FORSCHUNGSINITIATIVE Zukunft BAU

F 2940

Vlado Altmann, Dirk Timmermann

Webservices for Devices als Integrationsplattform für intelligente Dienste der Gebäudetechnik

Bundesinstitut für Bau-, Stadt- und Raumforschung
im Bundesamt für Bauwesen und Raumordnung

Fraunhofer IRB Verlag

F 2940

2015

ISBN 978-3-8167-9432-5

Fraunhofer IRB Verlag
Fraunhofer-Informationszentrum Raum und Bau

Postfach 80 04 69
70504 Stuttgart

Nobelstraße 12
70569 Stuttgart

Telefon 07 11 9 70 - 25 00
Telefax 07 11 9 70 - 25 08

E-Mail irb@irb.fraunhofer.de

www.baufachinformation.de

www.irb.fraunhofer.de/bauforschung

FORSCHUNGSINITIATIVE Zukunft BAU

Webservices for Devices als Integrationsplattform für intelligente Dienste der Gebäudetechnik

Endbericht

Der Forschungsbericht wurde mit Mitteln der Forschungsinitiative Zukunft Bau des Bundesinstitutes für Bau-, Stadt- und Raumforschung gefördert.
(Aktenzeichen: SF-10.08.18.7-11.4 / II 3-F20-11-004)
Die Verantwortung für den Inhalt des Berichtes liegt beim Autor.

Bearbeiter: M.Sc. Vlado Altmann

Projektleiter: Prof. Dr. Dirk Timmermann

09.09.2013

Universität Rostock Traditio et Innovatio

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zukunft Bau

Endbericht

Titel: Webservices for Devices als Integrationsplattform für intelligente Dienste der Gebäudetechnik

1. Einführung

In letzter Zeit hat ein schnelles 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 für ein enges Aufgabenspektrum geeignet und sind dabei weniger zukunftssicher. Im Vergleich dazu durchdringen die Web Service (WS)-Technologien andere Tätigkeitsbereiche wie z.B. Multimedia, Sensornetzwerke, Medizintechnik und beweisen damit ihre Zukunftsfähigkeit. Die dynamische Struktur und das modulare Designprinzip erlauben, 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 Device Language Message Specification/ Companion Specification for Energy Metering (DLMS/COSEM) [1], Smart Message Language (SML) [2], METERS&MORE [3] und andere zeigen keine Interoperabilität untereinander, obwohl sie demselben Zweck dienen. Die klassische Gebäudeautomation ist ebenfalls durch eine große Anzahl nicht interoperabler Protokolle wie z.B. LON [4], KNX [5], BACnet [6] 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 Benutzung der WS-Technologien gelöst werden, da diese über inhärente Plug&Play-Fähigkeiten verfügen und die Definition interoperabler Systeme erlauben. Devices Profile for Web Services (DPWS) [7] 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.

Das Ziel des Forschungsvorhabens ist es, die Einsatzmöglichkeiten von "Web Services for Devices" (WS4D) bzw. 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 des Vorhabens angefangen. Es werden dabei 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. Das Smart Metering-Profil wird dabei praktisch umgesetzt, getestet und evaluiert. Dabei werden die Mechanismen und Funktionsweise von DPWS sowie den Aufbau eines Profils basierend auf einem Protokoll erklärt. Hiermit wird erstmals die Möglichkeit der Abbildung eines Smart Metering-Protokolls auf DPWS bestätigt. Darüber hinaus wurde 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 ein analytischer Beweis geführt (siehe Abschnitt 5.3). Der analytische Beweis 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 bewiesenen 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.

2. Stand der Technik

Einige Smart Metering-Pilotprojekte haben bereits in Italien, Schweden, Frankreich, den Niederlanden, Griechenland, Deutschland und anderen Ländern stattgefunden [8]. In den meisten Fällen wurden diese Projekte durch die jeweilige Regierung unterstützt. Das italienische Projekt Telegestore wurde bereits im Jahr 2001 vorgestellt [9]. Das 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), welches SITRED genannt wurde, vorgestellt [10]. In Schweden wurde das Smart Metering durch die Gesetzgebung stark stimuliert [11]. Das entwickelte Smart Metering-System basiert auf dem proprietären Lontalk-Protokoll, welches von der Echelon Corporation entwickelt wurde [12]. In Deutschland wurde das neue Protokoll Smart Message Language eingeführt [2]. 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 [13]. Demnach soll die Kommunikation mit dem Smart Meter über ein Gateway ablaufen. 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 Transport Layer Security (TLS) abgesichert werden. Falls das Protokoll kein TLS unterstützt, müssen die Daten mit einem symmetrischen Verschlüsselungsverfahren und einer Signatur abgesichert werden.

Heutzutage ist das Internet Protocol (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 von National Institute of Standards and Technology (NIST) anerkannt [14]. Die Service-orientierte Architektur (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 [15]. 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.

Um ein Profil für DPWS zu entwickeln, wurde bei der Erfassung des Stands der Technik besonderer Wert auf die von BSI priorisierten Protokolle M-Bus [16], 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. Daraus resultiert aber ein höherer 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), 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 Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) Unterstützung mittels Bindings. SML ist ein reines Anwendungsschichtprotokoll und kann über beliebige Transportprotokolle wie TCP, Hypertext Transfer Protocol (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 1 gibt einen Überblick über die kumulativ übertragenen Datengrößen der einzelnen Protokolle auf der Anwendungsschicht (außer M-Bus). DLMS/COSEM führt den Verbindungsauf- und -abbau auf der Anwendungsschicht durch. Hierfür sind für die Übermittlung eines Messwertes mindestens drei Pakete notwendig. Die übertragenen Daten von DLMS/COSEM auf der Anwendungsschicht werden deswegen kumulativ als die Summe der drei Pakete betrachtet. Bei den gemessenen Datengrößen 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 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 1: Datengrößen der beispielhaften Protokollframes für die Übertragung eines Zählerwertes

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 Representational State Transfer (REST)-Architektur. Der Datenaustauch bei SML findet in Form einer Datei/eines Dokumentes statt. Die Messwerte und Parameter werden als eine Sequenz der Elemente durch die entsprechende Nachrichtentypen (Serviceaufrufe) abgefragt. Dadurch weist SML eine SOA-ähnliche Struktur auf. Einen Überblick über die Eigenschaften einzelner Protokolle liefert Tabelle 2. 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 2: Features ausgewählter Smart-Metering-Protokolle

3. Devices Profile for Web Services – Aufbau und Funktionsweise

DPWS ist eine Sammlung von WS-Standards, die für eine sichere WS-Funktionalität auf Geräten mit eingeschränkten Ressourcen entwickelt wurde [17]. Es ist eine Basistechnologie für die Gerätekommunikation, die mit den anderen WS-Spezifikationen kombiniert und erweitert werden kann [18]. DPWS weicht geringfügig von der Standard-WS-Architektur ab, um eine Geräte- und Service-orientierte Architektur und Infrastruktur zu ermöglichen. Es besteht aus WS-Discovery [19], WS-Eventing [20], WS-MetadataExchange [21], WS-Policy [22], WS-Transfer [23], WS-Security [24] und WS-Addressing (WSA) [25]. Der wesentliche Unterschied zwischen DPWS und Standard-WS besteht im Mechanismus, wie nach anderen Diensten gesucht werden kann. In DPWS erfolgt die Suche mittels eines Multicast Service Discovery - dem WS-Discovery. WS-Discovery benötigt keine zentrale Service-Registry wie bei Universal Description, Discovery and Integration (UDDI), dem ursprünglich für Web Services vorgeschlagenen zentralen Suchdienst [26]. Dabei wird der dezentralisierte Charakter der geräte-orientierten Netzwerke wie Wireless Sensor Networks, Home Automation Networks und Smart Meter Networks unterstützt. Der angebotene Service wird durch die Standardinterfaces aufgerufen, die vollständig mit der Standard-WS-Architektur kompatibel sind. Auf diese Weise werden ein hoher Interoperabilitätsgrad und Plug&Play-Funktionalitäten zwischen den DPWS-fähigen Geräten unter einander (wie z.B. in Gebäudeautomation) sowie DPWS-fähigen Geräten und Business-WS (wie z.B. Smart Metern und Energieanbieter) gewährleistet. In Abbildung 1 ist der DPWS-Protokoll-Stack abgebildet. Basierend auf der IP-Technologie ist Simple Object Access Protocol (SOAP) das Kernprotokoll des DPWS [26]. Die unteren Schichten werden mittels SOAP-Bindings wie z.B. SOAP-over-UDP [28], SOAP-over-HTTP [29] und anderen festgelegt. SOAP selbst basiert auf Extensible Markup Language (XML) [30]. Web Service Description Language (WSDL) wird für die Service-Beschreibung benutzt [31].

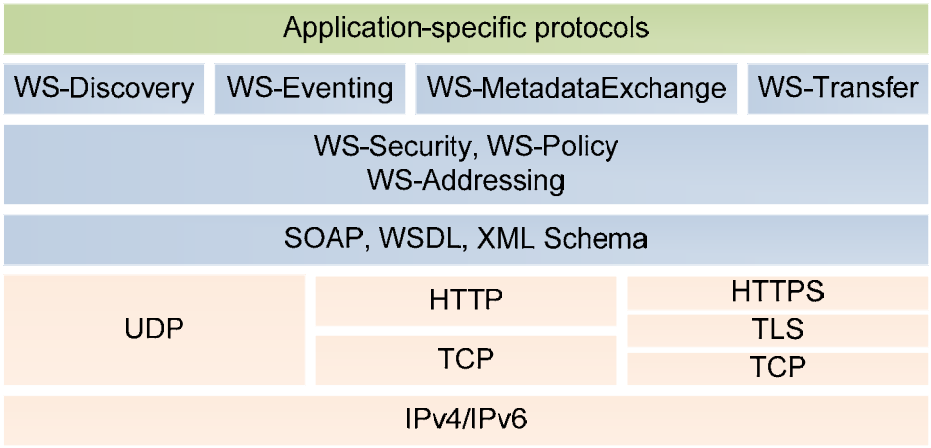

Abbildung 1: DPWS-Protokoll-Stack

Eine Besonderheit des DPWS ist das WS-Eventing. WS-Eventing definiert den Publishing- und Subscription-Mechanismus. Dabei wird der Service-Subscriber automatisch über Änderungen benachrichtigt, ohne eine zusätzliche Anfrage zu stellen. WS-Eventing stellt einen asynchronen Benachrichtigungsmechanismus dar.

4. Smart Message Language – Aufbau und Funktionsweise

SML ist ein Kommunikationsprotokoll für Smart Metering und wurde primär für Stromzähler entwickelt [2]. SML ist in die folgende Abschnitte unterteilt: Datenstrukturen für Payload-Übertragung, binäre SML-Codierung und das SML-Transportprotokoll. Das Letztere ist nur bei Punkt-zu-Punkt-Verbindungen anwendbar. Wie bereits erwähnt wurde, ist SML ein Anwendungsschichtprotokoll. Die Daten werden in Form einer Datei/eines Dokumentes 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 mit einer CloseRequest- oder CloseResponse-Nachricht entsprechend. Dazwischen können beliebig viele Request- und Response-Nachrichten platziert werden. Die spezifizierten SML-Nachrichten und deren Beschreibungen können der Tabelle 3 entnommen werden.

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 3: SML-Nachrichtentypen

Um die Funktionsweise von SML zu veranschaulichen, werden die GetProfileListRequest- und GetProfileListResponse-Nachrichten exemplarisch erklärt. Alle SML-Nachrichten folgen demselben Aufbau, der in Tabelle 4 dargestellt ist.

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

Tabelle 4: SML-Nachrichtenaufbau

Die *transactionId* wird vom Auftraggeber erstellt. Diese wird später verwendet, um die Antworten den Aufträgen zuzuordnen. Mit Hilfe 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 die 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 Aufbau der *GetProfileListRequest*-Nachricht ist in Tabelle 5 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 5: Aufbau der GetProfileListRequest-Nachricht

Wenn die Anfrage erfolgreich verarbeitet wurde, werden die angefragten Werte in der GetProfileListResponse-Nachricht übertragen. Der Aufbau dieser Nachricht ist in Tabelle 6 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. Der Aufbau anderer Nachrichten kann in der SML-Spezifikation nachgeschlagen werden.

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 6: Aufbau der GetProfileListResponse-Nachricht

5. Smart Meter-Profil für Embedded Web Services

In diesem Abschnitt wird die Möglichkeit einer Abbildung des SML-Protokolls auf DPWS gezeigt und anschließend die Performance beider Protokolle verglichen. Bei SML sind die Nachrichten ein Teil einer Datei/eines Dokumentes. 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 [32], 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 die Uniform Resource Identifier (URI) repräsentiert [33]. 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 7 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 7: 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.

5.1. Evaluierung des DPWS-Profils

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 [34]. Das Hauptproblem beim Einsatz von Web Services in Umgebungen mit eingeschränkten Ressourcen stellt weiterhin der hohe Kommunikationsoverhead dar. 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 [35]) oder drahtlose Low Power-Technologien wie 6LoWPAN zum Einsatz [36]. 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 Object Identification System (OBIS) (6 Gruppen mit Trennzeichen) [37].

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.

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

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 2 dargestellt. Wie erwartet, erzeugt DPWS einen sehr hohen Kommunikations-Overhead wegen des

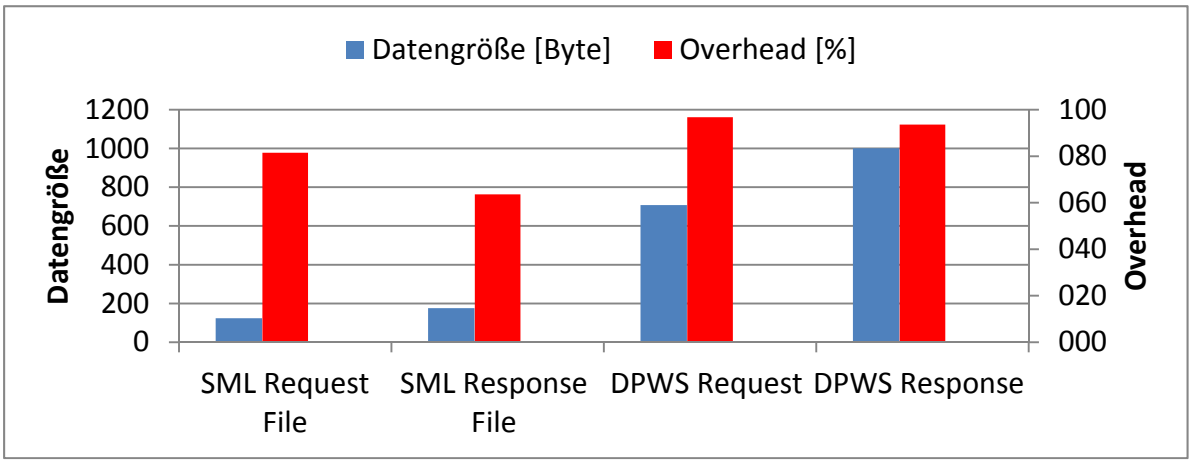

Abbildung 2: Vergleich des Kommunikations-Overheads von SML und DPWS

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 [38] oder Deflate [39] sind ungeeignet, da diese 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.

