik, Frank Golatowski, Dirk Timmermann: "Optimization of Ad Hoc Device and Service Discovery in Large Scale Networks", *18th IEEE Symposium on Computers and Communications (ISCC)*, pp. 833-838, ISBN: 978-1-4799-3755-4, DOI: 10.1109/ISCC.2013.6755052, Split, Croatia, July 2013.
10. Jan Skodzik, Peter Danielis, Vlado Altmann, Dirk Timmermann: „Time Synchronization in the DHT-based P2P Network Kad for Real-Time Automation Scenarios", *2nd IEEE WoWMoM Workshop on the Internet of Things: Smart Objects and Services (IoT-SoS)*, ISBN: 978-1-4673-5828-6, DOI: 10.1109/WoWMoM.2013.6583490, Madrid, Spain, June 2013.
11. Jan Skodzik, Vlado Altmann, Benjamin Wagner, Peter Danielis, Dirk Timmermann: „A Highly Integrable FPGA-Based Runtime-Configurable Multilayer Perceptron", *27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 429-436, ISBN: 978-1-4673-5550-6, DOI: 10.1109/AINA.2013.19, Barcelona, Spain, March 2013.
12. Vlado Altmann, Jens Rohrbeck, Jan Skodzik, Peter Danielis, Dirk Timmermann, Maik Rönnau, Matthias Ninnemann: „SWIFT: A Secure Web Domain Filter in Hardware", *27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 678-683, ISBN: 978-1-4673-6239-9, DOI: 10.1109/WAINA.2013.15, Barcelona, Spain, March 2013.
13. Vlado Altmann, Jan Skodzik, Frank Golatowski, Dirk Timmermann: "Investigation of the Use of Embedded Web Services in Smart Metering Applications", *38th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pp. 6172-6177, ISBN: 978-1-4673-2419-9, DOI: 10.1109/IECON.2012.6389071, Montreal, Canada, October 2012.
14. Peter Danielis, Jan Skodzik, Jens Rohrbeck, Vlado Altmann, Dirk Timmermann, Thomas Bahls, Danielis Duchow: „Using Proximity Information between BitTorrent Peers: An Extensive Study of Effects on Internet Traffic Distribution", *International Journal on Advances in Systems and Measurements*, Vol. 4, Nr. 3&4, pp. 212-221, ISSN:1942-261x, December 2011.
15. Jens Rohrbeck, Vlado Altmann, Stefan Pfeiffer, Peter Danielis, Jan Skodzik, Dirk Timmermann, Matthias Ninnemann, Maik Rönnau: „The Secure Access Node Project: A Hardware-Based Large-Scale Security Solution for Access Networks", *International Journal On Advances in Security*, Vol. 4, Nr. 3&4, pp. 234-244, ISSN:1942-2636, December 2011.

16. Jan Skodzik, Peter Danielis, Vlado Altmann, Jens Rohrbeck, Dirk Timmermann, Thomas Bahls, Daniel Duchow: „DuDE: A Distributed Computing System using a Decentralized P2P Environment", *36th IEEE LCN, 4th International Workshop on Architectures, Services and Applications for the Next Generation Internet*, pp. 1048-1055, ISBN: 978-1-61284-926-3, DOI: 10.1109/LCN.2011.6115162, Bonn, Germany, October 2011.
17. Jan Skodzik, Peter Danielis, Vlado Altmann, Jens Rohrbeck, Dirk Timmermann, Thomas Bahls, Daniel Duchow: „DuDE: A Prototype for a P2P-based Distributed Computing System", *36th IEEE LCN, 4th International Workshop on Architectures, Services and Applications for the Next Generation Internet*, Bonn, Deutschland, October 2011.
18. Jens Rohrbeck, Vlado Altmann, Stefan Pfeiffer, Dirk Timmermann, Matthias Ninnemann, Maik Rönnau: „Secure Access Node: an FPGA-based Security Architecture for Access Networks", *International Conference on Internet Monitoring and Protection (ICIMP)*, pp. 54-57, ISBN: 978-1-61208-004-8, St. Maarten, The Netherlands Antilles, March 2011.
19. Peter Danielis, Stephan Kubisch, Harald Widiger, Jens Rohrbeck, Vladyslav Altmann, Jan Skodzik, Dirk Timmermann, Thomas Bahls, Daniel Duchow: „Trust-by-Wire in Packet-Switched IPv6 Networks: Tools and FPGA Prototype for the IPclip System", *6th IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 1-2, ISBN: 978-1-4244-2309-5, DOI: 10.1109/CCNC.2009.4785006, Las Vegas, Nevada, USA, January 2009.

Selbstständigkeitserklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken habe ich als solche kenntlich gemacht.

Vlado Altmann Rostock, 30.09.14

Thesen

1. Die aktuelle Situation der Gebäudeautomation ist dadurch gekennzeichnet, dass viele proprietäre und geschlossene Standards für die kabelgebundenen als auch drahtlosen Automatisierungslösungen koexistieren. Zudem weisen diese Standards nur eine geringe oder keine Interoperabilität zueinander auf, sodass es nur schwer möglich ist, auf einfache Weise herstellerübergreifende Lösungen zu integrieren und dabei einen ganzheitlichen Ansatz zu verfolgen.
2. Die WS-Technologien besitzen ein breites Spektrum an Tätigkeitsbereichen und beweisen damit ihre Zukunftsfähigkeit. Die dynamische Struktur und das modulare Designprinzip erlauben es, WS-Technologien an nahezu alle Bedürfnisse anzupassen. Außerdem sind die WS-Spezifikationen öffentlich verfügbar.
3. Durch die flexiblen, anpassbaren und durchdachten Mechanismen von Web Services ist es möglich, ein beliebiges Protokoll auf diese abzubilden. Es lassen sich nachweisbar u.a. alle Protokolle im Smart Home, Smart Metering und Smart Building-Bereich auf Web Services abbilden.
4. Damit Web Services auf Medien mit niedrigen Datenraten eingesetzt werden können, kann eine geeignete Kompression angewendet werden. Dadurch kann der Kommunikationsoverhead der Web Services um 90 % reduziert werden. Die resultierende Performance ist vergleichbar mit spezialisierten Protokollen bei gleichzeitig bleibendem dynamischen Charakter und Anpassbarkeit der Web Services.
5. Die Standardmethode für die Suche nach Services (WS-Discovery) ist wenig skalierbar. Der Grund dafür ist der Multicast-Charakter der Suche sowie die fehlende Berücksichtigung der Netzwerk- und Geräteparameter.
6. Verfügt ein Client während der Service-Suche nicht über die notwendige Rechenleistung, können Pakete verworfen werden und damit auch die Antworten. Im schlimmsten Fall kann dies zu einem Buffer Overflow und damit auch zum Ausfall des Geräts führen.
7. Durch die dynamischen Optimierungen der Suchparameter zur Laufzeit können die anfallenden Datenraten am Client reduziert bzw. an eine vorgegebene Datenrate angepasst werden. Damit wird die Skalierbarkeit der Web Services deutlich erhöht und ermöglicht den Einsatz in Netzwerken beliebiger Größe.
8. Die vorgeschlagenen Optimierungen sind zu der Standard-WS-Discovery-Spezifikation rückwärtskompatibel, unterliegen jedoch den Einschränkungen des Multicasts, d.h., es können nur Services im lokalen Netzwerk gefunden werden.
9. Der aus den Peer-to-Peer-Netzwerken bekannte Ansatz der Distributed Hash Table (DHT) kann für die Suche nach Services genutzt werden. Als Grundlage kann die DHT-Implementierung Kademlia Verwendung finden. Für die Kommunikationen zwischen den Geräten wird nur Unicast benutzt.
10. Der DHT-basierte Ansatz für die Service-Suche nutzt eine zweischichtige Architektur. Die erste Schicht repräsentiert den Haupt-DHT-Ring. Auf dem Hauptring können Service-Typen und auf dem Service-Ring alle Service Provider desselben Service-Typs gefunden werden.