5.2. Kompression

Um HTTP-Header zu komprimieren, wird das Constrained Application Protocol (CoAP) vorgeschlagen [40]. 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, Type, Option Count, Code und Message ID. Das Type-Feld spezifiziert den Typ einer Nachricht und das Code-Feld gibt an, ob es sich um eine Anfrage oder eine Antwort handelt. 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 8 aufgeführt. Content-Type-Option entspricht dem Content-Type-Header

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

Tabelle 8: Mindestsatz an CoAP-Options für DPWS

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 3 dargestellt ist. Diese Kompressionsmethode erzeugt keine zusätzliche CPU- und Speicherauslastung. 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.

EXI stellt eine binäre XML-Codierung dar [41]. Die XML-Elemente werden durch Events repräsentiert. Ein EXI-Dokument besteht aus einer Folge der Events. Mit Hilfe des zusätzlichen Kontextes, der den Events zugeordnet werden kann, können XML-Attribute beschrieben werden. Im Gegensatz zu einer

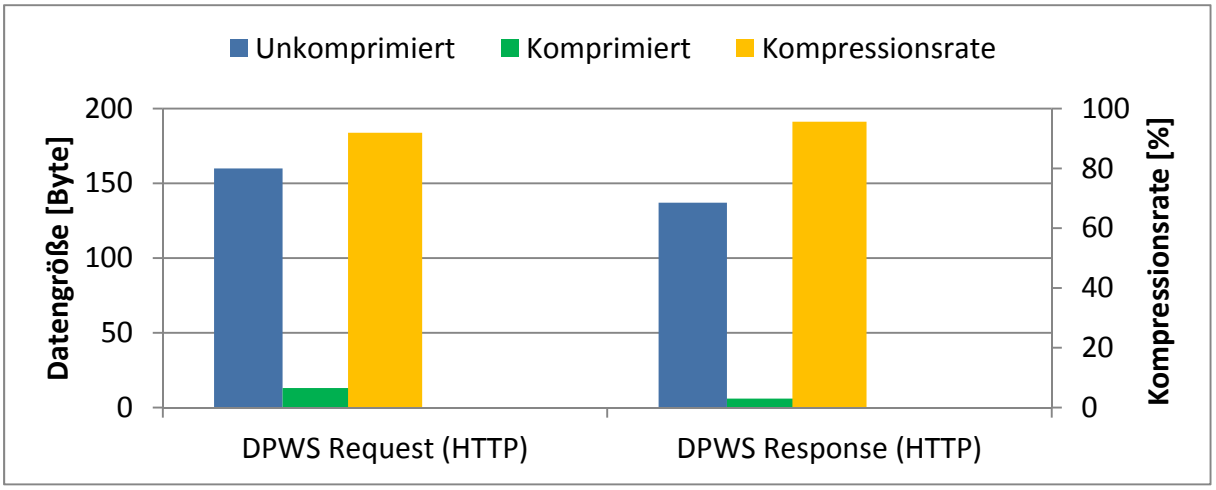

Abbildung 3: HTTP-Kompression

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. Zur Codierung eines XML-Dokumentes benutzt EXI Grammars (Grammatiken). Mit Hilfe 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 4).

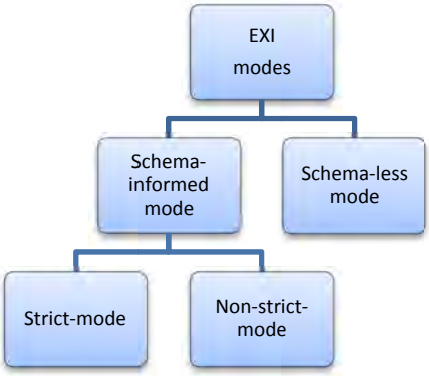

Abbildung 4: Exi Modes

Schema-less Mode wird benutzt, wenn keine Informationen über das XML-Schema vorliegen [42]. In diesem Fall werden die integrierten Grammars benutzt. Diese Grammars werden durch einen lernenden Mechanismus ständig erweitert.

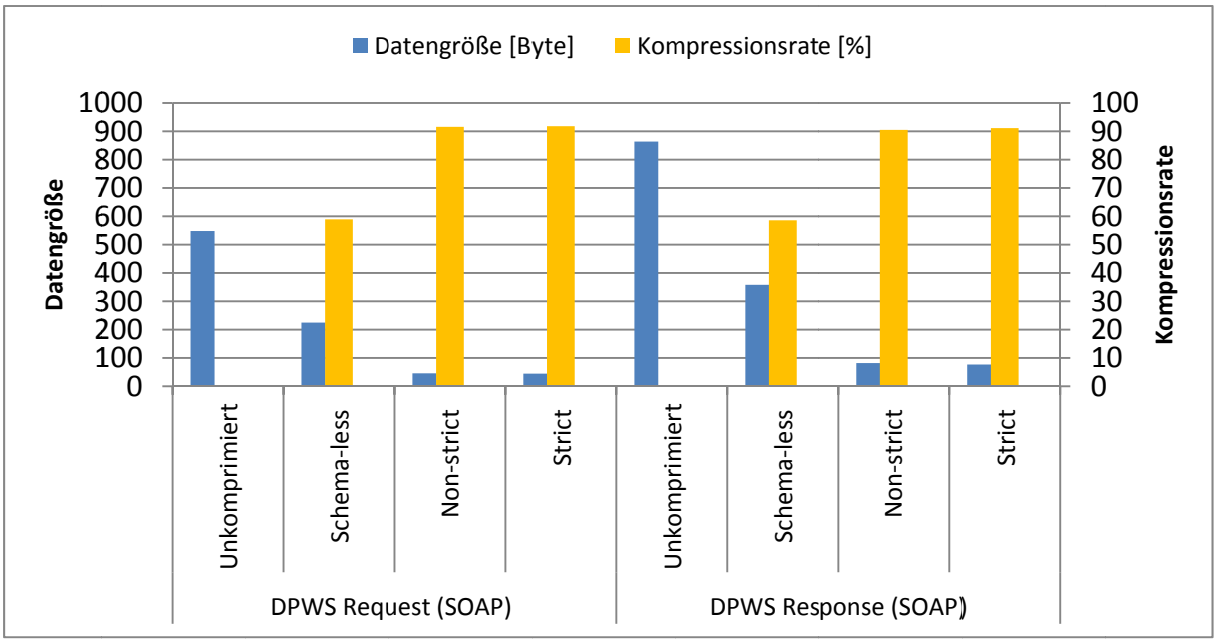

Abbildung 5: SOAP-Kompression

Daten, die einem Event zugeordnet sind, werden nur einmal codiert. Zum Beispiel wird der String eines schließenden Tags des XML-Dokumentes 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.

Für die weitere Overhead-Reduktion von DPWS wurde ein üblicher Universally Unique Identifier (UUID) [43], 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 5 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 6.

Die insgesamt erzielten DPWS-Kompressionsergebnisse sind in Abbildung 7 dargestellt. 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 dem SML und dem komprimiertem DPWS ist in Abbildung 8 dargestellt. Wie es in Abbildung 8 zu sehen ist, 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

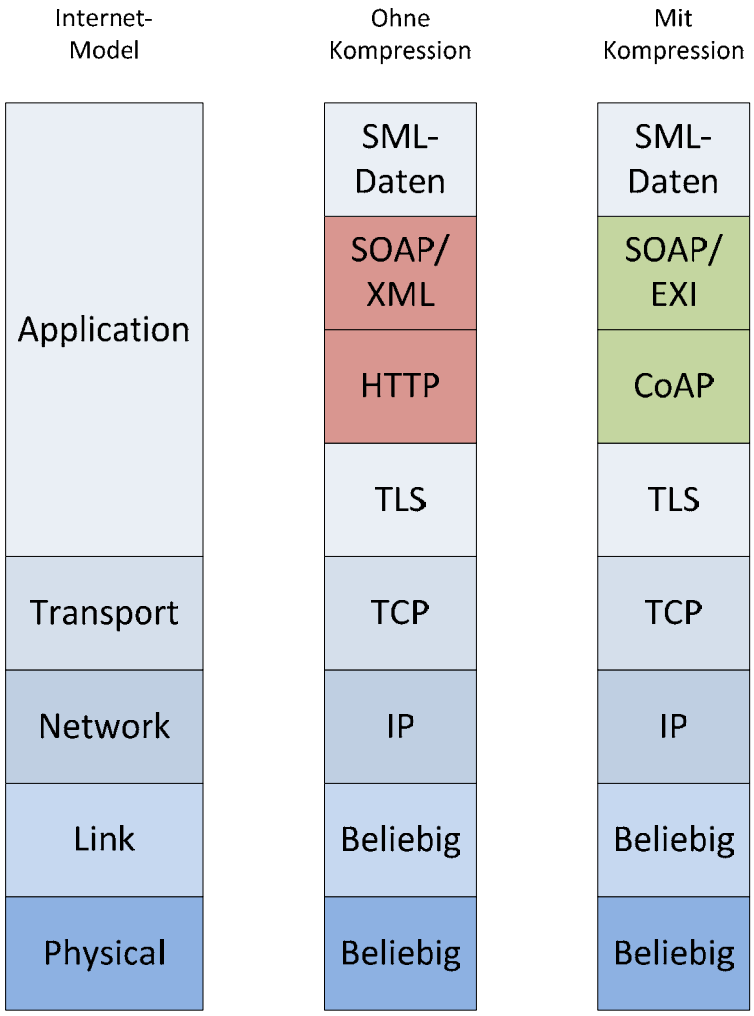

Abbildung 6: Überblick über die Kompressionsmethoden im Protokoll-Stack

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-/Antwortmethoden mit einer spezifischen Messeinrichtung zu verwenden.

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.

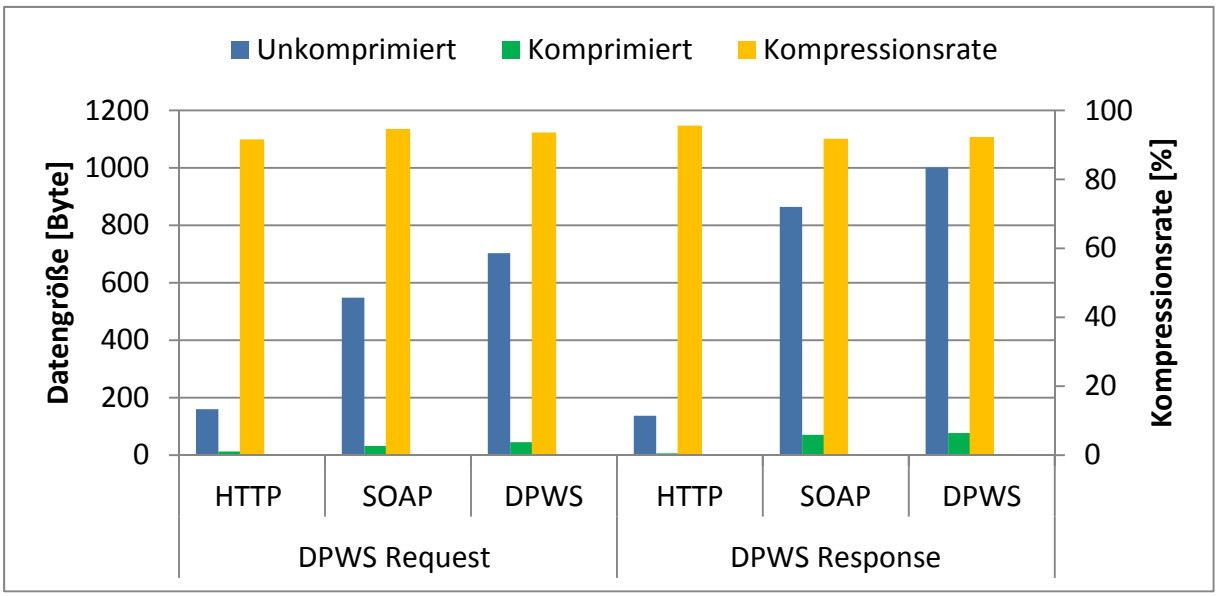

Abbildung 7: DPWS-Kompression

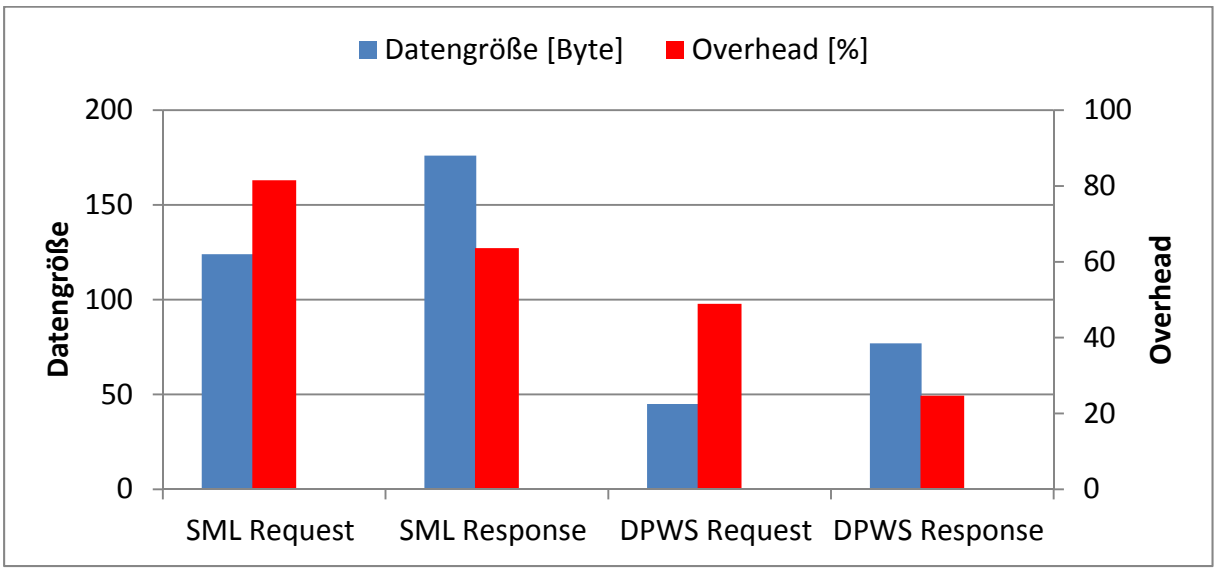

Abbildung 8: Vergleich des Kommunikations-Overheads von SML und komprimiertem DPWS

5.3. Analytische Betrachtung der Allgemeingültigkeit des vorgeschlagenen Ansatzes

Jedes Protokoll, unabhängig von der Anwendung und Einsatzgebiet, lässt sich in folgende Bausteine unterteilen, die in Tabelle 9 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 ist eine Menge von Protokollen *P* wie folgt definiert:

$$P = \{D, E, A, M, T\}$$

Als *P'* wird eine Menge von DPWS-Profilen angenommen, die eine Abbildung anderer Protokolle auf DPWS darstellen, für welche gilt:

$$P' = \{D', E', A', M', T'\}$$

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 beschreiben, welche Daten und ggf. in welcher Reihenfolge übertragen werden müssen.
Übertragung	Die Übertragung beschreibt, wie die Nachrichten zwischen Quelle und Senke übermittelt werden.

Tabelle 9: Generische Protokollbausteine

Wenn *p'* ein spezielles DPWS-Profil ist, dann ist es zu zeigen, dass

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

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 9). Das bedeutet:

$$\nexists d \in D: d \to d' \notin D'$$

Somit gilt:

$$\forall d \in D: d \to d' \in D'$$

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:

$$<N_E> V_E </N_E>$$

wobei $E = \{N_E, V_E\}$

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

<N_{E1}>
  <N_{E2}> V_{E2} </N_{E2}>
  <N_{E3}> V_{E3} </N_{E3}>
</N_{E1}>

Oder auch:

<N_{E1}>
  <N_{E2}>
    <N_{E3}> V_{E3} </N_{E3}>
  </N_{E2}>
</N_{E1}>

Die Kaskadierung kann dabei beliebige Tiefe haben. Somit gilt:

$$\forall e \in E : e \rightarrow e' \in E'$$

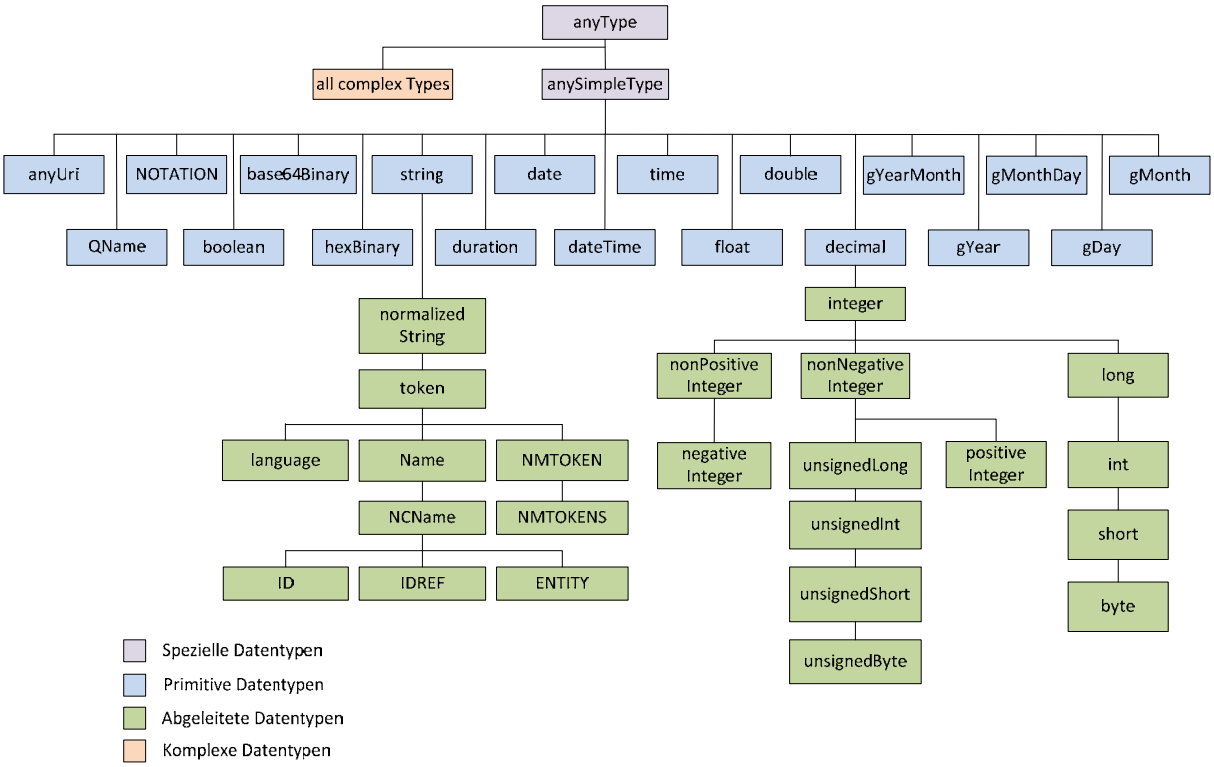

Abbildung 9: DPWS-Datentypen [44]

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:

$$\forall a \in A: a \rightarrow a' \in A'$$

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:

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

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 Source*Address* 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 Source*Address* 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 von einander 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 Addressierung 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:

$$\forall m \in M: m \rightarrow m' \in M'$$

Als nächstes wird die Übertragung betrachtet. Die Übertragung ist die Fähigkeit eines Protokolls die Nachrichten zwischen Quelle und Senke auszutauschen. DPWS unterstützt alle bekannten Nachrichtenaustauschmuster. Diese sind in Tabelle 10 dargestellt.

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:

$$\forall t \in T: t \to t' \in T'$$

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 10: Übertragungsarten

Es wurde hiermit gezeigt:

$$\forall p \in P: D \to D', E \to E', A \to A', M \to M', T \to T'$$

Damit gilt:

$$\forall p \in P: p \to p' \in P', \text{ was zu beweisen war.}$$

Somit wurde ein analytischer Beweis erbracht, 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 10). Konkret lassen sich also nachweisbar u.a. alle Protokolle im Smart Home, Smart Metering und Smart Building Bereich auf DPWS abbilden. Im Anhang A dieses Berichts befinden sich Beispiele, die eine Abbildung anderer Protokolle auf DPWS veranschaulichen. Die exemplarische Abbildung wird für ausgewählte Teile des ZigBee-Protokolls als einen Vertreter des Smart Home, des BACnet-Protokolls als einen Vertreter der klassischen Gebäudeautomation sowie des SML-Protokolls als einen Vertreter des Smart Metering gezeigt. Damit ist ein wesentliches Ziel des Vorhabens nun auch analytisch nachgewiesen.

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.

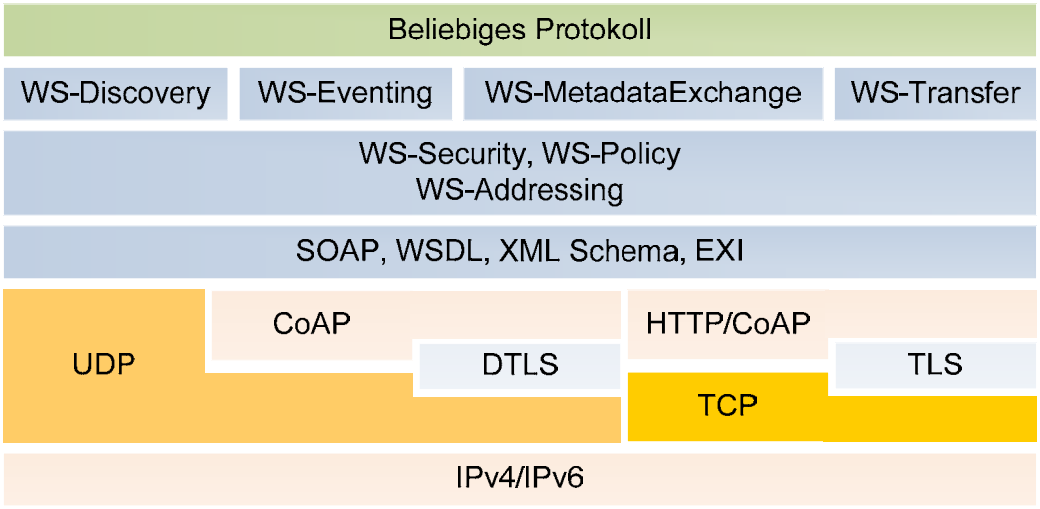

Abbildung 10: Abbildung eines beliebigen Protokolls auf DPWS

6. Optimierung von WS-Discovery für Großnetze

DPWS bietet ein automatisches Geräte-Discovery. 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 [19]. WS-Discovery stellt eine Grundlage für die Plug&Play-Kopplung der Geräte 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. 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. Der Discovery Proxy stellt einen *Single Point of Failure* (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, werden 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 Geräts führen. Um eine bessere Skalierbarkeit zu erzielen, wurde das Verhalten von WS-Discovery im Rahmen dieser Arbeit in Großnetzwerken mittels Simulation untersucht.

Für die Untersuchung wurde der *Network Simulator 3* (ns-3) verwendet [45]. Dieser bietet eine diskrete ereignisorientierte Netzwerksimulation. Der Vorteil von ns-3 ist es, dass eine Simulation bereits von der physikalischen Schicht her durchgeführt wird. Damit lassen sich komplexe Netzwerkabläufe und Verfahren testen. Als physikalisches Medium wurde *Switched Ethernet* als am meisten verbreitete Kommunikationstechnologie ausgewählt.

WS-Discovery nutzt SOAP-over-UDP Binding. Diese Spezifikation schreibt vor, ein zufälliges Delay zwischen 0 und 500 ms abzuwarten, bevor eine Antwort auf eine Discovery-Anfrage verschickt wird.

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. Mit wachsender Netzwerkgröße müssen mehrere Teilnehmer dasselbe Zeitintervall teilen, was zu einer steigenden Datenrate führt. Demnach würde ein dynamisch einstellbares Delay helfen, das Zeitintervall auf die Netzwerkgröße anzupassen. Mit Hilfe der Faustregel [40], dargestellt in Formel (2), ist es möglich, eine Schätzung über die mittlere Datenrate für den Client zu gewinnen:

$$D = S * \frac{G}{R} \tag{2}$$

Dabei bezeichnet D das Delay, S die Nachrichtengröße, G die Anzahl der Geräte und R die Datenrate. Allerdings ist für den Client nicht die mittlere sondern die Peak-Datenrate relevant, da zu Lastspitzen auch eine entsprechend hohe Rechenleistung verfügbar sein muss. Im Folgenden wird auf die einzelne Optimierungen von WS-Discovery eingegangen.

6.1. Einfügen der Hardware-Adresse

Für Device Discovery werden Probe-Nachrichten verwendet. Im Feld *Type* kann das gewünschte Gerät oder 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 Media Access Control (MAC) Adresse bekannt sein, um ein Paket zu erzeugen. Bei MAC-Adressen handelt es sich um weltweit eindeutige Adressen, die von Geräteherstellern vergeben werden. Im Falle einer Anfrage von einem unbekannten Gerät (Client) muss seine MAC-Adresse erfragt werden. Für diesen Zweck wird Address Resolution Protocol (ARP) verwendet [46]. 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 11 dargestellt.

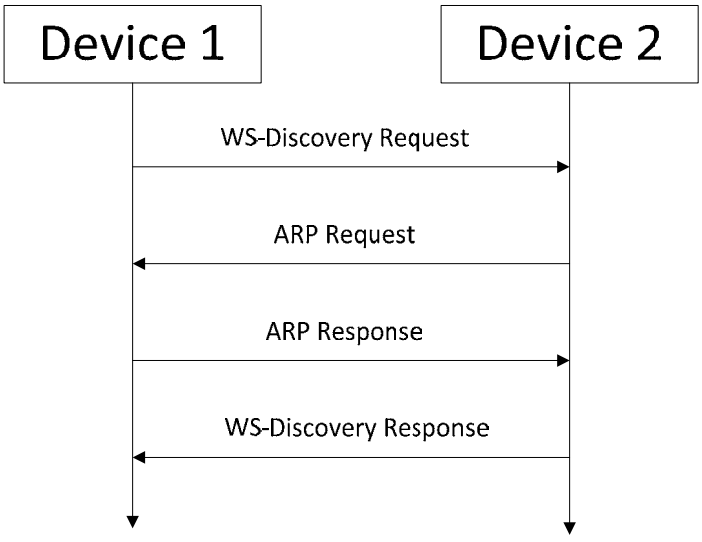

Abbildung 11: Nachrichtenaustausch während Discovery-Prozesses

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 dreimal so viele Nachrichten

verarbeiten wie Geräte im Netzwerk (ARP-Anfrage und 2 Mal WS-Discovery-Antwort) existieren. 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 Weglassen des ARP-Protokolls muss der Client nur noch die WS-Discovery-Antworten verarbeiten.

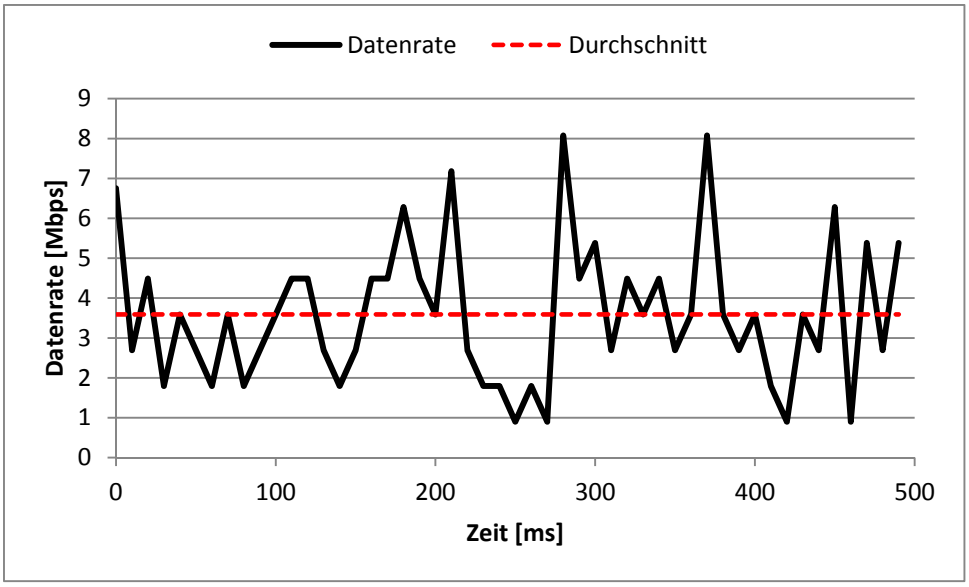

Abbildung 12: Vergleich der momentanen und der durchschnittlichen Datenraten

6.2. 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 (2) vergleichbar ist. Der Versuch wurde mit 200 Knoten ausgeführt. Wie in Abbildung 12 gezeigt ist, weicht die momentane Datenrate stark von der durchschnittlichen Datenrate ab. Die Peak-Datenrate kann doppelt so hoch

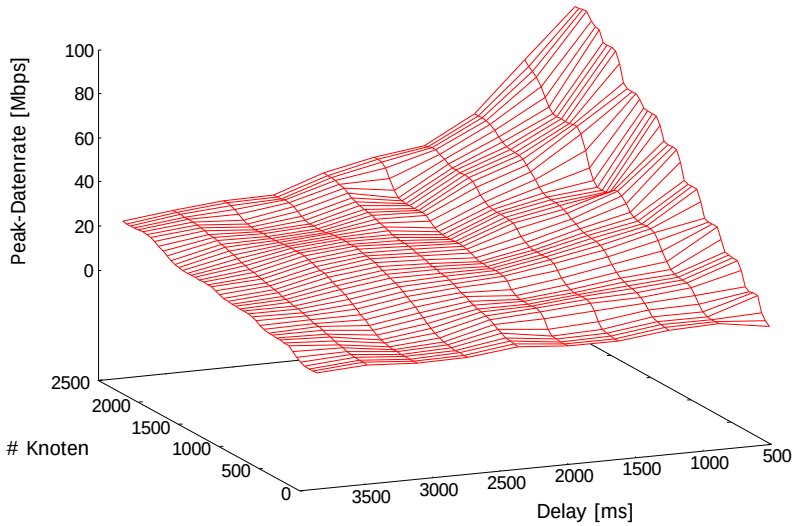

Abbildung 13: Peak-Datenrate in Abhängigkeit von Delay und Knotenanzahl

wie die durchschnittliche Datenrate sein. Dementsprechend ist die Berücksichtigung der Peak-Datenrate ein wichtiger Aspekt, um den Client im „worst case“ zu schützen.