11. Mittels DHT bzw. Kademlia kann das Discovery über mehrere Subnetze hinweg, im Gegensatz zu Standard WS-Discovery, ausgeführt werden. Das suchende Gerät behält die vollständige Kontrolle über den Discovery-Prozess und kann den Datenfluss entsprechend seinen freien Ressourcen und seiner Verarbeitungsgeschwindigkeit anpassen. Es können jedoch nur diejenigen Geräte an der Suche teilnehmen, die auch Teil des DHT sind.
12. Um DHT-Discovery in Plug&Play-Netzwerken einsetzen zu können, kann das automatische Bootstrapping verwendet werden. Der Ansatz basiert auf Multicast und einer inkrementellen Suchtoleranz. Die Multicast-Nachrichten werden jedoch ausschließlich für das Bootstrapping verwendet, um einen beliebigen Knoten im lokalen Netzwerk zu finden.
13. Zum Verarbeiten von Web Service-Nachrichten ist ein spezieller Parser notwendig, um die Nutzdaten aus einem XML-Dokument zu extrahieren. XML-Parser sind für nahezu alle Programmiersprachen verfügbar, da XML ein weitverbreitetes Datenaustauschformat ist. Das Parsen von XML-Dokumenten kann grundsätzlich auf String-Parsen zurückgeführt werden, was „softwarefreundlich“ ist.
14. Der größte Nachteil von XML ist die Dokumentgröße. Abhängig von der Dokumentstruktur kann der Overhead ein Vielfaches der Nutzdaten ausmachen. In bestimmten Automationsszenarien, in denen Echtzeitsysteme oder Deeply Embedded Devices eingesetzt werden, kann dieser Umstand kritisch werden. Um von Web Service-Technologien in solchen Systemen zu profitieren, wurde EXI entwickelt.
15. Für eine sehr schnelle Datenverarbeitung in Echtzeitsystemen wie z.B. Motion Control sind Verarbeitungsgeschwindigkeiten von einigen Mikrosekunden erforderlich. Die Verarbeitungsgeschwindigkeit von softwarebasierten Systemen hängt dabei von der jeweiligen Hardware ab und kann stark variieren.
16. Ein Hardware-EXI-Parser kann die Verarbeitungszeit um ca. 95 % verglichen mit dem Software-basierten Ansatz reduzieren. Der Parser kann dynamisch zur Synthesezeit erstellt und an ein beliebiges XML-Schema angepasst werden.
17. Der Einsatz des Hardware-EXI-Parsers kann besonders in Umgebungen mit harten Echtzeitanforderungen und sehr kurzen Antwortzeiten wie z.B. Motion Control vorteilhaft sein. Hierbei wird die dynamische Struktur und Interoperabilität der Web Services nicht durch die Anwendung eingeschränkt.
18. Der EXI-Parser kann als ein Co-Prozessor für die EXI-Verarbeitung eingesetzt werden und dabei den Hauptprozessor entlasten. Durch die Verwendung von Standardkomponenten für die Parser-Generierung kann dieser leicht in existierende FPGA-basierte Systeme integriert werden.

Kurzreferat

In letzter Zeit hat das schnelle Wachstum im Smart Home- und Smart Metering-Bereich zu einer Entwicklung vieler spezifischer Protokolle geführt. Die meisten dieser Protokolle sind weder miteinander noch mit bereits existierenden Standards kompatibel. Darüber hinaus sind die spezifischen Protokolle nur für ein enges Aufgabenspektrum geeignet und somit wenig zukunftssicher. Im Vergleich dazu durchdringen die WS-Technologien ein breites Spektrum an Tätigkeitsbereichen und beweisen damit ihre Zukunftsfähigkeit. Die dynamische Struktur und das modulare Designprinzip erlauben es, WS-Technologien an nahezu alle Bedürfnisse anzupassen. Außerdem sind die WS-Spezifikationen öffentlich verfügbar.

Im Rahmen dieser Arbeit wurde untersucht, inwieweit es möglich ist, existierende proprietäre Protokolle auf Web Services abzubilden. Hierfür wurde zunächst verschiedene Protokolle aus den Bereichen Gebäudeautomation, Smart Home und Smart Metering untersucht. Anschließend wurde ein Protokoll ausgewählt und anhand dessen exemplarisch eine Abbildung auf Web Services durchgeführt. Durch den Vergleich beider Protokolle wurde festgestellt, dass die Web Services einen sehr hohen Datenoverhead aufgrund ihrer XML-basierten Struktur und des Übertragungsprotokolls HTTP aufweisen. Daraufhin wurden geeignete Kompressionsmethoden erarbeitet, die eine Datenreduktion bis zu 90 % ermöglichen, ohne dabei zusätzliche Rechen- und Speicherressourcen zu beanspruchen. Damit konnte gezeigt werden, dass Web Services im Vergleich zu proprietären Protokollen einen vergleichbaren Datenoverhead verursachen. Darüber hinaus konnte in einer analytischen Betrachtung die Allgemeingültigkeit des Ansatzes gezeigt werden.