Um das Verhalten von WS-Discovery abzuschätzen, wurden die Anzahl der Knoten und das Delay variiert und die Datenrate am Client gemessen. Für eine bessere Genauigkeit wurden ca. 100 Simulationsläufe mit verschiedenen Werten durchgeführt. Die Ergebnisse sind in Abbildung 13 dargestellt.

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

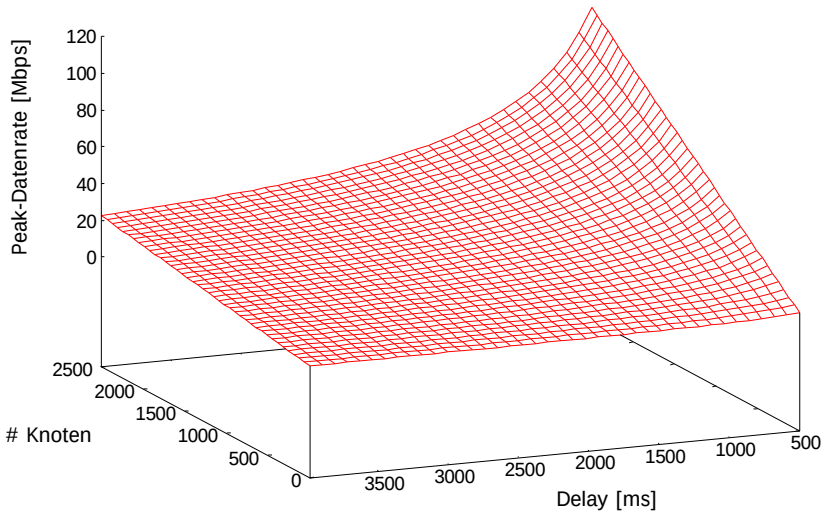

Abbildung 14: Approximation der Delay-Funktion

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 (3) wider.

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

Dabei ist D das Delay in ms, G ist die Knotenanzahl und R ist die Datenrate in Mbps. Die Approximationsfunktion ist in Abbildung 14 dargestellt.

Formel (3) stellt ein Kompromiss zwischen Peak-Datenrate und Systemantwortzeit dar. Da diese Formel empirisch aus Schätzungen ermittelt wurde, sollen 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 ist herstellerabhängig. 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 6.5 ein Mechanismus vorgeschlagen.

Nach der Ermittlung des Delays soll dieser 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. Eine Abweichung der Peak-Datenrate von der mittleren Datenrate bleibt jedoch dieselbe. Um diese zu reduzieren, wird eine Knotengruppierung vorgeschlagen.

6.3. Knotengruppierung

Beim Betrachten der Funktion in Abbildung 12 fällt auf, dass 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 das Verhalten der Datenrate ausbalancierter. Um dieses Verhalten zu erzielen, wird das gesamte Zeitintervall, welches für das Senden einer WS-Discovery-Antwort zur Verfügung steht (Delay-Zeit), 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 Dynamic Host Configuration Protocol (DHCP) Server [47]. 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 die meisten Geräte im niedrigen Teil des Adressraums landen. Folglich wird eine sequentielle Zuweisung der Zeitschlitze zu vollen und leeren Zeitschlitzen führen. Eine effizientere Methode ist es, 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 der Zeitschlitze zu bestimmen, wurden Simulationen für 250 bzw. 1000 Knoten und einer variablen Anzahl der Zeitschlitze durchgeführt. Darüber hinaus wird die kleinstmögliche Periode für einen Zeitschlitz 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 15 gezeigt ist, 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.

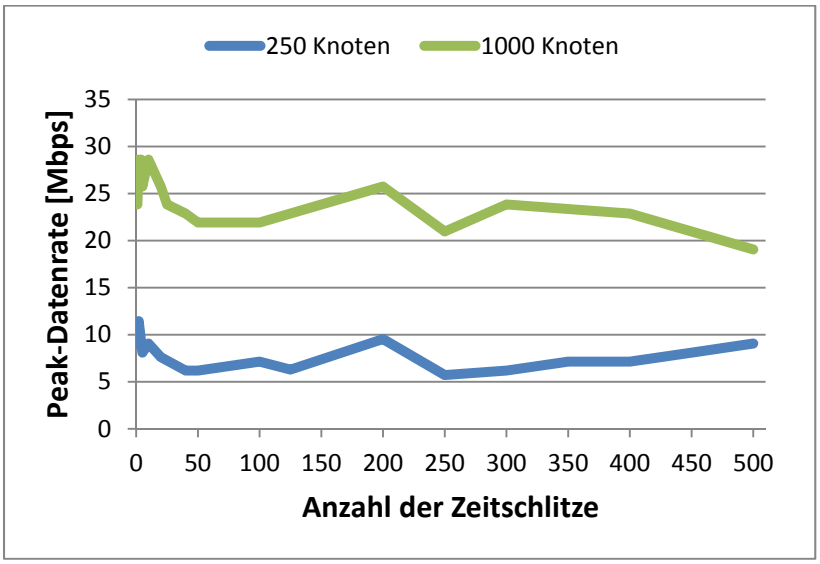

Abbildung 15: Peak-Datenrate in Abhängigkeit von der Anzahl der Zeitschlitze

Das bedeutet, die Peak-Datenrate ist am kleinsten, wenn es pro Zeitschlitz nur einen Knoten gibt. 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 aus dem gesammelten Wissen wurde eine Formel für die Schätzung der Anzahl der Zeitschlitze hergeleitet:

$$S = S_{max,} \text{ für } G > S_{max} \text{ und } S = G, \text{für } G < S_{max} \quad (4)$$

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, welches eine WS-Discovery-Antwort schicken will, muss seine Zeitschlitzzugehörigkeit bestimmen, d.h., in welchem Zeitschlitz das Gerät senden darf. 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 (5) 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.

$$S_{start} = \left\lceil (IP \bmod S) * \frac{D}{S} \right\rceil$$
$$S_{end} = \left\lceil \left((IP \bmod S) + 1\right) * \frac{D}{S} \right\rceil \quad (5)$$

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.

6.4. Optimierung der Paketwiederholung

In allen vorherigen Betrachtungen wurden die Paketwiederholungen entsprechend dem SOAP-over-UDP Binding ausgelassen. Die Wiederholungpakete kollidieren offensichtlich mit den ersten verzögerten WS-Discovery-Antworten anderer Geräte. Wenn das erste Beispielszenario (vgl. Abbildung 12) mit der Paketwiederholung durchgeführt wird, treten noch höhere Datenraten auf (vgl. Abbildung 16). Die durchschnittliche Datenrate wurde dem gesamten Zeitintervall und der Anzahl der gesendeten Paketen entsprechend angepasst.

Da SOAP-over-UDP Binding eine Paketwiederholung nach 50-250 ms nach dem Versenden des ersten Pakets vorschreibt, kann das Wiederholungspaket mit dem ersten Paket eines anderen Geräts kollidieren, welches 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 die zufälligen Netzwerkfehler zu vermeiden, soll das Wiederholungsintervall eine Spiegelung des Hauptintervalls darstellen. In diesem Fall wird die zeitliche Entfernung zwischen dem Originalantwort und 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.

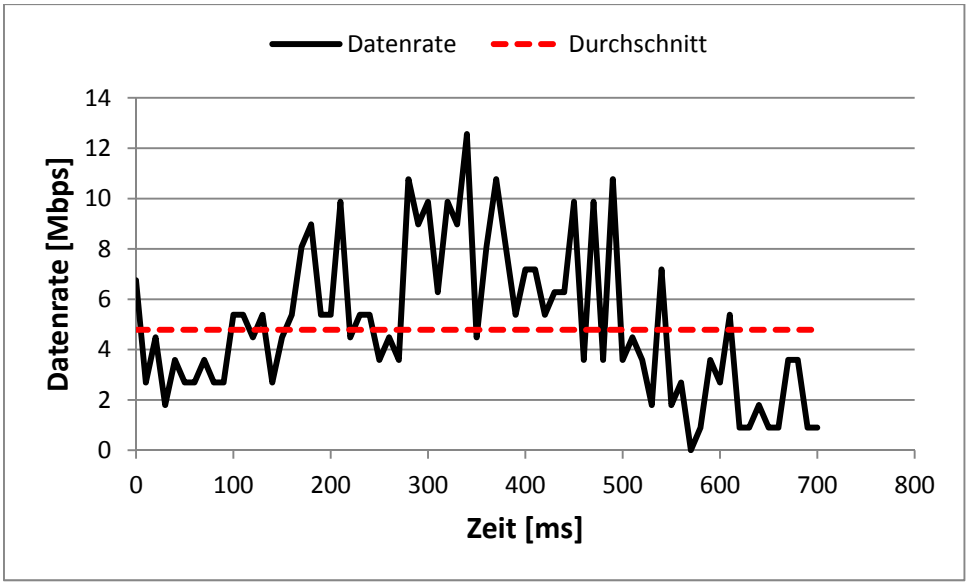

Abbildung 16: Datenrate von WS-Discovery mit Paketwiederholungen

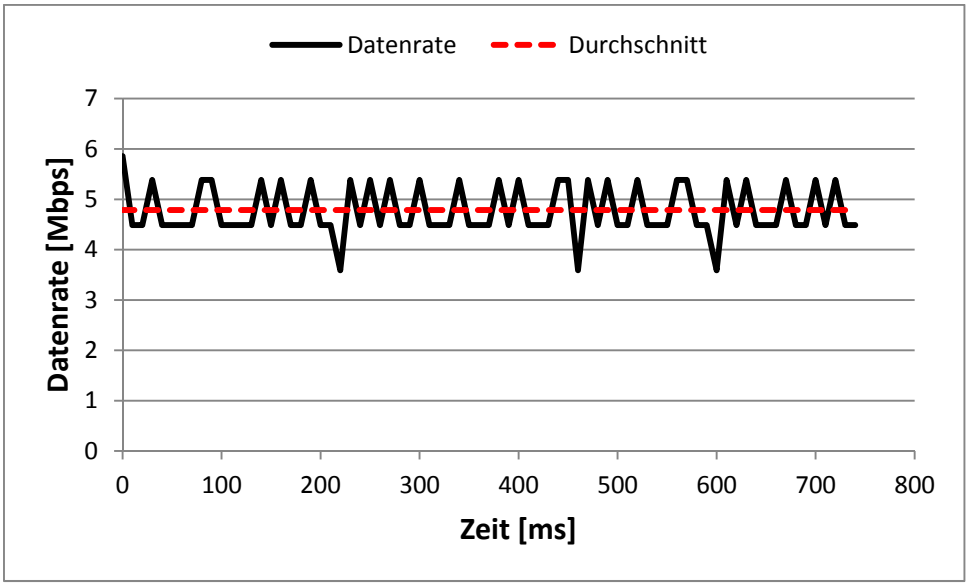

Abbildung 17: Datenrate von WS-Discovery mit angewandten Verbesserungen

Wenn alle Optimierungen auf das beschriebene Szenario (vgl. Abbildung 12) angewendet werden, kann eine wesentliche Reduzierung der Peak-Datenrate (über 50 %) auf der Client-Seite erreicht werden (siehe Abbildung 17). Wie bereits erwähnt wurde, 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 (2), mit etwas pessimistischen Werten bereits den Anforderungen an die Peak-Datenrate genügen. Auf diese Weise kann die Formel (2) 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 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.

6.5. 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. Diese Vorgehensweise ist allerdings in Automations- und Plug&Play-Netzwerken nicht erwünscht, da der direkte Gerätezugriff nicht möglich ist oder nicht im Interesse des Betreibers liegt. Zur Lösung dieses Problems wird ein neuer Mechanismus *Network Size Discovery* vorgeschlagen, welcher im Rahmen des Projektes entwickelt wurde. 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 (WS-Discovery Standard) 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 zum Ausfall einzelner Netzwerkteilnehmer oder -Komponenten führen. Zur Vermeidung der hohen 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, welches 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 NetSize-Nachricht über die aktuelle Netzwerkgröße informiert werden. Wie bereits erwähnt wurde, sind mindestens 3 NetSize-Nachrichten notwendig, 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 NetSize-Nachricht schickt. In der ersten Phase werden 3 Nachrichten erwartet. Wenn ein Gerät bereits 3 NetSize-Nachrichten von anderen Geräten detektiert, soll die eigene Nachricht verworfen werden. Trotz Sendewahrscheinlichkeit und Verwurf von Nachrichten können etwas mehr als 3 NetSize-Nachrichten 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 NetSize-Nachrichten) und das Zeitintervall verdoppelt. Diese Prozedur kann wiederholt werden, bis der richtige Wert ermittelt ist. Wenn die Anzahl der Geräte kleiner 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 (WS-Discovery Standard). Alle anderen Geräte müssen darauf hin 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 NetSize-Nachrichten 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 in Abhängigkeit 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 in der 1. Phase verschickt wurden oder bei einer Diskrepanz der Werte, wechseln alle Geräte in die nächste Phase. In jeder nachfolgenden Phase werden die Sendewahrscheinlichkeit und das Zeitintervall verdoppelt.

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 passieren, wenn alle Geräte gleichzeitig eingeschaltet werden. Hierfür wird eine Einschaltwartezeit von mindestens einer Minute vorgeschlagen, um eine Netzwerkverstopfung und Geräteauslastung zu vermeiden.

6.6. Zusammenfassung: 5-Schritte-Ansatz zur Reduzierung des Datenverkehrs beim Discovery

1. **Einfügen der Hardware Adresse**
   Ziel: Reduzierung der Anzahl der Nachrichten durch 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 von hohen Datenraten, die vom Client nicht verarbeitet werden und diesen u.U. außer Betrieb setzen können.
   Ergebnis: Die Datenraten können dynamisch an die Anforderungen 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.

7. Gebäudeautomation mit DPWS

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.

7.1. Web Services Description Language

WSDL ist eine Metasprache, mit der Services in Form von XML-Dateien beschrieben werden können. Diese Beschreibung definiert die Art und Weise, wie der Client mit dem Service interagieren kann. Dazu gehören die Definitionen von Operationen, funktionelle Angaben zur Schnittstelle, Angaben zum Netzwerkprotokoll etc. Ein WSDL-Dokument beschreibt den Aufbau eines Services als eine Menge von *Endpoints* und deren Interaktionen (Operationen) mittels Nachrichten (Messages) [31]. 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, welcher 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 11 gezeigt ist. Es wird hierbei auf WSDL 1.1 eingegangen, da nur diese Version von DPWS unterstützt wird [17].

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 11: Elemente eines WSDL-Dokuments (WSDL 1.1)

Ein WSDL-Dokument kann in einen konkreten und einen abstrakten Teil aufgeteilt werden. Der Aufbau eines WSDL-Dokumentes ist in Abbildung 18 dargestellt.

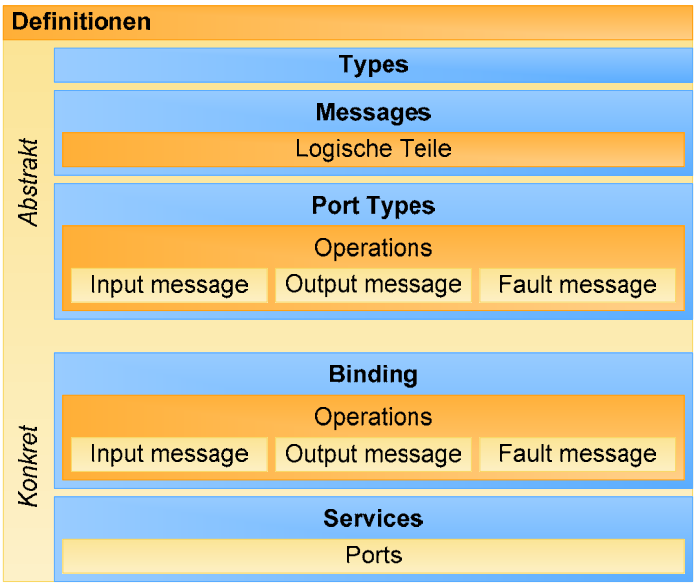

Abbildung 18: Aufbau eines WSDL-Dokumentes

Abstrakter Teil: Der abstrakte Teil eines WSDL-Dokumentes 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. WSDL nutzt 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. Eine beispielhafte Request-Response-Operation „ExampleOperation“ kann wie folgt durch Input- und Output-Messages dargestellt werden:

```
<wsdl:operation name="ExampleOperation">
    <wsdl:input name="ExampleOperationRequest"
        message="ExampleOperationRequestMessage" />
    <wsdl:output name="ExampleOperationResponse"
        message="ExampleOperationResponseMessage" />
</wsdl:operation>
```

Wären im oberen Beispiel die Input- und Output-Messages vertauscht, würde es sich um eine Solicit-Response-Operation handeln.

Konkreter Teil: Der konkrete Teil enthält die technischen Details zu WS. Ein Binding weist ein Port Type, seine Operationen und Messages einem bestimmten Protokoll und Datenformat zu. Mehrere Bindings können dabei zu einem Port Type erstellt werden. Alle definierten Port Types müssen sich im konkreten Teil widerspiegeln. Der Service stellt eine Sammlung von Ports dar. Ports sind die Netzwerkadressen für die Service Endpoints.

Die Funktionalität der WSDL wird anhand eines Beispiels verdeutlicht. Als Beispiel wird eine einfache Klimaanlagensteuerung verwendet. Die Klimaanlage bietet einen Service namens *AirCondService* an, über den es möglich ist, die aktuelle Raumtemperatur abzufragen und eine gewünschte Temperatur zu setzen:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=”http://www.demo.com/bbsr“
  xmlns:tns=”http://www.demo.com/bbsr” xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/“
  xmlns:xs=”http://www.w3.org/2001/XMLSchema”
  xmlns:wsam=”http://www.w3.org/2007/05/addressing/metadata“
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
  <wsdl:types>
    <xs:schema targetNamespace=”http://www.demo.com/bbsr” elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <xs:element name="temperature" type="tns:TemperatureType" />
      <xs:element name="status" type="xs:string" />
      <xs:complexType name="TemperatureType">
        <xs:sequence>
          <xs:element name="value" type="xs:int" />
          <xs:element name="unit" type="xs:string" />
          <xs:element name="timeStamp" minOccurs="0" type="xs:time" />
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="GetCurrentTemperatureMessage" />
  <wsdl:message name="GetCurrentTemperatureResponseMessage">
    <wsdl:part name="parameters" element="tns:temperature" />
  </wsdl:message>
  <wsdl:message name="SetDesiredTemperatureMessage">
    <wsdl:part name="parameters" element="tns:temperature" />
  </wsdl:message>
  <wsdl:message name="SetDesiredTemperatureResponseMessage">
    <wsdl:part name="parameters" element="tns:status" />
  </wsdl:message>
  <wsdl:portType name="AirCondInterface">
    <wsdl:operation name="GetCurrentTemperature">
```

```
      <wsdl:input name="GetCurrentTemperature"
        message="tns:GetCurrentTemperatureMessage"
        wsam:Action="http://www.demo.com/bbsr/AirCondInterface/GetCurrentTemperature" />
      <wsdl:output name="GetCurrentTemperatureResponse"
        message="tns:GetCurrentTemperatureResponseMessage"
        wsam:Action="http://www.demo.com/bbsr/
          AirCondInterface/GetCurrentTemperatureResponse" />
    </wsdl:operation>
    <wsdl:operation name="SetDesiredTemperature">
      <wsdl:input name="SetDesiredTemperature" message="tns:SetDesiredTemperatureMessage"
        wsam:Action="http://www.demo.com/bbsr/AirCondInterface/SetDesiredTemperature" />
      <wsdl:output name="SetDesiredTemperatureResponse"
        message="tns:SetDesiredTemperatureResponseMessage"
        wsam:Action="http://www.demo.com/bbsr/
          AirCondInterface/SetDesiredTemperatureResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="AirCondInterfaceBinding" type="tns:AirCondInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetCurrentTemperature">
      <soap:operation
        soapAction="http://www.demo.com/bbsr/AirCondInterface/GetCurrentTemperature" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="SetDesiredTemperature">
      <soap:operation
        soapAction="http://www.demo.com/bbsr/AirCondInterface/SetDesiredTemperature" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AirCondService">
    <wsdl:port name="AirCondInterfacePort0" binding="tns:AirCondInterfaceBinding">
      <soap:address location="http://192.168.1.151:37683/bbsr/AirCondService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Der Service hat einen Port von Typ *AirCondInterface* (festgelegt durch das *AirCondInterfaceBinding*) und wird über die Netzwerkadresse http://192.168.1.151:37683/bbsr/AirCondService angesprochen. Dieser Port Type bietet zwei Operationen an *SetDesiredTemperature* und *GetCurrentTemperature.* Die beiden Operationen werden durch die jeweiligen *Action*-Namen gekennzeichnet. Beide Operationen verfolgen das Request-Response-Muster, da eine Output-Message nach einer Input-Message folgt. Um die aktuelle Raumtemperatur abzufragen, muss eine Anfrage ohne Parameter gestellt werden. Als Antwort wird die aktuelle Temperatur, bestehend aus einem Zahlenwert (*value*) als Integer (*xs:int*), einer Maßeinheit (*unit*) als String (xs:string) und einem optionalen Zeitstempel (*timeStamp*) vom Typ Time (*xs:time)* geschickt. Bei der *SetDesiredTemperature* muss die gewünschte Temperatur an den Service geschickt werden. Hierfür werden dieselben Parameter für die Temperatur wie oben beschrieben verwendet. Als Antwort wird ein Status in Form eines Strings geschickt. Wie aus dem Beispiel ersichtlich ist, kann die Benutzung eines Services vollständig aus der WSDL-Beschreibung gelernt werden. Darüber hinaus kann WSDL einer schnellen und kostengünstigen Geräteentwicklung verhelfen. Damit bildet WSDL neben WS-Discovery eine weitere Grundlage für eine Plug&Play-Gerätekommunikation.

7.2. 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. 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 19). Die ResolveMatch-Nachricht entspricht der ProbeMatch-Nachricht und trägt die

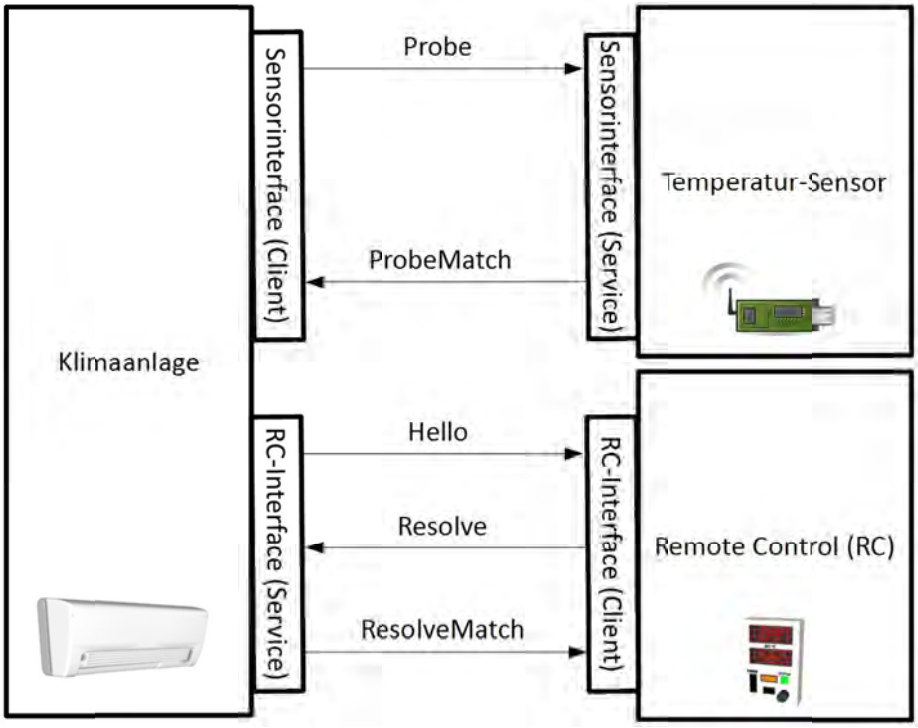

Abbildung 19: Geräte-Pairing mit DPWS

Transportadresse des Gerätes (Xaddrs). Mit Hilfe 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 Softwareinformation. WSDL enthält die Service-Beschreibung, d.h. welche Funktionen zur Verfügung stehen und wie diese aufgerufen werden müssen. 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 mit Hilfe 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. Das heißt, es muss dafür ein anderer Übertragungskanal benutzt werden. Für diesen Zweck bietet sich die Radio-Frequency Identification (RFID) 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. Die festgelegten Geräte-Namen können dann während des Discovery-Prozesses in Probe- und Hello-Nachrichten sowie in den Geräte-Metadaten verwendet werden.

8. Demonstrationsszenario

Um die Möglichkeiten von DPWS im Smart Home zu veranschaulichen, wird ein Demonstrationsszenario vorgeschlagen. Dabei werden verschiedene Konfigurationsmöglichkeiten wie statische und dynamische Konfiguration dargestellt.

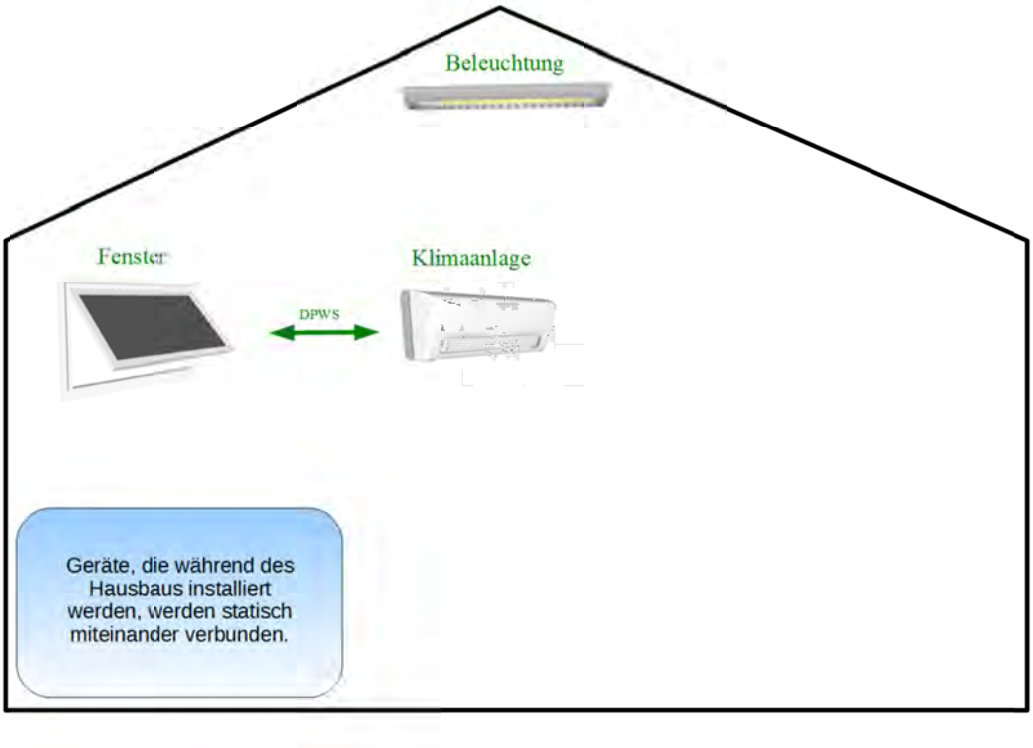

Abbildung 20: Demonstrationsszenario - Statische DPWS-Konfiguration

Statische Konfiguration findet in der Regel bei der Installation durch einen Fachmann statt. Dabei können Betriebsparameter mit Verzicht auf die Plug&Play-Eigenschaften gesetzt werden. So kann z.B. beim Hausbau eine Klimaanlage mit dem Fenster verbunden werden (vgl. Abbildung 20). Dabei werden auch weitere Geräte wie z.B. Beleuchtung installiert. Diese verfügen über eine DPWS-Funktionalität, können aber frei mit den nutzereigenen Geräten verbunden werden. Der Nutzer kann die Klimaanlage mit einem zusätzlichen Außentemperatursensor ausstatten. Die Sensoreinheit bietet darüber hinaus auch einen Lichtsensor, der gleichzeitig mit der Beleuchtung verbunden werden kann. Aktuelle Lichtverhältnisse können für das automatische Dimmen benutzt werden. Der Nutzer kann auch eine zentrale Steuereinheit für die Klimaanlage und die Wärmepumpe in Form eines kleinen Bedienpanels mit einer LCD-Anzeige erwerben. Sowohl der Außensensor als auch die Steuereinheit werden mittels Plug&Play mit den jeweiligen Geräten verbunden (vgl. Abbildung 21).

Zur Steuerung der „intelligenten“ Umgebung sind mobile Geräte gut geeignet. Der Nutzer kann mit seinem Handy, Tablet oder auch Fernseher die Temperatur oder Beleuchtung im Raum anpassen. Da die vorgeschlagene Lösung vollständig auf Internettechnologien basiert, ist für die Steuerung mit einem Handy lediglich die Installation einer geeigneten App notwendig. Dieser Ansatz ist besonders für Menschen mit eingeschränkter Mobilität wichtig (vgl. Abbildung 22).

Um den aktuellen Stromverbrauch oder die Stromerzeugung zu visualisieren und zu überwachen, können die mobilen Geräte auf die Messdaten über das Smart Meter Gateway zugreifen. Das Smart Meter Gateway kommuniziert dabei mit dem Smart Meter mittels des vorgeschlagenen DPWS-Profils für Smart Metering (vgl. Abbildung 23). Das Gateway kann dabei die verschlüsselten komprimierten Daten vom Smart Meter anderen Geräten im Haus in unverschlüsselter und nichtkomprimierter Form zur Verfügung stellen. Der Gateway-Ansatz wurde in Anlehnung an die BSI-Richtlinie gewählt.