Es wurde des Weiteren die Skalierbarkeit von Web Services untersucht. Dabei geht es insbesondere um die Gerätesuche in großen Netzwerken. Es wurde festgestellt, dass die Standardmethode für die Gerätesuche nur wenig skaliert. Der Grund dafür ist der Multicast-Charakter der Suche sowie die fehlende Berücksichtigung der Netzwerk- und Geräteparameter. Es wurden daraufhin Optimierungen zur Verbesserung der Skalierbarkeit der Web Services vorgeschlagen. Hierbei wurden zwei verschiedene Methoden untersucht, die sich aus mehreren Teiloptimierungen zusammensetzen. Mit den beiden vorgeschlagenen Methoden konnte die Skalierbarkeit der Gerätesuche wesentlich verbessert werden. Sie weisen jedoch unterschiedliche Vor- und Nachteile auf. Die Verbesserung der Skalierbarkeit konnte sowohl in Simulationen als auch in Feldversuchen bestätigt werden. Damit wurde gezeigt, dass Web Services in Netzwerken beliebiger Größer eingesetzt werden können.

Ein weiterer Aspekt dieser Arbeit war die Untersuchung der Web Services im Hinblick auf die Echtzeitfähigkeit. Hierbei wurde die Untersuchung auf die echtzeitfähige Verarbeitung der Web Service-Nachrichten beschränkt. Es konnte gezeigt werden, dass die Verarbeitung der Nachrichten in die Hardware verlagert werden kann. Hierbei kann die Verarbeitungszeit um

ca. 95 % verglichen mit den üblichen Software-basierten Ansätzen reduziert werden. Der Einsatz der Hardware-Lösung kann besonders in Umgebungen mit harten Echtzeitanforderungen und sehr kurzen Antwortzeiten wie z.B. Motion Control vorteilhaft sein. Hierbei wird die dynamische Struktur und Interoperabilität der Web Services nicht durch die Anwendung eingeschränkt.

Abstract

Recently, the rapid growth in the smart home and smart metering sector led to a development of many new specific protocols. However, most newly created protocols are neither interoperable with each other nor with already existing standards. Furthermore, the specific protocols are suited to a narrow range of applications making them not future-proof. In contrast, Web-based technologies continuously penetrate other spheres of activity and thus improve their sustainability. The dynamic structure and modular design principles allow to adapt WS technologies to nearly all needs. Moreover, the WS specifications are publicly available.

In this work, the possibility to map existing proprietary protocols onto Web services was investigated. Therefore, different protocols from building automation, smart home, and smart metering areas were analyzed. Subsequently, one protocol was chosen for the exemplary mapping onto Web services. After the comparison of both protocols, a high data overhead of the Web services due to XML-based structure and communication protocol HTTP was determined. Thereupon, suitable compression methods was developed, which allow for data reduction up to 90 % without additional computational and memory resource consumption. It was thereby shown, that Web services cause a similar data overhead in comparison to proprietary protocols. Moreover, the universality of the approach was shown in an analytical consideration.

Furthermore, the scalability of Web services was investigated. Especially, device discovery in large scale networks was addressed. It was concluded that the standard device discovery method scales poorly. The reason is the multicast nature of the discovery as well as the lack of considering network and device parameters. As a solution, optimizations for improvement of scalability of Web services were suggested. For this purpose, two different methods, which consist of several partial optimizations, were explored. With these two methods, the scalability of device discovery was improved considerably. They show however different advantages and disadvantages. The improvement of scalability was proven in simulations as well as field experiments. Hence, it was shown that Web services can be deployed in networks of any size.

Another aspect of this work was an exploration of the real-time capabilities of Web services. The exploration was confined to a real-time message processing of Web services. It was shown that the message processing can be moved to hardware. Thus, the processing time can be reduced by about 95 % compared to conventional software-based solutions. The deployment of the hardware solution is especially advantageous in environments with hard real-time requirements and very short response time such as motion control. Thereby, the dynamic structure and interoperability of Web services are not limited by an application.