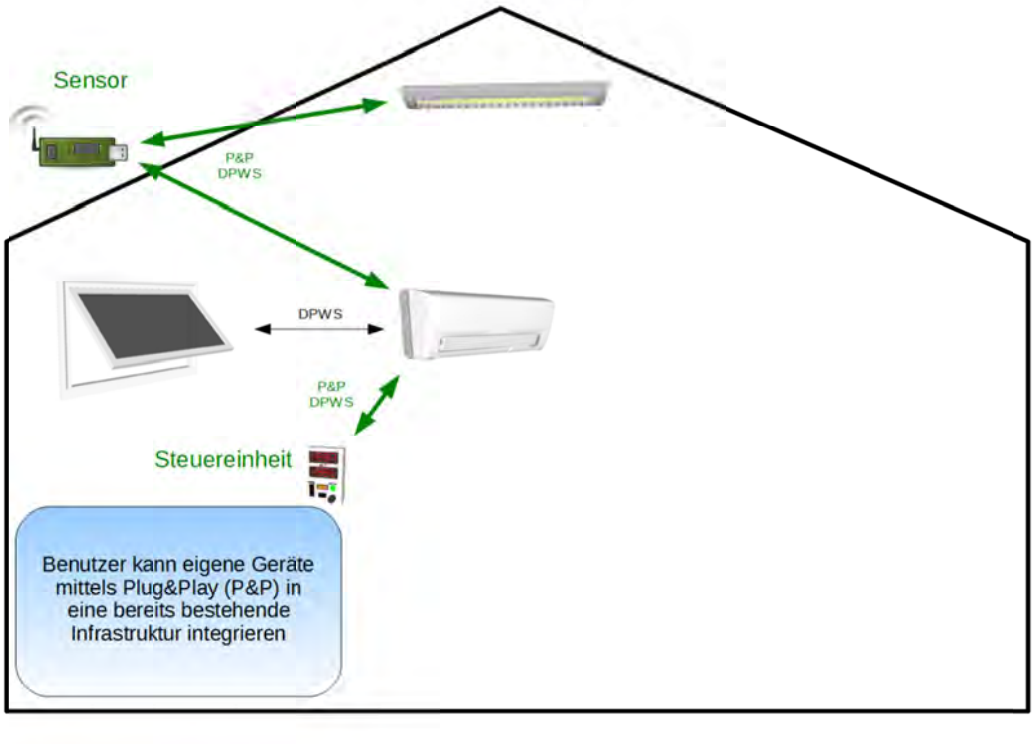

Abbildung 21: Demonstrationsszenario - Dynamische Integration neuer Geräte

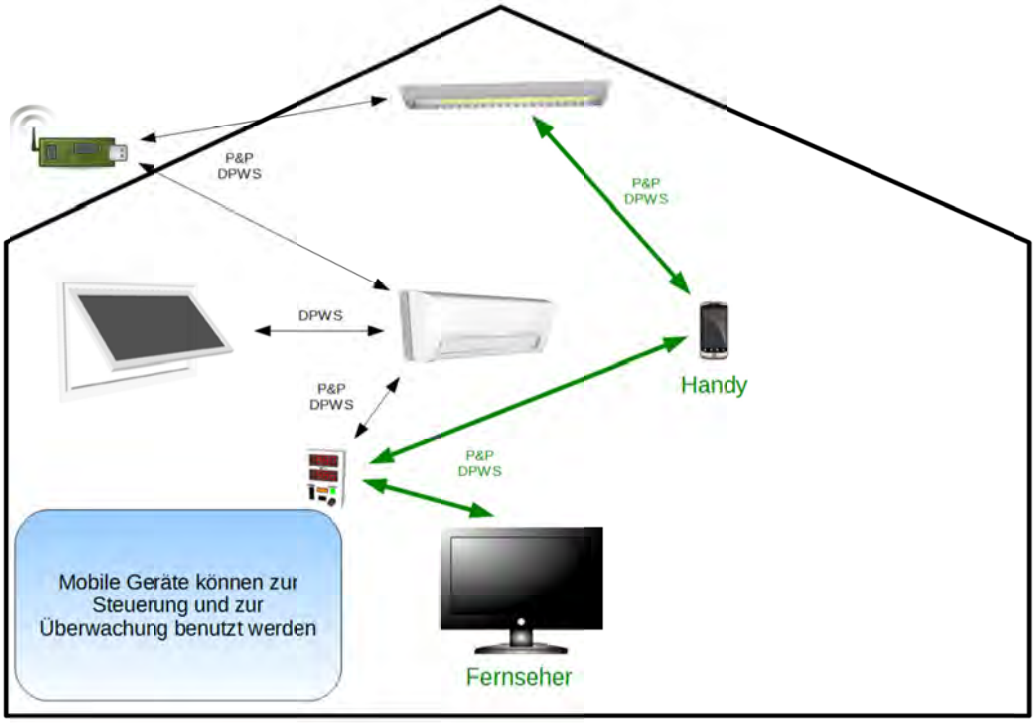

Abbildung 22: Demonstrationsszenario - Benutzung mobiler Geräte zur Smart-Home-Steuerung (P&P – Plug&Play)

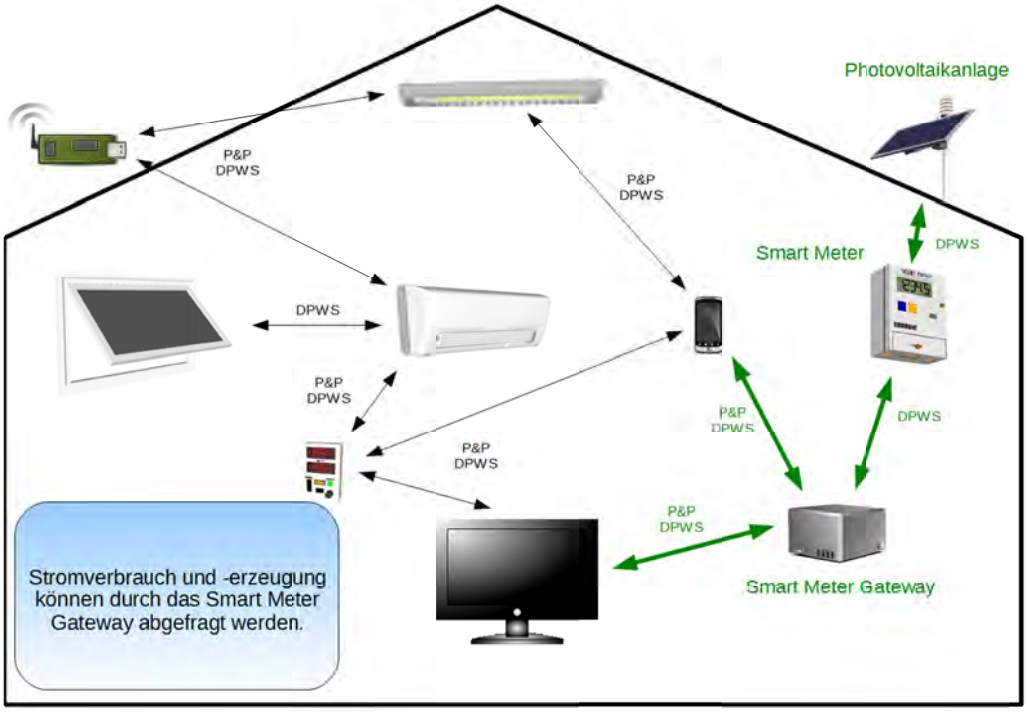

Abbildung 23: Demonstrationsszenario - Abfrage der Verbrauchsdaten mittels Smart Meter Gateway (P&P – Plug&Play)

8.1. Umsetzung

Die meisten Geräte, die in der Demonstration eingesetzt wurden, basieren auf dem Einplatinencomputer Raspberry Pi, Model B (siehe Abbildung 24). Die Spezifikationen von Raspberry Pi sind in Tabelle 12 aufgeführt. Als Betriebssystem kam Raspbian Linux zum Einsatz. Das Board wurde zusätzlich mit einem WLAN-Adapter erweitert, um eine drahtlose Kommunikation zu ermöglichen. Als drahtloser Sensor wurde SunSPOT verwendet (siehe Abbildung 25). Die Spezifikationen von SunSPOT können Tabelle 13 entnommen werden.

Abbildung 24: Raspberry Pi, Model B

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, I²C, UART
ROM	SD (SDHC und SDXC)
Leistungsaufnahme	max. 3,5W (700mA)
Stromversorgung	5 V

Tabelle 12: Raspberry Pi-Spezifikationen (Model B)

Abbildung 25: SunSPOT-Sensor

Größe	41 mm x 23 mm x 70 mm
CPU	ARM (400 MHz)
RAM	1 MB
ROM	8 MB (Flash)
Netzwerk	2.4 GHz IEEE 802.15.4, IR
Schnittstellen	4 Digitale und 4 analoge Pins, I²C, UART
Stromversorgung	3.7V 770 mAh Lithium-Ionen Akku
Sensoren	Beschleunigungssensor, Temperatursensor, Lichtsensor

Tabelle 13: SunSPOT-Spezifikationen

Um die elektrischen Geräte wie z.B. die Klimaanlage (Ventilator) anzusteuern bzw. Benutzereingaben mittels Tasten entgegenzunehmen, wurde eine zusätzliche Schaltung entwickelt. Die Ein- und Ausgänge der Schaltung werden mit den GPIO-Pins von Raspberry verbunden. Wie in Abbildung 26 beispielhaft dargestellt ist, wird die Klimaanlage mit dem GPIO-Pin 17 angesteuert. Eine Taste der Klimaanlage wird von GPIO-Pin 18 ausgelesen.

Um die DPWS-Funktionalität auf Geräten zu ermöglichen, wurde Java Multi Edition DPWS Stack (JMEDS) verwendet. Die JMEDS-API ist unter [48] zu finden. Da der SunSPOT-Sensor keine vollwertige Java Virtual Machine zur Verfügung stellt, wurde eine speziell angepasste JMEDS-Version benutzt. Die Funktionalität der Geräte wird am Beispiel der Klimaanlage erklärt. Da die Klimaanlage sowohl einen Service zur Verfügung stellt als auch andere Services in Anspruch nimmt, hat diese sowohl DPWS Device- als auch DPWS Client-Funktionalitäten. Das DPWS Device liefert eine Device-Beschreibung in Form von Metadaten und eine Service-Beschreibung in Form von WSDL. Die WSDL-Service-Beschreibung der Klimaanlage wurde bereits in Abschnitt 7 erklärt. In der prototypischen Umsetzung besitzt die Klimaanlage zwei Tasten, die für die Temperatureinstellung benutzt werden. Das Display der Klimaanlage kann die Temperatur von einem angebundenen Sensor sowie einen vom Benutzer eingestellten Sollwert anzeigen. Nachdem die Klimaanlage eine Pairing-Anfrage bekommen hat, wird dieses dem Benutzer auf dem Display signalisiert. Um eine Pairing-Anfrage selbst zu initiieren oder eine erhaltene Pairing-Anfrage zu bestätigen, müssen die beiden Tasten gleichzeitig gedrückt werden. Werden die Tasten innerhalb der Pairing-Zeit von 10 s nicht gedrückt, wird die Anfrage verworfen.

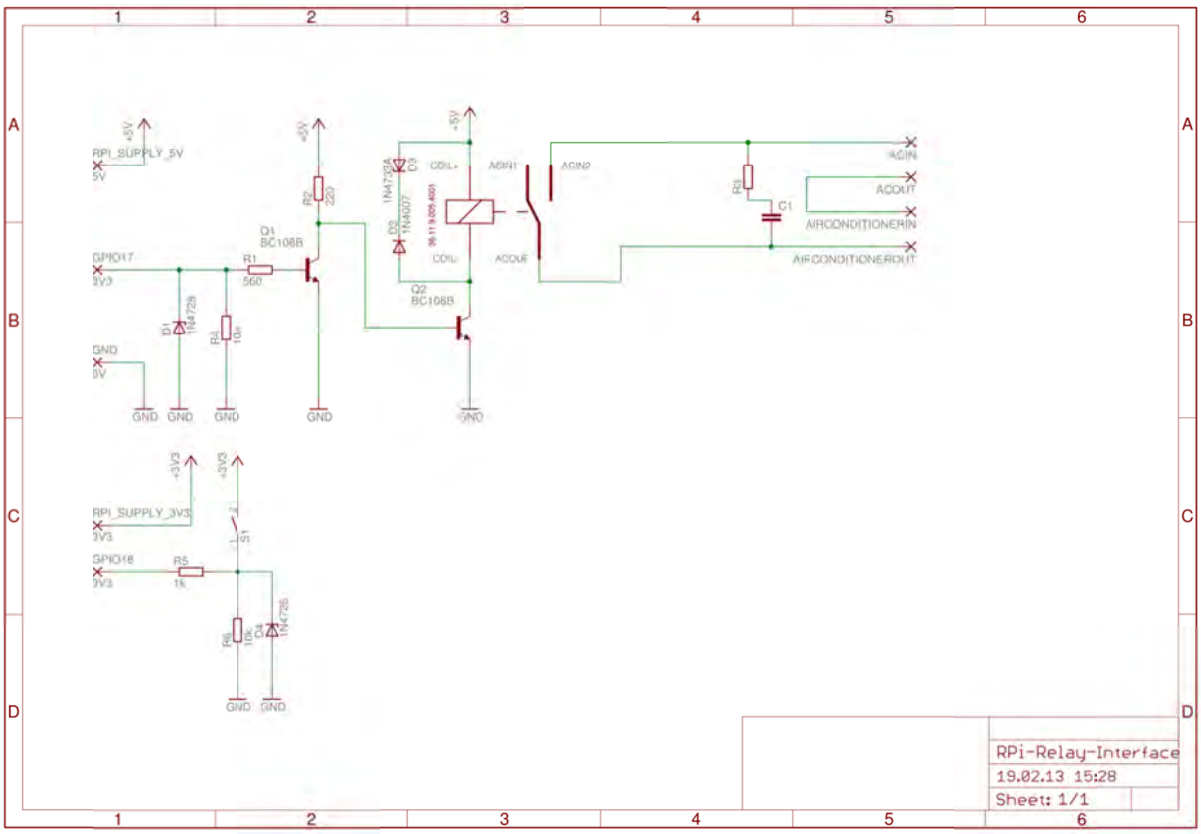

Abbildung 26: Zusatzschaltung zur Ansteuerung von elektrischen Geräten und Nutzerinteraktion mittels Tasten

Die Verarbeitung der Nachrichten innerhalb der Klimaanlage erfolgt in den folgenden Schritten. Nachdem eine DPWS-Nachricht empfangen wurde, wird diese zunächst geparst. Bei SOAP-over-HTTP oder SOAP-over-CoAP Bindings werden als erstes HTTP Header oder CoAP Options verarbeitet. Anschließend wird die SOAP-Nachricht innerhalb von HTTP Body oder CoAP Payload geparst. Bei SOAP-over-UDP Binding beginnt die Verarbeitung gleich mit der SOAP-Nachricht. Je nach SOAP-Kodierung wird die Nachricht mit einem XML- oder einem EXI-Parser geparst. Der Parser stellt dann die geparsten Strukturen in einem internen Datenformat dem Dispatcher zur Verfügung. Mehrere Verbindungen werden dabei durch einen Kommunikationsmanager verwaltet. Je nachdem, welche Nachricht empfangen wurde, wird die entsprechende Verarbeitungsroutine im Kern des Stacks aufgerufen. Handelt es sich beispielsweise um eine Service Invocation, wird zunächst entschieden, welche Service-Operation angefragt wurde. Danach wird die Verarbeitungsroutine dieser Service-Operation aufgerufen und die empfangenen Daten der speziellen Anwendung übergeben. Die spezielle Anwendung kann beispielsweise die Klimaanlage sein. Die spezielle Anwendung soll dann bei Bedarf die Daten zur Verfügung stellen, die als Antwort zurückgeschickt werden sollen. Bei einer fehlerhaften Anfrage kann auch eine Fehlermeldung erzeugt werden.

Der DPWS Client, der auch Teil der Software der Klimaanlage ist, sorgt für das Auffinden kompatibler Services wie z.B. den eines Sensors. Wie in Abschnitt 7 bereits erklärt wurde, verschickt die Klimaanlage eine Probe-Nachricht, um nach Services zu suchen. Die zweite Aufgabe des DPWS-Clients der Klimaanlage ist die Kommunikation mit einem Service. Hierbei können beispielsweise Werte von einem Sensor-Service abgefragt werden. Ähnlich wie bei der Device-Funktionalität erstellt die spezielle Anwendung der Klimaanlage Daten, die für eine Anfrage notwendig sind. Diese Daten

werden dem DPWS Stack übergeben und die Anfrage wird verschickt. Trifft eine Antwort auf die Anfrage ein, wird diese wie oben beschrieben verarbeitet und die empfangenen Daten werden der speziellen Anwendung bereitgestellt.

Der beschriebene Verarbeitungsablauf ist in Abbildung 27 dargestellt. Als Anwendung kann beispielsweise die Klimaanlage fungieren. Die einzelnen Stack-Teile wie z.B. HTTP- oder XML-Parser können durch CoAP- oder EXI-Parser ausgetauscht werden. Dies wurde im Rahmen des Projektes auch realisiert. Die Anwendung kann dabei beliebig sein. Diese bekommt die für sie notwendigen Daten oder stellt anderen Anwendungen Daten bereit. DPWS tritt hierbei als Vermittler auf, um die Daten zwischen den Anwendungen transparent auszutauschen.

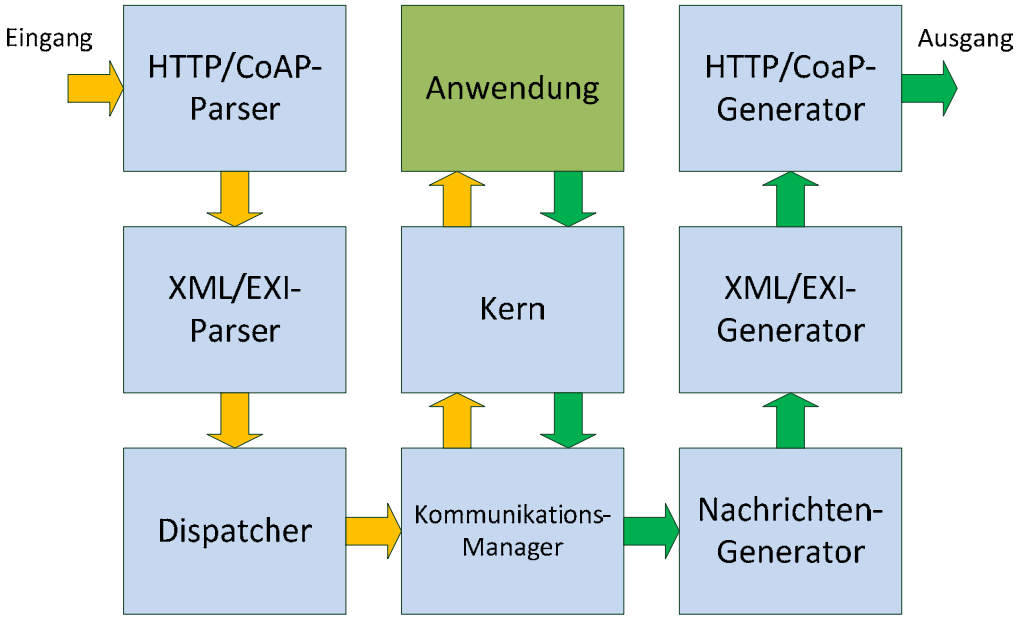

Abbildung 27: DPWS-Verarbeitungsprozess

8.2. Entwickelte Software

In Tabelle 14 ist ein Überblick über die im Rahmen des Projektes entwickelten Programme bzw. eingesetzten Bibliotheken gegeben. Die eingesetzte Bibliothek JMEDS sowie dpws_ws4d_basestation_host und dpws_ws4d_spot wurden im Rahmen des Projektes an die beschriebene Demonstration bzw. das entwickelte DPWS-Profil für Smart Metering angepasst. Alle eingesetzten Programme und Bibliotheken sind in der Programmiersprache Java entwickelt. Zur Ausführung der Programme ist Java Runtime Environment (JRE) ab Version 1.6 notwendig. Die Programme für den PC haben grundsätzlich keine besonderen Hardware-Anforderungen. Die Programme für das Raspberry Pi erfordern lediglich einen GPIO-Controller sowie den entsprechenden Java-Treiber Pi4J. Über den GPIO-Controller wurde die Interaktion mit dem Nutzer mittels Tasten realisiert. Das Programm AndroidDPWSExplorer ist grundsätzlich auf jedem Smartphone mit der Android Version 4.0 und höher lauffähig. Das Programm dpws_ws4d_basestation_host stellt das Gateway zum 6LoWPAN-Netzwerk dar und erfordert für eine korrekte Funktion die RXTX-Bibliothek. Diese ist ein Java-Treiber für die serielle Schnittstelle. Über diese Schnittstelle erfolgt die Kommunikation mit dem 6LoWPAN-Transceiver. RXTX kann direkt aus den Raspbian Repositories bezogen werden und wird daher als Teil des Betriebssystems angesehen. Das Programm dpws_ws4d_spot ist nur auf dem SunSPOT-Sensor lauffähig.

Name	Entwickler	Plattform	Erklärung
AirConditioner	URO	RPi	Implementierung einer DPWS-Klimaanlage (Demo)
AndroidDPWSExplorer	URO	Handy	Implementierung eines generischen DPWS-Clients auf dem Handy (Demo)
ControlUnit	URO	RPi	Implementierung einer DPWS-Wandsteuereinheit für die Klimaanlage (Demo)
EnergyMeter	URO	RPi	Implementierung eines DPWS Smart Meters (Demo)
Exificient	Siemens AG	PC	EXI-Bibliothek zum Lesen und Schreiben von EXI-Streams
GPIO	URO	RPi	Bibliothek zur Ansteuerung der Ein- und Ausgabe-Pins auf dem Raspberry Pi (Demo)
JMEDS	Materna/URO	Alle	DPWS-Bibliothek
LightBulb	URO	RPi	Implementierung einer DPWS-Lampe (Demo)
Pi4J	Pi4J Project	RPi	Java-Treiber für RPi GPIO Controller
SmartWindow	URO	RPi	Implementierung eines DPWS-Fensters (Demo)
SMLEnergyMeterEXI	URO	PC	Eine prototypische Umsetzung des DPWS-Profils für Smart Metering, mit deren Hilfe der Ansatz untersucht wurde
dpws_ws4d_basestation_host	TCD/URO	RPi	Eine Anwendung zur Umsetzung zwischen den Medien 6LoWPAN und Ethernet/WLAN. Diese ist zur Anbindung des Sensors notwendig (Demo)
dpws_ws4d_spot	TCD/URO	SunSPOT	Implementierung des Sensors (Demo)

* URO – Universität Rostock, TCD – Trinity College Dublin, RPi – Raspberry Pi

Tabelle 14: Überblick über die eingesetzte Software

Die eingesetzte Software kann aus folgenden Quellen bezogen werden:

- Exificient: http://exificient.sourceforge.net
- JMEDS: http://sourceforge.net/projects/ws4d-javame
- Pi4J: http://pi4j.com
- Entwickelte Software: https://gitlab.amd.e-technik.uni-rostock.de/vlado.altmann/intelligente-dienste/tree/master

8.3. Kostenbetrachtung

In diesem Abschnitt sollen die zusätzlichen Kosten betrachtet werden, die für die Aufwertung der Geräte mit einem zusätzlichen Kommunikationsmodul anfallen würden. Die Kosten der Geräte (ohne

Kommunikationsmodul) sind stark herstellerabhängig und werden von der Kommunikationserweiterung grundsätzlich nicht beeinflusst. Bei der Kostenbetrachtung werden die Kosten für den Prototypen bzw. Anschaffungskosten der Komponenten im Einzelhandel betrachtet. Der tatsächliche Produktpreis kann deutlich geringer sein. Im Rahmen des Projektes wurde für die Kommunikationssteuerung diverser Geräte das Raspberry Pi Model B eingesetzt, welches dem Institut MD der Universität Rostock in hinreichender Stückzahl zur Verfügung stand. Eine Umsetzung wäre mit der günstigeren Variante Raspberry Pi Model A ebenfalls möglich. Zu einer Annäherung an den Endproduktpreis wird für die Kalkulation der Einzelhandelspreis für das Raspberry Pi Model A genommen. In Tabelle 15 sind die notwendigen Komponenten für die Umsetzung einer Kommunikationslösung dargestellt. Da die Software des Kommunikationsmoduls komplett quelloffen und lizenzfrei ist, fließen in die Kostenkalkulation lediglich die Hardware-Preise ein. Der Gesamtpreis des prototypischen Kommunikationsmoduls beläuft sich somit auf 38,99 €. Der Preis des Kommunikationsmoduls kann darüber hinaus bei Geräten, die bereits eine Steuerungselektronik besitzen wie z.B. eine Klimaanlage deutlich geringer ausfallen. In diesem Fall kann dieselbe Steuerungselektronik mehrere Aufgaben u.a. die Kommunikationssteuerung übernehmen. Die Prototypkosten stellen darüber hinaus ein Vielfaches der Endproduktkosten dar.

Komponente	**Stückpreis**	**Anzahl**
Raspberry Pi Model A	18,87 €	1
WLAN-Adapter	5,31 €	1
SD Card	4,12 €	1
Spannungswandler	3,99 €	1
Kabel	1,66 €	1
Relais	1,63 €	1
Transistor	0,78 €	2
Weitere Bauelemente (Widerstände, Dioden etc.) Kit	~1 €	1
Lochrasterplatine	0,85 €	1

Tabelle 15: Komponentenpreise für das Kommunikationsmodul

9. Projektevaluierung

Im Folgenden werden die gesetzten Ziele erfasst und an das Projekt gestellten Fragen zusammenfassend beantwortet:

Interoperabilität zwischen isolierten Anwendungen der Gebäudeautomation (zwischen Energiemanagement, Smart Home, Smart Metering, betreutem Wohnen oder der Telematik):
Die Interoperabilität zwischen isolierten Anwendungen konnte durch DPWS erreicht werden. Es wurde am Beispiel von Smart Home und Smart Metering gezeigt, dass das generische offene Protokoll DPWS für den Einsatz in beiden Teilbereichen geeignet ist. Auf dieselbe Weise kann DPWS auf weitere Bereiche erweitert werden (siehe Kapitel 5, 7, 8).

Darstellung der technischen Möglichkeiten und Grenzen anhand eines Szenarios einer Gewerbeimmobilie:
Innerhalb der Arbeitsgruppe wurde das Szenario für Smart Home und Smart Metering neu definiert (siehe Kapitel 8). Die technischen Möglichkeiten von DPWS sind prinzipiell unbegrenzt. Die Geräte müssen jedoch DPWS-fähig sein. Die nicht DPSW-fähigen Geräte können über Gateways angebunden werden. Das stellt jedoch kein Hindernis dar, da zukünftig Smart Home-Gateways zu erwarten sind.

Evaluation und Demonstration von DPWS bezüglich Kosten:
Die zu erwartenden Kosten sind gering, da DPWS sowohl Standard-Hardware als auch standardkonforme Software nutzt (siehe Abschnitt 8.3). Die Software selbst ist quelloffen, frei und kostenlos verfügbar.

Evaluation und Demonstration von DPWS bezüglich des Einsatzspektrums:
Im Rahmen des Projektes wurde ein Cross Domain-Einsatz von DPWS in Smart Home und Smart Metering betrachtet (siehe Kapitel 5, 7, 8).

Evaluation und Demonstration von DPWS bezüglich Datensicherheit (Security):
DPWS erfüllt vollständig die BSI-Richtlinie für sichere Kommunikation. Der vollständige Satz an Sicherheitsfeatures wie Authentifizierung, Integrität und Datenschutz wird von DPWS unterstützt (siehe Kapitel 5, 8).

Evaluation und Demonstration von DPWS bezüglich Skalierbarkeit:
Die Skalierbarkeit von DPWS wurde mit zusätzlichen vorgeschlagenen Anpassungen nachgewiesen (siehe Kapitel 6).

Evaluation und Demonstration von DPWS bezüglich Betriebssicherheit (Safety):
Die Betriebssicherheit ist nach unserer Einschätzung mindestens ebenso groß wie die der durch DPWS zu substituierenden Software. Die im Projektverlauf auftretenden Schwächen bzgl. der Skalierbarkeit wurden durch die neuen Vorschläge vollständig behoben. In unseren umfangreichen Untersuchungen mit Prototypen und Demonstratoren wurden keinerlei Probleme bzgl. der Betriebssicherheit festgestellt.

Evaluation und Demonstration von DPWS bezüglich Echtzeitverhalten:
DPWS steht ursprünglich kein echtzeitfähiges Protokoll dar. Die vorgeschlagenen Kompressionsmethoden können jedoch das Echtzeitverhalten von DPWS verbessern. Um eine

Echtzeitfähigkeit von DPWS zu garantieren, sollen Quality of Service-Prioritäten auf Stack- und Netzwerkebene gewährleistet werden. Für eine sehr hohe Performance ist weiterhin eine Beschleunigung des XML/EXI-Parsings durch Hardware-Unterstützung bzw. durch die Erhöhung der CPU-Ressourcen möglich.

Evaluation und Demonstration von DPWS bezüglich Marktpotentials:
DPWS gewinnt zunehmend an Bedeutung. Marktpotenzial besteht, die Verbreitung am Markt muss jedoch von einem großen Hersteller vorangetrieben werden. Nach unserer Kenntnis wird dies zukünftig u.a. durch die Firma Schneider Electric geschehen. Es wird ein großes Potenzial besonders im Bereich Smart Home gesehen, da es hier aktuell keine etablierten Standards gibt. Laut einer Analyse der Firma Cisco soll es im Jahr 2020 50 Milliarden vernetzte Geräte geben. Da DPWS vollständig auf Internettechnologien basiert, lässt es sich perfekt in die Vision Internet der Dinge integrieren.

Demonstration der genannten Aspekte anhand eines Anwendungsszenarios:
Das Anwendungsszenario ist in Kapitel 8 beschrieben.

Empfehlungen für die Gebäudeautomation (Facility Management):
Ausgehend aus den Projektergebnissen können folgende Empfehlungen abgeleitet werden:

- Verteilte eingebettete Systeme sollten einen offenen Cross Domain-Standard wie DPWS einsetzen
- Spezielle oder proprietäre Protokolle lassen sich auf DPWS abbilden
- Zur Datenreduktion von DPWS empfiehlt sich CoAP zur HTTP-Kompression und EXI zur XML-Kompression
- Zur Einhaltung der BSI-Richtlinie für Smart Metering sollte CoAP-over-TCP-Binding verwendet werden
- Die Geräte-Kopplung lässt sich mittels einer Taste mit Hilfe von DPWS-Nachrichten „Probe – ProbeMatch“ bzw. „Hello – Resolve – ResolveMatch“ realisieren
- Für eine sichere Geräte-Kopplung können die Zertifikate mittels NFC ausgetauscht werden
- Zur Service-Discovery sollte WS-Discovery verwendet werden
- Eine Skalierbarkeit der WS-Discovery kann durch folgende Optimierungen erreicht werden:
    - Einfügen der Hardware Adresse
    - Dynamisches Delay
    - Knotengruppierung
    - Optimierung der Paketwiederholung
    - Network Size Discovery

Erweiterung bestehender Standards hin zu offenen Lösungen:
Die bestehenden Standards können wie im Projekt erfolgreich gezeigt durch Abbildung auf Web Services offene Schnittstellen anbieten (siehe Kapitel 5).

Vorschlag zu offenen Lösungen in Form von spezialisierten Profilen:
Im Rahmen des Projektes wurde beispielhaft ein spezialisiertes Profil für Smart Metering auf Basis von SML erarbeitet (siehe Kapitel 5).

Exploration bestehender Standards (Bussysteme der Gebäudeautomation wie z.B. LON, BACnet, KNX, KNX-RF, ZigBee, Z-Wave, EnOcean) im Hinblick auf die Belange von Smart Metering, Energiemanagement, Smart Home, Heimautomation, Ambient Assisted Living, Telematik:
Viele Bussysteme der Gebäudeautomation wie z.B. LON, BACnet, KNX, KNX-RF, ZigBee, Z-Wave, EnOcean erfüllen nicht oder nur teilweise die hohen Sicherheitsstandards der BSI-Richtlinie. Daher schreibt die BSI-Richtlinie den Einsatz von M-Bus, DLMS/COSEM und SML für Smart Metering vor (siehe Kapitel 2). Die bestehenden Standards der klassischen Gebäudeautomation bzw. Telematik sind zwar technisch für den Einsatz in Heimautomation oder Ambient Assisted Living geeignet, bringen jedoch sehr hohe Installation- und Wartungskosten mit sich. Das liegt vor allem an der fehlenden Möglichkeit einer automatischen Konfiguration der Geräte. Für die Installation eines neuen Gerätes oder die Umkonfiguration einer bestehenden Installation sind nicht nur Fachkenntnisse, sondern oft teure proprietäre Tools erforderlich. Deshalb haben sich diese Standards im Smart Home-Bereich bis heute nicht durchgesetzt.

Evaluation der bestehenden Standards hinsichtlich ihrer Integrationsfähigkeit in WS4D:
Die bestehenden Standards lassen sich auf WS4D-Technologien wie DPWS abbilden, wie in Kapitel 5 und Anhang A gezeigt wurde. Die Abbildung erfolgt auf der Anwendungsschicht. Für die Abbildung sollen zunächst die Bestandteile eines Protokolls wie Datentypen, Elemente, Attribute, Nachrichten und Übertragungsarten identifiziert werden. Diese sollen anschließend, wie in Kapitel 5 und Anhang A gezeigt, auf die jeweiligen generischen Bestandteile von DPWS abgebildet werden. Darüber hinaus kann zur Reduzierung der übertragenen Datenmengen eine sehr effiziente Kompression, wie in Abschnitt 5.2 beschrieben, angewandt werden. Dadurch kann das resultierende DPWS-Profil sogar ähnliche Datenmengen erzeugen wie das ursprüngliche (binäre) Protokoll.

Damit auch ein Feldgerät auf DPWS umgestellt werden kann, sollte es über eine Netzwerkschnittstelle zu einem Übertragungsmedium verfügen, das eine IP-Unterstützung bietet. Darüber hinaus sollte das Gerät über die für DPWS notwendigen Hardware-Ressourcen (mindestens 46 KB ROM und 7 KB RAM) verfügen. Da viele Feldgeräte eine spezielle proprietäre Hardware besitzen ist u.U. eine Portierung des Quellcodes eines DPWS-Stacks erforderlich. Geräte, auf denen DPWS aus diversen Gründen nicht eingesetzt werden kann, können über ein Gateway eingebunden werden.

Berücksichtigung neuer Entwicklungstendenzen und Ansätze (z.B. 6LoWPAN, digitalStrom):
Der 6LoWPAN-Standard kann für die drahtlose Low-Power-Übertragung von DPWS benutzt werden (siehe Kapitel 2, 5, 7). digitalStrom ist eine proprietäre Technologie für die Kommunikation über das Stromnetz. Die Anbindung der digitalStrom-Geräte ist über die REST API in WS4D-Technologien möglich.

Berücksichtigung und Integration von XML als offene Sprache in Kombination mit den Datenübertragungstechnologien moderner Internetstandards:
DPWS setzt standardmäßig XML als Datenbeschreibungssprache ein (siehe Kapitel 3).

Ablösung der historisch gewachsenen kryptischen Busprotokolle durch XML:
Eine Möglichkeit der Abbildung der Busprotokolle auf XML wurde in Kapitel 5 gezeigt.

Definition offener Integrationslösungen:
DPWS wird als offene Integrationslösung vorgeschlagen (siehe Kapitel 5, 7).

Ableitung von Empfehlungen für offene Integrationslösungen:
Siehe Punkt Empfehlungen für die Gebäudeautomation.

Adaption bestehender Standards:
Im Rahmen des Projektes wurde SML als bestehender Smart Metering Standard adaptiert (siehe Kapitel 5).

Erarbeitung spezifischer Profile:
Im Rahmen des Projektes wurde beispielhaft ein spezialisiertes Profil für Smart Metering auf Basis von SML erarbeitet (siehe Kapitel 5).

Definition und Demonstration eines Zielszenarios:
Das Zielszenario ist in Kapitel 8 beschrieben.

Integration der Ergebnisse in kommende Standardisierungsbemühungen:
Die im Rahmen des Projektes entstandenen Ergebnisse wie Smart Meter-Profil für Embedded Web Services und Optimierung von WS-Discovery für Großnetze wurden auf internationalen Konferenzen vorgestellt sowie in der digitalen Bibliothek des führenden Institute of Electrical and Electronics Engineers (IEEE) Xplore veröffentlicht (siehe Kapitel 12). An diese Arbeiten können sich die zukünftigen Standardisierungsbemühungen anlehnen.

10. 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. Im Rahmen des Projekts 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 ein analytischer Beweis geführt, welcher die Allgemeingültigkeit des vorgeschlagenen Ansatzes bestätigt hat. Damit wurde gezeigt, dass jedes beliebige Protokoll der Gebäudeautomatisierung, 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.

Für die Suche nach Geräten in Großnetzen wie Smart Meter Networks wurden Optimierungen für WS-Discovery vorgeschlagen. Hierbei werden die statischen Parameter durch die dynamischen Parameter ersetzt. Der Discovery-Mechanismus kann demnach für jede beliebige Netzwerkgröße und Datenrate angepasst werden. Der vorgeschlagene Ansatz kann darüber hinaus die Lastspitzen glätten und die resultierende Peak-Datenrate um 50 % senken. Das Gerät, das eine Discovery-Anfrage stellt, wird dadurch deutlich entlastet. Die dynamischen Parameter erlauben es, die Datenrate auf einen gewünschten herstellerspezifischen Wert einzustellen, was die Betriebssicherheit erhöht. Es wurde dabei gezeigt, dass Web Services einen zusätzlichen Mehrwert durch die dynamische Suche nach Geräten für die große Smart Meter- oder Gebäudeautomationsnetzwerke schaffen können. Die Skalierbarkeit der Technologie kann durch die entsprechenden Optimierungen erreicht werden.

Ein solches Forschungsprojekt wie hier im Bereich Smart Metering / Smart Home ist derart komplex, dass sich erst bei der Bearbeitung das eigentliche technisch/wissenschaftliche Problem und Ansatzmöglichkeiten zur Lösung heraus kristallisieren. Es wurden hervorragende Ergebnisse mit großer Forschungs- und ebenso praktischer Relevanz erzielt. In diesem Projekt konnte erstmals für den Bereich Smart Metering/Smart Home nachgewiesen werden, dass mit offenen, standardkonformen und szenarienübergreifenden Ansätzen und innovativen Ideen die gleiche und sogar bessere Effizienz verglichen mit einer proprietären und anwendungsspezifischen Lösung erzielt werden kann. Dieses methodisch / konzeptionelle Ergebnis kann auf viele andere Bereiche der Automatisierungssysteme sinngemäß übertragen werden. Die Übertragung dieser Konzepte ist für einen Experten ohne weiteres möglich.

Datum	Ziel	Status
Okt. 2011	Untersuchung des Standes der Technik	
Jan. 2012	Definition eines Anwendungsszenarios	
Apr. 2012	Ableitung allgemeiner Anforderungen und Spezifikation bzgl. DPWS	
Jul. 2012	Ausarbeitung einer offenen Kommunikationslösung für die Gebäudeautomation	
Nov. 2012	Spezifikation der Service-Infrastruktur / Architektur des Gesamtsystems	
Mär. 2013	Ableitung von Empfehlungen für die Kommunikations- und Integrationslösung für die Gebäudeautomation	
Sep. 2013	Umsetzung einer Demonstration am Anwendungsszenario	

Abbildung 28: Zeitplan des Projektes

11. Quellenverzeichnis

[1] DLMS/COSEM User Association, “DLMS/COSEM for smart metering”, IEC 62056 Standard, 2011

[2] M. Wisy, “Smart Message Language 1.03”, EMSYCON GmbH, 2008.

[3] Enel S.p.A., “METERS AND MORE”, prEN/TS 5ZZZZ Draft, 2011.

[4] Echelon Corporation, “Local Operating Network (LON)”, EN 14908 Standard, 2011.

[5] KNX Association, “KNX”, EN 50090 Standard 2011.

[6] ASHRAE, “Building Automation and Control Networks (BACnet)”, ISO-Norm 16484-5, 2011.

[7] D. Driscoll und A. Mensch, “Devices profile for web services version 1.1”, OASIS Standard, 2009.

[8] A. Weidlich, S. Karnouskos und J. Ringelstein, “Technology trends for smarthouse/smartgrid,” European Commision, 2010.

[9] S. Rogai, “Telegestore Project Progress & Results”, *IEEE ISPLC,* 2007.

[10] B. Botte, V. Cannatelli und S. Rogai, “The Telegestore project in Enel’s metering system”, *18th International Conference on Electricity Distribution*, 2005.

[11] 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]. Verfügbar: http://www.echelon.com/company/press/2006/eonse.htm.

[12] J. Söderbom, “Smart Meter roll out experiences from Vattenfall”, Vattenfall, 2012.

[13] Bundesamt für Sicherheit in der Informationstechnik (BSI), „Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen“, Technische Richtlinie BSI TR-03109, 2011.

[14] National Institute of Standards and Technology (NIST), “NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0”, 2011.

[15] SMB Smart Grid Strategic Group (SG3), “IEC Smart Grid Standardization Roadmap”, 2010.

[16] M-Bus, EN 13757 Standard, 2011.

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

[18] G. Moritz, E. Zeeb und andere, “Device Profile for Web Services and the REST,” 8th IEEE International Conference on Industrial Informatics (INDIN), 2010.

[19] V. Modi und D. Kemp, “Web Services Dynamic Discovery (WS-Discovery) Version 1.1”, OASIS Standard, 2009.

[20] D. Box, L. F. Cabrera und andere, “Web Services Eventing (WS-Eventing)", W3C Draft, 2006.

[21] K. Ballinger, B. Bissett und andere, „Web Services Metadata Exchange 1.1 (WS-MetadataExchange)“, W3C Draft, 2008.

[22] A. Vedamuthu, D. Orchard und andere, “Web Services Policy 1.5 – Framework”, W3C Standard, 2007.

[23] J. Alexander, D. Box und andere, „Web Services Transfer (WS-Transfer)“, W3C Draft, 2006.

[24] A. Nadalin, C. Kaler und andere, “Web Services Security: SOAP Message Security 1.1 (WS-Security)”, OASIS Standard, 2006.

[25] M. Gudgin, M. Hadley und T. Rodgers, “Web Services Addressing 1.0 - Core”, W3C Standard, 2006.

[26] T. Bellwood, S. Capell und andere, “UDDI Version 3.0.2”, OASIS Standard, 2004.

[27] M. Gudgin, M. Hadley und andere, “SOAP Version 1.2 Part 1: Messaging Framework”, W3C Standard, 2007.

[28] R. Jeyaraman, “SOAP-over-UDP Version 1.1”, OASIS Standard, 2009.

[29] M. Gudgin, M. Hadley und andere, “SOAP Version 1.2 Part 2: Adjuncts (Second Edition)”, W3C Standard, 2007.

[30] T. Bray, J. Paoli und andere, „ Extensible Markup Language (XML) 1.0 (Fifth Edition)“, W3C Standard, 2008.

[31] E. Christensen, F. Curbera und andere, “Web Services Description Language (WSDL) 1.1”, W3C Standard, 2001.

[32] R. Fielding, J. Gettys und andere, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616 Standard, 1999.

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

[34] C. Lerche, N. Laum und andere, “Implementing Powerful Web Services for Highly Resource-Constrained Devices,” Pervasive Computing and Communication Workshops, 2011.

[35] HomePlug PowerLine Aliance, “HomePlug GREEN PHY Specification Release Version 1.00”, 2010.

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

[37] DLMS/COSEM User Association, „Datenkommunikation der elektrischen Energiemessung - DLMS/COSEM - Teil 6-1: COSEM Objekt Identification System (OBIS)“, EN 62056-6-1 Standard, 2011.

[38] P. Deutsch, “GZIP file format specification version 4.3”, RFC 1952 Standard, 1996.

[39] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3”, RFC 1951 Standard, 1996.

[40] Z. Shelby, K. Hartke und andere, “Constrained Application Protocol (CoAP),” IETF Internet-Draft, 2012.

[41] J. Schneider und T. Kamiya, “Efficient XML Interchange (EXI) Format 1.0,” W3C Standard, 2011.

[42] D.C. Fallside und P. Walmsley, „XML Schema Part 0: Primer Second Edition“, W3C Standard, 2004.

[43] P. Leach, M. Mealling und R. Salz, “A Universally Unique IDentifier (UUID) URN Namespace”, RFC 4122 Standard, 2005.

[44] P.V. Biron, A. Malhotra, “XML Schema Part 2: Datatypes Second Edition”, W3C Standard, 2004.

[45] NS-3 Consortium, “Network Simulator 3 (ns-3)”, [Online]. Verfügbar: http://www.nsnam.org.

[46] D. C. Plummer, “An Ethernet Address Resolution Protocol”, RFC 826, 1982.

[47] R. Droms, “Dynamic Host Configuration Protocol”, RFC 2131, 1997.

[48] WS4D, “Java Multi Edition DPWS Stack (JMEDS)”, 2013. [Online]. Verfügbar: www.ws4d.org

12. Veröffentlichungen

- Vlado Altmann, Jan Skodzik, Frank Golatowski, Dirk Timmermann:
  Investigation of the Use of Embedded Web Services in Smart Metering Applications
  The 38th Annual Conference of the IEEE Industrial Electronics Society (IECON), pp. 6176-6181, Montreal, Kanada, Oktober 2012

- Vlado Altmann, Jan Skodzik, Peter Danielis, Frank Golatowski, Dirk Timmermann:
  Optimization of Ad Hoc Device and Service Discovery in Large Scale Networks
  The 18th IEEE Symposium on Computers and Communications (ISCC), pp. 833-838, Split, Kroatien, Juli 2013

A. Anhang

A.1. 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, welches 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.2. 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, welches 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.3. 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:**

z.B. 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